НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

«На правах рукопису»
УДК _____

До захисту допущено:
Завідувач кафедри
_____Сергій СТІРЕНКО
«_ __»_ _____2023 р.

Магістерська дисертація

на здобуття ступеня магістра
за освітньо-професійною програмою «Інженерія програмного забезпечення комп'ютерних систем» зі спеціальності 121 «Інженерія програмного забезпечення» на тему: «Платформа пошуку менторів для онлайн навчання»

Виконав: студент 6 курсу, групи ІМ-21мп
Мінченко Володимир Юрійович _____
(підпис)

Науковий керівник: доц. каф. ОТ, к.т.н., доцент
Волокита Артем Миколайович _____
(підпис)

Консультант з нормоконтролю:
проф. каф. ІСТ, д.т.н., професор
Жабін Валерій Іванович _____
(підпис)

Рецензент: доц. каф. ІСТ, к.т.н., доцент _____
Шимкович Володимир Миколайович (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Київ – 2023 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет (інститут) *Інформатики та обчислювальної техніки*
(повна назва)

Кафедра *Обчислювальної техніки*
(повна назва)

Рівень вищої освіти – другий (магістерський) за освітньо-професійною програмою

Спеціальність *121. Інженерія програмного забезпечення*
(код і назва)

Спеціалізація *121. Інженерія програмного забезпечення комп'ютерних систем*
(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
«___» _____ 2023 р.

ЗАВДАННЯ

на магістерську дисертацію студенту

Мінченку Володимиру Юрійовичу
(прізвище, ім'я, по батькові)

1. Тема дисертації Платформа пошуку менторів для онлайн навчання

Науковий керівник дисертації Волокита Артем Миколайович, доцент каф. ОТ, к.т.н.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «09» жовтня 2023 р. № 4115-с

2. Термін подання студентом дисертації 30.12.2023

3. Об'єкт дослідження процес взаємодії компонентів в архітектурі програмного забезпечення для онлайн платформи, призначеної для пошуку та залучення менторів, забезпечення ефективної взаємодії між учнями та

менторами, а також управління навчальними процесами у межах створеної системи.

4. Предмет дослідження Методи та технології проектування, розробки, тестування та оптимізації програмного забезпечення для створення і підтримки онлайн платформи пошуку менторів, включаючи інтерфейс користувача, систему рекомендацій, базу даних менторів та учнів, а також забезпечення безпеки та надійності функціонування платформи.

5. Перелік завдань, які потрібно розробити: визначення основних функцій платформи, вимог до інтерфейсу користувача, безпеки, навантаження, інтеграції з іншими сервісами, проектування архітектури системи, вибір стеку технологій, розробка моделі бази даних, створення архітектурного плану інтерфейсу користувача, серверної частини та інтеграцій, створення дизайну інтерфейсу, розробка і тестування користувацького досвіду для різних категорій користувачів (учні, ментори, адміністратори), програмування та імплементація функціоналу, інтеграція з зовнішніми API та сервісами електронної пошти, платіжних систем, соціальних мереж для автентифікації та інших потрібних інтеграцій, проведення юніт-тестування, інтеграційного тестування, навантажувального тестування та приймального тестування для забезпечення стабільності, безпеки та відповідності функціоналу вимогам налаштування серверного середовища, розгортання програмного забезпечення, налаштування бази даних та інших компонентів інфраструктури, розробка плану підтримки, моніторинг системи, виправлення помилок, оновлення функціоналу та оптимізація продуктивності, створення технічної документації, гайдів для користувачів та адміністраторів, а також документації коду, імплементація заходів безпеки, таких як шифрування, аутентифікація, захист від загроз і забезпечення конфіденційності даних.

6. Консультанти розділів дисертації:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1.	д.т.н., проф. Жабін В. І.		
2.	д.т.н., проф. Жабін В. І.		
3.	д.т.н., проф. Жабін В. І.		
4.	д.т.н., проф. Жабін В. І.		

7. Дата видачі завдання 01.09.2023

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	*Затвердження теми дослідження. Визначення предмету дослідження*	*01.09.2023*	
2	*Дослідження існуючих рішень та проблем конструювання трафіка в досліджуваній області*	*05.09.2023*	
3	*Побудова та обґрунтування архітектурного рішення*	*23.09.2023*	
4	*Розробка структур окремих інтерфейсів програми*	*01.10.2023*	
5	*Програмна реалізація*	*25.10.2023*	
6	*Розробка стартап-проекту*	*25.11.2023*	
7	*Оформлення пояснювальної записки*	*25.12.2023*	
8	*Захист*	*15.01.2024*	

Студент В. Ю. МІНЧЕНКО

Науковий керівник А.М. ВОЛОКИТА

РЕФЕРАТ

на магістерську дисертацію виконану на тему:

Платформа пошуку менторів для онлайн навчання

студентом Мінченком Володимиром Юрійовичем

Дипломна робота другого (магістерського) рівня вищої освіти на тему «Платформа пошуку менторів для онлайн навчання» складається зі вступу та 4 розділів. Загальний обсяг роботи: 126 сторінок, 12 таблиць, 50 рисунків, 3 додатки. Перелік посилань нараховує 15 найменувань.

Актуальність роботи: Засвоєння інформації в наш час є найбільш цінним активом. У умовах постійних змін на ринку праці виникає критична необхідність у підвищенні кваліфікації або навіть кардинальній зміні професійної сфери. Онлайн-освіта, яка розвивається стрімкими темпами, особливо під час карантину чи повномасштабного вторгнення, коли деякі навчальні заклади були змушені перейти до онлайн-формату, відповідає на ці потреби. Вкрай зростає потреба у сервісах та платформах для отримання знань на відстані, забезпечуючи, щоб рівень освіти не втрачався.

Дистанційне навчання стало не просто вибором, а необхідністю, пропонуючи гнучкість та мобільність, які багато хто шукає, дозволяючи навчатися за власним розкладом. Веб-платформи для вивчення онлайн курсів, як альтернатива Інтернет-коледжам, пропонують гнучкість та можливість дослідження специфічних або незвичних тем. Однак, з розвитком таких платформ, з'являється потреба у кваліфікованих менторах, які можуть надавати індивідуальну підтримку та керівництво, адаптуючи навчання під індивідуальні потреби кожного учня.

У цьому контексті, платформа пошуку менторів для онлайн навчання набуває особливої актуальності. Вона може радикально покращити якість онлайн освіти, забезпечуючи адаптацію навчальних планів, сприяння мотивації та залученості учнів, формування глибокого розуміння матеріалу та сприяння крос-культурному обміну та професійному зростанню. Така платформа зможе забезпечувати не тільки гнучкість та доступність, але й

особистісно-орієнтоване навчання, яке стає все більш важливим у світі, що швидко змінюється.

Мета роботи: Основна мета полягає в дослідженні, розробці та оцінці інформаційно-технологічного рішення для ефективної веб-платформи, яка забезпечує з'єднання учнів з кваліфікованими менторами в різних областях знань для індивідуального та цілеспрямованого навчання. Ця платформа має на меті оптимізувати процес пошуку та взаємодії між учнями та менторами, підвищити якість онлайн освіти через індивідуалізовані навчальні плани та методи, а також забезпечити гнучкість та доступність навчання для широкого кола користувачів. Окрім технічної реалізації платформи, мета включає розробку стратегій для підтримки та залучення користувачів, забезпечення безпеки даних та створення ефективної системи управління та моніторингу навчального процесу.

Завдання дослідження: під час розробки застосунка було опрацьовано наступні етапи:

- Аналіз вимог та збір специфікацій: Визначення основних функцій платформи, вимог до інтерфейсу користувача, безпеки, навантаження, інтеграції з іншими сервісами.
- Проектування архітектури системи: Вибір стеку технологій, розробка моделі бази даних, створення архітектурного плану інтерфейсу користувача, серверної частини та інтеграцій.
- Розробка користувацького інтерфейсу: Створення дизайну інтерфейсу, розробка і тестування користувацького досвіду для різних категорій користувачів (студенти, ментори, адміністратори).
- Програмування та імплементація функціоналу: Розробка основних компонентів системи, включаючи систему пошуку та відбору менторів, управління профілями, систему зворотного зв'язку та оцінювання.
- Інтеграція з зовнішніми API та сервісами: Підключення до сервісів електронної пошти, платіжних систем, соціальних мереж для

автентифікації та інших потрібних інтеграцій.

- Тестування: Проведення юніт-тестування, інтеграційного тестування, навантажувального тестування та приймального тестування для забезпечення стабільності, безпеки та відповідності функціоналу вимогам.
- Запуск платформи та деплоймент: Налаштування серверного середовища, розгортання програмного забезпечення, налаштування бази даних та інших компонентів інфраструктури.
- Підтримка та оновлення: Розробка плану підтримки, моніторинг системи, виправлення помилок, оновлення функціоналу та оптимізація продуктивності.
- Документування: Створення технічної документації, руководств для користувачів та адміністраторів, а також документації коду.
- Забезпечення безпеки: Імплементація заходів безпеки, таких як шифрування, аутентифікація, захист від загроз і забезпечення конфіденційності даних.

Об’єкт дослідження: Об'єктом дослідження є процес взаємодії та розробки архітектури та програмного забезпечення для онлайн платформи, призначеної для пошуку та залучення менторів, забезпечення ефективної взаємодії між учнями та менторами, а також управління навчальними процесами у межах створеної системи.

Предмет дослідження: Предметом дослідження є методи та технології проектування, розробки, тестування та оптимізації програмного забезпечення для створення і підтримки онлайн платформи пошуку менторів, включаючи інтерфейс користувача, систему рекомендацій, базу даних менторів та учнів, а також забезпечення безпеки та надійності функціонування платформи.

Особистий внесок здобувача. Магістерське дослідження є роботою виключно самостійною, в якій відображено особистий авторський підхід та особисто отримані теоретичні та прикладні результати, що відносяться до

вирішення даної задачі. Формулювання мети та завдань дослідження проводилось спільно з науковим керівником.

Практична цінність: Практична цінність даної роботи полягає в розробці архітектури та програмного забезпечення для онлайн платформи, призначеної для пошуку та залучення менторів, забезпечення ефективної взаємодії між учнями та менторами та управління навчальними процесами у межах створеної системи.

Ключові слова: застосунок, ментор, менті, студент, курс, архітектура, API.

ABSTRACT

for master's degree dissertation

with topic: «Mentor search platform for online learning»

made by: Minchenko Volodymyr Yuriyovych

The thesis of the second (master's) level of higher education on the topic "The platform for finding mentors for online learning" consists of an introduction and 4 chapters. Total volume of work: 126 pages, 12 tables, 50 figures, 3 appendices. The list of references includes 15 items.

Relevance of the work: The assimilation of information is the most valuable asset in our time. In the conditions of constant changes in the labor market, there is a critical need to improve qualifications or even radically change the professional sphere. Online education, which has grown at a rapid pace, especially during a quarantine or full-scale invasion, when some educational institutions have been forced to go online, meets these needs. There is a growing need for services and platforms for distance learning, ensuring that the level of education is not lost.

Distance learning has become not just a choice, but a necessity, offering the flexibility and mobility that many are looking for, allowing them to study on their own schedule. As an alternative to online colleges, web-based platforms for learning online courses offer flexibility and the ability to explore specific or unusual topics. However, with the development of such platforms, there is a need for qualified mentors who can provide individual support and guidance, adapting learning to the individual needs of each student.

In this context, the platform for finding mentors for online learning becomes especially relevant. It can radically improve the quality of online education by adapting curricula, promoting student motivation and engagement, building deeper understanding of the material, and facilitating cross-cultural exchange and professional growth. Such a platform will be able to provide not only flexibility and accessibility, but also person-oriented learning, which is becoming more and more important in a rapidly changing world.

Purpose of work: The main purpose is to research, develop and evaluate an information technology solution for an effective web-based platform that connects

students with qualified mentors in different areas of expertise for individualized and targeted learning. This platform aims to optimize the search process and interaction between students and mentors, to improve the quality of online education through individualized curricula and methods, and to ensure the flexibility and accessibility of learning for a wide range of users. In addition to the technical implementation of the platform, the goal includes the development of strategies to support and engage users, ensure data security, and create an effective system for managing and monitoring the educational process.

Research task: during the development of the application, the following stages were worked out:

- Analysis of requirements and collection of specifications: Determination of the main functions of the platform, requirements for the user interface, security, load, integration with other services.
- Designing the system architecture: Choosing a technology stack, developing a database model, creating an architectural plan for the user interface, backend and integrations.
- User interface development: Creation of interface design, development and testing of user experience for different categories of users (students, mentors, administrators).
- Programming and implementation of functionality: Development of the main components of the system, including the search and selection system for mentors, profile management, feedback and evaluation system.
- Integration with external APIs and services: Connection to email services, payment systems, social networks for authentication and other necessary integrations.
- Testing: Conducting unit testing, integration testing, load testing and acceptance testing to ensure stability, security and functional compliance.
- Platform launch and deployment: Server environment setup, software deployment, database setup and other infrastructure components.
- Support and Updates: Development of a support plan, system monitoring, bug fixes, functionality updates and performance optimization.

- Documentation: Create technical documentation, user and administrator guides, and code documentation.
- Ensuring security: Implementing security measures such as encryption, authentication, threat protection and data privacy.

Object of research: The object of research is the process of development architecture and software for an online platform designed to find and attract mentors, ensure effective interaction between students and mentors, and manage educational processes within the created system

Subject of the study: The subject of the study is the methods and technologies of designing, developing, testing, and optimizing software for creating and maintaining an online platform for finding mentors, including a user interface, a recommendation system, a database of mentors and students, as well as ensuring the safety and reliability of the platform's operation.

Personal contribution of the acquirer. The master's research is an exclusively independent work, which reflects the author's personal approach and personally obtained theoretical and applied results related to the solution of this problem. Formulation of the goal and tasks of the research was carried out together with the scientific supervisor.

Practical value: The practical value of this work lies in the development of architecture and software for an online platform designed to find and attract mentors, ensure effective interaction between students and mentors, and manage educational processes within the created system.

Keywords: application, mentor, mentee, student, course, architecture, API.

Пояснювальна записка до магістерської дисертації

на тему “Платформа пошуку менторів для онлайн навчання”

Зміст

ВСТУП..... 15
РОЗДІЛ 1. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ..... 17
1.1 Тенденції в онлайн навчанні..... 17
1.2 Потреби користувачів у менторстві..... 19
1.3 Аналіз предметної області..... 20
1.3.1 Користувачі..... 20
1.3.2 Навчальні компоненти..... 21
1.3.3 Комунікація..... 23
1.4 Приклад існуючих ресурсів..... 24
1.4.1 Moodle..... 25
1.4.2 iSpring..... 28
1.4.3 Coursera..... 31
1.4.4 Udemy..... 34
1.4.5 EnglishDom..... 36
1.4.6 Порівняльна таблиця існуючих онлайн платформ..... 38
ВИСНОВКИ ДО РОЗДІЛУ 1..... 40
РОЗДІЛ 2. ПРОЕКТУВАННЯ КОМПОНЕНТІВ СИСТЕМИ ТА ВИБІР ТЕХНОЛОГІЙ..... 41
2.1 Архітектура проекту..... 41
2.2 Вибір шаблону архітектури системи..... 42
2.3 Масштабованість для систем і служб..... 43
2.4 Балансувальник навантаження..... 46
2.5 Загальний огляд безпеки даних..... 49
2.5.1. Основні виклики та загрози у сфері безпеки веб-застосунків..... 49
2.5.2 Важливість безпеки даних у сучасному світі інтернету..... 49
2.5.3. Захист від міжсайтових скриптів (XSS)..... 50
2.4.4. Захист від підробки міжсайтових запитів (CSRF)..... 52
2.4.5. Захист від SQL-ін'єкцій в Node.js..... 54
2.5 Аутентифікація JWT..... 55
2.6 OAuth 2..... 58
2.7 Огляд технічного стеку..... 59
2.7.1 JavaScript..... 59
2.7.2 Платформа Node.js..... 60
2.7.3 Typescript..... 60
2.7.4 Fastify - backend фреймворк..... 61
2.7.5 OpenAPI специфікація..... 61
2.7.6 React - frontend бібліотека..... 62

2.7.7 CRA vs Vite.....62
2.7.8 Docker.....63
2.7.8 PostgreSQL.....63
ВИСНОВКИ ДО РОЗДІЛУ 2.....64
РОЗДІЛ 3. РОЗРОБКА ТА РОЗГОРТАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.....65
3.1 Проєктування бази даних.....65
3.2 Структура проєкту.....70
3.3 Підтримка консистентності коду.....71
3.4 Реалізація бекенд частини додатку.....72
3.4 Реалізація документації ендпоінтів.....77
3.5 Реалізація клієнтської частини додатку.....78
3.6 Організація деплойменту.....83
3.6.2 AWS.....85
3.6.3 Схема деплойменту.....85
ВИСНОВКИ ДО РОЗДІЛУ 3.....91
РОЗДІЛ 4. ІНТЕРФЕЙС РОЗРОБЛЕНОЇ СИСТЕМИ.....92
ВИСНОВКИ ДО РОЗДІЛУ 4.....105
РОЗДІЛ 5 РОЗРОБКА СТАРТАП-ПРОЄКТУ.....106
5.1 Опис ідеї проєкту.....106
5.2 Технологічний аудит проєкту.....107
5.3 Аналіз ринкових можливостей запуску стартап-проєкту.....108
4.3 Розробка ринкової стратегії проєкту.....114
5.4 Розробка маркетингової програми стартап-проєкту.....120
Визначення ключових переваг концепції потенційного товару.....120
4.5 Концепція маркетингових комунікацій.....121
ВИСНОВКИ ДО РОЗДІЛУ 5.....122
ВИСНОВКИ.....123
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ:.....124
ДОДАТКИ.....126
ДОДАТОК 1 Вихідний код програми.....127

ВСТУП

Сьогодні процес дистанційного навчання особливо актуальний у світлі сучасних соціокультурних і технологічних трансформацій. На жаль, ніхто завчасно не передбачав пандемію чи повномасштабну війну, і ці несподівані виклики вимагають більше уваги та розробки нових рішень для освіти. Навчальні заклади усіх рівнів та типів не завжди були готові до масштабного переходу на дистанційну освіту, і це створило серйозні виклики для якості навчання та доступності освітніх послуг.

Однією з найважливіших складових дистанційного навчання є обрана платформа, на якій відбувається освітній процес. Спрощена або недостатньо функціональна платформа може значно ускладнити процес навчання та обмежити можливості як вчителів, так і учнів. Важливою задачею є розробка та вдосконалення інструментів для оптимізації цього аспекту.

Дистанційне навчання є самостійною та важливою формою здобування освіти, де інформаційні технології виступають як провідний елемент в передачі знань та сприяють взаємодії між учнями та вчителями. Це процес віддаленого отримання освіти, який не потребує фізичної присутності в навчальних аудиторіях та забезпечує гнучкість у навчанні.

Історично поняття "дистанційне навчання" в основному асоціювалося з вищою освітою та програмами навчання в коледжах та університетах, де студенти мали можливість вивчати матеріал поза стінами навчальних закладів. Проте в сьогоденні дистанційне навчання поширюється на учнів початкової, середньої та старшої школи, забезпечуючи доступ до якісної освіти для різних груп населення.

Для підтримки якісного дистанційного навчання та взаємодії між учнями та вчителями, планується створити веб-додаток, який допоможе у розподілі домашніх завдань та забезпечить зручну платформу для взаємодії. Цей веб-додаток надасть можливість надсилати завдання, надавати інструкції з їх виконання та вести процес контролю за успішністю та виконанням завдань.

Дипломний проект орієнтується на створення веб-сервісу для дистанційного навчання та системи контролю успішності, яка сприятиме покращенню навчального процесу як для учнів, так і для вчителів. Віримо, що цей веб-додаток стане надійним і корисним інструментом у сучасній освіті.

РОЗДІЛ 1. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

1.1 Тенденції в онлайн навчанні

Тенденції в галузі онлайн-освіти стають все більш помітними в сучасному світі. Швидкий розвиток технологій неодмінно впливає на системи навчання та освіти. У цьому розділі я розгляну ключові тенденції в онлайн-навчанні та їх вплив на освітню сферу.

Однією з основних тенденцій є стрімкий ріст популярності дистанційного навчання. Це стає особливо актуальним у зв'язку з глобальними подіями, що змусили освітні установи переходити до онлайн-формату. Онлайн-навчання стає доступним та зручним для різних категорій користувачів, включаючи учнів, студентів та фахівців.

Сучасні методи навчання акцентують увагу на інтерактивності. Онлайн-платформи активно застосовують різноманітні засоби, такі як відео, аудіо, чати та форуми, для залучення учнів до активної участі у процесі навчання. Це сприяє покращенню якості освіти в онлайн-форматі.

Онлайн-освіта стає все більш індивідуалізованою, адаптуючись до потреб кожного користувача. Платформи для онлайн-навчання дають можливість обирати курси, теми та методи відповідно до індивідуальних цілей та рівня знань.

Великі можливості для отримання нових навичок відкриваються завдяки онлайн-навчанню. Це особливо важливо в умовах стрімкого розвитку технологій та вимог сучасного ринку праці.

Використання штучного інтелекту та аналітики в онлайн-навчанні дозволяє створювати більш ефективні та індивідуалізовані підходи до навчання. Аналіз даних про навчання допомагає вдосконалювати програми та методи навчання.

Використання віртуальної реальності та розширеної реальності включається в процес онлайн-навчання, створюючи імерсивні середовища для навчання та досліджень. Це сприяє виробленню процесу навчання більш захопливим та ефективним.

Онлайн-навчання розширює межі географічних обмежень, забезпечуючи якісну освіту для студентів і учнів, незалежно від їхнього місця проживання. Це робить освіту більш доступною для людей з віддалених регіонів та країн. Розуміння цих тенденцій є ключовим для успішного створення сервісу пошуку менторів для онлайн-навчання.

Особлива увага в онлайн навчанні приділяється аналізу змін та нововведень, які формують сучасний ландшафт дистанційної освіти, що безпосередньо впливають на розвиток платформ пошуку менторів. Персоналізація навчальних програм, забезпечена штучним інтелектом та машинним навчанням, дозволяє підлаштовувати навчальний процес під індивідуальні потреби та стилі навчання учнів. Важливою тенденцією є зростання мікронавчання, що пропонує короткі, але інтенсивні освітні сесії, зручні для зайнятих людей, які прагнуть навчатися в ритмі сучасного життя.

Соціальне навчання та формування освітніх спільнот на платформах забезпечують обмін знаннями та досвідом, створюючи віртуальне середовище для співпраці та підтримки між учнями та менторами. Мобільне навчання відкриває нові можливості для доступу до освіти, роблячи її доступною з будь-якої точки світу. Також існує зв'язок між онлайн навчанням та професійним розвитком, де освітні платформи все частіше пропонують курси та програми, спрямовані на підвищення кваліфікації та розвиток кар'єри.

Забезпечення безпеки даних стає критично важливим, особливо у контексті збільшення обсягу особистої та навчальної інформації, яка циркулює в онлайн просторі. Відкриті освітні ресурси набувають популярності, пропонуючи вільний доступ до якісних навчальних матеріалів та ресурсів.

Усі ці тенденції вказують на важливість створення гнучких, адаптивних та безпечних платформ для пошуку менторів, здатних відповідати зростаючим вимогам сучасного освітнього середовища.

1.2 Потреби користувачів у менторстві

У цьому розділі зосереджується увага на основні аспектах та вимогах, які користувачі висувають до менторства в онлайн навчанні. Ключові підпункти:

1. Потреба в особистому підході:

- Індивідуалізація навчання: Студенти шукають менторів, здатних адаптувати навчальний процес до їхніх індивідуальних потреб, стилів навчання та освітніх цілей.
- Вирішення специфічних проблем: Ментори можуть надати цінні поради та рішення для конкретних академічних завдань або труднощів, з якими стикаються учні.

2. Підтримка у вирішенні завдань:

- Академічна допомога: Ментори допомагають студентам розуміти складні концепції, підготуватися до іспитів та виконувати домашні завдання.
- Розвиток навичок та вмінь: Підтримка у розвитку важливих навичок, необхідних для успішного вивчення предметів та професійного розвитку.

3. Кар'єрні консультації:

- Професійне наставництво: Менторство, спрямоване на допомогу у виборі кар'єрного шляху, розвитку професійних навичок та підготовці до ринку праці.
- Мережеві можливості: Допомога в налагодженні корисних професійних контактів та доступу до індустріальних ресурсів.
- Мотивація та психологічна підтримка:
    - Емоційна підтримка: Надання моральної підтримки, включаючи заохочення та допомогу в подоланні навчальних викликів.
    - Підвищення самооцінки: Допомога учням у формуванні впевненості в своїх здібностях та навчальних досягненнях.

Кожен з цих підпунктів вказує на глибокий зв'язок між потребами

користувачів та якістю менторської підтримки, що повинна надаватися через онлайн платформи. Розуміння цих потреб є критично важливим для розробки ефективної системи менторства, яка не тільки відповідає на академічні та професійні запити користувачів, але й забезпечує необхідну емоційну та мотиваційну підтримку.

1.3 Аналіз предметної області

1.3.1 Користувачі

Основною метою проєктування навчальної веб-платформи є організація та супровід навчального процесу та можливості взаємодії різного характеру між учасниками навчального процесу. У зв'язку з цим, при аналізі предметної області важливо звернути увагу на навчальне середовище та специфічні потреби різних груп користувачів, що включають студентів, викладачів та адміністраторів. Усі вони взаємодіють з веб-платформою, але мають різні ролі та очікування.

Студенти - основні користувачі, що прагнуть отримати знання та навички. Для них важливо мати доступ до якісних освітніх матеріалів, інтерактивних завдань та можливості зворотного зв'язку з викладачами. Середовище навчання має бути інтуїтивно зрозумілим, гнучким та забезпечувати необхідну взаємодію з іншими користувачами.

Ментори - передають знання та контролюють освітній процес. Важливо, щоб система надала їм інструменти для ефективного викладання, видачі завдань, перевірки робіт та ведення зворотного зв'язку зі студентами. Викладачам потрібні засоби для створення та модифікації курсового контенту та адміністрування оцінювання.

Адміністратори - забезпечують організаційну та технічну підтримку навчальної платформи. Їхнє завдання - контролювати роботу системи, забезпечувати безпеку даних, вносити зміни на основі зворотного зв'язку та забезпечувати безперервність навчального процесу та контролювати межі доступу до даних.

До аналізу предметної області - користувачів, додано наступні аспекти:

- Демографічні характеристики користувачів: Важливо розуміти вік, освітній рівень, професійні інтереси користувачів для адаптації контенту та функціоналу платформи.
- Мотиваційні фактори: Чітке розуміння того, чому користувачі шукають менторство або навчальні курси, допоможе створити більш цілеспрямований та ефективний досвід.
- Поведінкові моделі та взаємодія з технологіями: Аналіз того, як користувачі використовують платформу, які інструменти вони вважають зручними, та які бар'єри можуть виникати під час взаємодії.
- Переваги та вимоги користувачів: З'ясування ключових вимог та очікувань користувачів до платформи, включаючи якість контенту, доступність менторів, вартість послуг, тощо.

Цей комплексний підхід дозволить глибше зрозуміти потреби та очікування користувачів, а також дасть змогу забезпечити створення умов для більш ефективного та корисного навчального середовища.

1.3.2 Навчальні компоненти

Крім аналізу користувачів в предметній області, важливо розглянути навчальні компоненти, що є основою будь-якої навчальної системи. Центральним елементом навчальної діяльності є процес отримання знань та практичних навичок, що здійснюється через різні типи навчальних матеріалів та інструментів оцінювання. Основні компоненти системи охоплюють завдання, оцінки, тести, навчальні матеріали, а також інтерактивні елементи для забезпечення активного навчання.

1. Навчальні матеріали:

- Текстові лекції, нотатки, відео, презентації: Забезпечення доступу до різноманітних форматів навчальних матеріалів, які можуть включати текстові файлы, слайди, відеоматеріали тощо.
- Інтерактивні завдання: Впровадження завдань різного характеру та типу, забезпечення механізмів для їх завантаження та взаємодії з викладачем.

- Використання зовнішніх ресурсів: Інтеграція посилань на зовнішні освітні ресурси для розширення бази знань студентів.

2. Комунікаційні інструменти:

- Форуми та дискусії: Створення можливості для взаємодії між студентами та викладачами, обговорення навчального матеріалу.
- Спілкування: Надання можливостей для прямої комунікації між студентами та викладачами, включно з електронною поштою, чатами, відеоконференціями.

3. Оцінювання та тестування:

- Онлайн тести та квізи: Розробка та інтеграція різних типів тестів, що дозволяють оцінювати знання студентів.
- Система оцінок: Механізми для виставлення, редагування та перегляду оцінок, включаючи фінальні оцінки за дисципліну.
- Зворотний зв'язок: Впровадження можливостей для викладачів надавати зворотній зв'язок студентам щодо виконаних завдань та тестів.

4. Управління контентом та дисциплінами:

- Структурування курсів: Організація матеріалів за темами та модулями для зручного доступу студентів.
- Адміністрування предметів: Надання можливостей адміністраторам створювати, редагувати та видаляти предмети, контролювати хід навчання.

5. Технологічна підтримка:

- Інфраструктура: Розробка та підтримка надійної технічної інфраструктури для навчальної платформи.
- Безпека даних: Забезпечення захисту персональних даних та навчальних матеріалів.
- Обслуговування та підтримка: Створення системи підтримки для вирішення технічних та організаційних питань користувачів.

Даний аналіз предметної області - навчальні компоненти - дає змогу глибше зрозуміти, як ефективно структурувати та управляти навчальним

матеріалом, інструментами комунікації, оцінюванням та іншими важливими аспектами для створення продуктивного та взаємопов'язаного навчального середовища.

1.3.3 Комунікація

Важливим аспектом взаємодії студентів та викладачів у навчальному процесі є спілкування між собою для ефективної координації дій та вирішення питань, що виникають. Спілкування у навчальному середовищі вимагає гнучкості та адаптивності, щоб відповідати потребам віддаленого навчання та імітувати взаємодію очного навчання. Різні формати, як-от відеоконференції, форуми, чати чи електронна пошта, дозволяють реалізувати це спілкування. Ці можливості можуть інтегруватися в систему або забезпечуватися за допомогою зовнішніх ресурсів.

Ось докладніший аналіз цієї тематики:

1. Комунікаційні канали:
- Форуми та дискусійні групи дозволяють студентам та викладачам вести структуровані обговорення та обмінюватися думками, створюючи записи дискусій, доступні для всіх учасників курсу.
- Пряме спілкування через чати, електронну пошту або месенджери забезпечує швидкий обмін інформацією та вирішення нагальних питань.
- Відеоконференції та вебінари надають можливості для живої взаємодії, близької до очного навчання, дозволяючи проводити лекції, семінари та консультації в реальному часі.
2. Стратегії комунікації:
- Синхронне та асинхронне спілкування: Визначення оптимального балансу та використання обох форм спілкування залежно від потреб навчального процесу.
- Модерування та підтримка: Забезпечення ефективного та ввічливого спілкування через активне модерування та підтримку з боку команди платформи.
3. Інструменти та технології:

- Використання різноманітних платформ: Аналіз та інтеграція найкращих інструментів для спілкування, забезпечуючи доступність та зручність для всіх користувачів.
- Інтеграція з соціальними мережами: Використання популярних соціальних мереж для розширення можливостей спілкування та залучення користувачів.

4. Бар'єри та виклики:

- Мовні та культурні бар'єри: Адаптація комунікаційних стратегій для різноманітної аудиторії з урахуванням мовних та культурних особливостей.
- Технічні проблеми: Розробка рішень для забезпечення стабільності та доступності комунікаційних інструментів.

5. Ефективність та вплив комунікації:

- Оцінка ефективності: Регулярний аналіз та оцінка комунікаційних практик з метою виявлення та впровадження покращень.
- Зворотний зв'язок: Створення механізмів для збору та аналізу зворотного зв'язку від користувачів для постійного покращення комунікаційних процесів.

Цей комплексний аналіз важливий для розуміння та вдосконалення комунікаційних аспектів навчального процесу, адже ефективне спілкування є ключовим для успішної взаємодії, обміну знаннями, мотивації та підтримки студентів та викладачів.

1.4 Приклад існуючих ресурсів

У даному порівнянні робиться акцент саме на онлайн-ресурси, що представляють собою веб-додатки для дистанційного навчання, у яких представлена роль ментора. Нижче проведено аналіз декількох існуючих зразків, їх зміст, структуру, додаткові можливості.

1.4.1 Moodle

Moodle (Modular Object-Oriented Dynamic Learning Environment, вимовляється «Мудл») - це модульне об’єктно-орієнтоване динамічне навчальне середовище, відоме також як система управління навчанням, система управління курсами, віртуальне навчальне середовище або просто платформа для навчання. Це відкрите програмне забезпечення, яке надає викладачам, учням та адміністраторам великий набір інструментів для комп’ютеризованого навчання, включаючи дистанційне навчання. Ця платформа є однією з найпопулярніших у світі для створення навчальних курсів онлайн і надає можливості для ефективного управління навчальним процесом.

В контексті менторства, Moodle пропонує ряд можливостей:

- Персоналізація навчання: Ментори можуть створювати індивідуалізовані навчальні плани та матеріали, адаптовані під потреби менті.
- Гнучкі комунікаційні інструменти: Платформа надає форуми, чати, та інші засоби для ефективної взаємодії.
- Оцінювання та зворотній зв'язок: Ментори мають доступ до інструментів для створення тестів, квізів, та відстеження прогресу менті.
- Планування та відстеження: Інструменти для організації та моніторингу навчального процесу.
- Гнучкість та масштабованість: Можливість інтегрувати різні плагіни та додатки, що розширюють функціональність Moodle.
- Мобільний доступ: Ментори та менті можуть легко доступатися до курсів та комунікацій через мобільні додатки Moodle.
- Безпека та приватність: Забезпечення захисту даних та конфіденційності користувачів.

Використовуючи Moodle для менторських програм, організації можуть створювати адаптивні, інтерактивні та індивідуально налаштовані навчальні середовища, що враховують специфічні потреби та цілі як менторів, так і менті.

Окрім вище перечисленного, Moodle - безкоштовна система дистанційного навчання навчання з відкритим кодом. На сьогодні Moodle

підтримує понад тисячу різних плагінів та існують переклади більш ніж на сто різних мов.

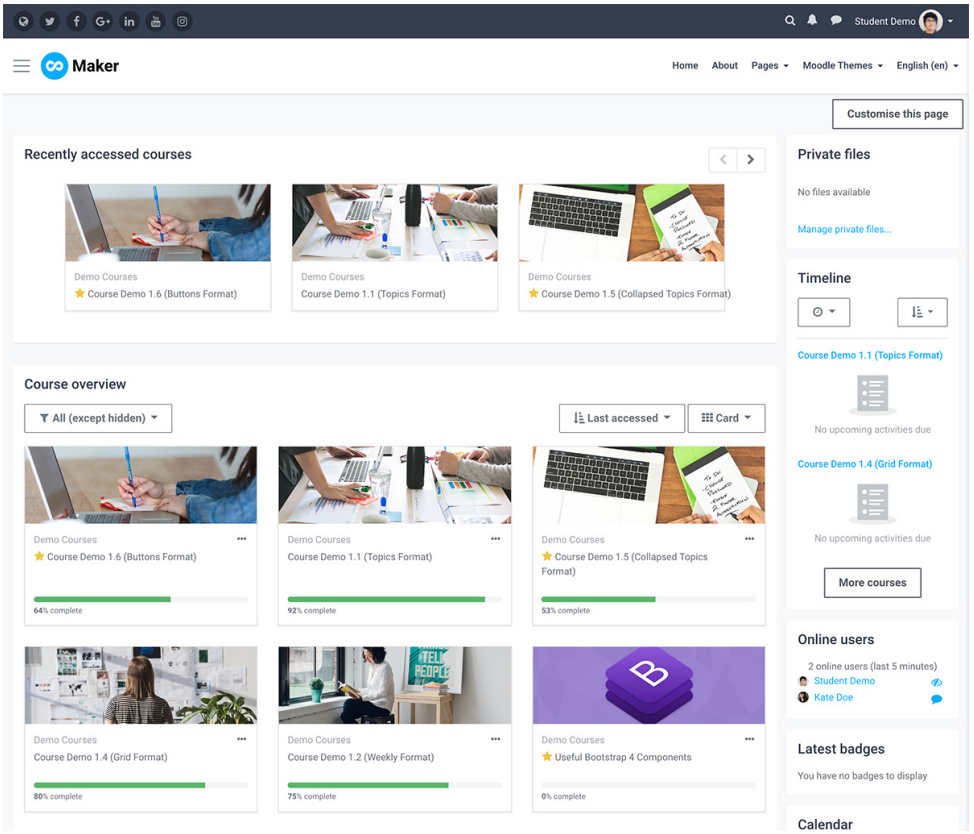


Рис 1.1. Сторінка курсів Moodle

Moodle використовується для організації навчання в ВНЗ та корпоративного навчання, проте порівняно з комерційними платформами вона вважається відносно складною в налаштуванні. Особливості Moodle включають можливість використання плагінів для головних налаштувань, при цьому весь функціонал та дизайн може бути змінений за допомогою цих плагінів, які можна отримати безкоштовно або створити самостійно. Система є з доступним кодом, дозволяючи користувачам розробляти плагіни та вносити їх у відкритий доступ. Також Moodle добре співпрацює з іншими сервісами, такими як вебінар Zoom.

Керування користувачами в Moodle дозволяє надавати різні ролі та об'єднувати користувачів в групи.

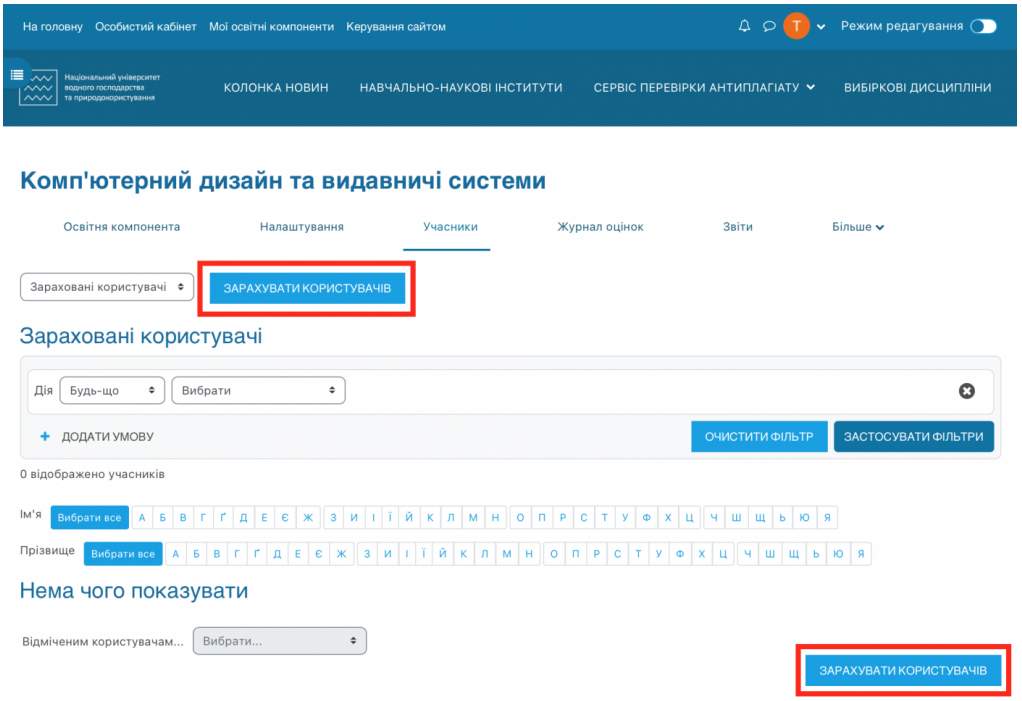


Рис 1.2 Зарахування користувачів

Усі інші функції, такі як масове призначення курсів та настройки умов реєстрації, додаються завдяки відповідним плагінам.

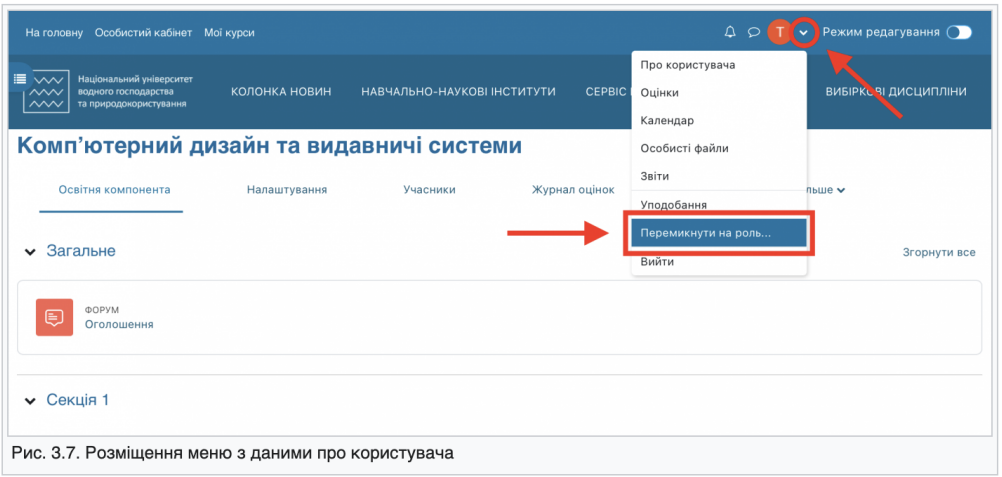


Рис. 3.7. Розміщення меню з даними про користувача

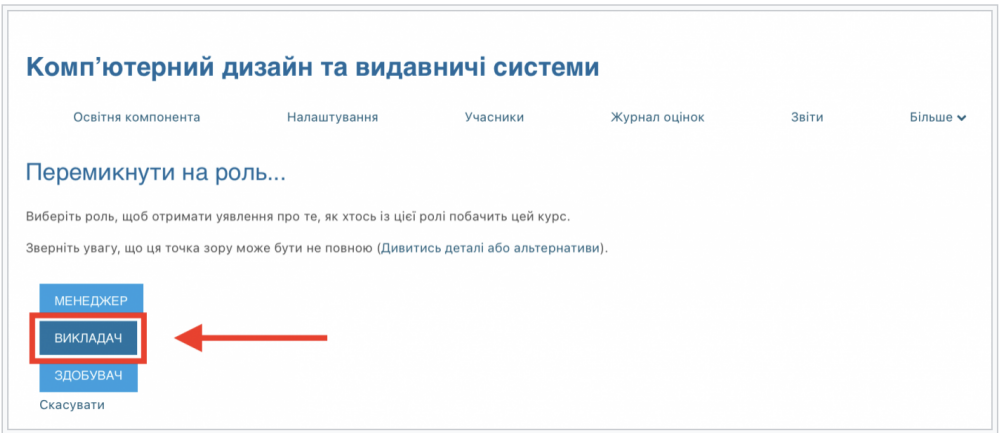


Рис 1.3 Менеджинг ролей

Система звітності у Moodle надає можливість налаштувати власну систему звітів, використовуючи лише необхідні дані для аналізу учнів. Звіти можуть включати інформацію про час витрачений на вивчення курсу, регулярність відвідування платформи, а також аналіз помилок в тестах. Тип звіту залежить від встановленого плагіна. Також є можливість налаштувати статистику успішності користувачів та їх активності, популярність курсу чи ефективність роботи платформи.

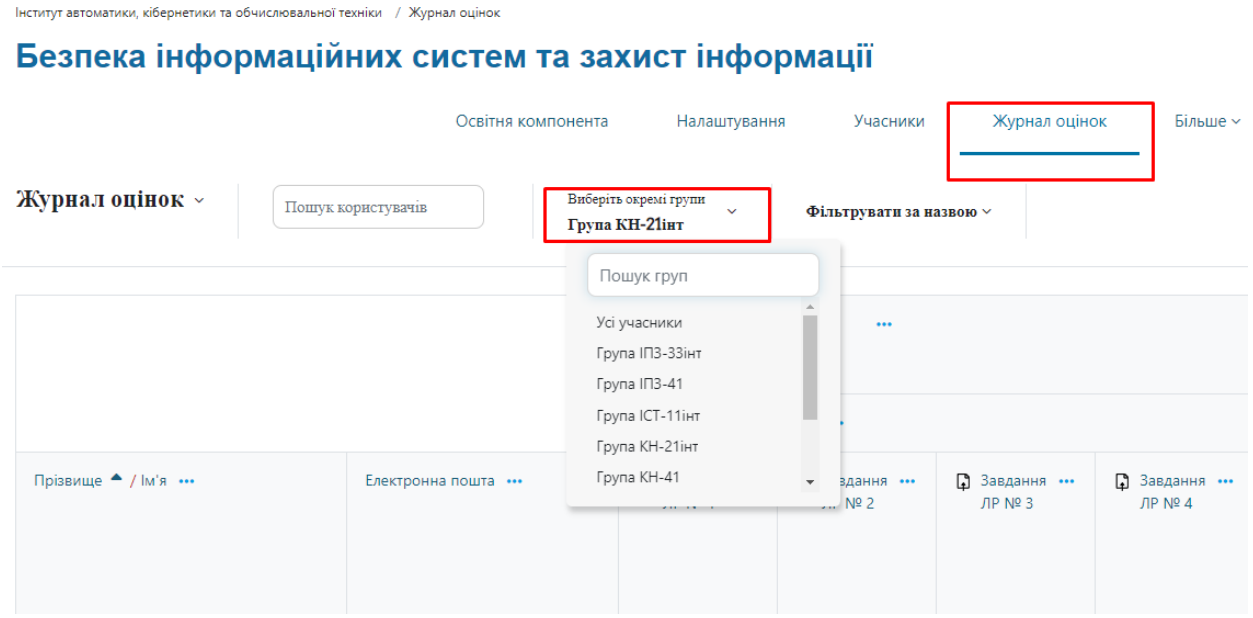


Рис 1.4 Сторінка журналу оцінок

1.4.2 iSpring

iSpring є потужною платформою для створення електронних курсів та навчального контенту, яка в контексті менторства може використовуватися для створення, управління та доставки навчального матеріалу та програм. Вона надає широкий спектр інструментів для створення інтерактивного та залучаючого контенту, що робить її корисною для менторів, які шукають ефективні способи передачі знань та навичок. Ось деякі ключові можливості iSpring, які роблять її придатною для менторських програм:

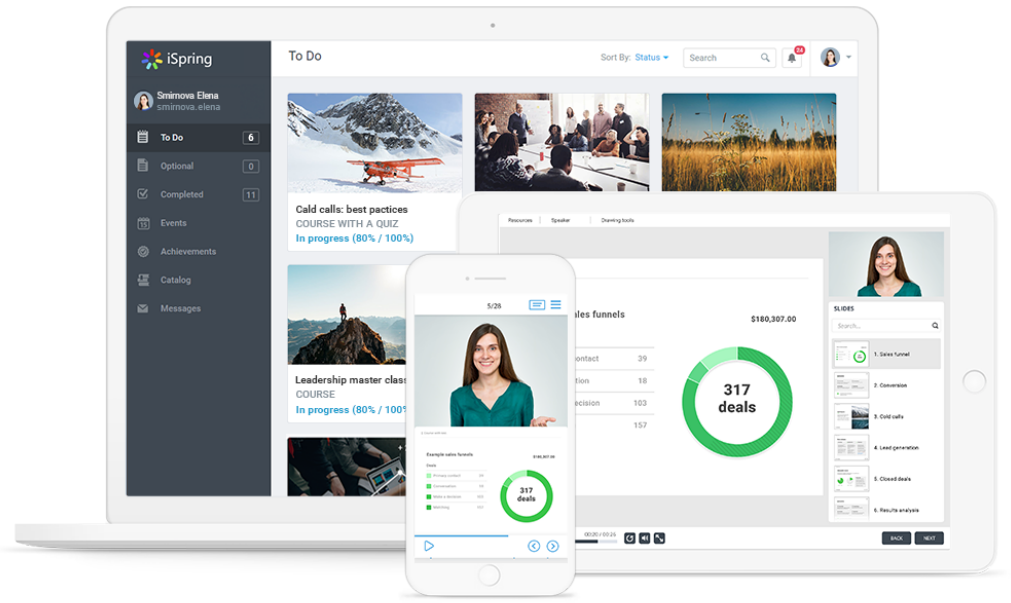


Рисунок 1.5. Платформа iSpring

- Інтуїтивний конструктор курсів: iSpring дозволяє менторам легко створювати інтерактивні курси та оцінки, використовуючи перетягування об'єктів, шаблони та інші інструменти, не вимагаючи глибоких технічних знань.
- Інтеграція з PowerPoint: iSpring є потужним доповненням до Microsoft PowerPoint, що дозволяє менторам перетворювати звичні презентації на інтерактивні навчальні курси з квізами, симуляціями, відео та іншими мультимедійними елементами.
- Легка співпраця з іншими сервісами. Наприклад, iSpring Learn, завдяки відкритому API, часто використовують з іншими системами клієнта, що втілюється як кадрова система чи корпоративний портал.
- Жвавий запуск платформи. Платформа не вимагає комплексного налаштовування. Достатньо зареєструватися, завантажити курси і покликати користувачів.
- Мобільне навчання: Платформа підтримує мобільне навчання, дозволяючи менті доступати курси на різних пристроях, забезпечуючи гнучкість та доступність навчання.
- Симуляції діалогів: Ментори можуть створювати симуляції діалогів та

рольові ігри для навчання м'яких навичок, комунікації та інших важливих компетенцій.

- Трекінг та аналітика: iSpring надає інструменти для відстеження прогресу та аналізу результатів навчання, дозволяючи менторам адаптувати програму та відповідати на індивідуальні потреби менті.
- Спілкування та обговорення: Інструменти для організації форумів, чатів та інших форм спілкування між ментором та менті забезпечують високий рівень взаємодії та обговорення матеріалу.
- Широкий спектр контенту: Від відео та аудіо до інтерактивних квізів і сценаріїв, iSpring дозволяє менторам використовувати різноманітні формати контенту для створення залучаючого та ефективного навчального досвіду.
- Система звітності

  У iSpring Learn є можливість різних видів звіту. Для зручності їх поділили по групам:

  • По тестам, діалогам чи завданням. Є можливість з’ясувати які помилки в тестах співробітники допустили, чи вдалося на сто відсотків пройти, що вдалось легко, а що складно протягом навчання.

  • Популярні користувачі. Існує можливість знайти, користувачів чи групи, які найактивніші та переглянути їхні матеріали.

  • По матеріалам. Є можливість передавитися дії, які вчинені над матеріалом, кількість переглядів, а також успішність користувачів.

  • По заходам. Є можливість переглянути, скільки заходів було проведено і їх учасників.

  • За програмами навчання. Можна дізнатися, як добре учасники проходять програми навчання.

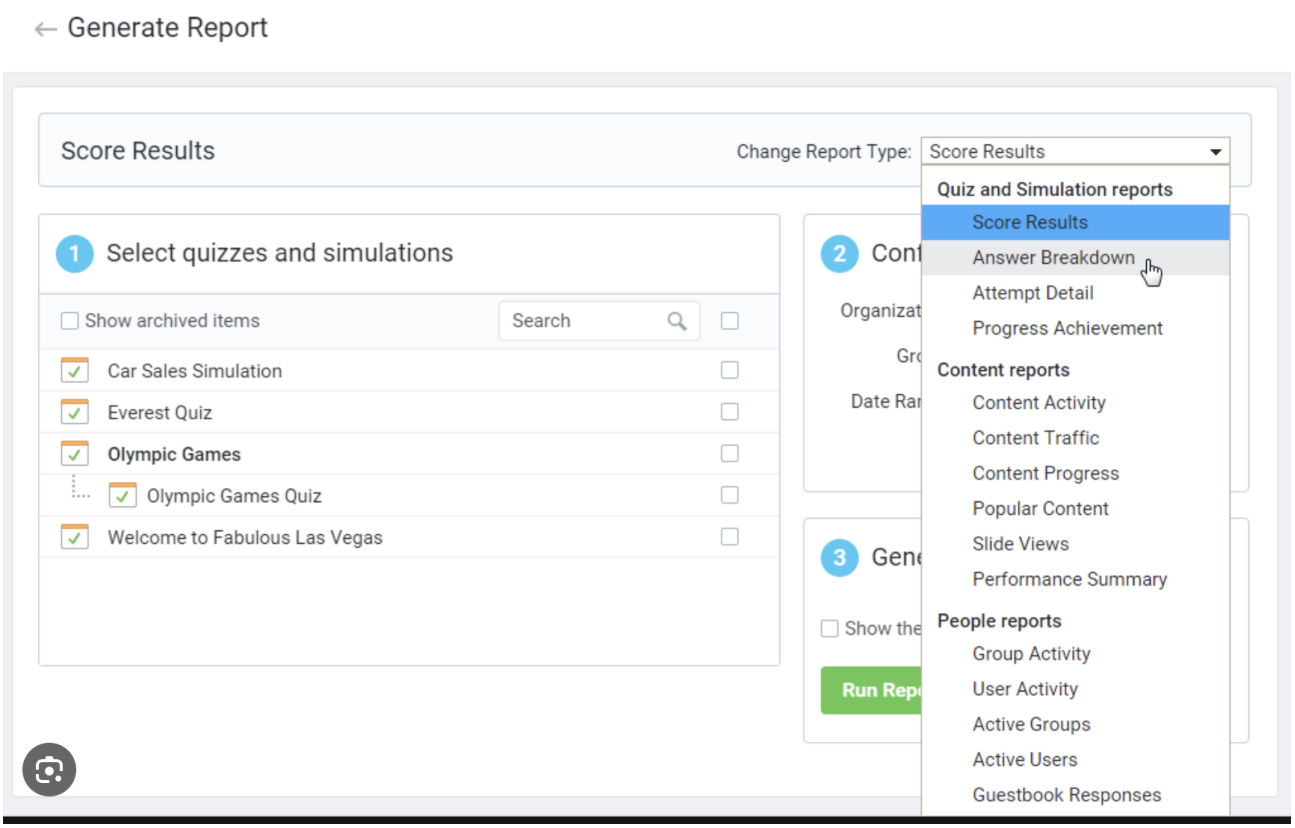


Рис 1.6 Створення звіту

iSpring є ефективним рішенням для менторів, які шукають сучасну, інтуїтивно зрозумілу та багатофункціональну платформу для розробки та доставки навчальних курсів. Її багатий функціонал і гнучкість роблять її ідеальним інструментом для створення індивідуалізованих та ефективних менторських програм.

1.4.3 Coursera

Coursera - це інтернет-платформа для дистанційного навчання, яка надає університетам та організаціям можливість публікувати власні навчальні матеріали. Сервіс пропонує велику кількість безкоштовних онлайн-курсів з різних галузей знань. Після успішного завершення курсу студент отримує сертифікат, який підтверджує його участь.

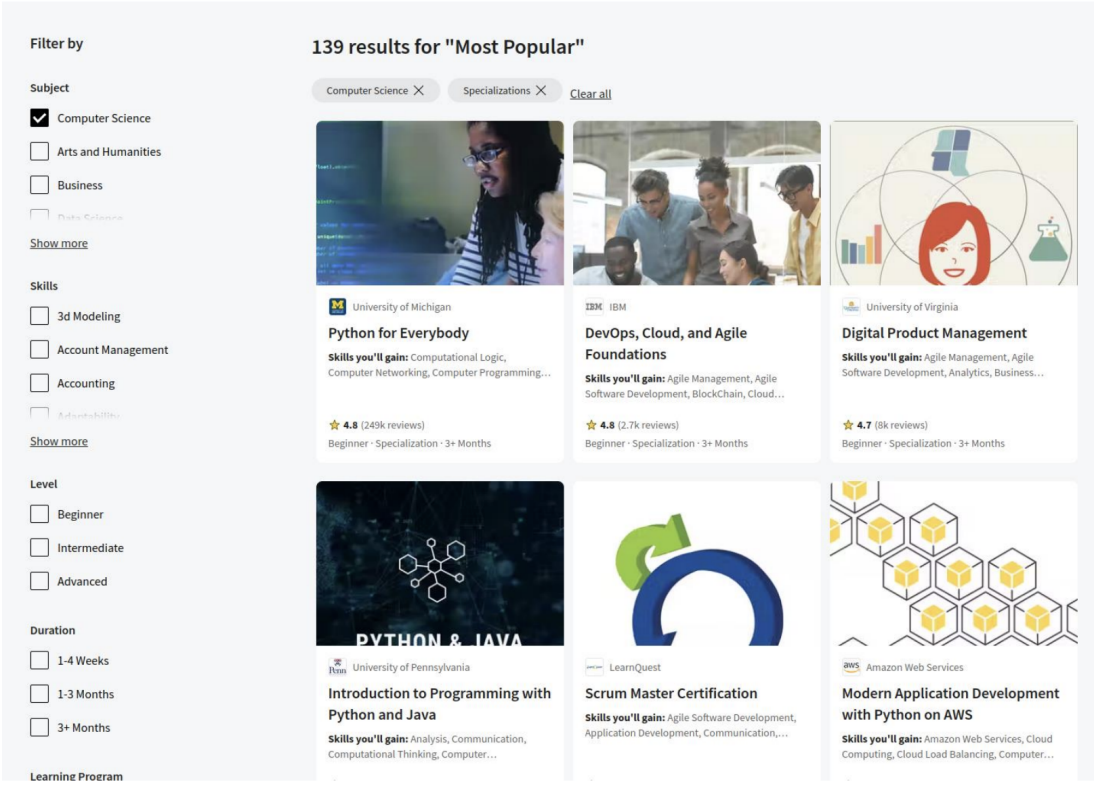


Рис 1.7 Сторінка курсів на ресурсі Coursera

Coursera, в контексті платформи для менторів, може слугувати потужним інструментом у наданні індивідуалізованого навчання, розвитку навичок та кар'єрного просування для менті. Ось як можливості Coursera можуть бути використані менторами:

- Індивідуальні навчальні плани: Ментори можуть використовувати широкий спектр курсів Coursera для створення індивідуалізованих навчальних планів, які відповідають конкретним потребам і цілям кожного менті, включаючи розвиток спеціалізованих навичок або галузевих знань.
- Підтримка та наставництво: Ментори можуть використовувати комунікативні інструменти Coursera, такі як форуми та дискусійні сесії, для надання зворотного зв'язку, відповідей на питання та підтримки менті у процесі навчання.
- Спільнота навчання: Ментори можуть організовувати або залучати менті до спільнот на Coursera для спільного навчання, обміну досвідом та мережевої взаємодії з іншими учасниками курсів, що розширює

можливості професійного та особистісного розвитку.

- Доступ до світових знань: Завдяки співпраці з провідними університетами та інституціями, Coursera дозволяє менторам надавати менті доступ до найновіших знань, досліджень та інновацій з усього світу.
- Гнучкість та доступність: Coursera забезпечує гнучкий графік навчання та доступність курсів, що дозволяє менторам та менті планувати навчання відповідно до своїх графіків та потреб.
- Можливості сертифікації: Після завершення курсів на Coursera, менті можуть отримати сертифікати, що є доказом їхніх досягнень та можуть бути використані для кар'єрного росту.

Але, в Coursera менторство є скоріш винятком з правила, замість цього широко використовується підхід в експертному оцінюванні - peer-review[5].

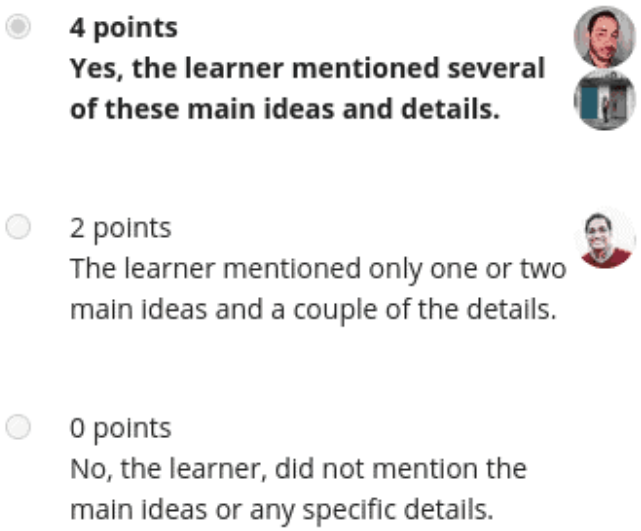


Рис 1.8 Приклад експертного оцінювання

Експертні оцінки становлять важливий аспект онлайн-навчання, особливо на платформі як Coursera, де вони використовуються у значній кількості курсів. Для менторів, цей тип завдання може бути цінним інструментом для оцінювання та залучення менті до глибшого розуміння матеріалу. Процес експертних оцінок на Coursera відбувається наступним

чином:

- Виконання завдань: Студенти виконують завдання, які часто включають відкриті запитання та вимагають розгорнутих відповідей. Після завершення, завдання відправляється на оцінювання.
- Взаємне оцінювання: Студенти оцінюють роботи одне одного за допомогою заздалегідь визначеної рубрики. Це дозволяє не тільки отримати зворотний зв'язок але й розвинути критичне мислення та об'єктивність у оцінюванні.

Незважаючи на переваги, експертні оцінки мають певні обмеження пов'язані з людським фактором, такі як потенційний обман, затримки у оцінюванні та розбіжності в оцінках. Однак, багато учнів підходять до цього процесу відповідально, надаючи конструктивні та справедливі оцінки, що збагачує навчальний досвід.

1.4.4 Udemy

Udemy є однією з провідних онлайн платформ, яка пропонує широкий спектр курсів у різних областях, від бізнесу та технологій до особистого розвитку. Для менторів, Udemy може слугувати як вартісним ресурсом для підтримки і розвитку їх менті, забезпечуючи доступ до різноманітних навчальних матеріалів та ресурсів.

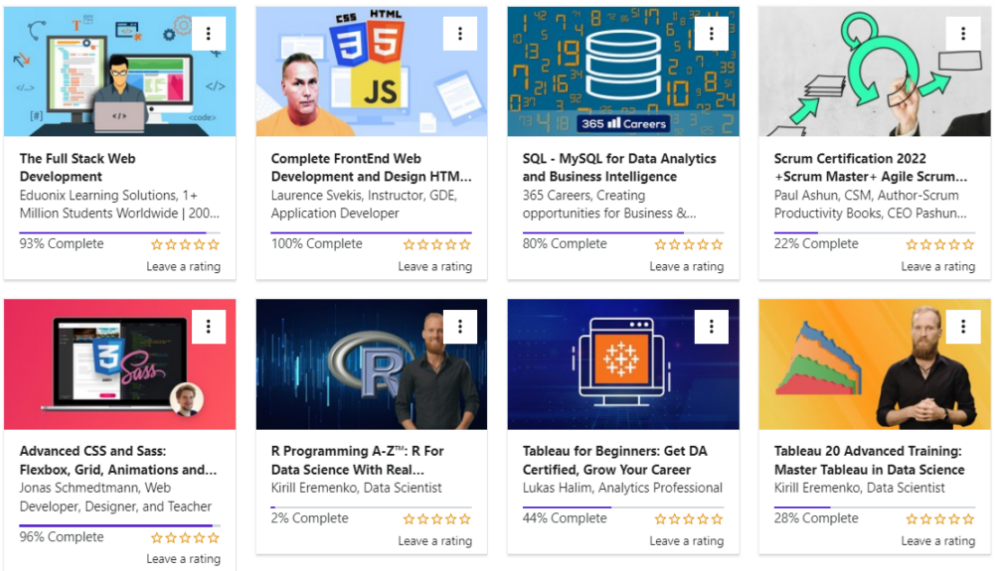


Рис 1.9 Ресурс Udemy

Ось деякі ключові характеристики Udemy як платформи для менторів:

- Великий вибір курсів: Udemy пропонує тисячі онлайн курсів на різноманітні теми, що дозволяє менторам знайти відповідні матеріали для підтримки індивідуальних навчальних потреб менті.
- Доступність та гнучкість: Більшість курсів на Udemy доступні за відносно низькою вартістю, або навіть безкоштовно, та пропонують гнучкий графік навчання, що дозволяє менті навчатися у власному темпі.
- Якість контенту: Курси на Udemy розробляються досвідченими інструкторами та експертами у своїх областях, забезпечуючи високу якість та актуальність навчального матеріалу.
- Взаємодія та залученість: Ментори можуть використовувати курси Udemy для створення інтерактивних та залучаючих навчальних сесій, включаючи відео лекції, практичні завдання, тести та форуми для обговорення.
- Індивідуальний підхід: Ментори можуть підібрати курси та програми відповідно до конкретних цілей та інтересів менті, створюючи персоналізовані навчальні шляхи.
- Сертифікація: По завершенню курсів на Udemy, менті можуть отримати сертифікати, що підтверджують їхні знання та навички, що може бути корисним для їх професійного зростання та розвитку.

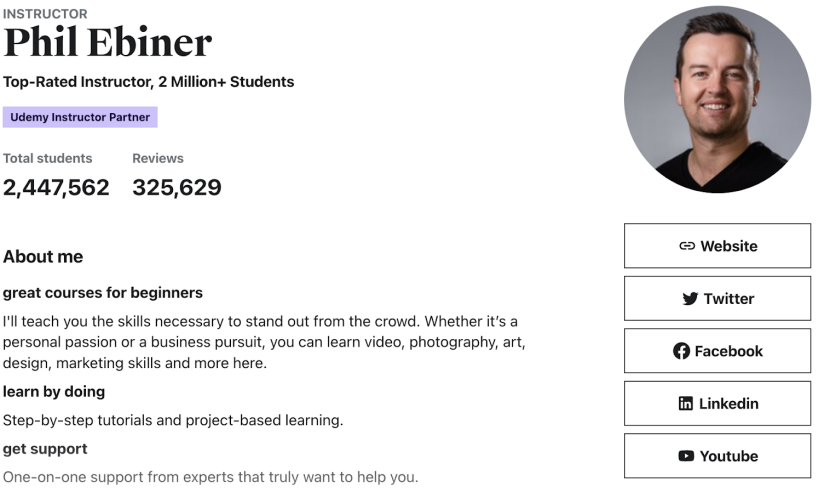


Рис 1.10 Сторінка ментора

Використовуючи Udemy як платформу для менторства, ментори отримують доступ до багатого ресурсу навчальних матеріалів та можливостей, які можуть допомогти їм підтримувати, мотивувати та розвивати своїх менті у відповідності до їхніх індивідуальних потреб і цілей.

1.4.5 EnglishDom

EnglishDom — це онлайн-платформа, яка спеціалізується на навчанні англійської мови. Вона пропонує широкий спектр курсів, від початкового до просунутого рівня, і зосереджена на покращенні мовних навичок через індивідуальні та групові уроки з носіями мови та кваліфікованими викладачами. У контексті платформи для менторів, EnglishDom може слугувати цінним ресурсом для тих, хто бажає підвищити свою майстерність англійської або навчити цьому інших.

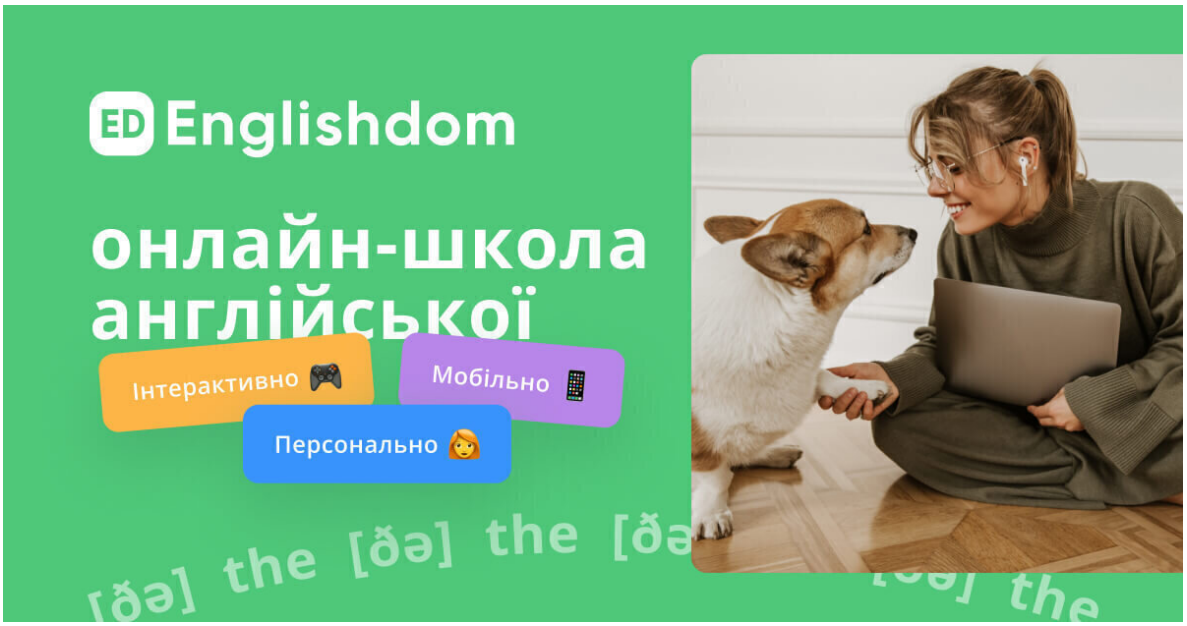


Рис 1.11 EnglishDom

Ось як може бути використана EnglishDom менторами:

- Індивідуалізовані навчальні плани: EnglishDom дозволяє менторам створювати або вибирати курси, що відповідають конкретним мовним потребам та цілям менті. Це може бути зосередження на певних аспектах мови, таких як граматика, лексика, розмовна англійська, бізнес-англійська тощо.

- Гнучкість та доступність: Платформа пропонує гнучкий графік і можливість навчання з будь-якої точки світу, що дозволяє менторам і менті координувати уроки відповідно до своїх зайнятостей та часових зон.
- Кваліфіковані викладачі: EnglishDom співпрацює з досвідченими викладачами та носіями мови, що забезпечує високу якість навчання та можливість поглибленого занурення в мовне середовище.
- Інтерактивність та залученість: Курси часто включають інтерактивні завдання, ігри, відео та аудіоматеріали, що робить процес навчання більш захоплюючим та ефективним.
- Відстеження прогресу: EnglishDom забезпечує інструменти для моніторингу та оцінки прогресу менті, дозволяючи менторам адаптувати навчальний процес та зосередитися на аспектах, які потребують додаткової уваги.
- Спільнота та підтримка: Ментори можуть спілкуватися з іншими викладачами та менторами через форуми, вебінари та спеціалізовані групи, обмінюючись досвідом та методиками навчання.

Використовуючи EnglishDom як платформу для менторства, ментори отримують доступ до комплексного навчального середовища, спрямованого на підвищення рівня володіння англійською мовою, що може бути надзвичайно корисним для менті, які прагнуть покращити свої мовні навички для особистого розвитку, професійного росту або міжнародного спілкування.

1.4.6 Порівняльна таблиця існуючих онлайн платформ

Таблиця 1.1

Сервіс	**Moodle**	**iSpring**	**Coursera**	**Udemy**	**EnglishDom**
Підтримка менторів	Так	Так	Ні	Так	Так
Керування ролями студентів/менторів	Так	Так	Так	Так	Так
Змога керування статистикою та успішністю	Так	Так	Так	Ні	Так
Платформа з відкритим кодом	Так	Ні	Ні	Ні	Ні
Типи послуг ментора (безкоштовні/платні)	Залежить від реалізації	Платний	Залежить від курсу	Залежить від курсу	Платний
Можливість додавати власні курси	Так	Так	Ні	Так	Так
Імпорт з інших платформ	Ні	Так	Ні	Ні	Ні

На основі наданої таблиці з порівняльного аналізу, можна зробити короткий висновок: кожна з платформ - Moodle, iSpring, Coursera, Udemy та EnglishDom - володіє рядом корисних функцій для електронного навчання. Однак, жодна з них не покриває повного спектра можливостей. Наприклад, підтримка менторів, керування ролями та статистикою присутня на більшості платформ, але питання відкритого коду, типів послуг менторів та імпорту з інших платформ варіюється. Жодна існуюча платформа не зможе забезпечити повний спектр функціоналу, який був би корисний для платформи для менторів.

ВИСНОВКИ ДО РОЗДІЛУ 1

У даному розділі проведено ретельний аналіз можливостей онлайн-навчання для менторів, досліджуючи різноманітність і специфіку веб-платформ. Було розкрито переваги та виклики, що вони представляють, і виокремлено ключові аспекти, які роблять їх ефективними для менторства. Огляд популярних платформ, таких як Coursera, Udemy, і EnglishDom, показав, що кожна з них надає унікальні інструменти та ресурси для підтримки різноманітних навчальних цілей.

.Освіта сьогодні є критично важливою для успіху, і постійний пошук ресурсів для вдосконалення знань є необхідним. Онлайн-курси, як засіб досягнення цієї мети, демонструють потенціал стати значною частиною освітньої системи майбутнього. Результати нашого аналізу підкреслюють значення теми і підтверджують актуальність дистанційного навчання як комплементарної, а іноді й необхідної складової освітнього процесу.

Дослідження визначило основні напрямки та проблеми пов'язані з онлайн-навчанням, враховуючи потреби та вимоги сучасних студентів в контексті менторства. Широкий огляд інструментів та ресурсів для дистанційного навчання висвітлив їхню різноманітність і вказав на важливість їх вибору для ефективного менторства.

У розділі також було висвітлено сучасні тенденції в онлайн-навчанні та окреслено які потреби студентів можуть бути задоволені через оптимізовані менторські програми. Це створює міцну основу для подальшої розробки та дослідження платформи пошуку менторів, спрямованої на задоволення цих потреб та підтримку освітнього прогресу користувачів.

Наступні частини роботи зосередяться на детальному проектуванні, розробці та оцінці такого сервісу, розглядаючи його практичну реалізацію та вплив у реальному освітньому середовищі.

РОЗДІЛ 2. ПРОЕКТУВАННЯ КОМПОНЕНТІВ СИСТЕМИ ТА ВИБІР ТЕХНОЛОГІЙ

Проектування системи є ключовим елементом в документації будь-якого проекту, що включає в себе детальний опис архітектури, компонентів, інтерфейсів та інших аспектів системи, які будуть розроблені або модифіковані. Цей розділ є фундаментальним у документації будь-якого проекту програмного забезпечення. Він описує структурне впорядкування системи, включаючи її компоненти, взаємодію між ними, а також середовище, в якому програмне забезпечення функціонує технічні та функціональні вимоги системи, забезпечує вихідну точку для розробки та дає зрозуміле уявлення про кінцевий продукт.

2.1 Архітектура проекту

Архітектура програмного забезпечення представляє собою осмислену модель для виконання алгоритмів через програмування структури системи. Визначаючи, які компоненти входять до системи, які функції виконують ці компоненти, і як вони взаємодіють між собою, архітектура відображає загальну структуру всієї системи а також визначає вищий рівень організації системи та є ключовим для розуміння того, як виконувати проектування, розробку та подальше утримання системи.

- Під час проектування архітектури важливо враховувати різні фактори, такі як призначення системи, цільова аудиторія чи користувачі, на яку спрямована робота системи, і функції, які є найбільш важливими для користувачів.
- Архітектура програмного забезпечення має велике значення, особливо для великих систем. Якщо на етапі розробки є чіткий проект загальної системи, розробники мають основу, на яку вони можуть спиратися під час розробки. Це допомагає уникнути конфліктів, дублювання та зайвої роботи і сприяє полегшенню обслуговування, повторному використанню та адаптації системи.

2.2 Вибір шаблону архітектури системи

Визначення оптимального шаблону архітектури системи є одним з ключових етапів при створенні ефективної та стабільної платформи для сервісу з пошуку менторів для онлайн навчання. Шаблон архітектури визначає основну структуру та взаємозв'язки між компонентами системи. Правильний вибір шаблону сприятиме досягненню балансу між продуктивністю, масштабованістю та здатністю системи до адаптації до змін.

Монолітна архітектура є традиційним підходом до побудови програмного забезпечення, де всі компоненти програми тісно інтегровані та працюють як єдиний, неділимий блок. Вона зазвичай має одну кодову базу і всі компоненти програми, включаючи базу даних, серверну логіку та клієнтську частину, запускаються у єдиному процесі, кілька компонентів об'єднані в єдиний великий додаток.

Ось основні характеристики монолітної архітектури:

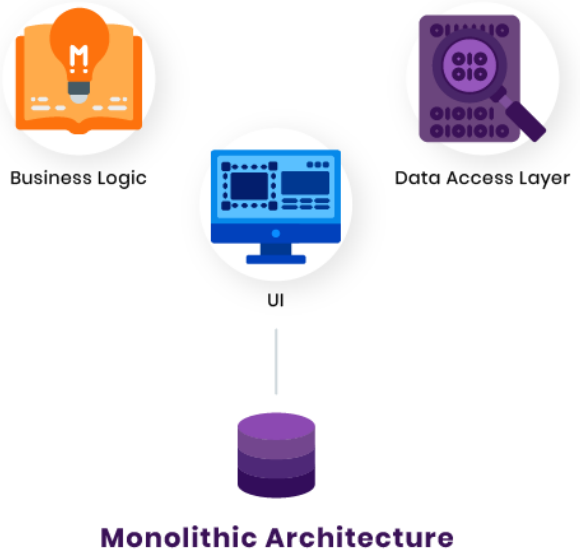


Рис 2.1 Монолітна архітектура

Особливості монолітної архітектури включають:

- Простота розгортання, розробки та вдосконалення: Оскільки система об'єднана в одному застосунку, розробка, розгортання та тестування можуть бути здійснені легше та ефективніше.
- Завдяки єдиній базі коду можна достатньо швидко перевірити будь-яку

необхідну функцію, оскільки всі функції знаходяться в одному репозиторії. Також легше побачити вплив нових подій через залежності цього флоу.

- Повторне використання коду: сскільки весь код знаходиться прямо всередині програми, його легко використовувати повторно та створювати на основі наявних функцій
- Консистентність: Моноліти часто використовують реляційні бази даних OLTP (онлайн-обробка транзакцій), які мають надійні гарантії узгодженості. Ці типи баз даних у монолітних програмах зазвичай мають гарантії ACID (атомарність, послідовність, ізоляція та довговічність), які надають традиційні гарантії узгодженості, до яких ми звикли
- Спрощене тестування: Однорівнева програма дозволяє проводити тестування швидко, оскільки система є централізованим модулем.
- Відсутність накладних витрат на мережу та обмежені зовнішні залежності: усі різні модулі викликають один одного безпосередньо всередині програми. Немає віддалених викликів через мережу через зовнішні API або брокери подій.

Недоліки монолітної архітектури включають:

- Сильно взаємозалежні компоненти. Це може стати проблемою для впровадження змін у такому великому та складному додатку з високим зв’язком. Будь-яка зміна коду впливає на всю систему, тому створює більше технічних проблем і має бути ретельно скоординовано.
- Ускладнення розвитку та розгортання: Великі проекти можуть зазнати ускладнення у розвитку та розгортанні нових функцій та змін.
- Надійність: Помилка в одному модулі може вплинути на доступність всієї програми(але масштабування може впоратись з цим).

2.3 Масштабованість для систем і служб

Масштабована система - це та, яка здатна ефективно реагувати на зміни робочого навантаження та вимог користувачів. Рівень масштабованості

визначається тим, наскільки добре система може адаптуватися до змін, додаючи чи видаляючи ресурси для задоволення потреб. Архітектура включає в себе апаратне забезпечення, програмне забезпечення, технології та оптимальні практики, які формують усю систему

Система вважається масштабованою, якщо вона:

- Здатна додавати ресурси та розширюватися для ефективної роботи при зростанні попиту та робочого навантаження.
- Легко та безперешкодно може зменшувати ресурси при зниженні попиту та робочого навантаження.

Шаблон масштабованості - це визначений та перевірений спосіб розробки або конфігурації системи, який дозволяє ефективно збільшувати або зменшувати її масштаби відповідно до змін робочого навантаження чи вимог користувачів. Шаблони масштабування можуть включати стратегії горизонтального та вертикального масштабування, кешування, реплікацію даних, використання CDN (мережі доставки контенту) та інші техніки, що дозволяють оптимально вирішувати проблеми масштабованості.

Горизонтальне масштабування та вертикальне масштабування представляють собою два відмінних підходи до масштабування системи, обидва з яких можуть бути використані для покращення продуктивності та потужності системи на рис 2.4.

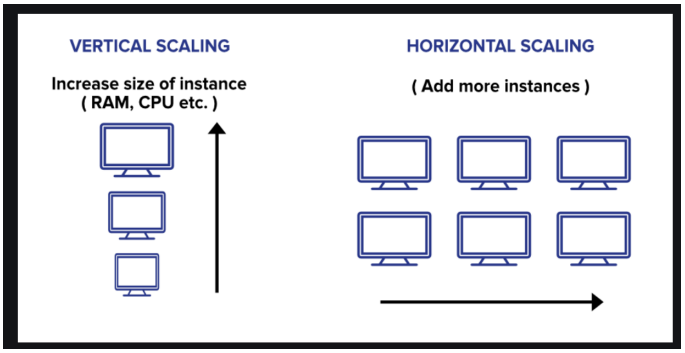


Рис 2.2 Вертикальне та горизонтальне масштабування

Зміна потужності одного обчислювального вузла або перехід до більш потужного обладнання охарактеризовано як вертикальне масштабування. Збільшення потужності відбувається за рахунок покращення параметрів,

таких як процесори чи оперативна пам'ять, або внесення інших змін для підвищення продуктивності. Цей вид масштабування відносно простий, оскільки вимагає лише переходу від менших до більших машин, не втручаючись у код.

У горизонтальному масштабуванні покращується продуктивність сервера через додавання більше фізичних машин/серверів до мережі, розподіляючи робоче навантаження з обробки та пам’яті між кількома пристроями. Це дає змогу ефективно розподілити навантаження між цими серверами, і при цьому підході немає необхідності змінювати потужність сервера або замінювати його.

Горизонтальне масштабування	Вертикальне масштабування
Потрібне балансування навантаження	Не потрібне балансування навантаження
Стійкість до системних збоїв	Єдина точка відмови
Використовує мережні виклики	Міжпроцесова комунікація
Невідповідність даних	Відповідність даних
Простота масштабування	Ліміт по апаратній частині

Таблиця 2.1 Порівняння вертикального і горизонтального масштабування

- Розподіл навантаження: У горизонтальному масштабуванні потрібно забезпечити рівновагу навантаження для розподілу трафіку між різними машинами. У вертикальному масштабуванні, де використовується лише одна машина для виконання завдань, баланс навантаження не є обов'язковим.
- Стійкість до відмов: Горизонтальне масштабування виявляється більш стійким до відмов системи. У разі виходу з ладу однієї машини, можна перенаправити запити на іншу, уникнувши простою. Це не є характерною рисою для вертикального масштабування, де існує ризик створення єдиної точки відмови.
- Мережевий зв'язок: Горизонтальне масштабування передбачає мережевий зв'язок або взаємодію між машинами, що може бути

повільним та схильним до збоїв. У вертикальному масштабуванні використовується міжпроцесний зв'язок, який є достатньо швидким.

- Узгодженість даних: У горизонтальному масштабуванні дані можуть бути неузгодженими, оскільки різні машини обробляють різні запити, що може викликати розсинхронізацію. У вертикальному масштабуванні, де всі запити направляються на одну машину, проблем з неузгодженістю даних відсутні.
- Обмеження: У горизонтальному масштабуванні можна додавати стільки серверів, скільки потрібно, в залежності від бюджету та вимог. У вертикальному масштабуванні існує обмежена ємність, що може спричинити простій та має свою верхню межу.

2.4 Балансувальник навантаження

Балансувальники навантаження ефективно розподіляють запити користувачів та робочі навантаження між групою внутрішніх серверів. Головна ідея полягає в тому, щоб розподілити завдання між різними ресурсами так, щоб жоден з них не був перевантажений. Це допомагає забезпечити доступність та масштабованість послуг.

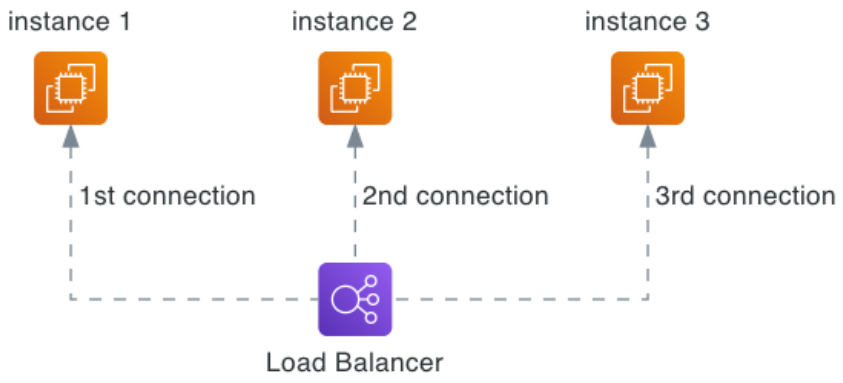


Рис 2.3 Балансувальник навантаження

Основні завдання балансувальника навантаження включають:

- Виявлення ресурсів, доступних у системі.
- Перевірка працездатності цих ресурсів для визначення того, які з них не лише доступні, а й працюють належним чином для обробки робочого навантаження. У випадку пошкодження доступного сервера балансатор

навантаження автоматично переключає трафік на інший сервер, уникнувши затримок чи простою для користувачів.

- Визначення алгоритму балансування роботи між різними функціональними серверами.

На прикладі платформи для менторів використовується балансувальник з інфраструктури AWS - AWS ALB.

AWS ALB представляє собою розподілений балансувальник навантаження, що спрямований на розподіл вхідного трафіку між різними екземплярами додатків. Це дозволяє збільшити масштабованість вашого додатку і відзначити кілька ключових можливостей:

- Автоматичне масштабування: AWS ALB легко інтегрується з AWS Auto Scaling, що дозволяє автоматично збільшувати або зменшувати кількість ресурсів вашого додатку в залежності від обсягу трафіку. Якщо трафік зростає, ALB розподіляє його між новими екземплярами, які можуть бути автоматично створені.
- Забезпечення високої доступності: ALB гарантує високий рівень доступності і надійності, оскільки він автоматично маршрутизує трафік на доступні і здорові екземпляри додатку. Це допомагає уникнути відмов в роботі, навіть якщо один з ресурсів стає недоступним. Завдяки вбудуваній підтримці Health Check можна легко монітирити здорові інстанси задля коректної маршрутизації трафіку.
- Маршрутизація трафіку: AWS ALB дозволяє налаштовувати правила маршрутизації трафіку, щоб розподіляти його на основі URL, хоста, маркерів та інших параметрів. Це дає можливість точно керувати тим, як різні види трафіку обробляються вашим додатком.
- SSL/TLS термінація: ALB виконує термінацію SSL/TLS, розшифровуючи зашифрований трафік і передаючи його до екземплярів додатку в безпечному внутрішньому мережевому трафіку. Це допомагає зменшити навантаження на сервери та поліпшити продуктивність.

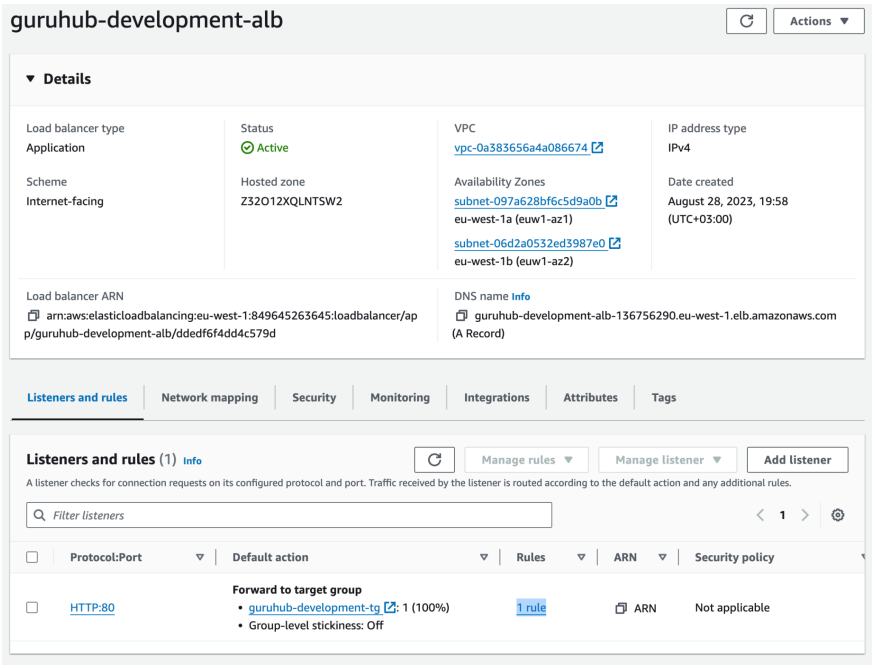


Рис 2.4 Конфігурація AWS ALB в платформі для менторів

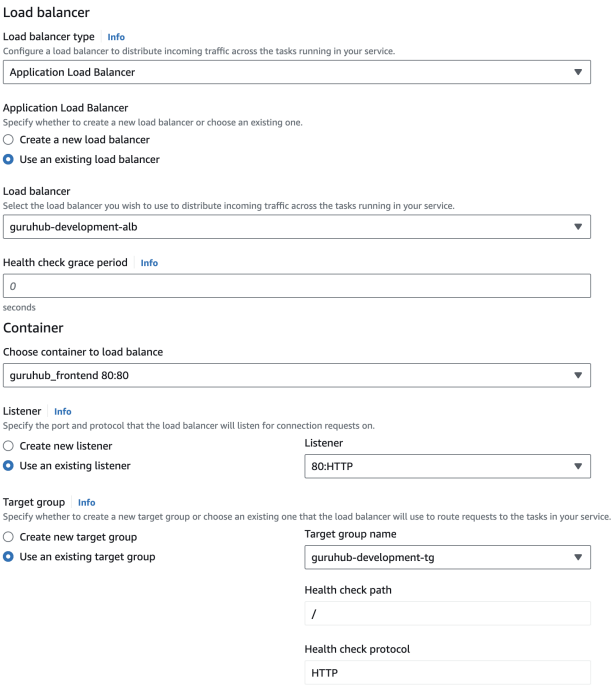


Рис 2.5 Налаштування балансувальника навантаження при ініціалізація інстансів сервісу

Даний сервіс спрощує використання балансувальника навантаження між інстансами, що дозволяє не задумуватись над використаними алгоритмами, легко моніторити ситуацію

2.5 Загальний огляд безпеки даних

2.5.1. Основні виклики та загрози у сфері безпеки веб-застосунків

У сучасному цифровому світі, безпека даних у веб-застосунках є критичним питанням. Головні виклики та загрози включають:

- Атаки міжсайтового сценарію (XSS): Зловмисники вбудовують клієнтські сценарії в браузери користувачів, можливо, зберігаючи шкідливі сценарії в базі даних або надаючи можливість користувачам вибирати посилання, які виконують JavaScript-код зловмисника.
- Атаки підробки міжсайтових запитів (CSRF): Зловмисний користувач виконує дії з використанням облікових даних іншого користувача без його відома чи згоди.
- SQL-ін'єкції: Можливість виконання зловмисним користувачем довільного SQL-коду в базі даних, що може призвести до втрати або витоку даних.
- Clickjacking: Атака, при якій шкідливий сайт упаковує інший сайт під фрейм, може викликати не навмисні дії користувача на цільовому сайті.
- Відсутність використання SSL / HTTPS: Беззахисний обмін даними може призвести до перехоплення облікових даних та модифікації інформації.

2.5.2 Важливість безпеки даних у сучасному світі інтернету

У світі цифрової трансформації та розвитку Інтернету речей, коли веб-застосунки набувають все більшого значення у обробці та обміні даними, безпека інформації стає важливішою ніж коли-небудь. Основні аспекти включають в себе:

- Збереження конфіденційності: Захист від незаконного доступу та небажаного розголошення конфіденційної інформації.
- Гарантія цілісності: Забезпечення непорушності даних та їх захист від незаконних змін чи пошкоджень.
- Забезпечення доступності: Гарантія безперебійного доступу до ресурсів

та послуг в межах веб-застосунку.

- Збереження довіри: Підтримка довіри користувачів до безпеки та конфіденційності їх особистих даних.
- Відповідність: Дотримання вимог законодавства та стандартів у сфері безпеки даних.

У наступних розділах будуть розглянуті конкретні заходи та стратегії для забезпечення безпеки даних у веб-застосунках.

2.5.3. Захист від міжсайтових скриптів (XSS)

Атаки XSS є потенційно небезпечними для веб-застосунків, і належні стратегії захисту грають ключову роль у запобіганні їхнім негативним наслідкам. Ось деякі ефективні підходи для захисту від XSS у застосунках, розроблених з використанням Node.js:

- Використання шаблонів та бібліотек убезпеченого відображення:

Бажано слідувати за безпечними шаблонізаторами та бібліотеками для відображення контенту на сторінках. Приклади таких бібліотек включають EJS, Pug (раніше відомий як Jade) або Handlebars. До прикладу реалізація шаблонізатора в платформі для менторів:

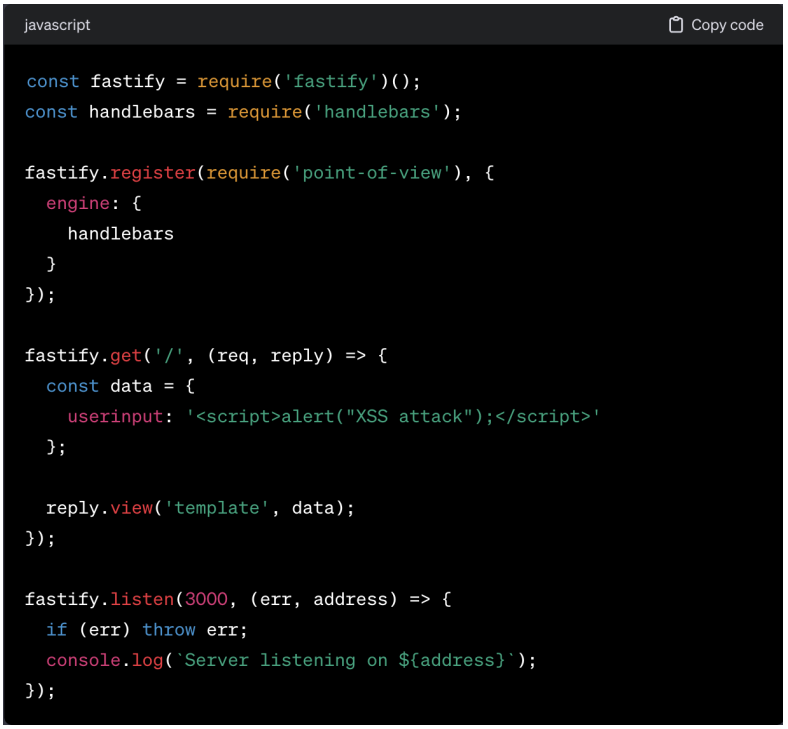


Рис 2.6 Використання шаблонізатора handlebars в платформі для менторів

- Екранування та очищення введених даних:

Слід застосовувати екранування та очищення введених даних перед їхнім відображенням через спеціальні функції для екранування HTML-коду та інших потенційно небезпечних символів. До прикладу використання функції з бібліотеки *sanitize-html,* яка є санітайзером HTML:

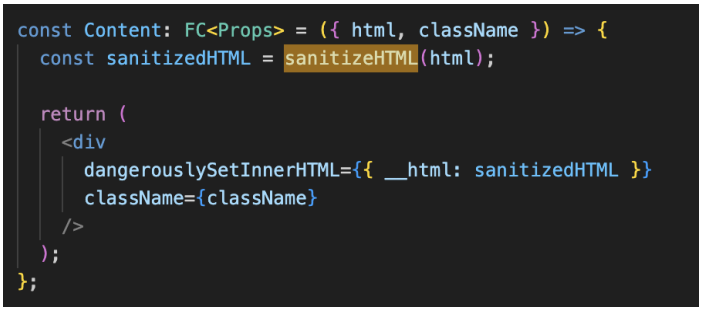


Рис 2.7 Використання санітайзера HTML в платформі для менторів

Опція *dangerouslySetInnerHTML* дає можливість задати динамічне значення для елементу *div.* Сама назва вже містить означення, що це може призвести до вразлості в категорії XSS, тому слід вжити заходів по очищенню HTML коду, який задається для елементу.

- Використання заголовків безпеки браузера:

Слід встановлювати заголовки безпеки браузера, такі як Content Security Policy (CSP), щоб обмежити джерела, з яких можна завантажувати ресурси. Це зменшить ризик впровадження шкідливого коду. До прикладу використання обмеження ресурсів з платформи для менторів:

Рис 2.8 Content Security Policy в платформі для менторів

1. default-src 'self';: вказує, що за замовчуванням сторінка може мати лише ресурси з того самого походження.
2. script-src 'self';: Дозволяє виконувати код JavaScript лише з того самого джерела.
3. img-src https://*.my-s3-endpoint.com;: Визначає, що дозволено завантажувати лише зображення з вказаної кінцевої точки S3 та її субдоменів.
4. media-src https://*.my-s3-endpoint.com;: дозволяє завантажувати аудіо- та відеофайли лише з вказаної кінцевої точки S3 та її субдоменів.

Загальною метою є забезпечення недопущення впровадження шкідливого коду в контент та збереження цілісності та безпеки даних користувачів.

2.4.4. Захист від підробки міжсайтових запитів (CSRF)

CSRF-атаки можуть викликати серйозні загрози для безпеки веб-застосунків, тому важливо розглядати відповідні заходи захисту при розробці застосунків на платформі Node.js. Нижче наведено кілька стратегій захисту від CSRF:

- Використання токенів CSRF:

Гарною практикою є вбудовування унікальних токени CSRF у форми та запити, які забезпечують додатковий рівень перевірки автентичності. Ці токени повинні бути генеровані сервером та валідуватися при кожному запиті.

```
fastify.register(require('@fastify/cookie'), { secret })
fastify.register(require('@fastify/csrf-protection'), { cookieOpts: { signed: true } })


fastify.route({
  method: 'GET',
  path: '/',
  handler: async (req, reply) => {
    const token = await reply.generateCsrf()
    return { token }
  }
})


fastify.route({
  method: 'POST',
  path: '/',
  onRequest: fastify.csrfProtection,
  handler: async (req, reply) => {
    return req.body
  }
})
```

Рис 2.9 Генерування та валідація на токени CSRF в платформі для менторів

- Заголовок CSRF та AJAX:

Перевірка наявність токенів CSRF в заголовках запитів, особливо у випадку використання AJAX-запитів може дати бажані плоди. Використання спеціальних заголовків, таких як X-Requested-With зменшить ризик CSRF.

- Використання захищених куків та запитів:

Визначення параметрів куків Secure і HttpOnly, щоб вони передавалися лише через захищені канали і були недоступні для JavaScript-коду в браузері.

- Синхронізація CSRF з сесійними токенами:

Токени CSRF та сесійні токени співпадають, і вони відповідають лише конкретному користувачеві, щоб ускладнити можливість атак.

```
fastify.register(require('@fastify/secure-session'), { secret: "a string which is longer than 32 chara
fastify.register(require('@fastify/csrf-protection'), { sessionPlugin: '@fastify/secure-session' })


fastify.route({
  method: 'GET',
  path: '/',
  handler: async (req, reply) => {
    const token = await reply.generateCsrf()
    return { token }
  }
})


fastify.route({
  method: 'POST',
  path: '/',
  onRequest: fastify.csrfProtection,
  handler: async (req, reply) => {
    return req.body
  }
})
```

Рис 2.10 Генерування та валідація на сесійні токени CSRF в платформі для менторів

- Обмеження запитів:

Обмеження на типи HTTP-запитів, які мають доступ до чутливих операцій, таких як POST або PUT. Це дасть змогу знизити можливість здійснення атак.

- Перевірка реферера:

Включення перевірки реферера, щоб гарантувати, що запити надходять від легітимних джерел.

```
import fastifyCors from '@fastify/cors';
import Fastify from 'fastify';
import Knex from 'knex';

import { ENV } from '~/common/enums/enums';

const app = Fastify({
  logger: {
    transport: {
      target: 'pino-pretty',
    },
  },
});

app.register(fastifyCors, {
  origin: ENV.APP.NODE_ENV === 'development' ? '*' : ENV.APP.FRONTEND_URL,
  methods: ['GET', 'POST', 'PUT', 'DELETE', 'OPTIONS'],
});
```

Рис 2.11 Перевірка реферера та обмеження методів запитів в платформі для менторів

2.4.5. Захист від SQL-ін'єкцій в Node.js

SQL-Ін'єкції є серйозною загрозою для безпеки веб-застосунків, тому важливо вживати належні заходи безпеки при використанні Node.js. Нижче подано кілька стратегій захисту від SQL-Ін'єкцій:

- Використання параметризованих запитів:

Замість конкатенації рядків слід використовувати параметризовані запити для взаємодії з базою даних. Це дозволяє додатково відокремити дані від SQL-коду, ускладнюючи можливість впровадження шкідливих даних.

```
await knex(TABLE_NAME)
  .whereIn(
    'key',
    permissions.map((permission) => permission.key),
  )
  .del();
```

Рис 2.12 Приклад параметризованого запиту в платформі для менторів

- Використання ORM-Бібліотек:

Можна розглянути використання ORM-бібліотек, таких як Sequelize, TypeORM, Knex&Objection, які автоматично генерують безпечні SQL-запити. ORM використовує моделі даних та об'єктно-орієнтований підхід, зменшуючи

ризик SQL-Ін'єкцій.

- Екранування введених даних:

Перш ніж включити введені дані в SQL-запит, має бути екранування для уникнення впровадження шкідливого SQL-коду. Використання вбудованих функцій екранування дозволяє правильно обробляти дані.

```
const { mentors } =
  (await this.#CourseModel
    .query()
    .where({ 'courses.id': courseId })
    .andWhere((builder) => {
      if (mentorName) {
        builder.whereRaw('fullName ilike %:mentorName:%', { mentorName });
      }
    })
    .whereIn('mentors.id', filteredMentorIds)
    .select('mentors')
    .withGraphJoined(
      'mentors(withoutPassword).[userDetails(withoutMoneyBalance).[avatar]]',
    )
    .first()
    .castTo<CourseAllMentorsDto>()) ?? {};
```

Рис 2.13 Приклад екранування даних на рівні ORM в платформі для менторів

2.5 Аутентифікація JWT

Аутентифікація представляє собою суттєву складову будь-якої веб-програми, оскільки вона визначає ідентичність користувача і контролює доступ до захищених ресурсів.

Аутентифікація за допомогою JSON Web Token (JWT) - це метод без збереження стану, який забезпечує безпечну передачу інформації у форматі JavaScript Object Notation (JSON) між різними сторонами.

Термін "без стану" у контексті автентифікації вказує на те, що сервер не зберігає інформацію про стан сеансу між послідовними запитами. У системі автентифікації без зберігання стану кожен запит обробляється індивідуально і містить всю необхідну інформацію для перевірки та підтвердження ідентичності та прав доступу користувача чи організації. У випадку аутентифікації за допомогою JWT ця інформація передається у вигляді токену.

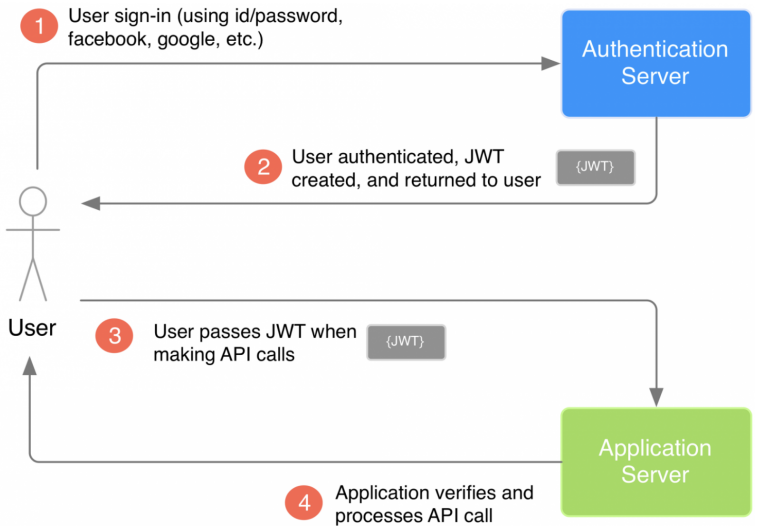


Рис 2.14 Принцип роботи аутентифікації JWT

- Увійшовши в систему: користувач передає свої облікові дані, такі як ім'я користувача та пароль, веб-додатку або системі для перевірки. Ці дані передаються на сервер автентифікації.

```
const encryptionData = {
  data: signInUserDto.password,
  salt: user.passwordSalt,
  passwordHash: user.passwordHash,
};

const isPasswordValid = await this.#encryptService.compare(encryptionData);

if (!isPasswordValid) {
  throw new AuthError({
    status: HttpCode.BAD_REQUEST,
    message: ExceptionMessage.BAD_CREDENTIALS,
  });
}
```

Рис 2.15 Перевірка облікових даних юзера в платформі для менторів

- Створення токенів: після успішної автентифікації сервер створює JSON-токен, який містить важливу інформацію про користувача та сеанс автентифікації. Сервер надсилає цей токен клієнту для подальшої перевірки.

```
public async create(data: TokenPayload): Promise<string> {
  const secretKey = await generateSecret(this.#alg);

  return new SignJWT(data)
    .setProtectedHeader({ alg: this.#alg })
    .setExpirationTime(this.#expiresIn)
    .sign(secretKey);
}
```

Рис 2.16 Реалізація методу генерації токена за допомогою бібліотеки jose

- Зберігання токенів: клієнт зберігає токен, зазвичай, у файлі cookie або в спеціальному локальному сховищі та включає його у наступні запити до сервера.

Рис 2.17 Реалізація виклику методу на збереження токена в localStorage в платформі для менторів

- Перевірка користувача та відповідь сервера: коли клієнт відправляє запит на сервер додатків, він перевіряє підпис у токені та перевіряє твердження в корисному навантаженні, щоб переконатися, що користувач має доступ до запитуваного ресурсу.

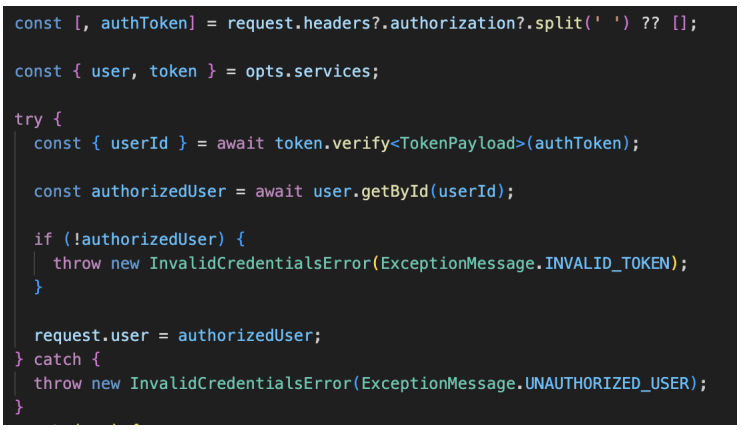


Рис 2.18 Реалізація перевірки користувача маючи токен

- Термін дії токена: коли термін дії JWT закінчується, клієнт повинен отримати новий токен, авторизувавшись знову.

Рис 2.17 Приклад перевірки токена

2.6 OAuth 2

OAuth 2 - це архітектура авторизації, яка дозволяє програмам, таким як Facebook, GitHub і Google, отримувати обмежений доступ до облікових записів користувачів у HTTP-сервісі. Вона працює шляхом передачі доступу над процесом автентифікації користувача в службу, яка зберігає обліковий запис користувача, і надання дозволу програмам сторонніх розробників на доступ до цього облікового запису користувача.

- Додаток запитує у користувача дозвіл на доступ до ресурсів сервісу.
- Якщо користувач надає дозвіл на авторизацію, програма отримує дозвіл на авторизацію.
- Додаток надсилає запит на отримання маркера доступу до сервера авторизації (API), при цьому представляючи свою власну ідентифікацію та дозвіл на авторизацію.
- Якщо ідентифікатор програми автентифікований і дозвіл на авторизацію є дійсним, сервер авторизації (API) видаватиме маркер доступу до програми, завершуючи процес авторизації.
- Програма може запитувати ресурс із сервера ресурсів (API) та представляти маркер доступу для авторизації.
- Якщо маркер доступу є дійсним, сервер ресурсів (API) надає ресурс програмі.

На прикладі платформи для менторів можна розглянути OAuth2 авторизацію із сервісом Google:

```
fastify.register(fastifyOauth2, {
  name: 'googleOAuth2',
  scope: ['profile', 'email'],
  credentials: {
    client: {
      id: clientId,
      secret: clientSecret,
    },
    auth: fastifyOauth2.GOOGLE_CONFIGURATION,
  },
  startRedirectPath: `${ApiPath.AUTH}${AuthApiPath.GOOGLE_SIGN}`,
  callbackUri: `${baseURL}${ApiPath.AUTH}${AuthApiPath.GOOGLE_CALLBACK}`,
  callbackUriParams: {
    access_type: 'offline',
  },
});
```

Рис 2.18 Реєстрація ендпоінтів та ключів доступу для запитів на авторизацію

Дана конфігурація створює підписку на 2 роути: *startRedirectPath* - ендпоінт, після виклику якого виконується запит на ресурс Google на авторизацію використовуючи ключі доступу. У відповідь, сервер Google виконає запит на *callbackUri* ендпоінт, підписка на який реалізована на рис 2.19 з авторизаційним токеном, який отримає дані про юзера через інший запит на API з заданим токеном.

```
fastify.route({
  method: HttpMethod.GET,
  url: AuthApiPath.GOOGLE_CALLBACK,
  async handler(req, rep) {
    const {
      token: { access_token: accessToken, token_type: tokenType },
    } = await this.googleOAuth2.getAccessTokenFromAuthorizationCodeFlow(req);

    const { name: fullName, email } = await httpService.load<
      Record<'name' | 'email', string>
    >('https://www.googleapis.com/oauth2/v2/userinfo', {
      headers: { Authorization: `${tokenType} ${accessToken}` },
      method: HttpMethod.GET,
    });

    const { token } = await authService.signOAuth({ fullName, email });

    return rep.redirect(
      `${frontendURL}${AuthApiPath.SIGN_REDIRECT}?token=${token}`,
    );
  },
});
```

Рис 2.19 Реалізація колбек ендпоінту, який слухає на відповідь ресурсу Google після авторизації

2.7 Огляд технічного стеку

2.7.1 JavaScript

JavaScript — це легка, кросплатформна, однопотокова та інтерпретована компільована мова програмування. Він також відомий як мова сценаріїв для веб-сторінок. Він добре відомий розробкою веб-сторінок, і багато середовищ без браузера також потрібно його.

JavaScript можна використовувати як для розробки на стороні клієнта, так і для розробки на стороні сервера.

- На стороні клієнта: надає об’єкти для керування браузером і його об’єктною моделлю документа (DOM).
- На стороні сервера: він надає об’єкти, які стосуються запуску JavaScript на сервері. Корисний фреймворк, який сьогодні є найвідомішим, — це

Node.js.

2.7.2 Платформа Node.js

Node.js - це потужна та гнучка платформа для розробки серверних додатків, яка базується на мові програмування JavaScript. Вона використовується для створення високопродуктивних та масштабованих веб-застосунків. Нижче розглянуті кілька ключових аспектів платформи Node.js:

- Асинхронність та події: дозволяє ефективно обробляти багато запитів одночасно.
- Пакетний Менеджер npm: інструмент для управління залежностями та розповсюдженням пакетів у величезній екосистемі.
- Розширюваність: підтримує розширення та модульність, що полегшує створення масштабованих систем.
- Активна спільнота: користується великою популярністю серед розробників та має активне та велике ком'юніті, яке надає підтримку, обмін знанням та сприяє обміну досвідом, росту та розвитку платформи.
- Швидкість та Продуктивність: завдяки вбудованому двигуну V8 від Google, Node.js забезпечує високу швидкість виконання коду.

Недоліки Node.js:

- Недостатня підтримка CPU-інтенсивних завдань: може виявитися менш ефективним у виконанні завдань, які сильно навантажують процесор

2.7.3 Typescript

JavaScript — це динамічна мова програмування без системи типів. Typescript як надмножина JavaScript, тобто компілюється в простий JavaScript, додає статичні типи та класові структури до мови. Система типів підвищує якість коду, його читабельність і полегшує підтримку та рефакторинг кодової бази. Додатковою особливістю є той факт, що помилки можна виявити під час компіляції, а не під час виконання.

TypeScript підтримує новіші стандарти ECMAScript, та навіть дозволяє

використовувати майбутні функції JavaScript завдяки транспіляції в код, сумісний з поточними браузерами.

2.7.4 Fastify - backend фреймворк

Fastify - це ефективний та швидкий веб-фреймворк для Node.js, що зосереджений на забезпеченні високої продуктивності та ефективності розробки. Він був створений з метою бути легким та простим у використанні, при цьому забезпечуючи швидкість обробки запитів, орієнтований на забезпечення найкращого досвіду розробника з найменшими витратами та потужною архітектурою плагінів, що робить його ідеальним для будівництва швидких HTTP API, веб-додатків та сайтів

Ось основні функції та принципи, на яких побудовано Fastify:

- Модульність та розширюваність: Легко інтегрувати з іншими модулями та плагінами для розширення функціональності.
- Спрощення розробки: Простий у використанні інтерфейс та документація роблять розробку швидкою та ефективною.
- Схеми валідації: Він використовує схеми валідації JSON Schema для валідації вхідних та вихідних даних, що забезпечує безпеку та стабільність API.
- Підтримка асинхронності: Він підтримує асинхронну обробку запитів з використанням async/await, що дозволяє писати більш чистий та зрозумілий код.

2.7.5 OpenAPI специфікація

OpenAPI специфікація використовує стандартизований формат для опису всіх аспектів RESTful API, включаючи кінцеві точки, методи, параметри, тіло запитів та відповідей, коди стану, що може бути легко сприйнята як людьми, так і комп'ютерами, сприяючи кращому розумінню та використанню API. Хоча специфікація часто використовується разом з HTTP API, вона не залежить від жодної мови програмування, що робить її універсальною та використовуваною в різноманітних середовищах.

Swagger – це набір відкритих інструментів, розроблений на основі специфікації OpenAPI, які сприяють проектуванню, створенню, документуванню та використанню REST API.

2.7.6 React - frontend бібліотека

React — популярна бібліотека JavaScript для створення інтерфейсів користувача. React дозволяє розробникам швидко розробляти складні користувацькі інтерфейси з мінімальною кількістю коду, що робить розробку швидше, ніж будь-коли раніше. Особливістю є використання компонентів - фрагменти коду для багаторазового використання замість шаблонів або розмітки HTML під час розробки інтерфейсу програми.

React використовує віртуальний DOM (Document Object Model), що підвищує продуктивність, оскільки зміни в інтерфейсі користувача відбуваються спочатку у віртуальному DOM, а потім, за допомогою алгоритмів порівняння, мінімально необхідні зміни вносяться у реальний DOM.

Однією з особливостей є створенні SPA, де весь контент завантажується один раз, а подальші зміни в інтерфейсі користувача відбуваються без перезавантаження сторінки.

2.7.7 CRA vs Vite

CRA (Create React App) і Vite є інструментами для спрощення початкового налаштування та розробки проектів, що базуються на React.

CRA - це офіційний стартовий шаблон від команди React для створення нових проектів. Він включає конфігурацію на основі Babel[20] і Webpack[20] з нуля для розробки, тестування та збірки React додатків

Vite - є сучасним інструментом для розробки, що використовує оптимізацію для збірки, засновану на імпортах та пропонує швидший час ініціалізації та оновлення за допомогою використання ES модулів для розробки та гарячого перезавантаження.

Відмінності від CRA або конфігурації збірки на основі комплектування

полягають в тому, що Vite не створює усю програму перед обслуговуванням. Він розподіляє модулі програми на дві категорії: залежності та вихідний код.

Залежності представляють собою стандартний JavaScript, який рідко змінюється під час розробки, попередньо об’єднуючи залежності за допомогою esbuild, що в 10-100 разів швидше, ніж збірники CRA на основі JavaScript. Попереднє групування гарантує, дає можливість уникати зайвих накладних витрат та перевантаження мережі. З огляду на те, що залежності залишаються незмінними, їх можна також кешувати, що дозволяє уникнути повторного об'єднання.

2.7.8 Docker

Docker представляє собою відкриту платформу, спрямовану на розробку, доставку, автоматизацію розгортання, запуск програм та масштабування та управління додатками в контейнерах. Контейнеризація дозволяє упакувати додаток із усіма його залежностями в стандартний виконавчий блок, що забезпечує консистентність незалежно від середовища. Ця технологія дозволяє ізолювати ваші програми від інфраструктури, щоб забезпечити швидку поставку програмного забезпечення. За допомогою Docker можна керувати як своєю інфраструктурою, так і самими програмами для доставки, тестування та розгортання коду ви можете суттєво зменшити час між написанням коду та його запуском у виробництві.

2.7.8 PostgreSQL

PostgreSQL - це потужна система керування реляційними базами даних (RDBMS) з відкритим кодом для керування структурованими даними та підтримки зв’язків між різними наборами даних. Широко використовується для керування та організації структурованих даних і відомий своєю надійністю, масштабованістю та підтримкою складних запитів.

З особливостей можна відмітити відповідність ACID (атомність, узгодженість, ізоляція, довговічність), забезпечуючи надійну основу для цілісності даних, масштабованість для обробки великих обсягів даних і одночасних транзакцій.

ВИСНОВКИ ДО РОЗДІЛУ 2

У цьому розділі була розглянути усі аспекти, що стосуються проектування системи, включаючи різноманітні архітектури, масштабованість та безпеку даних, виявилися важливими для успішності проекту. Цей висновок підкреслює ключові принципи та рекомендації, які визначають успішну реалізацію системи та гарантують її ефективність.

Етап обміркованої розробки системи передбачає детальне вивчення та вибір найбільш підходящої архітектури відповідно до потреб проекту, було взято за основу монолітну архітектуру, де всі компоненти програми тісно інтегровані та працюють як єдиний, неділимий блок, що дозволяє ефективно керувати системою та забезпечує гнучкість у виборі технологій для окремих компонентів. Ключовим аспектом ефективної системи є її здатність до масштабованості, гарантує високу доступність та продуктивність системи.

Гарантування безпеки даних є невід'ємною частиною успішного проекту. Застосування технологій, таких як HTTPS, встановлення строгих політик контролю доступу та шифрування даних, – це лише деякі засоби захисту від потенційних загроз. Врахування ефективної валідації та екранування вхідних даних, захист від SQL-ін'єкцій та систематичне оновлення безпеки – ключові аспекти у забезпеченні безпеки даних. Також озглянуто певні аспекти з авторизації, такі як JWT і OAuth2.

Були розглянуті основні технології, що використовуються в проекті, включно з мовами програмування, фреймворками, базами даних та інструментами розробки. Ці рішення були обрані на основі їх продуктивності, гнучкості, спільноти підтримки та відповідності цілям проекту.. Важливо згадати, обрані технології впливають на архітектуру, продуктивність, масштабованість та підтримку проекту.

Узагальнюючи, успішне проектування системи – це умілий баланс між різноманітними факторами, спрямованими на забезпечення продуктивності, гнучкості та надійності. Строге врахування цих принципів дозволяє створити ефективну систему, яка відповідає вимогам та очікуванням користувачів.

РОЗДІЛ 3. РОЗРОБКА ТА РОЗГОРТАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Проєктування бази даних

Для проектування, візуалізації, конструювання та документування баз даних використовуються діаграми різних нотацій, в кожної з яких є свої переваги та цілі. На основі визначених взаємозв'язків і залежностей між різними сутностями можна впорядкувати дані в даталогічну структуру, яку потім можна відобразити в об'єкти зберігання, які підтримуються конкретною СУБД.

Для більшої наглядності розподілю схему на кілька частин для їх опису.

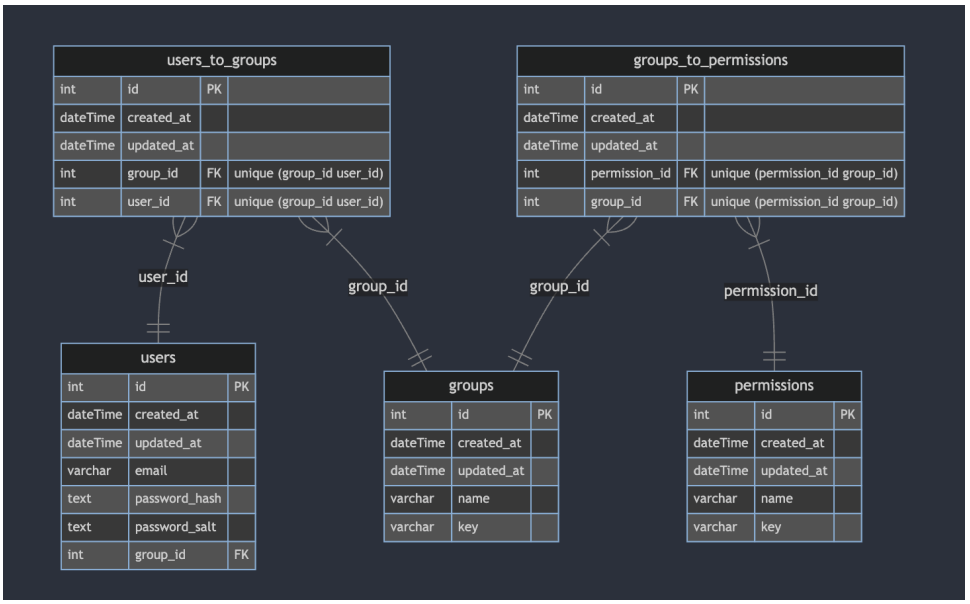


Рис 3.1 Модель зв'язку юзерів з пермішенами

- Користувачі (Users):

Ця сутність представляє користувачів у системі. Поля включають унікальний ідентифікатор (id), мітки часу для створення та оновлення (created_at, updated_at), електронну адресу користувача, хеш пароля, сіль пароля, та зовнішній ключ до сутності groups (group_id).

- Групи (Groups):

Ця сутність представляє групи, до яких можуть належати користувачі. Поля включають унікальний ідентифікатор (id), мітки часу для створення та оновлення, назву групи, та ключ.

- Користувачі до Груп (Users_to_Groups):

Це проміжна таблиця для представлення багато-до-багатьох зв'язків між користувачами та групами. Кожен користувач може належати до багатьох груп, і кожна група може мати багато користувачів. Поля включають унікальний ідентифікатор, мітки часу для створення та оновлення, та зовнішні ключі до сутностей users (user_id) та groups (group_id). Також забезпечується унікальність комбінації group_id та user_id.

- Дозволи (Permissions):

Ця сутність представляє дозволи, які можуть бути надані групам. Поля включають унікальний ідентифікатор (id), мітки часу для створення та оновлення, назву дозволу, та ключ.

- Групи до дозволів (Groups_to_Permissions):

Це проміжна таблиця для представлення багато-до-багатьох зв'язків між групами та дозволами. Кожна група може мати багато дозволів, і кожен дозвіл може бути призначений багатьом групам. Поля включають унікальний ідентифікатор, мітки часу для створення та оновлення, та зовнішні ключі до сутностей permissions (permission_id) та groups (group_id). Також забезпечується унікальність комбінації permission_id та group_id.

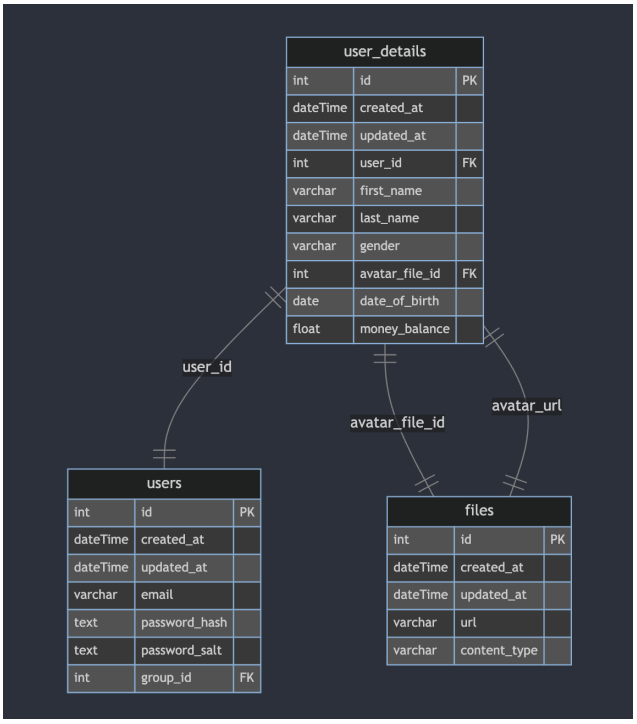


Рис 3.2 Модель зв’язків курсів юзерів, деталей юзерів та файлами

- Деталі користувача (user_details):

Ця сутність представляє детальну інформацію про користувачів. Поля включають унікальний ідентифікатор (id), мітки часу для створення та оновлення (created_at, updated_at), зовнішній ключ до сутності users (user_id), особистісні дані такі як ім'я, прізвище, стать, дату народження, та фінансовий атрибут money_balance. Також є зовнішній ключ до сутності files (avatar_file_id), що вказує на зв'язок з файлом, представляє аватар користувача.

- Файли (files):

Ця сутність представляє файли, які можуть бути аватарами профілю для користувачів або будь-яким іншим типом файлу. Поля включають унікальний ідентифікатор (id), мітки часу для створення та оновлення, URL до файлу url), та тип контенту файлу (content_type).

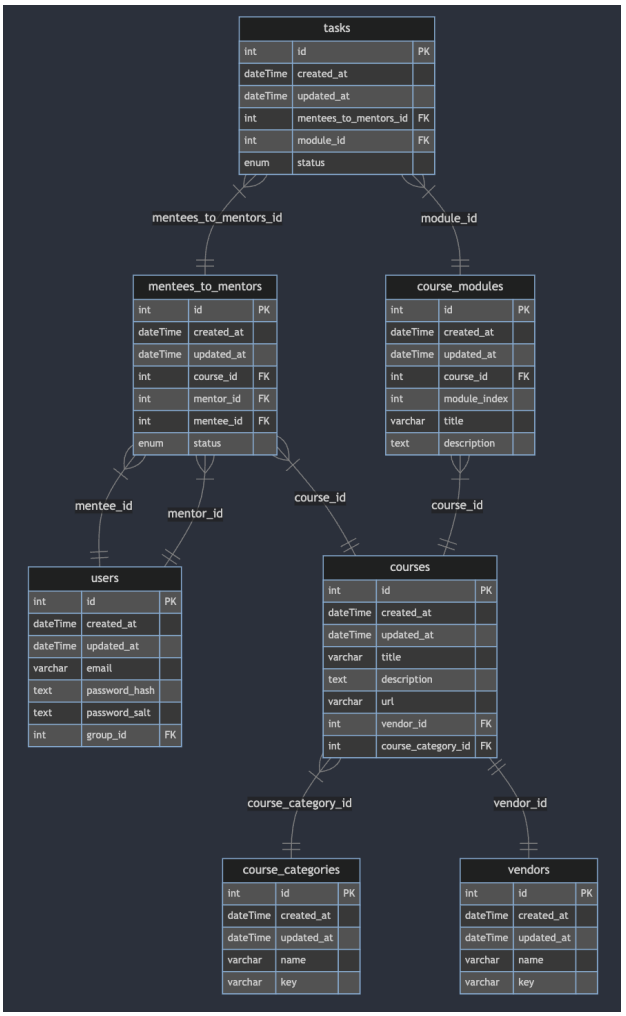


Рис 3.3 Модель зв’язків курсів, вендорів, категорій, менторів і менті, модулів-курсів

- Користувачі (Users): Сутність було описано в попередній частині даталогічної моделі
- Менті-ментор (Mentees_to_Mentors): Проміжна таблиця для відносин між менторами та їх менті. Поля включають ідентифікатор, мітки часу, зовнішні ключі до курсів (course_id), ментора (mentor_id), та менті (mentee_id), а також статус відносин.
- Завдання (Tasks): Сутність для представлення завдань, призначених менті або у рамках курсів. Поля включають ідентифікатор, мітки часу, зовнішні ключі до відносин менті-ментор (mentees_to_mentors_id) та модуля курсу (module_id), та статус завдання.
- Нотатки завдань (Task Notes): Сутність для зберігання нотаток або коментарів до завдань. Поля включають ідентифікатор, мітки часу, зовнішні ключі до завдання (task_id) та автора нотаток (author_id), та статус нотатки.
- Модулі курсів (Course Modules): Сутність для представлення модулів в межах курсів. Поля включають ідентифікатор, мітки часу, зовнішній ключ до курсу (course_id), індекс модуля (module_index), назву (title) та опис модуля (description).
- Категорії Курсів (Course Categories): Сутність для категоризації курсів. Поля включають ідентифікатор, мітки часу, назву категорії (name) та ключ категорії (key).
- Постачальники (Vendors): Сутність для представлення постачальників або видавців курсів. Поля включають ідентифікатор, мітки часу, назву (name) та ключ постачальника (key).

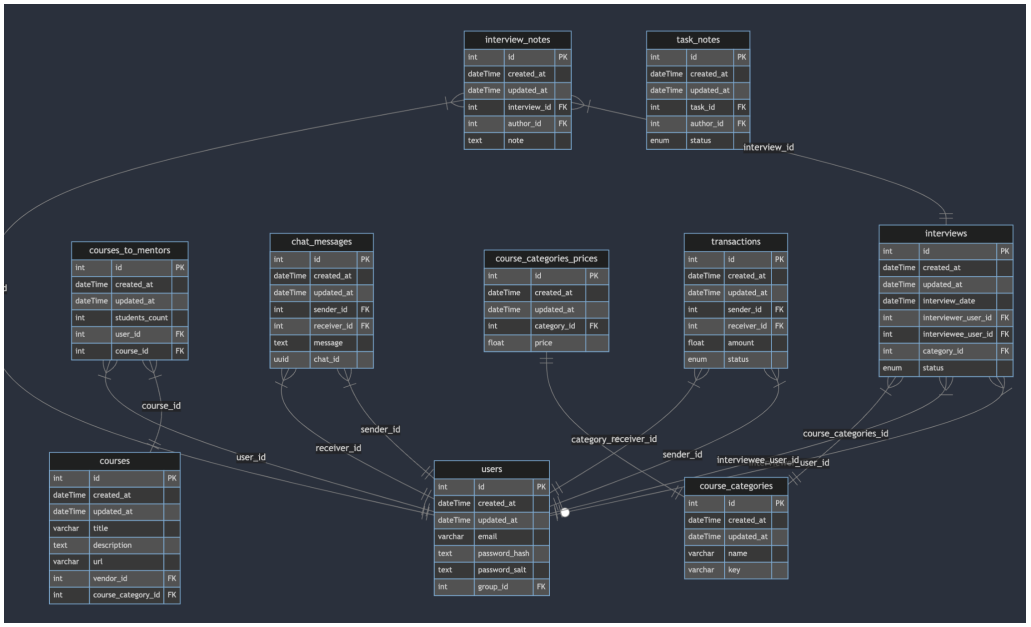


Рис 3.4 Модель зв’язків інтервю, ментора і менті, чату, оплати курсів, цін курсів

- Інтерв'ю (Interviews): Сутність для представлення інтерв'ю. Поля включають ідентифікатор (id), мітки часу створення та оновлення, дату інтерв'ю (interview_date), зовнішні ключі для інтерв'юера (interviewer_user_id), інтерв'ює (interviewee_user_id), категорію курсу (category_id), та статус інтерв'ю.
- Замітки інтерв'ю (Interview Notes): Сутність для зберігання заміток, пов'язаних з інтерв'ю. Поля включають ідентифікатор, мітки часу, зовнішній ключ до інтерв'ю (interview_id), автора замітки (author_id), та текст замітки.
- Повідомлення чату (Chat Messages): Сутність для представлення повідомлень в чаті між користувачами. Поля включають ідентифікатор, мітки часу, зовнішні ключі для відправника (sender_id) та одержувача (receiver_id), текст повідомлення та унікальний ідентифікатор чату (chat_id).
- Користувачі (Users): Сутність було описано в попередній частині даталогічної моделі.
- Категорії Курсів (Course Categories): Сутність для представлення категорій курсів. Поля включають ідентифікатор, мітки часу, назву

категорії та ключ.

- Ціни на категорії курсів (Course Categories Prices): Сутність для представлення цін на категорії курсів. Поля включають ідентифікатор, мітки часу, зовнішній ключ до категорії курсу (category_id), та ціну.
- Транзакції (Transactions): Сутність для представлення фінансових транзакцій між користувачами. Поля включають ідентифікатор, мітки часу, зовнішні ключі для відправника (sender_id) та одержувача (receiver_id), суму транзакції та статус.

3.2 Структура проєкту

Хотілось спершу почати з кількох слів про архітектуру проєкту. Вона представлена монорепозиторієм, в якому ключі модулі backend і frontend знаходяться в одній батьківській директорії.

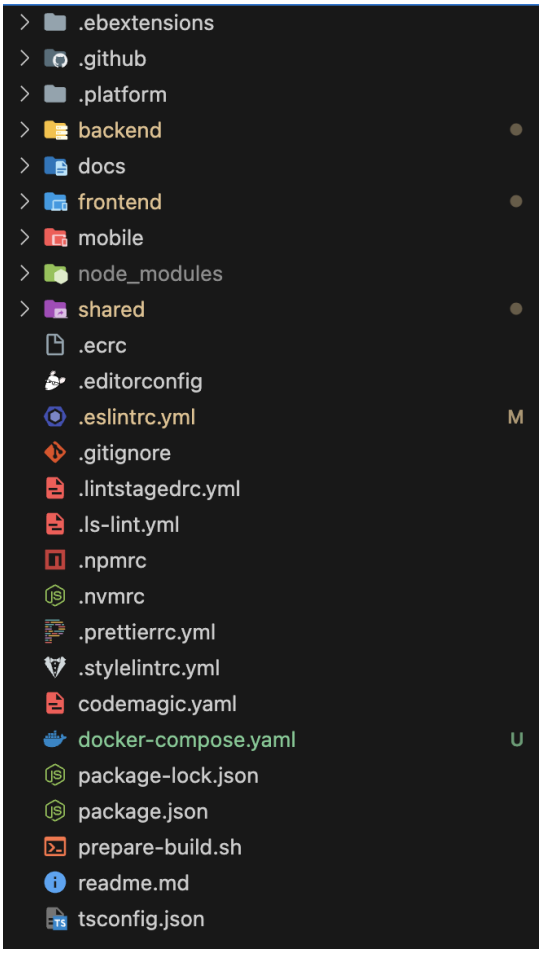


Рис 3.5 Структура монорепозиторія проєкту

Це дає змогу менеджить спільні залежності з однієї вхідної точки. Чудовим прикладом є підмодуль shared, який містить спільні компоненти для

обох вище перечислених модулів. Завдяки лінкуванню, який надає npm - можна доволі просто імпортувати цей модуль як локальний пакет накриклад через *"guruhub-shared": "file:../shared/build"* лінку. Але з цим є маленький нюанс: npm надає абстракцію workspaces для монорепозиторіїв у вигляді, що забезпечує легке імпортування локальних пакетів без додаткових описів. Достатньо лише зазначити конфігурацію:

```json
{ "workspaces": ["shared", "backend", "frontend"] }
```

Додатковою особливістю є можливість запускати скрипти для кожного воркспейсу з батьківської директорії одночасно без потреби наприклад відкривати кілька терміналів чи переміщуючи в директорії модулів. Накриклад, для встановлення всіх залежностів в кожному підмодулі достатньо лише команди:

npm install -ws

3.3 Підтримка консистентності коду

Eslint - npm пакет, який аналізує ваш код, щоб знайти проблеми та автоматично їх виправити. Це дозволяє застосовувати власні правила, наприклад, чи має рядок мати одинарні чи подвійні лапки.

Prettier допомагає відформатувати код, наприклад правильні відступи, кінцеві коми або максимальну довжину рядка. Він поставляється як пакет npm, а також розширення VS Code.

Simple-git-hooks — це інструмент попередньої фіксації, який дозволяє запускати команди або сценарії перед фіксацією або надсиланням нашого коду в git. Він гарантує форматування та виправлення коду перед фіксацією.

Lint-staged може запускати кілька лінтерів для поетапної перевірки, який форматує та виправляє більшу частину файлу перед фіксацією. Lint-staged може запускатися як сценарій попередньої фіксації, який форматує зміни в кодовій частині перед фіксацією за допомогою simple-git-hooks.

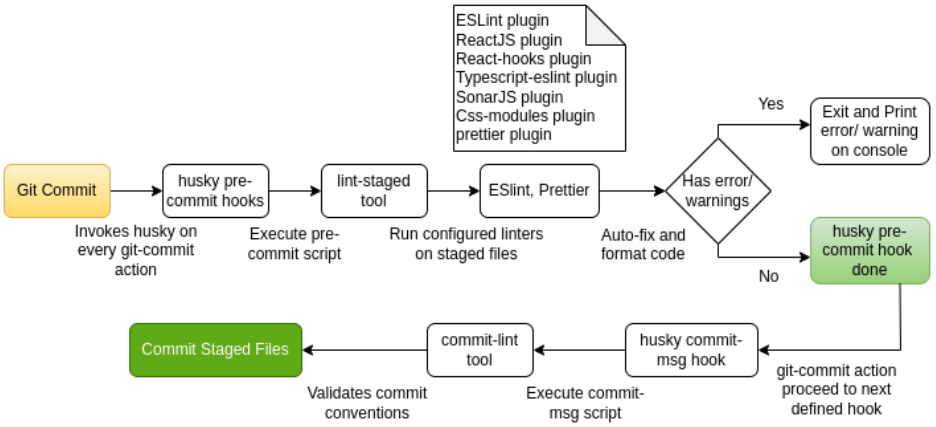


Рис 3.6 Приклад флоу для коміту змін з форматуванням та аналізом коду(husky і simple-git-hooks мають однакові функції)

Приклади конфігурації для цих інструментів наведені в додатку 1.

3.4 Реалізація бекенд частини додатку

Звісно, перший етап є формування стандартизованої структури як на рис 3.7

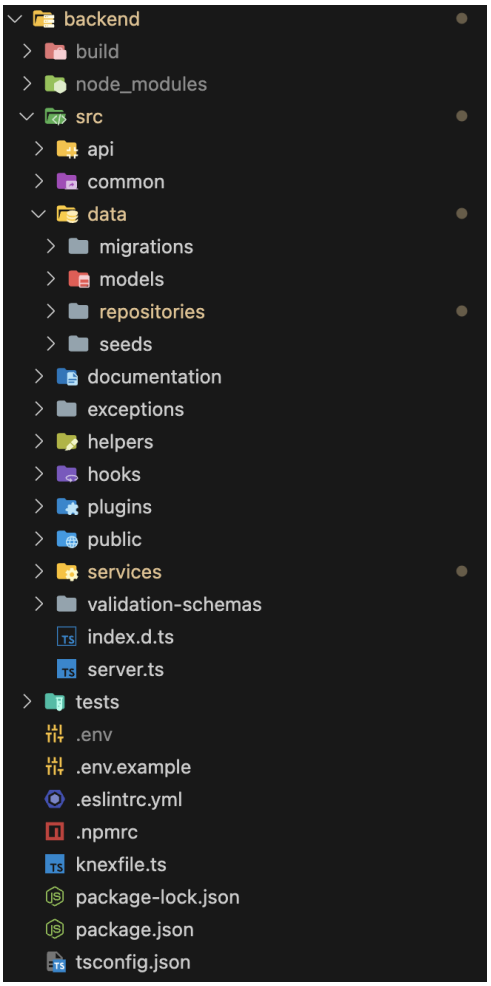


Рис 3.7 Структура бекенд воркспейсу

Відповідно для кожної сутності створено директорії для кожного шару,

чи контролер, чи сервіс, чи репозиторій, чи модель.

Саме через бекенд частину додаток має підключення до бази даних. Підключення до бази даних контролюється Knex&Objection ORM через наступний конфігураційний код:

```
const DEFAULT_ENV_CONFIG: Knex.Config<ConfigPropType> = {
  client: ENV.DB.DIALECT,
  connection: ENV.DB.CONNECTION_STRING,
  pool: {
    min: ENV.DB.POOL_MIN,
    max: ENV.DB.POOL_MIN,
  },
  migrations: {
    directory: './src/data/migrations',
    tableName: DbTableName.MIGRATIONS,
  },
  seeds: {
    directory: './src/data/seeds',
  },
  debug: false,
  ...knexSnakeCaseMappers({ underscoreBetweenUppercaseLetters: true }),
};

const knexConfig: Record<AppEnvironment, Knex.Config<ConfigPropType>> = {
  [AppEnvironment.DEVELOPMENT]: DEFAULT_ENV_CONFIG,
  [AppEnvironment.PRODUCTION]: DEFAULT_ENV_CONFIG,
};

export default knexConfig;
```

Рис 3.11 Підключення до бази даних

З основних компонентів це ініціалізація підключення через облікові дані та додавання можливості мати різні конфігурації в залежності від вибраної опції програмного оточення: development чи production

Для контролю важливих змінних таких як ключі доступу використовується ENV змінні як на прикладі:

```
wilhelm wesser, 15 months ago | 4 authors (fshabanov and others)
#
# APPLICATION
#
NODE_ENV=development
PORT=3001

#
# DATABASE
#
DATABASE_URL=[db_client]://[db_username]:[db_user_password]@localhost:[db_port]/[db_name]
DB_POOL_MIN=2
DB_POOL_MAX=10
DB_DIALECT=pg

#
# UDEMY
#
UDEMY_CLIENT_ID=[udemy_client_id]
UDEMY_CLIENT_SECRET=[udemy_client_secret]
UDEMY_BASE_URL=https://www.udemy.com/api-2.0/
```

Рис 3.12 Приклад опису env змінних

Завдяки цьому, можна доволі легко контролювати важливі облікові дані з однієї точки входу, та й крім того це має на меті убезпечити важливі зміні від втручання ззовні.

Далі наведено приклад опису кожного із шарів додатку для огляду:

```
import { Model, Modifiers, QueryBuilder, RelationMappings } from 'objection';

import { DbTableName } from '~/common/enums/enums';

import { Abstract } from '../abstract/abstract.model';
import { Group, UserDetails } from '../models';

wilhelm wesser, 16 months ago | 6 authors (oleg.gavashi@yahoo.com and others)
class User extends Abstract {
  public 'email': string;

  public 'passwordHash': string;

  public 'passwordSalt': string;

  public static override get relationMappings(): RelationMappings {
    return {
      groups: {
        relation: Model.ManyToManyRelation,
        modelClass: Group,
        join: {
          from: `${DbTableName.USERS}.id`,
          through: {
            from: `${DbTableName.USERS_TO_GROUPS}.userId`,
            to: `${DbTableName.USERS_TO_GROUPS}.groupId`,
          },
          to: `${DbTableName.GROUPS}.id`,
        },
      },
      userDetails: {
        relation: Model.HasOneRelation,
        modelClass: UserDetails,
        join: {
          from: `${DbTableName.USERS}.id`,
          to: `${DbTableName.USER_DETAILS}.userId`,
        },
      },
    };
  }
```

Рис 3.13 Приклад імплементації моделі юзера

Як вже було згадано раніше, даний проєкт містить абстракцію ORM бібліотеки для роботи з базами даних. І складовою підходу є оперування моделями, а не інстансом підключення до бази даних при необхідності отримати ту чи іншу сутність. Тож на приклад рисунку 3.13 зображено визначення моделі юзера з основними складовими та залежностями на інші сутності, що є доволі зручним методом аби в подальшому з мінімальною кількістю коду отримувати юзерів з групами та деталями юзерів в одному запиті.

```
class Group {
  #GroupModel: typeof GroupM;

  public constructor({ GroupModel }: Constructor) {
    this.#GroupModel = GroupModel;
  }

  public async getById(id: number): Promise<GroupsWithPermissionIdsDto | null> {
    const group = await this.#GroupModel
      .query()
      .where('groups.id', id)
      .select('groups.*')
      .withGraphJoined('permissions')
      .castTo<GroupsWithPermissionsDto>()
      .first()
      .then((data) => {
        if (!data) {
          return null;
        }

        return {
          ...data,
          permissionIds: data.permissions.map((permission) => permission.id),
        };
      });

    return group ?? null;
  }
```

Рис 3.14 Приклад імплементації репозиторію групи

Саме цей рівень відповідає за імплементацію відповідних запитів до бази через моделі з подібним до SQL синтаксисом для вибору, фільтрації, сортування, склеюванням сутностей. Тільки для склеювання використовується особлива функція *.withGraphJoined*, яка під капотом використовує звичний для SQL синтаксис JOIN. Тільки обгортка допомагає абстрагуватись від цього. Це має доволі чудовий профіт при склеюваннях кількох сутностей з вкладеностями, що зменшує складність читання і підтримки коду.

```
public getLastMessage({
  userId,
  chatOpponentId,
}: ChatGetLastMessagesRequestDto): Promise<ChatMessageGetAllItemResponseDto> {
  return this.#ChatMessageModel
    .query()
    .select()
    .where({ senderId: userId, receiverId: chatOpponentId })
    .orWhere({ senderId: chatOpponentId, receiverId: userId })
    .orderBy('createdAt', SortOrder.DESC)
    .first()
    .withGraphJoined(
      '[sender(withoutPassword).[userDetails(withoutMoneyBalance).[avatar]], receiver(withoutPassword).[userDetails(withoutMoneyBalance).[avatar]]]',
    )
    .castTo<ChatMessageGetAllItemResponseDto>()
    .execute();
}
```

Рис 3.15 Приклад вкладеності склеювання зв’язаних сутностей в методі для ChatMessage репозиторію.

```
class Transaction {
  #transactionRepository: typeof transactionRep;

  public constructor({ transactionRepository }: Constructor) {
    this.#transactionRepository = transactionRepository;
  }

  public getById(id: number): Promise<TransactionGetAllItemResponseDto> {
    return this.#transactionRepository.getById(id);
  }

  public getHoldBySenderAndReceiverId(
    senderId: number,
    receiverId: number,
  ): Promise<TransactionGetAllItemResponseDto> {
    return this.#transactionRepository.getHoldBySenderAndReceiverId(
      senderId,
      receiverId,
    );
  }

  public getTransactionsByUserId(
    userId: number,
    { count, page }: EntityPaginationRequestQueryDto,
  ): Promise<EntityPagination<TransactionGetAllItemResponseDto>> {
    const zeroIndexPage = convertPageToZeroIndexed(page);

    return this.#transactionRepository.getTransactionsByUserId(userId, {
      count,
      page: zeroIndexPage,
    });
  }
```

Рис 3.15 Приклад імплементації сервісу транзакції

Рівень сервісу вже оперує відповідними методами з рівня репозиторію. Даний підхід є корисним, коли потрібно зробити запит на сутності та й провести якусь бізнес логіку наприклад як на рис 3.16

```
public async initHoldStudentPayment({
  menteeId,
  mentorId,
  rawPriceOfStudying,
}: BillingInitHoldStudentPaymentArgumentsDto): Promise<void> {
  const menteeBalance = await this.#userService.getByIdMoneyBalance(menteeId);

  const priceOfStudying =
    rawPriceOfStudying * Billing.DEFAULT_STUDYING_PRICE_COEFFICIENT;

  if (menteeBalance < priceOfStudying) {
    const symbolsAmount = 2;
    const requiredBalance = `You need to add $${(
      priceOfStudying - menteeBalance
    ).toFixed(symbolsAmount)} to your balance.`;

    throw new BillingError({
      message: `${ExceptionMessage.NOT_ENOUGH_FUNDS_TO_PAY_FOR_MENTORS_SERVICES} ${requiredBalance}`,
    });
  }

  const newMenteeBalance = menteeBalance - priceOfStudying;

  await this.#userDetailsService.updateMoneyBalance(
    menteeId,
    newMenteeBalance,
  );

  const transaction = await this.makeTransaction({
    senderId: menteeId,
    receiverId: mentorId,
    amount: priceOfStudying,
  });
  await this.holdTransaction(transaction.id);
}
```

Рис 3.16 Приклад методу сервіса з бізнес логікою

Рис 3.17 Приклад імплементації контроллера менторів

Це безпосередньо точка входу в кінцеві ендпоінти для бекенду, до яких звертається клієнт. Особливістю є те, що вхідні дані валідуються перед виконанням фнкції *handler* за допомогою схеми на рис 3.17

Рис 3.18 Приклад валідаційної схеми

3.4 Реалізація документації ендпоінтів

Завдяки інтеграція пакету Swagger з Fastify можна задати визначення API схем для кожного роуту з вхідними та вихідними даними, а також з додатковими умовами такі як авторизація та інше.

```yaml
get:
  summary: Get Current User
  tags:
    - auth
  responses:
    '200':
      description: Authorized user
      content:
        application/json:
          schema:
            $ref: '../../schemas/auth/current-user-response-dto.yaml'
    '401':
      description: Unauthorized user
  security:
    - bearerAuth: []
```

```yaml
type: object
properties:
  email:
    type: string
    example: john.doe@gmail.com
  fullName:
    type: string
    example: John Doe
  id:
    type: number
    example: 1
  createdAt:
    type: string
    example: 2022-08-09T08:20:47.167Z
  permissions:
    type: array
    items:
      type: object
      properties:
        id:
          type: number
          example: 1
        key:
          type: string
          example: manage_uam
        name:
          type: string
          example: Manage UAM
```

Рис 3.19 Визначення схем для роуту Get Current User

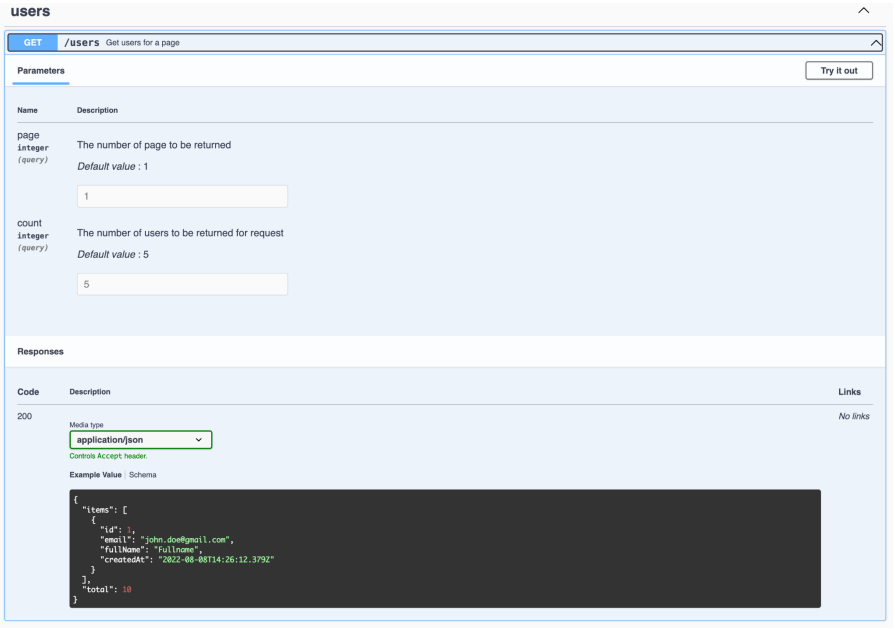


Рис 3.20 Згенерована UI інструментом Swagger

Маючи опис кожного ендпоінта, його вхідні та вихідні параметри, налаштування безпеки такі як авторизація, можна легко згенерувати UI результат, який можна використовувати не тільки як наглядне зображення запитів на сервер з отриманими даними, а й протестувати їх виконання.

3.5 Реалізація клієнтської частини додатку

Насамперед слід почати з структури воркспейсу, що на рисунку 3.21:

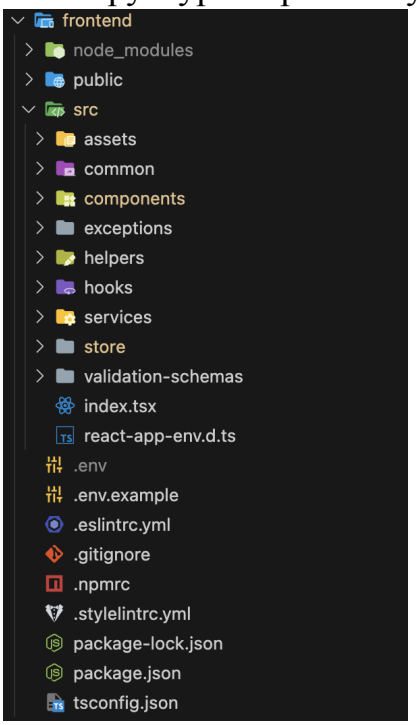


Рис 3.21 Структура фронтенд воркспейсу

З неї випливає, що додаток розбитий на кілька значущих рівнів, що забезпечує його простоту та здатність легко підтримувати в подальшому.

```
import reactPlugin from '@vitejs/plugin-react';
import { type ConfigEnv, defineConfig, loadEnv } from 'vite';
import { VitePWA as vitePWAPlugin } from 'vite-plugin-pwa';
import tsconfigPathsPlugin from 'vite-tsconfig-paths';
import svgrPlugin from 'vite-plugin-svgr';

const config = ({ mode }: ConfigEnv): ReturnType<typeof defineConfig> => {
  const {
    VITE_APP_PROXY_SERVER_URL,
    VITE_APP_API_ORIGIN_URL,
    VITE_APP_DEVELOPMENT_PORT,
  } = loadEnv(mode, process.cwd());

  return defineConfig({
    build: {
      outDir: 'build',
    },
    plugins: [
      tsconfigPathsPlugin(),
      reactPlugin(),
      vitePWAPlugin({
        registerType: 'autoUpdate',
        workbox: {      You, 3 months ago via PR #290 • cp-289: + deny list for
          navigateFallbackDenylist: [new RegExp(`^${VITE_APP_API_ORIGIN_URL}`)],
        },
      }),
      svgrPlugin(),
    ],
    server: {
      port: Number(VITE_APP_DEVELOPMENT_PORT),
      proxy: {
        [VITE_APP_API_ORIGIN_URL]: {
          target: VITE_APP_PROXY_SERVER_URL,
          changeOrigin: true,
        },
      },
    },
  });
};

export default config;
```

Рис 3.26 Конфігурація Vite для додатку

З рисунку 3.26 можна наглядно побачити конфігурацію Vite для модуля, яка має на меті підключити необхідні плагіни для коректної роботи та необхідні змінні оточення з рисунку 3.27.

```
# API
#
REACT_APP_API_ORIGIN_URL=/api/v1

# BUILD
#
GENERATE_SOURCEMAP=false
CI=false

# STRIPE
#
REACT_APP_STRIPE_PUBLIC_KEY=[stripe_public_key]
```

Рис 3.27 Змінні оточення

З особливостей проєкту, хотів би зупинитись на доволі важливому підході управління станом або ж ще як називають store[21].

Глобальні дані, які відносяться до додатку отримуються з сервера через екшени, далі зберігаються та інтегруються в компоненти. Також на основі змін цих даних відбуватиметься перемалювання візуалізації.

Для запиту на зміну чи отримання даних необхідні редакс екшени, які приєднані до глобального хендлера подій, який ще називають dispatch. Вони являють собою обробниками подій, коли якийсь компоненти викликає певну подію в залежності від ініціалізованої дії.

```
enum ActionType {
  GET_COURSES = 'courses-management/get-courses',
  GET_CATEGORIES = 'courses-management/get-categories',
  UPDATE_CATEGORY = 'courses-management/update-category',
}
```

```
const getCourses = createAsyncThunk<
  EntityPagination<CourseGetResponseDto>,
  EntityPaginationRequestQueryDto,
  AsyncThunkConfig
>(ActionType.GET_COURSES, async ({ count, page }, { extra }) => {
  const { coursesApi } = extra;

  const courses = await coursesApi.getAll({
    count,
    page,
  });

  return courses;
});
```

Рис 3.29 Назва події та прилінкована дія

```
useEffect(() => {
  dispatch(
    coursesManagementActions.getCourses({
      page,
      count: PaginationDefaultValue.DEFAULT_COUNT_BY_10,
    }),
  );
  dispatch(coursesManagementActions.getCategories());
}, [dispatch]);
```

Рис 3.20 Приклад виклику дії

І звісно, той функціонал, який буде відстежувати виконання дії на рис 3.21 через reducer.

```
const reducer = createReducer(initialState, (builder) => {
  builder.addCase(getCourses.pending, (state) => {
    state.dataStatus = DataStatus.PENDING;
  });
  builder.addCase(getCourses.fulfilled, (state, { payload }) => {
    state.dataStatus = DataStatus.FULFILLED;
    state.courses = payload.items;
    state.totalCoursesNumber = payload.total;
  });
  builder.addCase(getCourses.rejected, (state) => {
    state.dataStatus = DataStatus.REJECTED;
  });
```

Для кожного доменного модуля задані свої події, дії, обробники, що відповідає рис 3.22

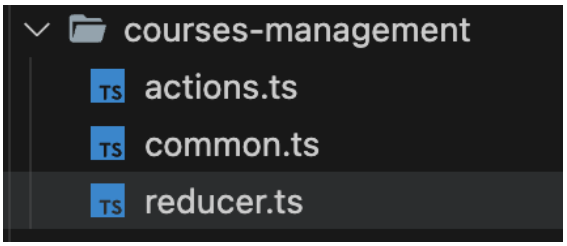


Рис 3.22 Redux підхід до імплементації подій, дій, обробників

```
const store = configureStore({
  reducer: rootReducer,
  middleware: (getDefaultMiddleware) => {
    return getDefaultMiddleware({
      thunk: { extraArgument },
    }).concat([handleUnauthorized, handleError, chatSocket]);
  },
});
```

Рис 3.23 Ініціалізація єдиного об’єкту redux store

```
root.render(
  <StrictMode>
    <Provider store={store}>
      <HistoryRouter history={history}>
        <App />
        <Toast />
      </HistoryRouter>
    </Provider>
  </StrictMode>,
);
```

Рис 3.24 Обгортка додатку за допомогою Provider для наскрізної доступності store.

Далі, в рутовому файлі підключаються всі редюсери і через глобальний провайдер додаються в рутовий компоненти, аби вони були доступні наскріз всього додатку.

Також, варто згадати про абстракцію для запитів на сервер, аби це було стандартизовано. Має бути рутовий сервіс, з якого виконуються всі запити, який мають наслідувати інші доменні сервіси.

```
class Http {
  #storage: Storage;

  public constructor({ storage }: Constructor) {
    this.#storage = storage;
  }

  public load<T = unknown>(
    url: string,
    options: Partial<HttpOptions> = {},
  ): Promise<T> {
    const {
      method = HttpMethod.GET,
      payload = null,
      contentType,
      hasAuth = true,
      queryString,
    } = options;
    const headers = this.getHeaders(contentType, hasAuth);

    return fetch(this.getUrlWithQueryString(url, queryString), {
      method,
      headers,
      body: payload,
    })
      .then(this.checkStatus)
      .then((res) => this.parseJSON<T>(res))
      .catch(this.throwError);
  }
```

Рис 3.25 Приклад Http сервісу

Рис 3.26 Приклад наслідування http сервісу

З деяких особливостей підходу з redux - це використовувати middlewares - функції, які виконуються між шарами логіки. Це доволі зручний спосіб контролювати специфічну поведінку і видавати певний результат. Найкращим прикладом цього функціоналу може бути обробка помилок в контексті показу сповіщень, що якийсь API запит виконався неуспішно як на рис 3.27.

Рис 3.27 - Імплементація middleware для показу помилок в нотифікаціях.

3.6 Організація деплойменту

3.6.1 Контейнеризація додатку

Згідно з вибраним планом контейнеризація додатку, реалізація цього здійснена через створення 2 контейнерів - для бекенду і фронтенду відповідно. Для кожного воркспейсу створено відповідні конфігураційні файли для Docker для висвітлення, що вміщувати в контейнери при деплойменті.

Відповідно для бекенд контейнеру наведений конфіг на рис 3.28 та рис 3.29 - для фронтенду

```
FROM node:18.16.0 as backend-builder

ARG NODE_ENV
ENV NODE_ENV=$NODE_ENV
ARG HOST
ENV HOST=$HOST
ARG PORT
ENV PORT=$PORT

WORKDIR /app
COPY ./package*.json ./tsconfig.json  ./
COPY ./shared/ ./shared/
COPY ./backend/package.json ./backend/package.json
RUN npm ci

COPY ./backend/ ./backend/
RUN npm run build:backend

EXPOSE 3001
CMD npm start -w backend
```

Рис 3.28 Dockerfile для бекенду

```
FROM node:18.16.0 as frontend-builder

ARG VITE_APP_NODE_ENV
ENV VITE_APP_NODE_ENV=$VITE_APP_NODE_ENV
ARG VITE_APP_API_ORIGIN_URL
ENV VITE_APP_API_ORIGIN_URL=$VITE_APP_API_ORIGIN_URL
ARG VITE_APP_PROXY_SERVER_URL
ENV VITE_APP_PROXY_SERVER_URL=$VITE_APP_PROXY_SERVER_URL
ARG VITE_APP_STRIPE_PUBLIC_KEY
ENV VITE_APP_STRIPE_PUBLIC_KEY=$VITE_APP_STRIPE_PUBLIC_KEY

WORKDIR /app
COPY ./package*.json ./tsconfig.json  ./
COPY ./shared/ ./shared/
COPY ./frontend/package.json ./frontend/package.json
RUN npm ci

COPY ./frontend ./frontend/
RUN npm run build:frontend

FROM nginx:1.22.0-alpine

COPY ./nginx/nginx.development.conf /etc/nginx/nginx.conf
RUN rm -rf /usr/share/nginx/html/*
COPY --from=frontend-builder /app/frontend/build/ /usr/share/nginx/html

EXPOSE 80

ENTRYPOINT ["nginx", "-g", "daemon off;"]
```

Рис 3.29 Dockerfile для фронтенду

Бекендове середовище інсталює необхідні пакети, компілює код з Typescript у Javascript та запускає веб-сервер. У випадку фронтенду процес також передбачає встановлення залежностей та компіляцію проекту в один загальний файл, після чого контент обслуговується через сервер nginx.

Нижче наведено конфігураційний файл для сервера nginx, який зображено на рисунку 3.30:

```
worker_processes auto;

events { worker_connections 1024; }

http {
  sendfile on;

  upstream backend {
    server localhost:3001;
  }

  server {
    listen 80;
    server_name localhost;

    client_max_body_size 10M;

    root /usr/share/nginx/html;
    index  index.html index.htm;
    include /etc/nginx/mime.types;

    location /api {
      proxy_http_version 1.1;
      proxy_set_header   Upgrade $http_upgrade;
      proxy_set_header   Connection keep-alive;
      proxy_set_header   Host $host;
      proxy_cache_bypass $http_upgrade;
      proxy_set_header   X-Real-IP $remote_addr;
      proxy_set_header   X-Forwarded-For $proxy_add_x_forwarded_for;
      proxy_set_header   X-Forwarded-Proto $scheme;
      proxy_set_header   X-Forwarded-Host   $host;

      proxy_pass         http://backend;
      proxy_redirect     off;
    }

    location / {
      try_files $uri $uri/ /index.html =404;
    }
  }
}
```

Рис 3.30 Конфіг файл nginx

3.6.2 AWS

Amazon Web Services (AWS) визначається як провідний постачальник хмарних сервісів та інфраструктури, який утримує лідерство на ринку хмарних обчислень від моменту свого виникнення. Під терміном AWS розуміється набір різноманітних програм, що пропонуються Amazon для хмарних обчислень. Визначена глобальним торговельним гігантом як «найбезпечніша, обширна та надійна хмарна платформа», AWS пропонує хмарні послуги, що надходять із численних центрів обробки даних по всьому світу.

3.6.3 Схема деплойменту

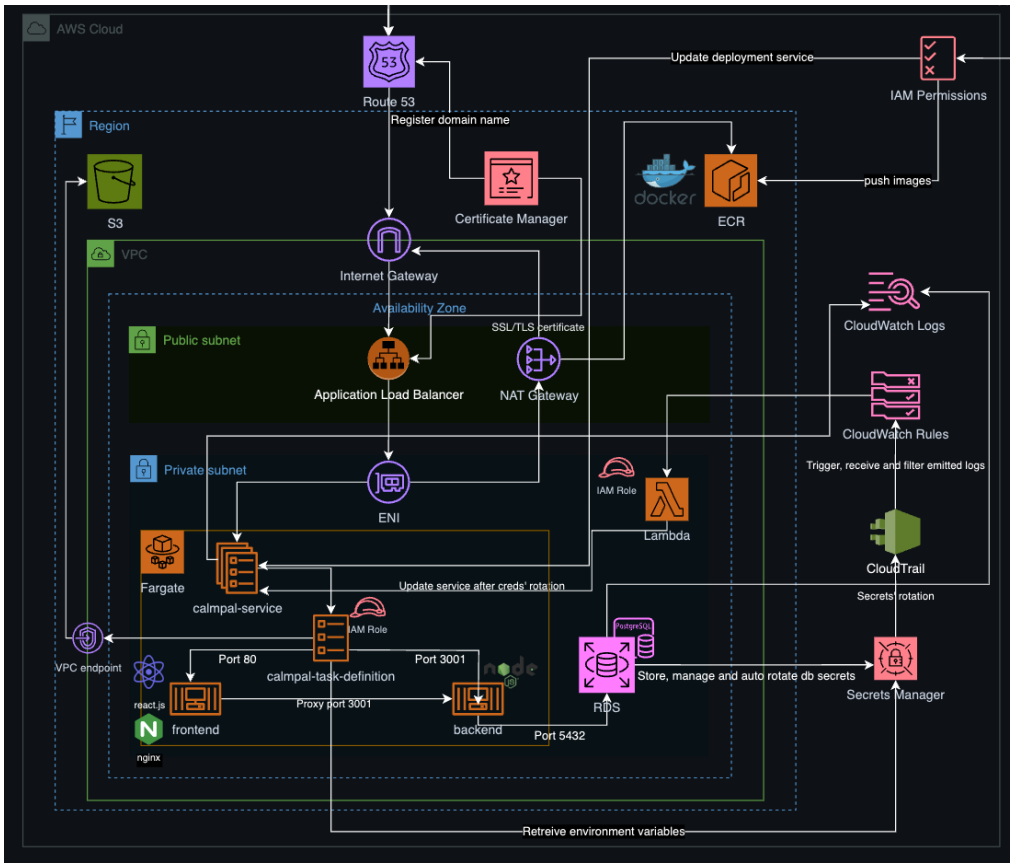


Рис 3.31 Схема деплойменту додатку

Варто почати з сервісу, який відповідає за збереження імеджів, які сформувались після білду воркспейсів в докері. Адже саме звідти машина-сервер буде стягувати собі відповідний імедж аби його запустити як контейнер.

AWS ECR

Amazon Elastic Container Registry (Amazon ECR) представляє собою безпечну, розширювану та надійну службу реєстрації зображень контейнерів, яку управляє AWS. Amazon ECR підтримує приватні репозитарії з дозволами, що базуються на ресурсах за допомогою AWS IAM. Це забезпечує можливість надавати доступ до ваших репозитаріїв контейнерів і зображень конкретним користувачам або екземплярам Amazon EC2.

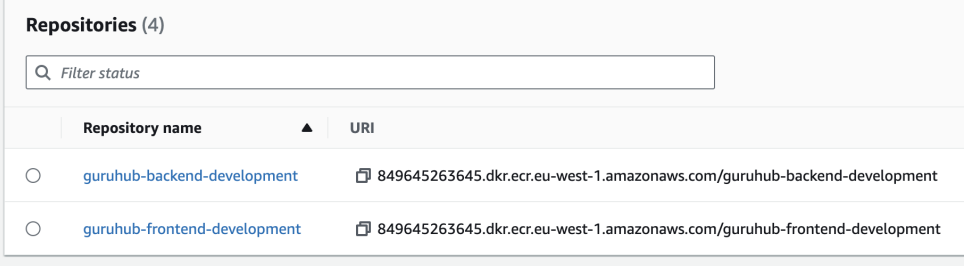


Рис 3.32 Задеплоєні 2 репозиторії для збереження імеджів воркспейсів

AWS ECS

AWS Fargate представляє собою технологію, яку можна використовувати разом з Amazon ECS для запуску контейнерів, уникаючи необхідності управління серверами чи кластерами екземплярів Amazon EC2. З Fargate не треба вирішувати питання створення, налаштування чи масштабування кластерів віртуальних машин для контейнерів. Це усуває необхідність обирати типи серверів, розглядати, коли масштабувати кластери, чи оптимізувати упаковку кластерів.

Щодо кластерів Amazon ECS, вони є логічним групуванням завдань чи служб, які дозволяють ізолювати ваші програми. Коли ви використовуєте Fargate для виконання завдань, Fargate також автоматично керує ресурсами кластера.

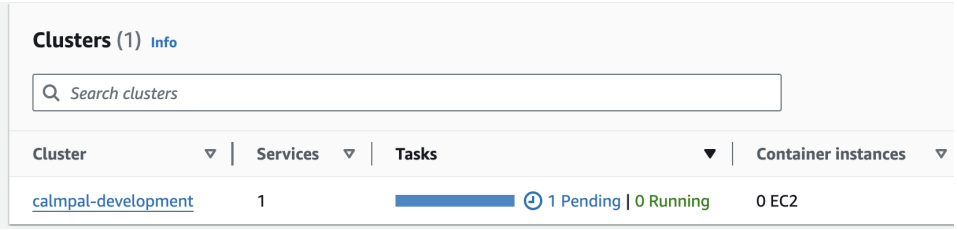


Рис 3.33 Приклад кластеру з деплойменту додатку

Визначення завдань представляє собою текстовий файл у форматі JSON, який описує один чи кілька контейнерів, які утворюють вашу програму як на рис 3.34.

Рис 3.34 Опис визначення завдання через JSON

Існує можливість використовувати його для визначення параметрів, таких як операційна система, використані контейнери, відкриті порти та обсяги даних для завдань. Весь стек додатків може складатися не з одного, а з декількох визначень завдань, об'єднаних в кожному власному компоненті.

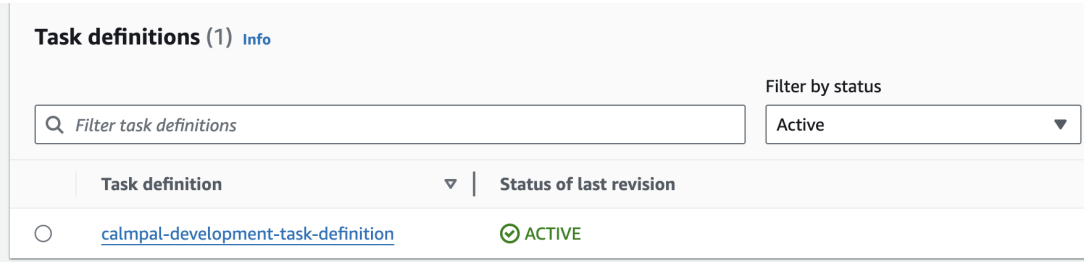


Рис 3.35 Визначення завдання в межах даного деплойменту

Завдання представляє собою екземпляр визначення завдання в кластері, яке можна запускати як окреме завдання або як частину служби. Служба Amazon ECS дозволяє виконувати та підтримувати бажану кількість завдань у кластері, замінюючи або додаючи екземпляри в разі необхідності.

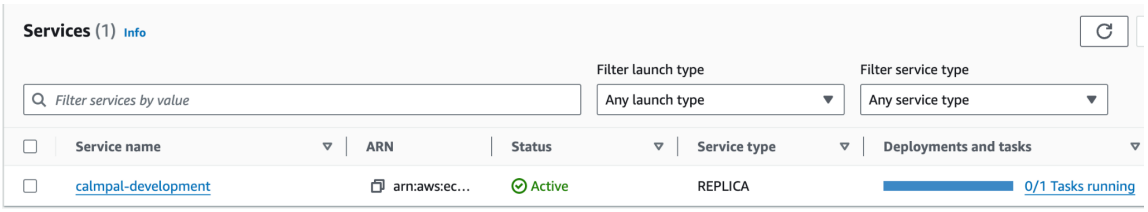


Рис 3.36 Визначення служби в межах даного деплойменту

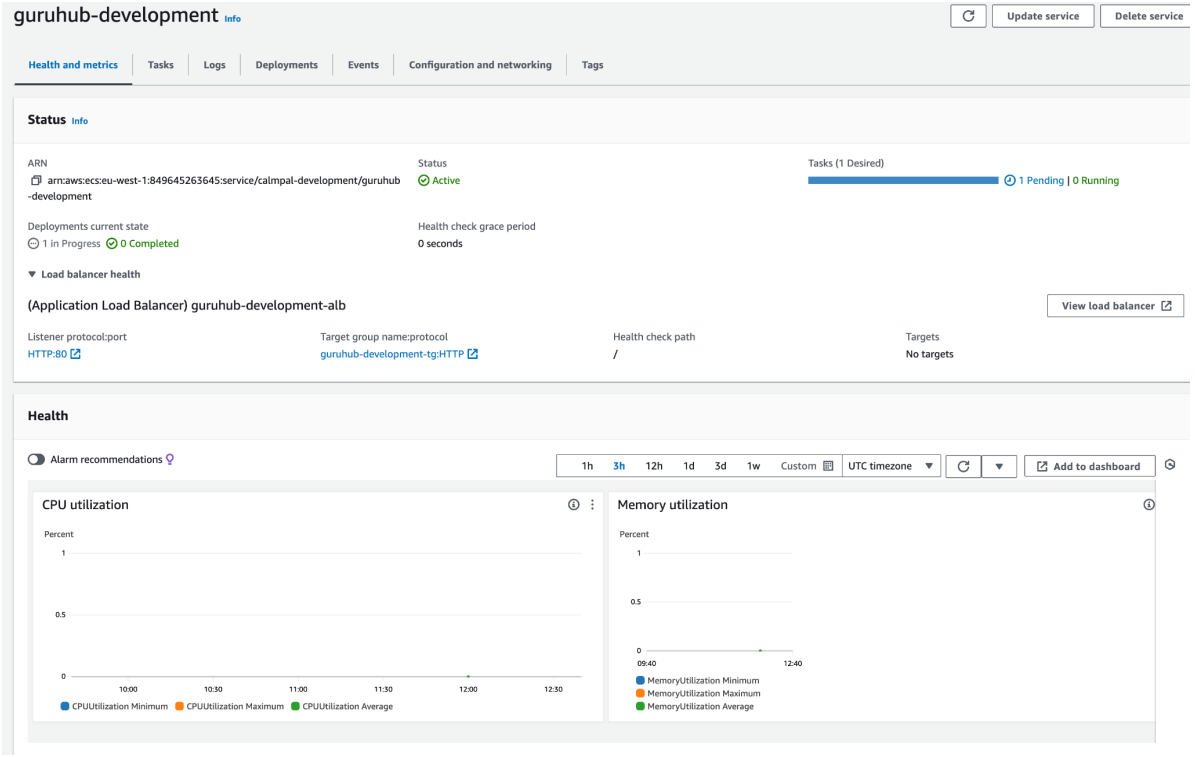


Рис 3.37 Дашборд служби в межах даного деплойменту

AWS IAM

AWS Identity and Access Management (IAM) є веб-службою, яка забезпечує безпечний контроль доступу до ресурсів AWS. За допомогою IAM можна централізовано управляти дозволами, які визначають, до яких ресурсів AWS мають доступ користувачі. Особливістю є використання IAM для контролю того, хто пройшов аутентифікацію (увійшов) і авторизований (має дозволи) на використання ресурсів.

AWS RDS

Служба реляційної бази даних Amazon (Amazon RDS) — це веб-служба, яка спрощує процеси налаштування, експлуатації та масштабування реляційних баз даних у хмарному середовищі AWS. Вона надає економічно

ефективні можливості для змінних за розміром стандартних реляційних баз даних і виконує загальні завдання адміністрування баз даних.

AWS VPC

За допомогою віртуальної приватної хмари Amazon (Amazon VPC) ви можете запускати ресурси AWS у логічно ізольованій віртуальній мережі, яку ви визначили. Ця віртуальна мережа дуже нагадує традиційну мережу, якою ви керуєте у власному центрі обробки даних, з перевагами використання масштабованої інфраструктури AWS. Використання кінцевої точки VPC дозволяє клієнтам безпечно та приватно з'єднуватися із підтримуваними службами AWS та іншими кінцевими точками VPC на основі AWS PrivateLink. Екземпляри Amazon VPC можуть взаємодіяти з ресурсами служби, не потребуючи публічних IP-адрес, і весь трафік між Amazon VPC та службою залишається в межах мережі Amazon, не виходячи за її межі.

AWS Secrets Manager

AWS Secrets Manager сприяє управлінню, отриманню та зміні облікових даних бази даних, облікових даних програм, маркерів OAuth, ключів API та інших секретів на протязі їхнього життєвого циклу.

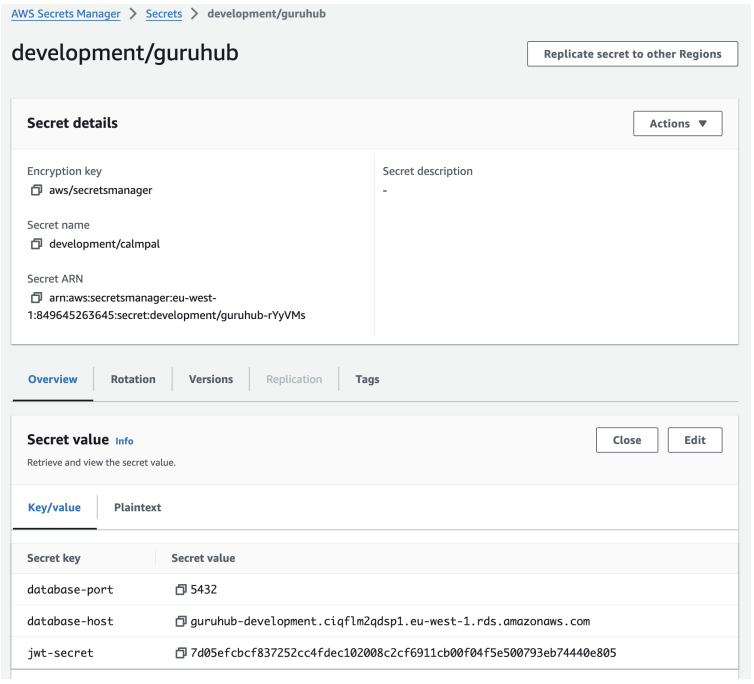


Рис 3.38 Збереження змінних оточення в межах деплойменту

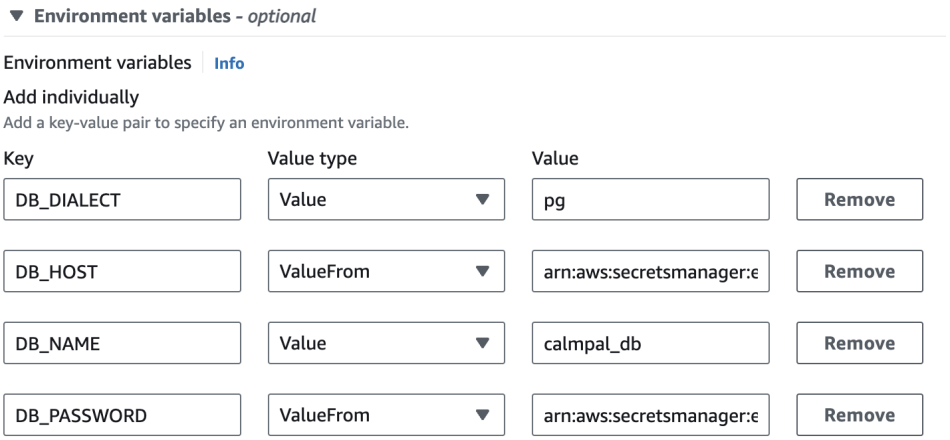


Рис 3.39 Приклад використання змінних оточення з Secrets Manager в ECS Task Definition

AWS Route 53

Amazon Route 53 - це надійна та масштабована веб-служба системи доменних імен (DNS), яку можна використовувати для виконання трьох основних функцій в будь-якій комбінації: реєстрації домену, маршрутизації DNS і перевірки доступності.

AWS Certificate Manager

Менеджер сертифікатів AWS (ACM) спрощує процес створення, зберігання та оновлення загальнодоступних і приватних сертифікатів і ключів SSL/TLS X.509, що забезпечують безпеку ваших веб-сайтів та програм AWS. ACM дозволяє видавати сертифікати для інтегрованих служб AWS безпосередньо або імпортувати сторонні сертифікати для управління ними. Сертифікати ACM можуть захищати окремі доменні імена, кілька конкретних доменних імен, домени з символами підстановки або їх комбінації.

ВИСНОВКИ ДО РОЗДІЛУ 3

В ході розробки використовувалися сучасні технології, що дозволили створити ефективну та функціональну систему, яка має на меті надати високу продуктивність та безпеку системи. Розроблена даталогічна модель бази даних відображає точні та зручні зв'язки між сутностями, забезпечуючи ефективне зберігання та витяг з інформації. Бекенд системи був розроблений з використанням сучасних фреймворків на базі Node.js, що дозволило створити стабільний та високопродуктивний сервер. Обробка запитів, бізнес-логіка та взаємодія з базою даних були ефективно імплементовані,

Представлено конкретні технічні рішення, які були використані під час створення системи. Подана структура файлів системи, а також докладно описаний процес кожного етапу розробки системи, супроводжуючись відповідними діаграмами та рисунками.

Було виділено окрему увагу процес деплойменту, що був оптимізований для ефективного впровадження системи в середовище виробництва, забезпечуючи швидкий та безпечний реліз, зменшуючи час простою системи.

Загалом, використання передових технологій у поєднанні з детальною даталогічною моделлю бази даних, добре розробленим бекендом та зручним фронтендом створило інтегровану систему, яка відповідає високим стандартам якості та відповідає потребам користувачів.

РОЗДІЛ 4. ІНТЕРФЕЙС РОЗРОБЛЕНОЇ СИСТЕМИ

В даному розділі представлений розгляд розробленої системи, а також зроблений загальний огляд основного функціоналу.

Почну з вхідної точки в веб застосунок. Всім користувачам, незалежно від того, чи вони авторизовані в системі, буде доступно сторінка дашборди з курсами як на рис 4.1:

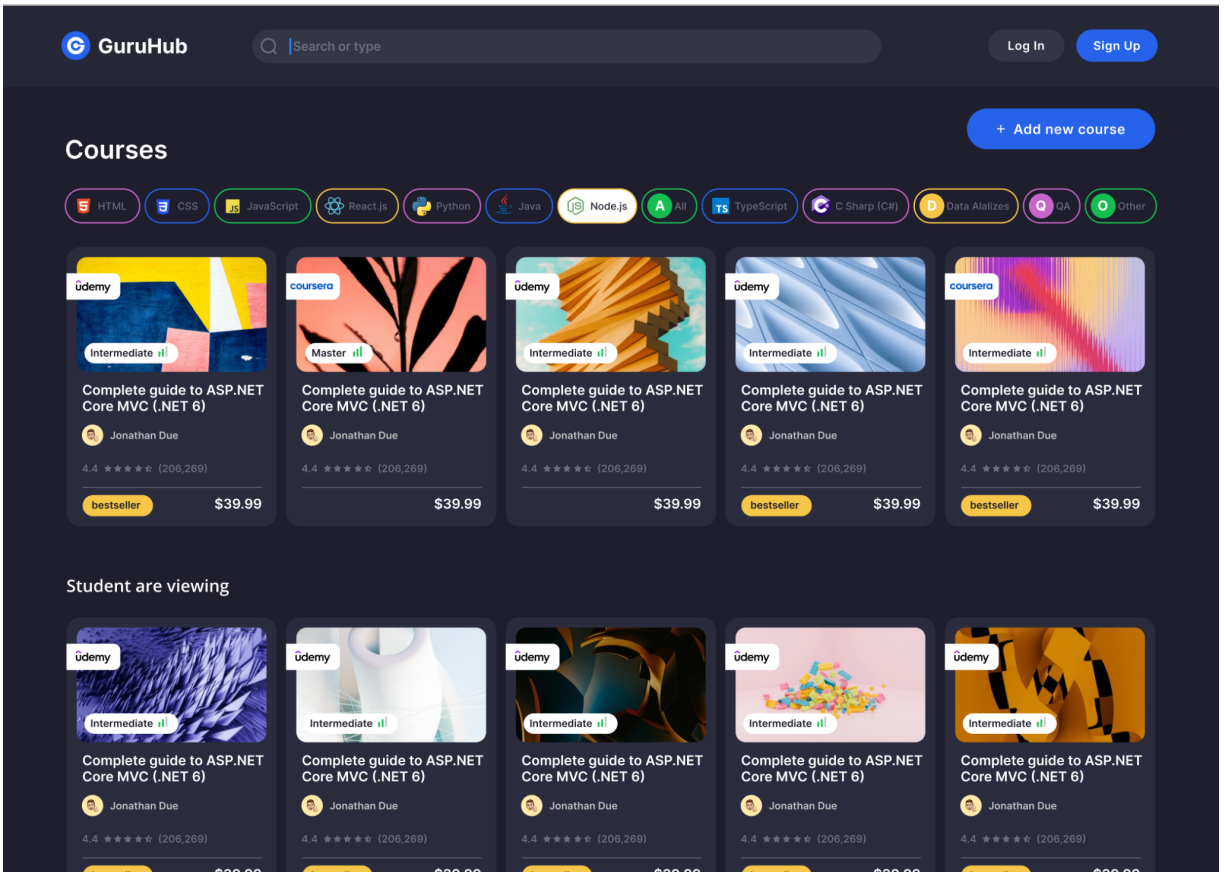


Рис 4.1 Сторінка дашборди курсів.

А також буде пропонуватись авторизуватись через кнопки у верхньому правому куті. У юзера є можливість відсортувати курси по технологіям, а також вибрати курс аби переглянути його опис як на рисунку 4.2.

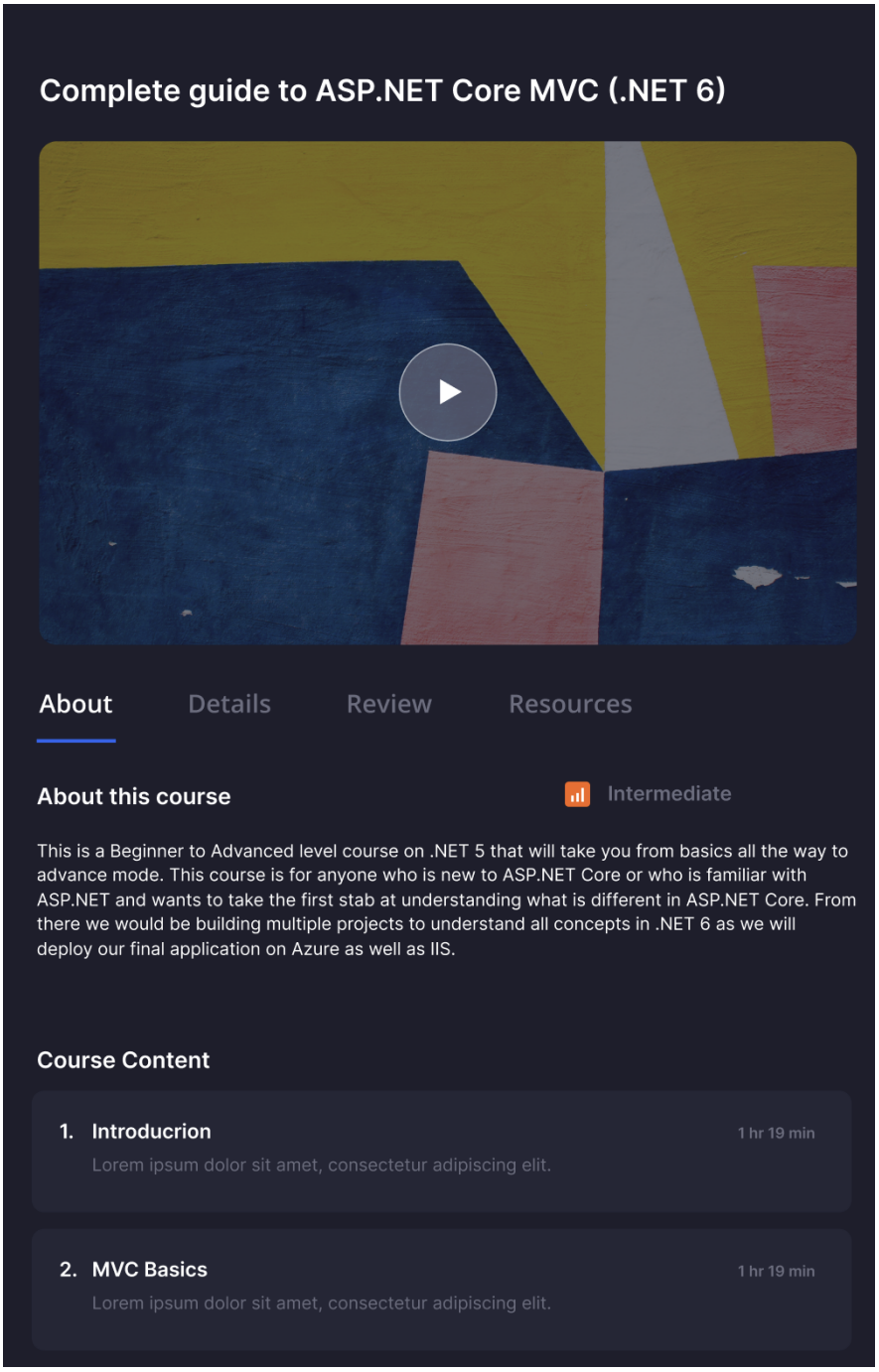
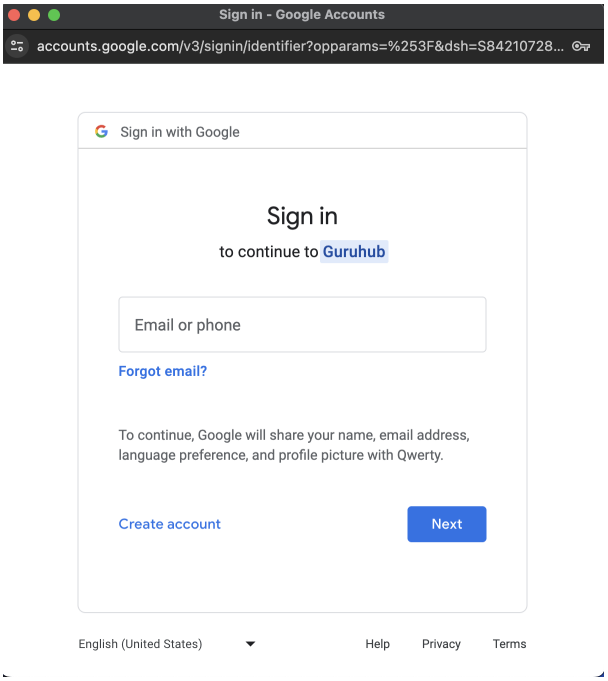


Рис 4.2 Опис вибраного курсу

Якщо користувач хоче додати свій курс, йому потрібно буде авторизуватись в системі, або увійти або створити нового юзера через відповідну сторінку авторизації відповідно до рисунка 4.3

Рис 4.3 Сторінка авторизації

Авторизуватись можна як ввівши пошту та пароль та за допомогою OAuth2 Google чи Github.



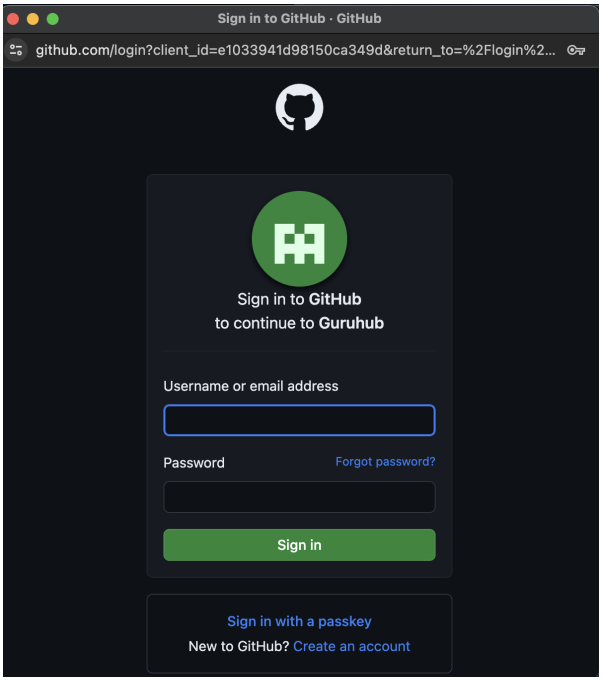


Рис 4.4 OAuth2 Google

Рис 4.5 OAuth2 Github

І далі відповідно додати новий курс або ж імпортувати існуючий з Coursera чи Udemy як на рис 4.6

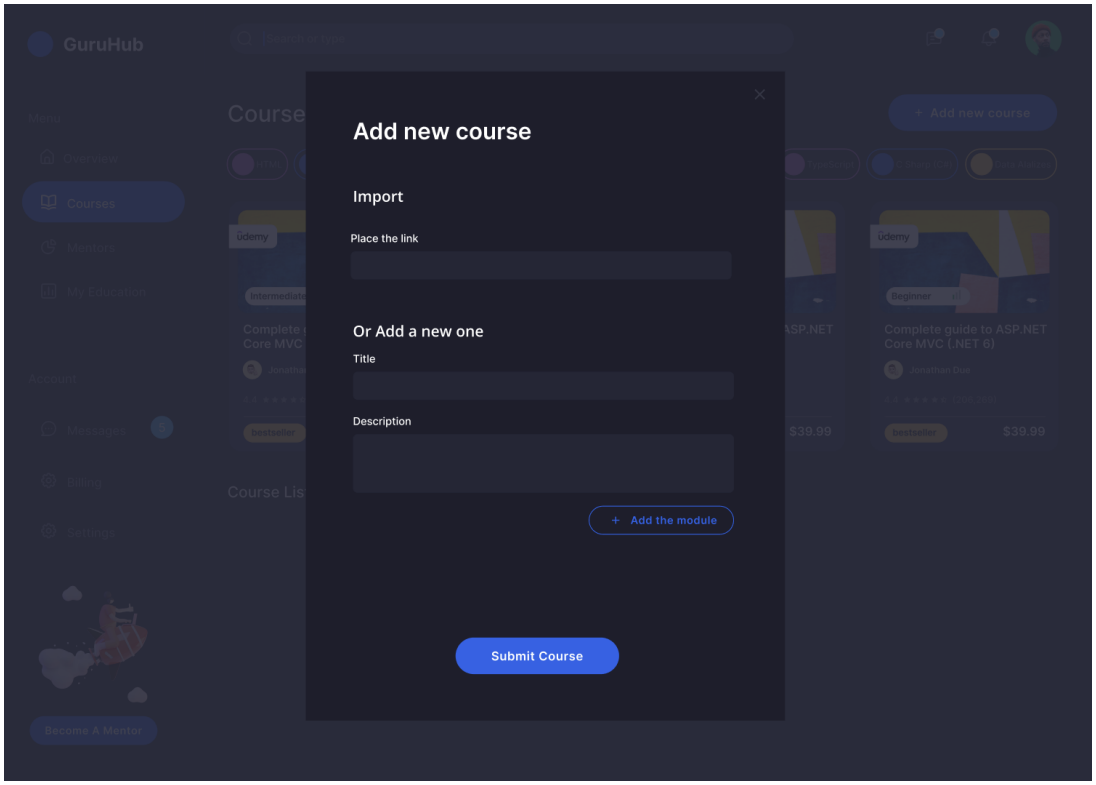


Рис 4.6 Модальне вікно для додавання нових курсів

Після авторизації студент зможе бачити додатково сайдбар з можливістю переходу на інші скріни як на рис 4.7

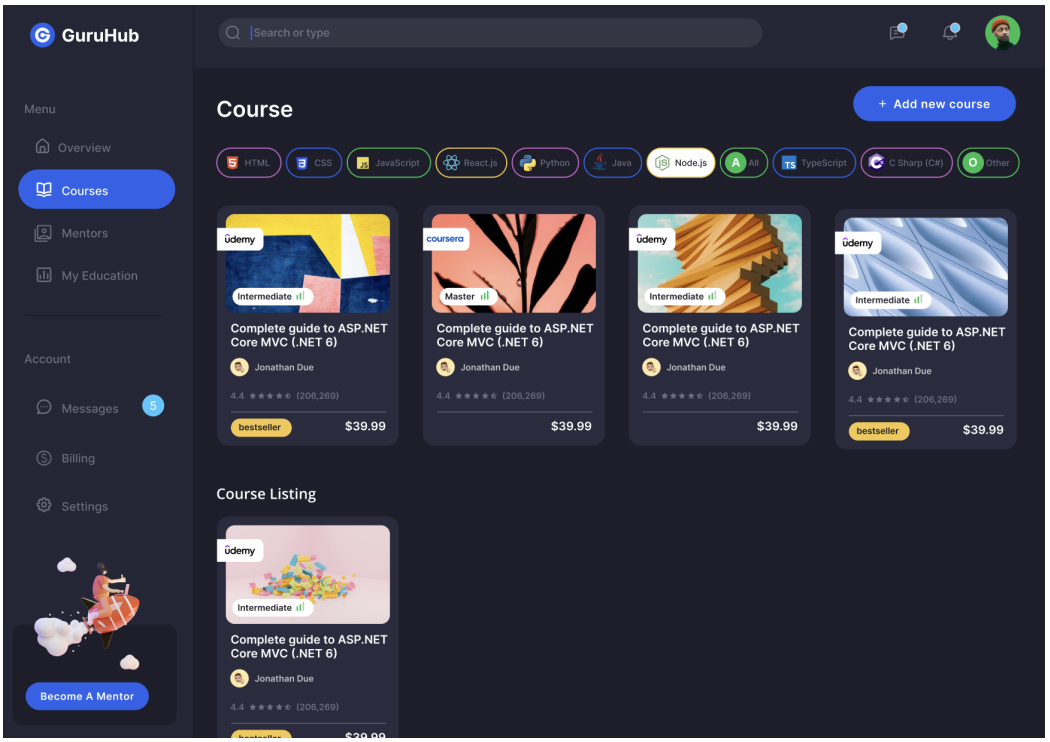


Рис 4.7 Зображення сайдбара на сторінці дашборди курсів

Перш ніж студент захоче знайти ментора, йому потрібно поповнити свій рахунок, з якого буде списувати плата за проходження курсу на сторінці Billing як на рис 4.8

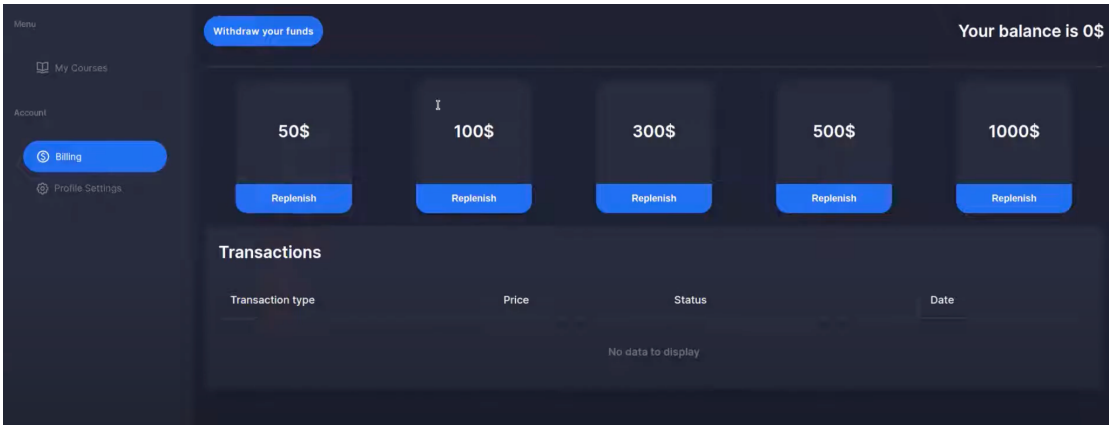


Рис 4.8 Сторінка поповнення рахунку

Відповідно ціни на курс формуються за принципом ціни на категорію як зображено на рис 4.9:

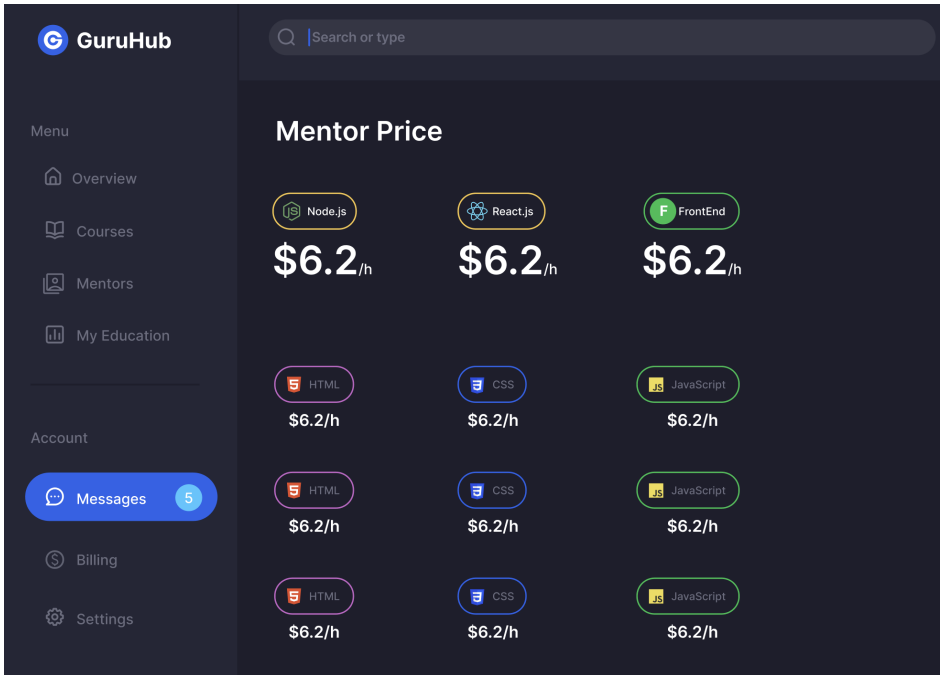


Рис 4.9 Цінова політика за категорію

Далі, зайшовши на опис курсу, можна або стати ментором або вибрати ментора як на рисунку 4.10

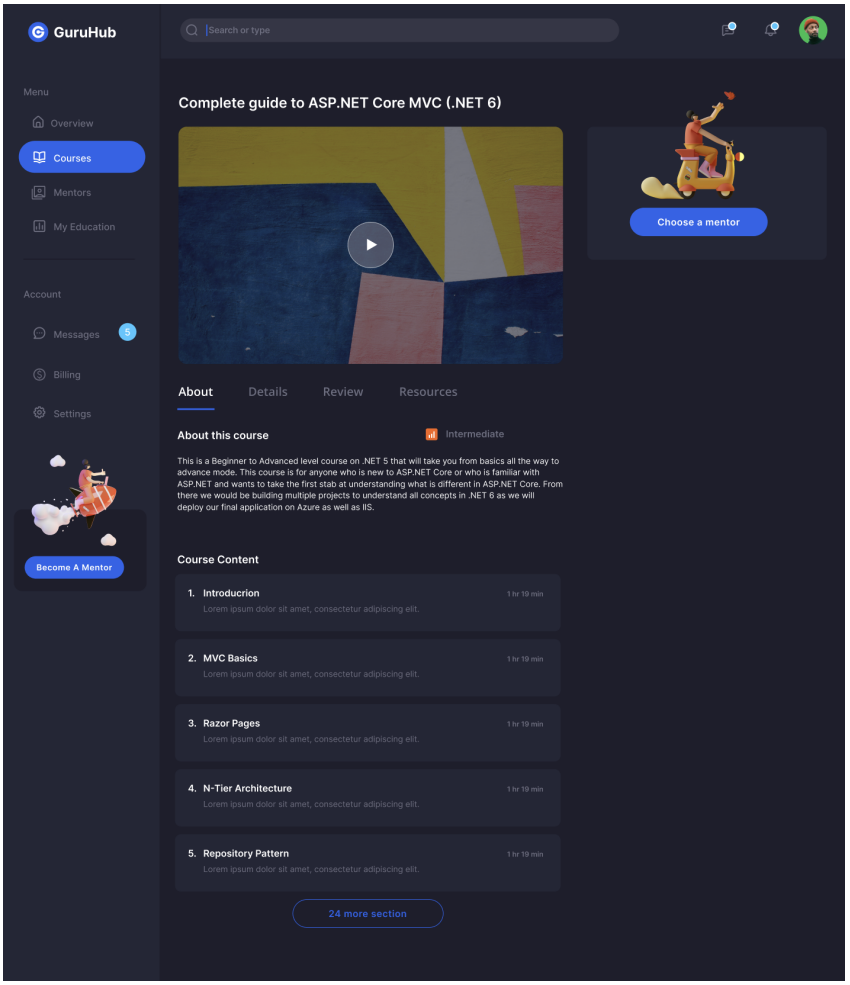


Рис 4.10 Сторінка опис курсу з можливостями стати ментором або вибрати

При натисканні на Вибрати ментора, з’являється модальне вікно з списком доступних менторів на цей курс як на рисунку 4.11

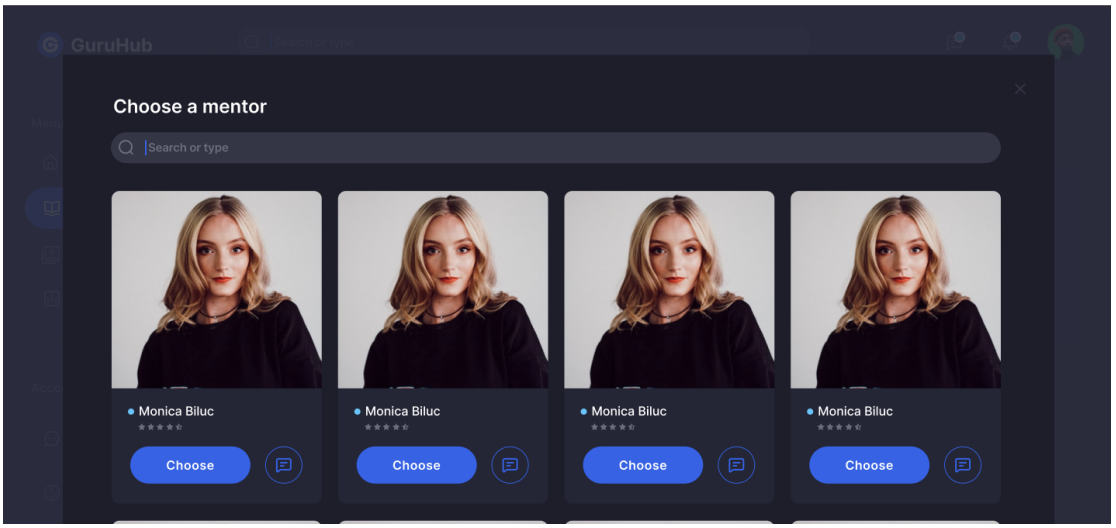


Рис 4.11 Модальне вікно вибору ментора

Тож, студент сам обирає собі ментора і після підтвердження вибору, юзер може бачити на сторінці опису курсу свого ментора як на рис 4.12

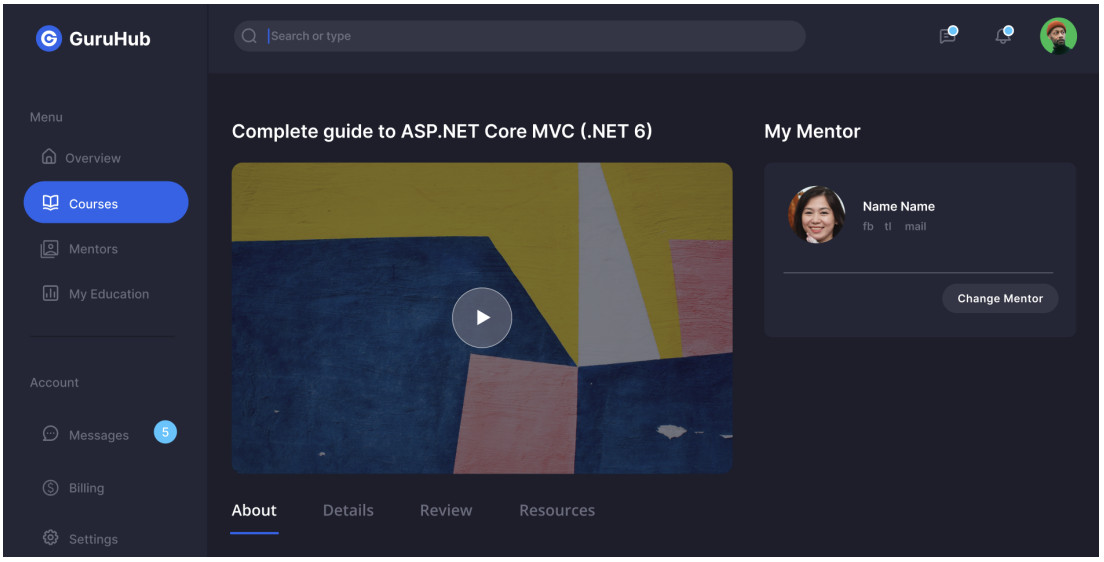


Рис 4.12 Відображення вибраного ментора на сторінці опису курсу

Авжеж, зберігається можливість змінити ментора в будь-який час.

Далі, студент може приступити до навчання по модулям, кожен модуль містить завдання до виконання студентом, яке потрібно засабмітити на рев'ю як на рис 4.13

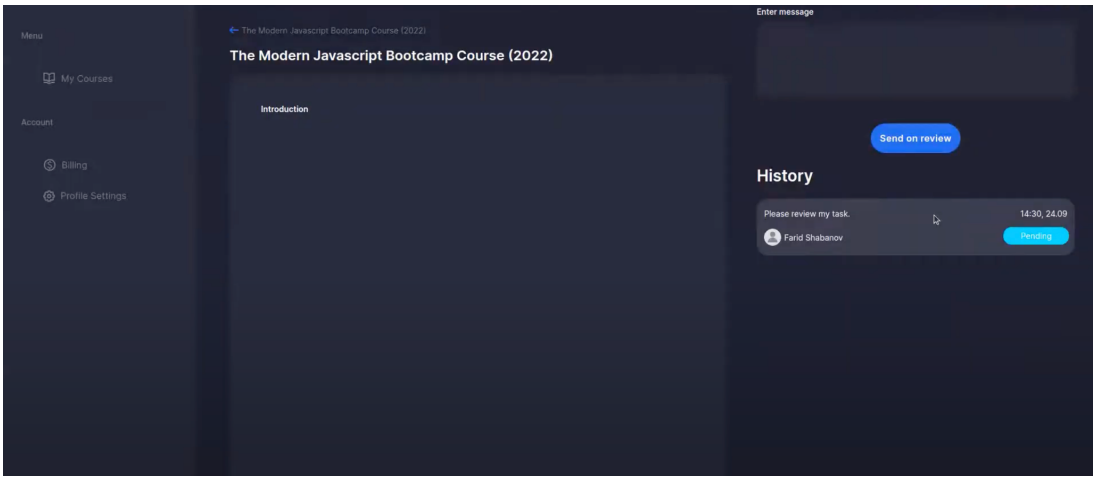


Рис 4.13 Сторінка здачі завдання по модулю

Зі сторони ментора це виглядатиме наступним чином: по-перше він бачитиме перелік своїх студентів як на рис 4.14 та заявку на здачу завдання по модулю як на рис 4.15

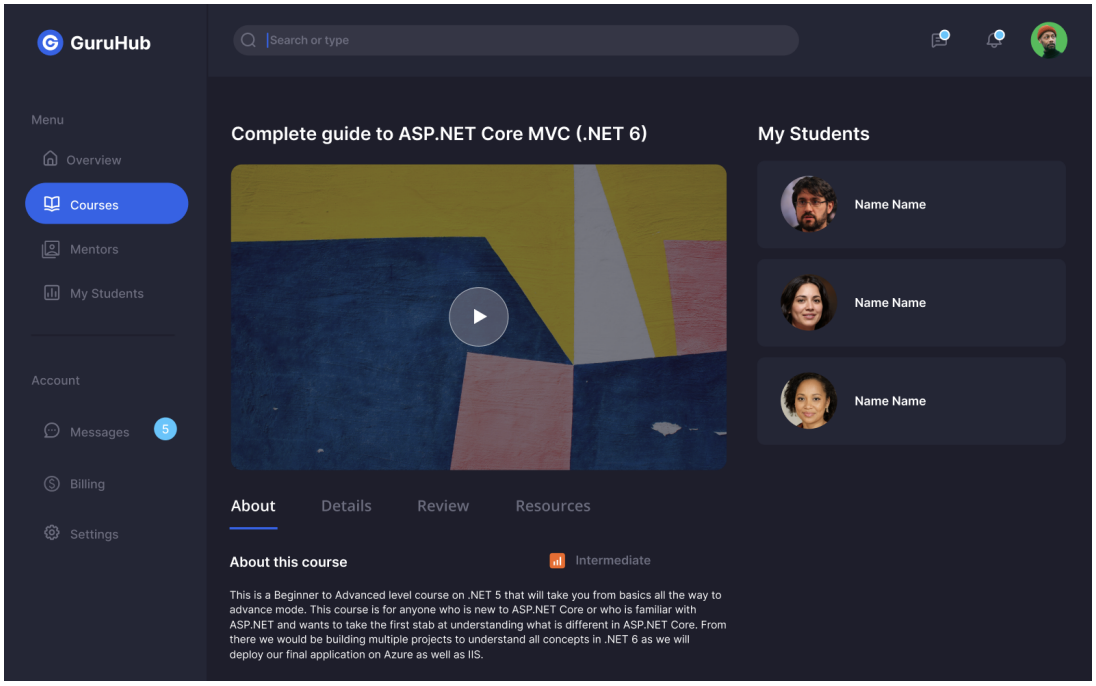


Рис 4.14 Сторінка з списком студентів для ментора

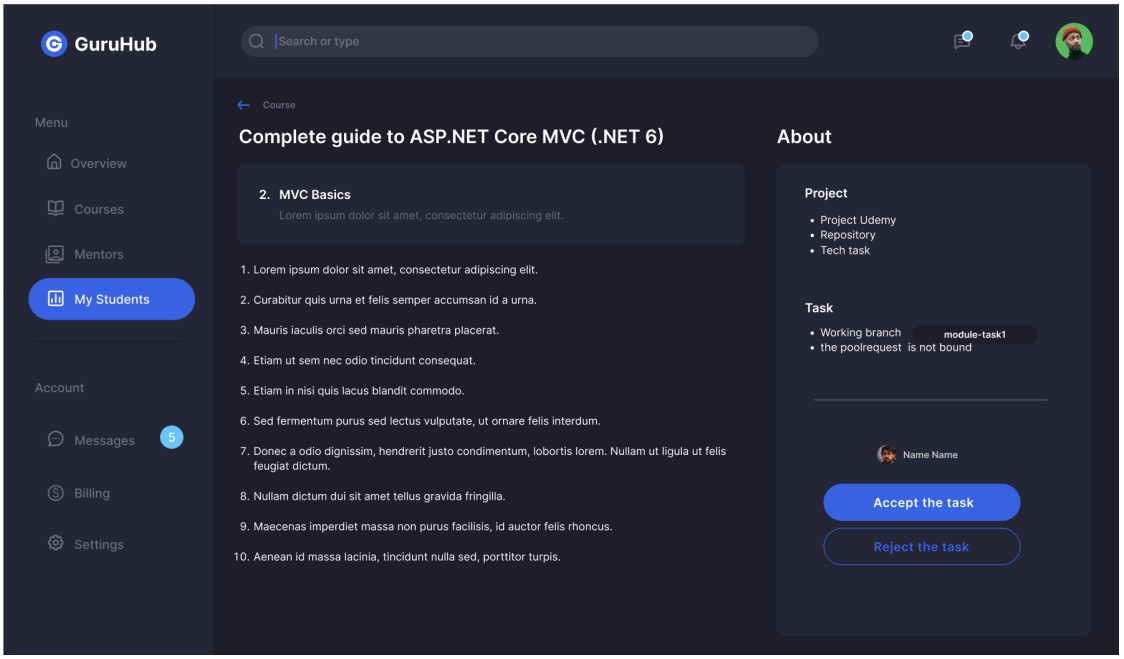


Рис 4.15 Сторінка із заявкою на здачу завдання студентом

Студент має можливість відслідковувати свій погрес в навчанні як на рис 4.16

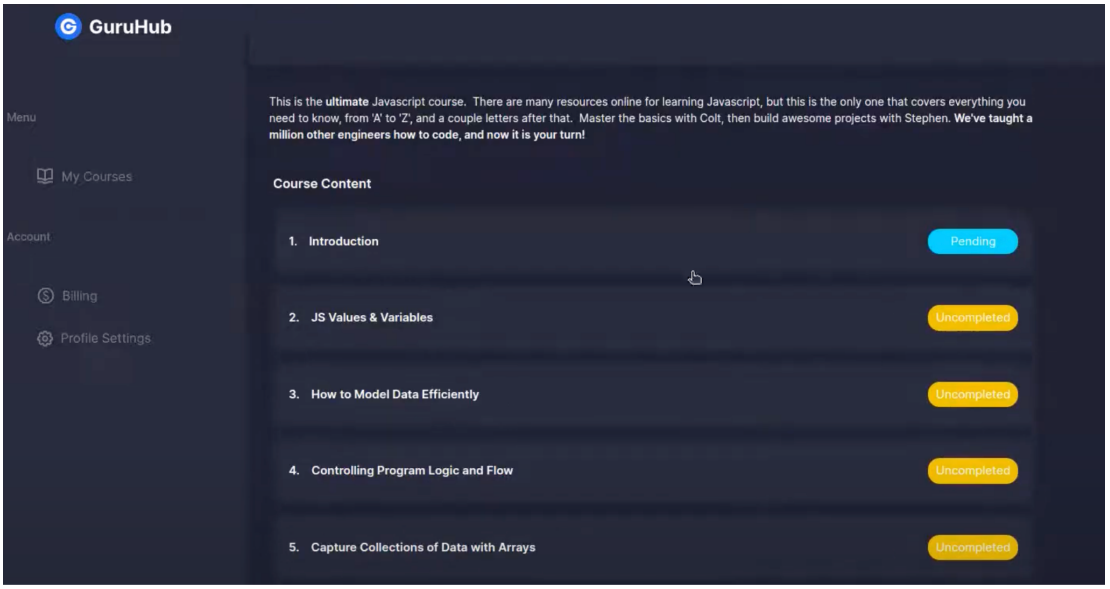


Рис 4.16 Прогрес студента по проходженню курсу

Для кожного юзера є можливість заповнити свій профіль актуальними даними, фото для аватарки як на рис 4.17

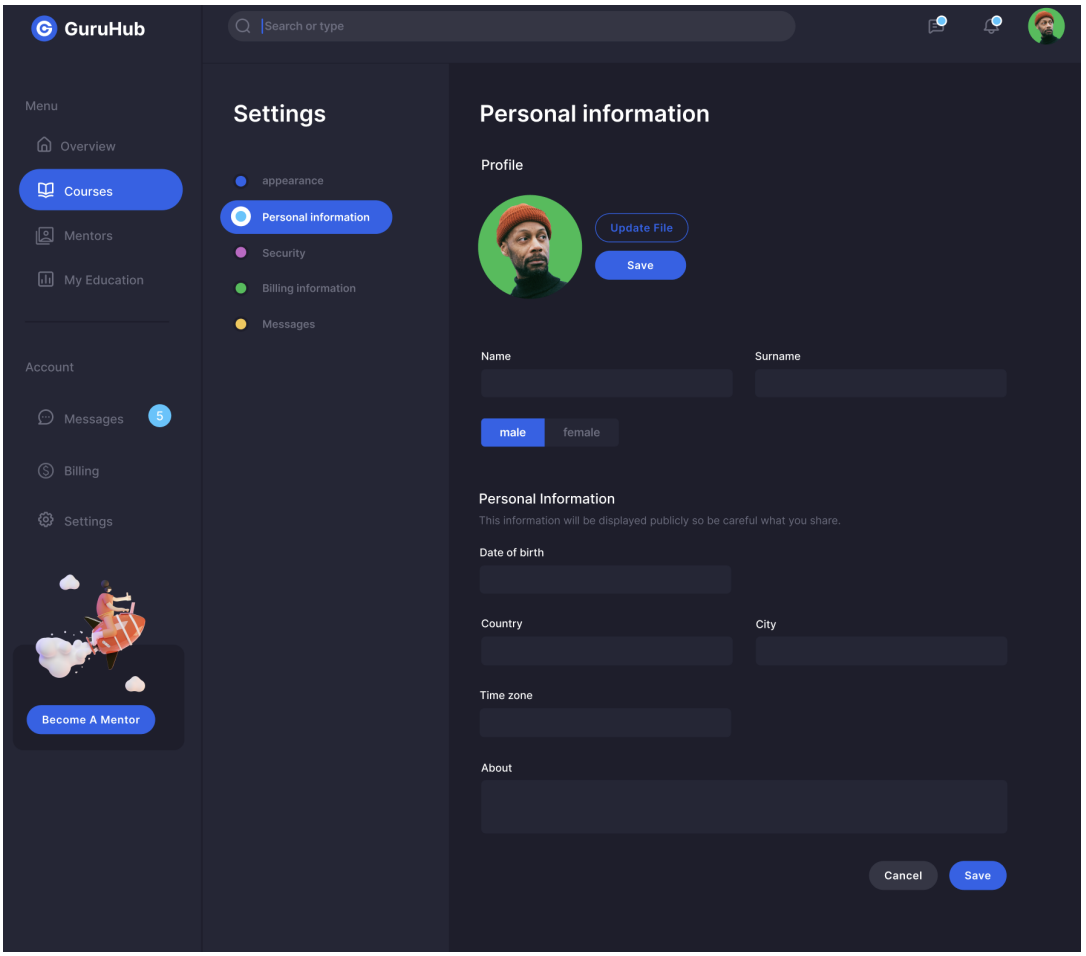


Рис 4.17 Заповнення профілю користувача

Будь-який юзер може як бути ментором, так і бути студентом на різних курсах, і він це може відстежувати з сторінки Мої курси як на рис 4.18

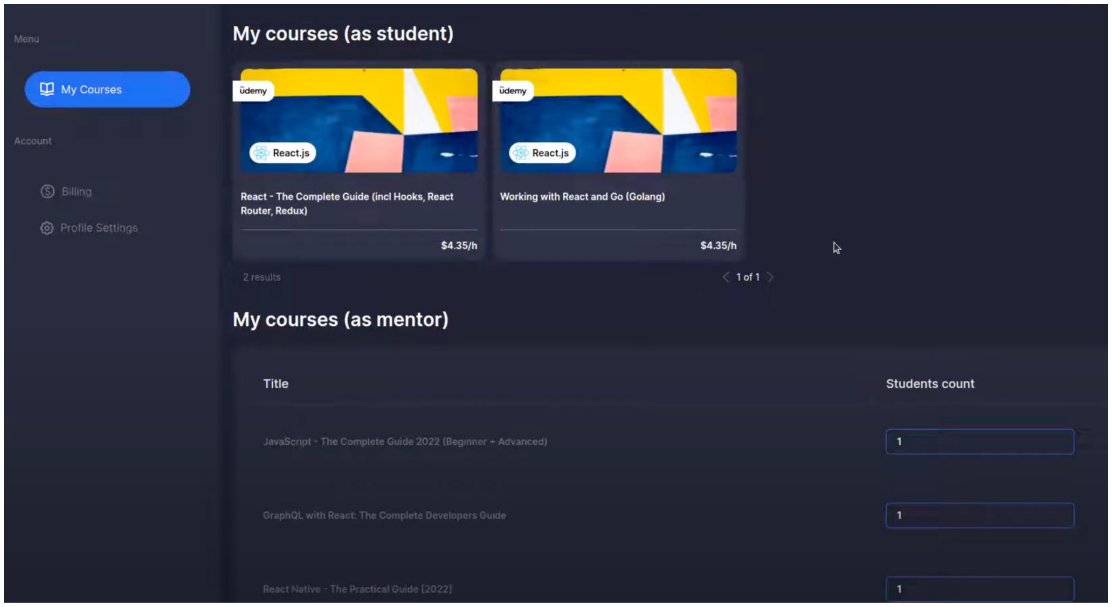


Рис 4.18 Сторінка Мої курси

Як ментор, користувач може налаштувати обмеження скільки студентів він може прийняти одночасно на курс.

Якщо юзер клікає на кнопку Стати ментором, він автоматично записується на інтерв’ю по даній категорії курсу. Далі адміністратор може назначити відповідного користувача, який може проводити інтерв’ю і записати результат співбесіди в описі інтерв'ю як на рис 4.19

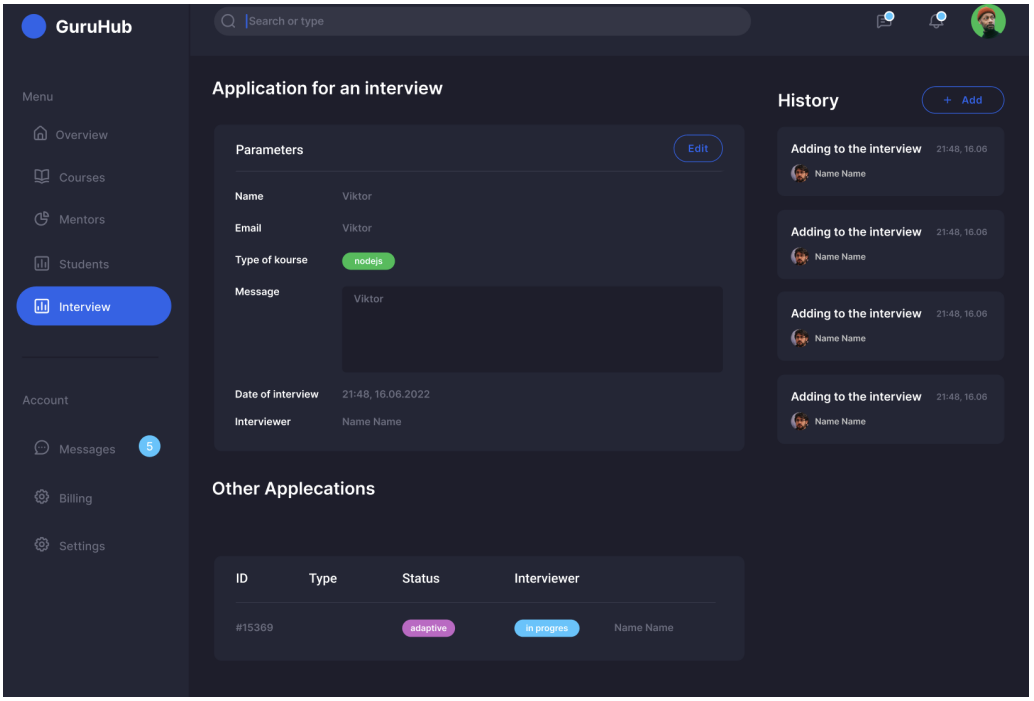


Рис 4.19 Сторінка опису інтерв’ю

Якщо інтерв’ю пройдене успішно, майбутній ментор отримає повідомлення про це і автоматично стане ментором для вибраного раніше курсу.

Як тільки студент записується на курс, з його рахунку списується сума, яка поки не завершиться курс не потрапить ментору. Ментор може лише бачити кількість транзакцій в статусі очікування на Billing сторінці як на рис 4.20

Рис 4.20 Транзакції в стані очікування на завершення студентом курсу

Транзакції, з підтвердженим переказом у зв'язку з проходженням студентом курсу мають такий вигляд як на рис 4.21

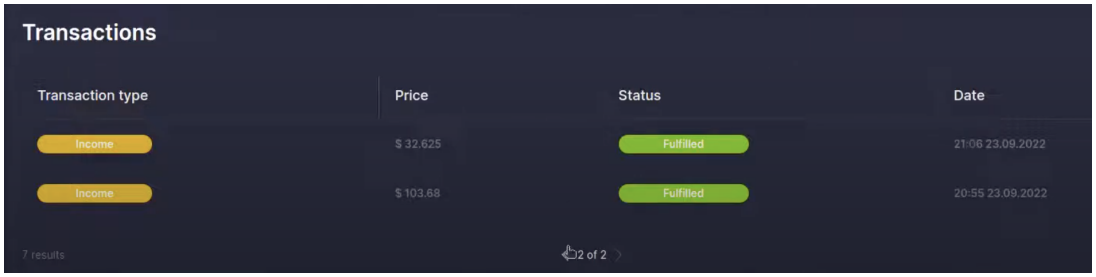
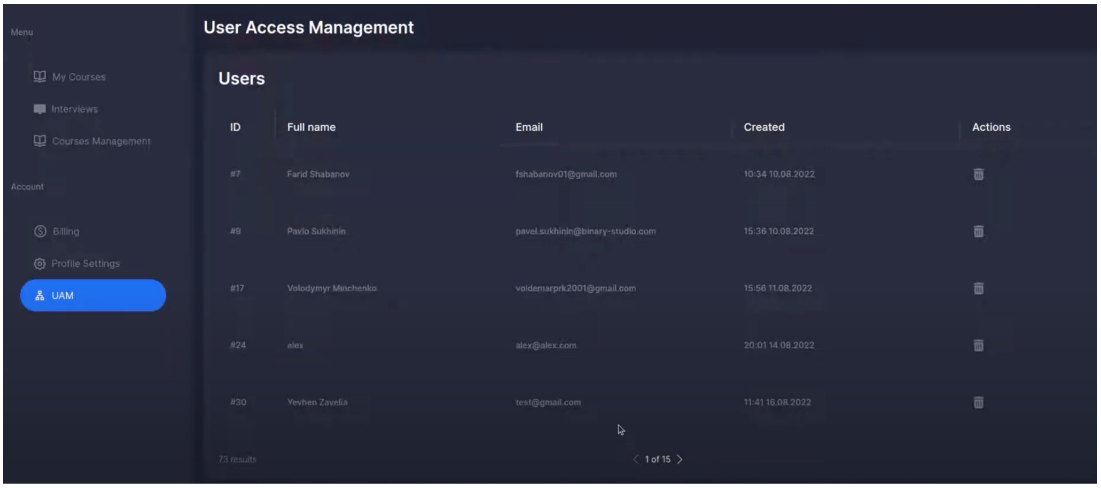


Рис 4.21 Підтверджені транзакції оплати за курс

Окрім менторів і студентів існує роль Адміністраторів, які можуть менеджити юзерів, додавати їх до груп, які мають певні обмежені дозволи. Це все здійснюється через User Access Managment сторінку як на рис 4.22



Продовження рис 4.22

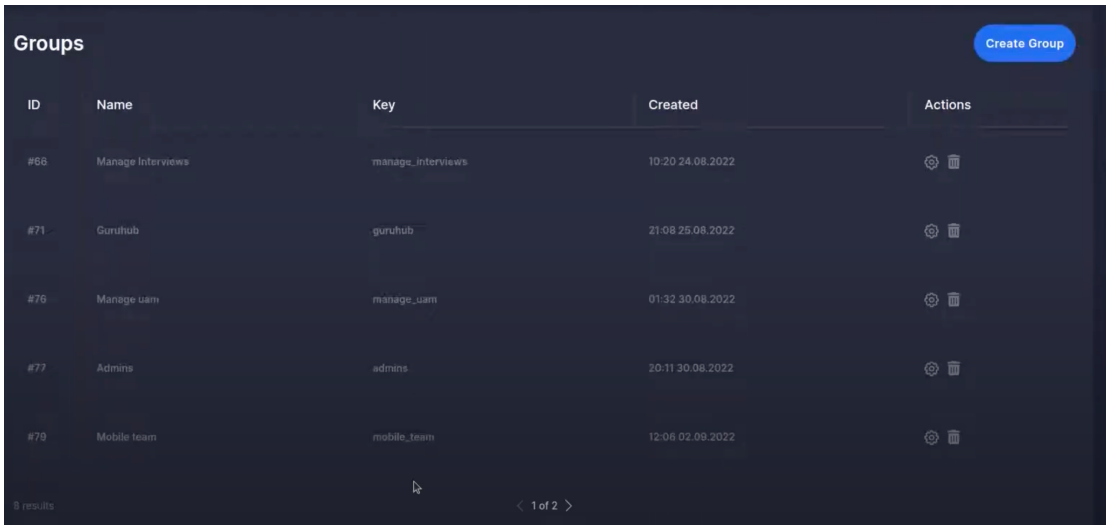
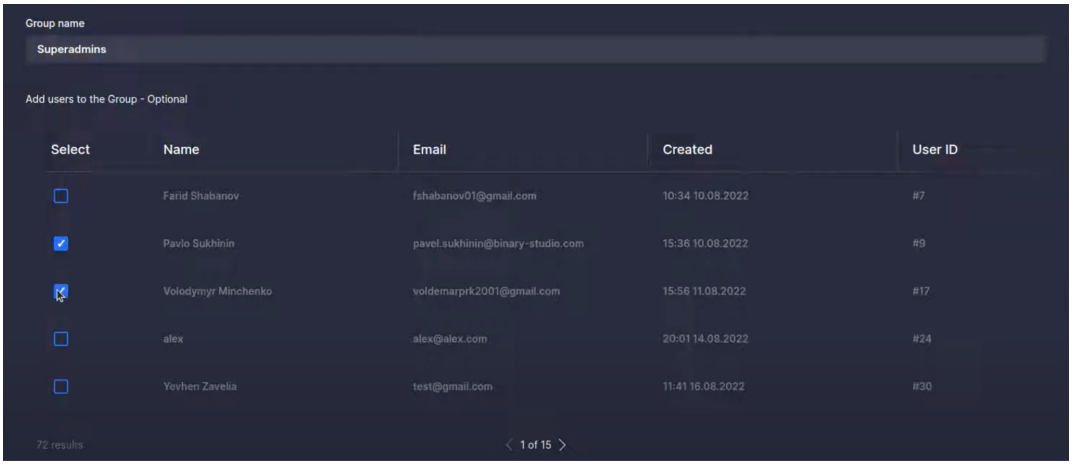


Рис 4.22 Сторінка User Access Management



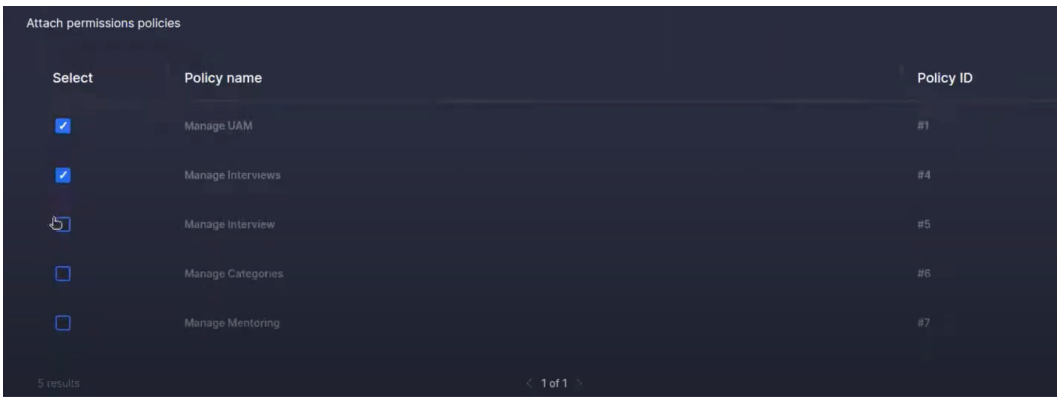


Рис 4.23 Функціонал створення нової групи з вибором юзерів та прикріпленням дозволів.

І останній функціонал це функціонал чату з підтримкою режиму реального часу як на рис 4.24

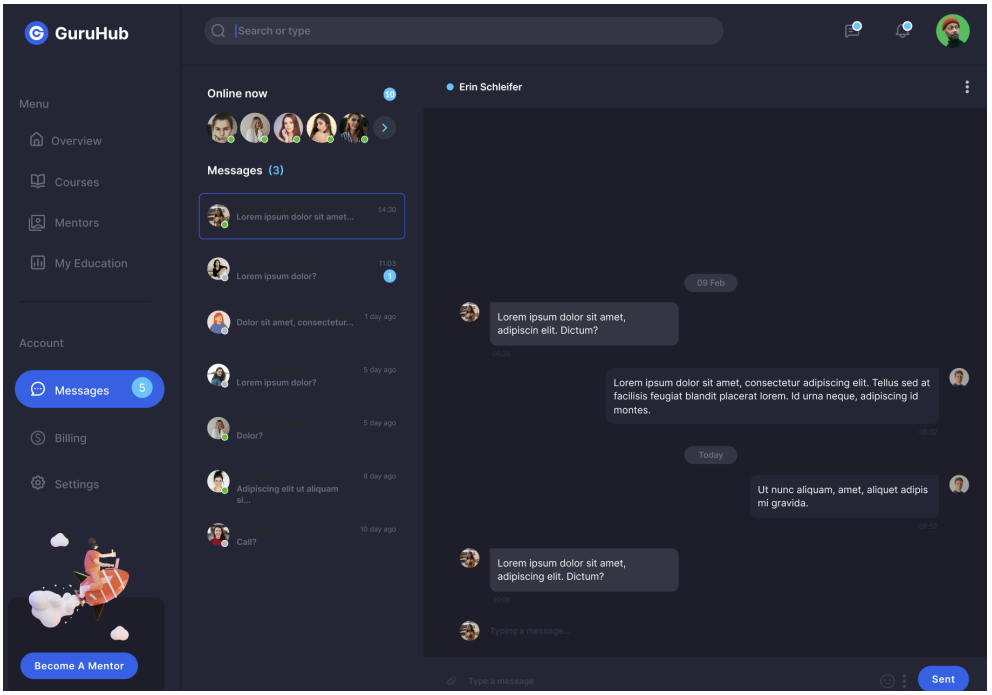


Рис 4.24 Сторінка чату

Даний функціонал чату дозволяє менторам і студентам комунікувати в межах спільного курсу, адже завжди можуть виникати уточнюючі питання, які десь треба задати. Окрім цього, чат має підтримку режиму реального часу, тож адресат завжди отримає повідомлення відразу без необхідності перезавантаження сторінки браузера.

ВИСНОВКИ ДО РОЗДІЛУ 4

Оглянуто розроблену систему та встановлено що вона працює згідно з встановленими для неї вимогами. Було розглянуто основний функціонал виконаної платформи покроково аби показати основний флоу як кожен модуль взаємодіє з іншими. Наприклад без авторизації, користувач зможе тільки переглянути всі наявні курси в платформи, але для того, щоб або стати ментором або ж менті йому слід авторизуватись в системі.

Зроблено огляд існуючого функціоналу, який складається з модулів:

- Авторизація через креди акаунту чи OAuth Google і Github
- Перегляд курсів
- Можливість записатись на курс під ментором чи менті
- Проходження інтерв'ю аби стати ментором
- Можливість оплати за навчання
- Функціонал по перегляду модулів курсів та форма здачі домашнього завдання
- Рев'ю ментором імплементації студентом
- Перегляд Моїх курсів студентом чи ментором
- Можливість комунікації за допомогою чату
- Перегляд прогресу навчання студента по курсу
- Можливість відслідковування проведення оплати за курс ментору через транзакції
- Функціонал організації доступів та ролей для користувачів
- Кастомізація профілю користувача

З огляду на поставлену задачу, даний функціонал відповідає вимогам поставленого завдання.

РОЗДІЛ 5 РОЗРОБКА СТАРТАП-ПРОЄКТУ

5.1 Опис ідеї проєкту

У світі стрімкого розвитку технологій та постійних змін в освітньому середовищі, виникає необхідність створення інноваційних рішень для полегшення процесу навчання та підвищення доступності освітніх ресурсів. Цей проєкт – це веб-платформа для вивчення онлайн курсів, яка спрямована на забезпечення ефективного та гнучкого навчання для різних категорій користувачів.

Основні характеристики проєкту:

- Гнучкість та мобільність: Платформа надає можливість навчання за власним розкладом, дозволяючи користувачам вибирати та пристосовувати графік навчання під свої потреби. Доступність з будь-якого пристрою (комп'ютера, планшета, мобільного телефону) робить навчання максимально мобільним та зручним.
- Широкий вибір курсів: Платформа пропонує різноманітні онлайн курси з різних сфер, включаючи технічні науки, мистецтво, бізнес, мови та інші. Користувачі можуть вибрати курси відповідно до своїх інтересів та професійних цілей.
- Взаємодія з менторами: Платформа забезпечує можливість взаємодії з викладачами через форуми, чати та відеоконференції. Це створює сприятливу атмосферу для обговорення матеріалів та отримання відповідей на питання.
- Система відстеження прогресу: Кожен користувач має особистий кабінет із системою відстеження прогресу. Вона включає в себе відомості про пройдені курси, здобуті навички та рекомендації для подальшого розвитку.
- Оцінювання та сертифікація: Після успішного завершення курсів користувачі отримують сертифікати, що підтверджують їхні знання та навички. Це може бути важливим елементом для резюме та професійного розвитку.

Тож мета - створити інноваційну освітню платформу, що відповідає сучасним вимогам та надає користувачам ефективні інструменти для саморозвитку та досягнення успіху в різних галузях.

5.2 Технологічний аудит проєкту

Таблиця 4.1

Технологічна здійсненність ідеї проєкту

№	Ідея проєкту	Технології її реалізації	Наявність технології	Доступність технологій
1	Система для управління проєктами	Мова програмування Javascript	Наявна	Доступна безкоштовно
2		Фреймворк React	Наявна	Доступна безкоштовно
3		Фреймворк Fastify	Наявна	Доступна безкоштовно
4		ORM Knex&Objection	Наявна	Доступна безкоштовно
5		База даних Postgres	Наявна	Доступна безкоштовно
6		Udemy API	Наявна	Доступна безкоштовно
7		Amazon Cloud	Наявна	Доступна безкоштовно і також платно в залежності від надаваного розміру хмарного середовища

Для створення веб-додатку використано наступні технології: Javascript, Typescript, та Postgres. Також у проекті використано такі бібліотеки як React, Fastify, Knex&Objection, jsonwebtoken, body-parser, CORS, Multer та NPM, котрі роблять розробку програмного забезпечення швидшою за рахунок надавання готового функціоналу, що використовуються для вирішення типових задач у розробці.

5.3 Аналіз ринкових можливостей запуску стартап-проєкту

Визначення ринкових можливостей, які можна використати під час ринкового впровадження проєкту, та ринкових загроз, які можуть завадити реалізації проєкту, дозволяє визначити та окреслити напрями розвитку проєкту із урахуванням поточного стану ринку, потреб потенційних клієнтів та пропозицій проєктів-конкурентів.

Таблиця 4.2

Попередня характеристика потенційного ринку стартап-проєкту

№	Показники стану ринку	Характеристика
1	Кількість головних гравців, од	> 3
2	Динаміка ринку (якісна оцінка)	Продукти цього типу наявні вже давно, тож ринок досить стабільний, але завжди відкритий для нового
3	Наявність обмежень для входу (вказати характер обмежень)	Велика кількість конкурентних підходів та наявність гравців з уже налаштованою клієнтською базою
4	Специфічні вимоги до стандартизації та сертифікації	Відсутні

Таблиця 4.3

Характеристика потенційних клієнтів стартап-проєкту

№	Потреба, що формує ринок	Цільова аудиторія	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару
1	Якість статистики роботи над проєктом в цілому та індивідуального розробника	Команди розробників та їх менеджмент	Для користувачів, котрим важливо знати стан проєкту в цілому буде цікавіше дивитись загальну статистику виконаних задач, а проєктним менеджерам може знадобитись також індивідуальна статистика для оцінки успішності конкретного розробника.	Аналіз вхідних даних з максимальною точністю
2	Зручний інтерфейс клієнтської частини системи	Команди розробників та їх менеджмент	Індивідуальне розуміння зручного інтерфейсу.	Інтерфейс клієнтської частини системи повинен бути інтуїтивно зрозумілим, з інтерактивними підказками

Таблиця 4.4

Фактори загроз

№	Фактор	Зміст загрози	Можлива реакція компанії
1	Висока конкуренція	Наразі на ринку присутні багато рішень, які можуть частково або повністю задовольнити потреби клієнтів	Швидка реакція команди розробників на побажання потенційних клієнтів, можливість розширення даного рішення та його доповнення.
2	Мала кількість доступних мов розпізнавання та інтерфейсу	Користувач не буде користуватися системою, якщо він не матиме змоги обрати зручну для себе мову інтерфейсу	Поступове збільшення кількості підтримуваних мов інтерфейсу

Таблиця 4.5

Фактори можливостей

№	Фактор	Зміст можливості	Можлива реакція компанії
1	Розширення клієнтської бази	За рахунок правильної стратегії розвитку та додавання нового функціоналу потенційно можна	Постійне додавання нового функціоналу до системи, аналіз відгуків

Таблиця 4.5 (продовження)

Фактори можливостей

№	Фактор	Зміст можливості	Можлива реакція компанії
		збільшити кількість клієнтів	користувачів про їхні побажання та проблеми у роботі з системою
2	Залучення інвестицій	Для залучення інвестицій потрібно розробити чіткий якісний бізнес план та можливості розширення і розвитку	Розробити чітку стратегію розвитку, щоб зацікавити потенційних інвесторів
3	Збільшення кількості підтримуваних платформ	Підтримка додаткових платформ дозволяє задовольнити потреби значно більшої кількості користувачів	Розробка системи з урахуванням поступового збільшення кількості підтримуваних платформ

Таблиця 4.6

Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
Тип конкуренції: **Нецінова**	Конкуренція, здійснюється через підвищення якості товарів, їх надійності, збільшення термінів служби, підвищення	Підвищення надійності та якості продукту

Таблиця 4.6 (продовження)

Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	**В чому проявляється дана характеристика**	**Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)**
Тип конкуренції: **Нецінова**	продуктивності, поліпшення умов реалізації за незмінних цін	Підвищення надійності та якості продукту
За рівнем конкурентної боротьби: **Міжнародний**	Конкуренти в різних країнах світ борють за клієнтів незважаючи на країну, у якій вони проживають	Ускладнення позиціонування компанії. Створення регіональних представництв для просування продукту
За галузевою ознакою: **внутрішньогалузева**	Конкуренція спостерігається між компаніями, що надають сервіси для онлайн навчання	Необхідність надання користувачу суттєвих переваг у порівнянні з конкурентами
За характером конкурентних переваг: **нецінова**	Конкуренція здійснюється зі сторони наявного функціоналу та якості реалізації продукту	Розширення функціоналу та якості товару
За інтенсивністю: **не марочна**	Бренд компанії не грає великої ролі	Заохочення клієнтів за рахунок можливостей товару, а не за рахунок бренду компанії

Таблиця 4.7

Обґрунтування факторів конкурентоспроможності

№	Фактор конкурентоспроможності	Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проєктів значущим)
1	Технічна підтримка	Для клієнтів важливо, щоб у разі виникнення позаштатних ситуацій вони могли оперативно отримати допомогу
2	Задоволення потреб	Продукт може якісно задовольняти потреби користувачів та підходить різним групам потенційних клієнтів.
3	Зручний користувацький інтерфейс	Інтуїтивно зрозумілий інтерфейс покращує загальне сприйняття
		системи користувачами та знижує ризик віддання переваги системам-конкурентам
4	Сегмент ринку	Даний сегмент ринку постійно зростає, тому у перспективі можемо отримати додаткових клієнтів, чиї вимоги може задовольнити продукт

Таблиця 4.8

Порівняльний аналіз сильних та слабких сторін

№	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-замінників у порівнянні з проєктом						
			-3	-2	-1	0	+1	+2	+3
1	Технічна підтримка	13			+				
2	Задоволення потреб	18					+		
3	Користувацький інтерфейс	18						+	
4	Сегмент ринку	15			+				

Таблиця 4.9

SWOT- аналіз стартап-проєкту

Сильні сторони: ● Користувацький інтерфейс; • Задоволення потреб;	**Слабкі сторони:** ● Технічна підтримка • Сегмент ринку
Можливості: ● Розширення клієнтської бази; • Залучення інвестицій	**Загрози:** • Висока конкуренція
● Збільшення кількості підтримуваних платформ	● Мала кількість доступних мов розпізнавання та інтерфейсу

4.3 Розробка ринкової стратегії проєкту

Розглянемо зацікавлені сторони, в тому числі споживачів і стратегії розвитку на ринку.

Таблиця 4.10

Вибір цільових груп потенційних споживачів

№	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1	Студенти	Висока, тому що безплатних продуктів, що економлять загальний бюджет навчання мало	70%	Середня	Середня, тому що на ринку присутні готові рішення, але частина з них має суттєві недоліки
2	Школярі	Середня, тому що далеко не усі школярі готові витрачати гроші чи багато грошей на освіту в цей час	30%	Середня	Середня, тому що на ринку присутні готові перевірені рішення

Таблиця 4.10 (продовження)

Вибір цільових груп потенційних споживачів

№	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
3	Люди з вищою освітою	Середня, тому що зазвичай можуть собі дозволити додаткові витрати, але не всі	50%	Середня	Низька, тому що на ринку присутні готові перевірені рішення
Які цільові групи обрано: стартапи, малобюджетні або малі проєкти.					

Таблиця 4.11

Визначення базової стратегії розвитку

№	Обрана альтернатива розвитку проекту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку
1	Розподілена система зі низьким навантаженням на браузер та ресурси комп’ютера	Стратегія диференційованого маркетингу	Швидкодія, відмовостійкість, масштабованість	Стратегія диференціації
2	Система з можливістю налаштування користувацького інтерфейсу	Стратегія диференційованого маркетингу	Зручність використання, інтуїтивна зрозумілість	Стратегія диференціації

Таблиця 4.12

Визначення базової стратегії конкурентної поведінки

№	Чи є проект «першопроходцем» на ринку?	Чи буде компанія шукати нових споживачів, або завойовувати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, і які?
1	Ні, існують аналоги	На початковому етапі – шукати нових, а потім – завойовувати існуючих у конкурентів	Частково (авторизація через різні сервіси, назви компонентів графічного інтерфейсу, зручний інтерфейс)

Таблиця 4.13

Визначення стратегії позиціонування

№	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного стартап-проекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)
1	Достовірність результатів статистики	Стратегія диференціації	Якісний аналіз вхідних даних,	● Достовірність результату

Таблиця 4.13 (продовження)

Визначення стратегії позиціонування

№	**Вимоги до товару цільової аудиторії**	**Базова стратегія розвитку**	**Ключові конкурентоспроможні позиції власного стартап-проекту**	**Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)**
			зручний інтерфейс	● Інтуїтивно зрозумілий інтерфейс
2	Зручність використання	Стратегія диференціації	Зручний інтерфейс	● Інтуїтивно зрозумілий інтерфейс ● Розподіленість складових системи
3	Швидкодія	Стратегія диференціації	Швидкість здійснення аналізу та завантаження даних	● Оптимальне розташування серверів ● Оптимізація програмного коду

5.4 Розробка маркетингової програми стартап-проєкту

Таблиця 4.14

Визначення ключових переваг концепції потенційного товару

№	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити
1	Швидкодія	Швидкість обробки запитів	Зменшення часу очікування користувачів
2	Простий інтерфейс	Простота використання	Користувачам зручніше використовувати систему
3	Якість статистики роботи над проєктом в цілому та індивідуального розробника	Якісний аналіз вхідних даних	Система здійснює аналіз вхідних даних з максимальною точністю
4	Інформативність результатів	Результати аналізу є інтуїтивно зрозумілими	Систему буде обирати більша кількість користувачів

Таблиця 4.15

4.5 Концепція маркетингових комунікацій

№	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного звернення	Концепція рекламного звернення
1	Клієнти дізнаються про товар з текстової, аудіо та відео реклами в інтернеті, тестових показів на виставках	Інтернет	Зручність інтерфейсу, швидкодія, якість статистики та інформативність результату	Повідомити про існування нового продукту та його переваг.	Новий погляд на управління платформи для онлайн навчання Рекламне звернення «Давайте не відставати від прогресу »

ВИСНОВКИ ДО РОЗДІЛУ 5

У даному розділі представлено результати розробки стартап-проекту до розроблюваної системи.

Викладено головну ідею проєкту, напрямки застосування та вигоди користувача. Визначено слабкі, сильні та нейтральні сторони ідеї проєкту. Проведено технологічний аудит проєкту, у ході якого описано функціонал розроблюваної системи та технології, що використовувалися у ході розробки.

Проаналізовано ринкові можливості запуску стартап-проєкту. Визначено фактори загроз та можливостей, альтернативи ринкового впровадження, фактори конкурентоспроможності. Проведено ступеневий аналіз конкуренції на ринку, аналіз конкуренції в галузі.

Визначено базову стратегію розвитку, стратегію конкурентної поведінки, стратегію позиціонування, ключові цільові групи потенційних користувачів, їх потреби та вимоги щодо наявної функціональності. Розроблено маркетингову програму стартап-проєкту, концепцію маркетингових комунікацій. Сформовано ключові переваги концепції потенційного товару та систему збуту.

В результаті виконання даного розділу було отримано стартап-проєкт, що покроково описує шлях для виведення розроблюваної системи на ринок та формування первинної бази потенційних користувачів.

ВИСНОВКИ

На основі п’ятьох розділів роботи можна зробити загальний висновок, що комплексний підхід до онлайн-навчання менторів, дизайну системи, її розробки та стратегії виведення на ринок стартап-проєкту відіграють вирішальну роль у формуванні ефективної та конкурентоспроможної освітньої платформи.

Ретельний аналіз існуючих ресурсів та потреб користувачів дозволив визначити ключові напрямки для створення інтегрованої системи, яка надає якісні менторські послуги. Обрані технології та архітектурні рішення підкреслюють важливість гнучкості та масштабованості у відповідності до зростаючих вимог безпеки даних та продуктивності.

Розробка продуманої даталогічної моделі та ефективного бекенду, разом з оптимізацією процесів деплойменту, демонструє потенціал створення надійної та високопродуктивної системи. Результати технологічного аудиту, аналізу ринкових можливостей та стратегічного планування забезпечують фундамент для реалізації стартап-проєкту, зорієнтованого на задоволення специфічних потреб цільових користувачів.

У сукупності, викладені в роботі стратегії та розробки створюють міцну основу для сталого розвитку освітньої платформи, що сприяє інноваційному та цілеспрямованому менторству в сучасному освітньому просторі.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ:

1. Richards M. Fundamentals of software architecture: an engineering approach / Mark Richards, Neal Ford. – [Б. м.] : O'Reilly Media, 2020. – 432 с.
2. Kazman R. Designing software architectures: a practical approach / Rick Kazman, Humberto Cervantes. – [Б. м.] : Addison-Wesley Professional, 2016. – 320 с.
3. Different types of elearning platforms and methods - benefits & key players [Електронний ресурс] // FATbit Blog. – Режим доступу: https://www.fatbit.com/fab/different-types-of-elearning-platforms-and-methods/.
4. Top 10 elearning features that everyone should know about [Електронний ресурс] // eLearning Industry. – Режим доступу: https://elearningindustry.com/elearningfeatures-top-everyone-know.
5. Getting and viewing grades for peer-reviewed assignments [Електронний ресурс] // eLearning Industry. – Режим доступу: https://www.coursera.support/s/article/208279946-Getting-and-viewing-grades-for-peer-reviewed-assignments?language=en_US
6. Top 10 elearning features that everyone should know about [Електронний ресурс] // eLearning Industry. – Режим доступу: https://elearningindustry.com/elearningfeatures-top-everyone-know.
7. Coursera`s mission, vision, and commitment to our community | coursera [Електронний ресурс]. – Режим доступу: https://about.coursera.org/.
8. Moodle [Електронний ресурс]. – Режим доступу: https://moodle.org/
9. Online Training Software | ispring [Електронний ресурс]. – Режим доступу: https://www.ispringsolutions.com/
10. Classroom | google for education [Електронний ресурс] // Google for Education. – Режим доступу: https://edu.google.com/workspace-for-education/classroom/.
11. Janssens T. Tom Janssens | The goal of software architecture [Електронний ресурс] / Tom Janssens. – Режим доступу: https://tojans.me/posts/the-goal-of-softwarearchitecture/.
12. What is a multi layered software architecture? | Packt Hub [Електронний ресурс] // Packt Hub. – Режим доступу: https://hub.packtpub.com/what-is-multi-layeredsoftware-architecture/.

13. What does scalability mean for systems and services? [Електронний ресурс]. Режим доступу: https://www.lucidchart.com/blog/what-does-scalability-mean-for-systems-and-services
14. Sébastien Goasguen. Docker Cookbook: Solutions and Examples for Building Distributed Applications / Sébastien Goasguen – O'Reilly Media, Inc, 2016.
15. Сибельшатц А., Сударшан С. Концепції системи баз даних / – New York: McGrawHill – 2011
16. Learning JavaScript: JavaScript Essentials for Modern Application Development 3rd Edition / Ethan Brown. – K.: O'Reilly Media, 2016 – 53 – 89 с.
17. Docker Documentation [Електронний ресурс]. – Режим доступу: URL: https://docs.docker.com/
18. PostgreSQL: a closer look at the object-relational database management system [Електронний ресурс] // IONOS Digitalguide. – Режим доступу: https://www.ionos.com/digitalguide/server/know-how/postgresql/.
19. What is swagger [Електронний ресурс] // API Documentation & Design Tools for Teams | Swagger. – Режим доступу: https://swagger.io/docs/specification/2-0/what-isswagger/.
20. React – JavaScript-бібліотека для створення користувацьких інтерфейсів [Електронний ресурс] // React – JavaScript-бібліотека для створення користувацьких інтерфейсів. – Режим доступу: https://uk.reactjs.org/.
21.Redux - A predictable state container for javascript apps. | redux [Електронний ресурс]. – Режим доступу: https://redux.js.org/
22. Installation | knex.js [Електронний ресурс] // SQL Query Builder for Javascript | Knex.js. – Режим доступу: https://knexjs.org/guide/#configuration-options.
23. What Is JWT Authentication? How Does It Work? [Електронний ресурс] // Robert MacDonald – Режим доступу: https://www.1kosmos.com/authentication/jwt-authentication/

ДОДАТКИ

ДОДАТОК 1 Вихідний код програми

Файл .eslint.yaml

```
env:
 es2022: true
parserOptions:
 ecmaVersion: 2021
 sourceType: module
extends:
 - eslint:recommended
 - plugin:import/recommended
 - plugin:import/typescript
 - plugin:@typescript-eslint/recommended
plugins:
 - simple-import-sort
rules:
 padding-line-between-statements:
   - off
 no-tabs:
   - error
   - allowIndentationTabs: true
 no-multiple-empty-lines:
   - error
   - max: 1
 no-var:
   - error
 no-console:
   - error
 arrow-parens:
   - error
   - always
 no-else-return:
   - error
   - allowElseIf: false
 jsx-quotes:
   - error
   - prefer-double
 max-params:
   - error
   - 2
 lines-between-class-members:
   - error
 '@typescript-eslint/explicit-function-return-type':
   - error
   - allowTypedFunctionExpressions: true
 '@typescript-eslint/quotes':
   - error
   - single
 '@typescript-eslint/object-curly-spacing':
   - error
   - always
 '@typescript-eslint/semi':
   - error
   - always
 '@typescript-eslint/comma-dangle':
   - error
   - always-multiline
 '@typescript-eslint/no-unused-vars':
   - error
   - vars: all
     args: after-used
     argsIgnorePattern: ^_
     caughtErrors: all
 '@typescript-eslint/no-empty-interface':
   - error
   - allowSingleExtends: true
'@typescript-eslint/padding-line-between-stateme
nts':
   - error
   - blankLine: always
     prev: '*'
     next:
       - switch
       - class
       - function
       - if
       - return
       - try
       - interface
       - type
'@typescript-eslint/explicit-member-accessibility':
   - error
 import/no-unresolved:
   - off
 import/no-default-export:
   - error
 import/newline-after-import:
   - error
   - count: 1
 import/first:
   - error
 import/no-duplicates:
   - error
 import/group-exports:
   - error
 simple-import-sort/imports:
   - error
 simple-import-sort/exports:
   - error
```

Файл .prettier.yaml

```
printWidth: 80
tabWidth: 2
useTabs: false
semi: true
singleQuote: true
quoteProps: preserve
trailingComma: all
bracketSpacing: true
arrowParens: always
```

```
overrides:
  - files: '*.scss'
    options:
```

Файл .lintstagedrc.yaml

```
'*.{ts,tsx,html,json,md}': prettier --write
'*': npm run lint:editorconfig && npm run lint:fs
'shared/**/*.ts': npm run lint:shared:js
```

Файл backend/.eslintrc.yml

```
extends:
  - ../.eslintrc.yml
env:
  node: true
overrides:
```

Файл frontend/.eslintrc.yml

```
extends:
  - ../.eslintrc.yml
env:
  browser: true
settings:
  react:
    version: '18'
```

Файл .stylelintrc.yml

```
extends:
  - stylelint-config-standard
plugins:
  - stylelint-order
rules:
  selector-class-pattern: null
  color-hex-length: long
  declaration-no-important: true
  max-nesting-depth: 0
  no-descending-specificity: true
  scss/at-import-partial-extension: always
  string-quotes: single
  unit-disallowed-list:
    - em
    - rem
  order/order:
    - custom-properties
    - declarations
  order/properties-order:
    - all
    - position
    - top
    - right
    - bottom
    - left
    - inset
    - inset-block-start
    - inset-block-end
    - inset-inline-start
    - inset-inline-end
    - z-index
      singleQuote: false
  - files: '*.html'
    options:
      printWidth: 9999
'backend/**/*.ts': npm run lint:backend:js
'backend/tests/**/*.ts': npm run
lint:backend:tests:js
'frontend/**/*.scss': npm run lint:frontend:css
'frontend/**/*.{ts,tsx}': npm run lint:frontend:js

  - files:
    - knexfile.ts
    rules:
      'import/no-default-export':
        - off

overrides:
  - files:
    -
src/common/types/react-table/react-table-config.d.
ts
    rules:
      '@typescript-eslint/no-explicit-any':
        - off

    - display
    - grid-template
    - grid-template-rows
    - grid-template-columns
    - grid-template-areas
    - grid-auto-rows
    - grid-auto-columns
    - grid-auto-flow
    - grid-area
    - grid-row
    - grid-column
    - grid-row-start
    - grid-row-end
    - grid-column-start
    - grid-column-end
    - flex
    - flex-grow
    - flex-shrink
    - flex-basis
    - flex-flow
    - flex-direction
    - flex-wrap
    - order
    - justify-content
    - justify-items
    - justify-self
    - align-content
    - align-items
    - align-self
    - grid-gap
    - gap
```

- grid-row-gap
- row-gap
- grid-column-gap
- column-gap
- float
- clear
- box-sizing
- writing-mode
- width
- min-width
- max-width
- height
- min-height
- max-height
- inline-size
- min-inline-size
- max-inline-size
- block-size
- min-block-size
- max-block-size
- margin
- margin-top
- margin-right
- margin-bottom
- margin-left
- margin-block-start
- margin-block-end
- margin-inline-start
- margin-inline-end
- padding
- padding-top
- padding-right
- padding-bottom
- padding-left
- padding-block-start
- padding-block-end
- padding-inline-start
- padding-inline-end
- overflow
- overflow-x
- overflow-y
- color
- font
- font-weight
- font-size
- font-family
- font-style
- font-display
- font-variant
- font-size-adjust
- font-stretch
- font-effect
- font-emphasize
- font-emphasize-position
- font-emphasize-style
- font-smooth
- line-height
- direction
- letter-spacing
- white-space
- text-align
- text-align-last
- text-transform
- text-decoration
- text-emphasis
- text-emphasis-color
- text-emphasis-style
- text-emphasis-position
- text-indent
- text-justify
- text-outline
- text-wrap
- text-overflow
- text-overflow-ellipsis
- text-overflow-mode
- text-orientation
- text-shadow
- vertical-align
- word-wrap
- word-break
- word-spacing
- overflow-wrap
- tab-size
- hyphens
- unicode-bidi
- columns
- column-count
- column-fill
- column-gap
- column-rule
- column-rule-color
- column-rule-style
- column-rule-width
- column-span
- column-width
- page-break-after
- page-break-before
- page-break-inside
- src
- list-style
- list-style-position
- list-style-type
- list-style-image
- table-layout
- empty-cells
- caption-side
- background
- background-color
- background-image
- background-repeat
- background-position
- background-position-x

- background-position-y
- background-size
- background-clip
- background-origin
- background-attachment
- background-blend-mode
- box-decoration-break
- border
- border-width
- border-style
- border-color
- border-top
- border-block-start
- border-top-width
- border-top-style
- border-top-color
- border-right
- border-inline-end
- border-right-width
- border-right-style
- border-right-color
- border-bottom
- border-block-end
- border-bottom-width
- border-bottom-style
- border-bottom-color
- border-left
- border-inline-start
- border-left-width
- border-left-style
- border-left-color
- border-radius
- border-top-left-radius
- border-top-right-radius
- border-bottom-right-radius
- border-bottom-left-radius
- border-image
- border-image-source
- border-image-slice
- border-image-width
- border-image-outset
- border-image-repeat
- border-collapse
- border-spacing
- outline
- outline-width
- outline-style
- outline-color
- outline-offset
- box-shadow
- visibility
- cursor
- mix-blend-mode
- backdrop-filter
- will-change
- transform
- transform-origin
- transform-style
- backface-visibility
- opacity
- filter
- perspective
- perspective-origin
- transition
- transition-delay
- transition-timing-function
- transition-duration
- transition-property
- animation
- animation-name
- animation-duration
- animation-play-state
- animation-timing-function
- animation-delay
- animation-iteration-count
- animation-direction
- animation-fill-mode
- appearance
- clip
- clip-path
- counter-reset
- counter-increment
- resize
- user-select
- nav-index
- nav-up
- nav-right
- nav-down
- nav-left
- pointer-events
- quotes
- touch-action
- zoom
- fill
- fill-rule
- clip-rule
- stroke
- stroke-width

Файл frontend/.stylelintrc.yml

```
extends:
  - stylelint-config-standard-scss
  - ../.stylelintrc.yml
plugins:
  - stylelint-scss
overrides:
  - files:
    -
src/components/billing/components/replenish-card
s-list/components/replenish-card/styles.scss
    rules:
      'declaration-no-important': - null
```

Файл package.json

```
{
  "name": "bsa-2022-guruhub",
  "private": "true",
  "engines": {
    "node": ">=16 <17",
    "npm": ">=8.8 <9"
  },
  "scripts": {
    "lint:editorconfig": "editorconfig-checker",
    "lint:fs": "ls-lint",
    "lint:shared:js": "cd shared && npm run lint:js",
    "lint:shared:type": "cd shared && npm run
lint:type",
    "lint:shared": "cd shared && npm run lint",
    "lint:frontend:css": "cd frontend && npm run
lint:css",
    "lint:frontend:js": "cd frontend && npm run
lint:js",
    "lint:frontend:type": "cd frontend && npm run
lint:type",
    "lint:frontend": "cd frontend && npm run lint",
    "lint:backend:js": "cd backend && npm run
lint:js",
    "lint:backend:type": "cd backend && npm run
lint:type",
    "lint:backend": "cd backend && npm run lint",
    "lint:backend:tests:js": "cd backend/tests &&
npm run lint:js",
    "lint:backend:tests:type": "cd backend/tests &&
npm run lint:type",
    "lint:backend:tests": "cd backend/tests && npm
run lint",
    "lint:mobile:js": "cd mobile && npm run
lint:js",
    "lint:mobile:type": "cd mobile && npm run
lint:type",
    "lint:mobile": "cd mobile && npm run lint",
    "lint:css": "npm run lint:frontend:css",
    "lint:js": "npm run lint:shared:js && npm run
lint:backend:js && npm run lint:backend:tests:js
&& npm run lint:frontend:js && npm run
lint:mobile:js",
    "lint:type": "npm run lint:shared:type && npm
run lint:backend:type && npm run
lint:backend:tests:type && npm run
lint:frontend:type && npm run lint:mobile:type",
    "lint": "npm run lint:editorconfig && npm run
lint:fs && npm run lint:shared && npm run
lint:backend && npm run lint:backend:tests &&
npm run lint:mobile && npm run lint:frontend",
    "install:shared": "cd shared && npm install",
    "install:frontend": "cd frontend && npm
install",
    "install:backend": "cd backend && npm
install",
    "install:mobile": "cd mobile && npm install",
    "install:backend:tests": "cd backend/tests &&
npm install",
    "install:all": "npm install && npm run
install:shared && npm run build:shared && npm
run install:frontend && npm run install:backend
&& npm run install:mobile && npm run
install:backend:tests",
    "install:build": "cd ./build/shared/build && npm
ci && cd ../../backend && npm ci",
    "build:shared": "cd shared && npm run build",
    "build:frontend": "cd frontend && npm run
build",
    "build:backend": "cd backend && npm run
build",
    "update:shared": "cd ./shared && npm version
patch && npm run build",
    "update:shared:backend": "cd backend && npm
run update:guruhub-shared",
    "update:shared:backend:tests": "cd
backend/tests && npm run
update:guruhub-shared",
    "update:shared:frontend": "cd frontend &&
npm run update:guruhub-shared",
    "update:shared:mobile": "cd mobile && npm
run update:guruhub-shared",
    "update:shared:all": "npm run install:shared &&
npm run build:shared && npm run
update:shared:backend && npm run
update:shared:backend:tests && npm run
update:shared:frontend && npm run
update:shared:mobile",
    "build": "npm run build:shared && npm run
build:frontend && npm run build:backend &&
npm run prepare-build",
    "prebuild": "npm run install:all",
    "prepare-build": "sh ./prepare-build.sh",
    "prestart": "npm run build && npm run
install:build",
    "start": "cd ./build/backend && npm start"
  },
  "devDependencies": {
    "@ls-lint/ls-lint": "1.11.2",
    "@typescript-eslint/eslint-plugin": "5.30.6",
    "editorconfig-checker": "4.0.2",
    "eslint": "8.19.0",
    "eslint-plugin-import": "2.26.0",
    "eslint-plugin-simple-import-sort": "7.0.0",
    "lint-staged": "13.0.3",
    "prettier": "2.7.1",
    "simple-git-hooks": "2.8.0",
    "stylelint": "14.9.1",
    "stylelint-config-standard": "26.0.0",
    "stylelint-order": "5.0.0",
    "typescript": "4.7.4"
```

```
  },
  "simple-git-hooks": {
    "pre-commit": "npx lint-staged"
  }
{
  "name": "backend",
  "private": "true",
  "engines": {
    "node": ">=16 <17",
    "npm": ">=8.8 <9"
  },
  "scripts": {
    "migrate": "knex migrate:latest",
    "seed": "knex seed:run",
    "migrate:dev": "node -r tsconfig-paths/register
node_modules/knex/bin/cli.js migrate:latest",
    "migrate:dev:make": "node -r
tsconfig-paths/register
node_modules/knex/bin/cli.js migrate:make -x ts",
    "migrate:dev:down": "node -r
tsconfig-paths/register
node_modules/knex/bin/cli.js migrate:down",
    "migrate:dev:rollback": "node -r
tsconfig-paths/register
node_modules/knex/bin/cli.js migrate:rollback
--all",
    "seed:dev": "node -r tsconfig-paths/register
node_modules/knex/bin/cli.js seed:run",
    "seed:dev:make": "node -r
tsconfig-paths/register
node_modules/knex/bin/cli.js seed:make -x ts",
    "lint:js": "npx eslint \"src/**/*.ts\"",
    "lint:type": "npx tsc --noEmit",
    "lint": "npm run lint:js && npm run lint:type",
    "start:dev": "nodemon --ignore ./tests --exec
ts-node --files -r tsconfig-paths/register
./src/server.ts",
    "prestart": "npm run migrate",
    "start": "node ./src/server.js",
    "start:build": "node ./build/src/server.js",
    "build:ts": "tsc && tsc-alias -p tsconfig.json",
```

Файл frontend/package.json

```
{
  "name": "frontend",
  "private": true,
  "engines": {
    "node": ">=16 <17",
    "npm": ">=8.8 <9"
  },
  "scripts": {
    "lint:js": "npx eslint \"src/**/*.{ts,tsx}\"
--resolve-plugins-relative-to .",
    "lint:css": "npx stylelint \"src/**/*.scss\"",
    "lint:type": "npx tsc --noEmit",
    "lint": "npm run lint:css && npm run lint:js &&
npm run lint:type",
    "start": "react-scripts start",
    "build": "react-scripts build",
    "update:guruhub-shared": "rm -r
node_modules/guruhub-shared && npm i -ES
guruhub-shared"
  },
  "dependencies": {
    "@hookform/error-message": "2.0.0",
    "@hookform/resolvers": "2.9.6",
    "@reduxjs/toolkit": "1.8.3",
    "clsx": "1.2.1",
```

```
    "build:copy": "cp -r package.json
package-lock.json build && cp -r
src/documentation build/src",
    "build": "npm run build:ts && npm run
build:copy",
    "update:guruhub-shared": "rm -r
node_modules/guruhub-shared && npm i -ES
guruhub-shared"
  },
  "devDependencies": {
    "@types/bcrypt": "5.0.0",
    "nodemon": "2.0.19",
    "pino-pretty": "8.1.0",
    "ts-node": "10.8.2",
    "tsc-alias": "1.6.11"
  },
  "dependencies": {
    "@aws-sdk/client-s3": "3.154.0",
    "@fastify/cookie": "9.2.0",
    "@fastify/cors": "8.5.0",
    "@fastify/multipart": "7.1.1",
    "@fastify/oauth2": "7.8.0",
    "@fastify/static": "6.5.0",
    "@fastify/swagger": "7.4.1",
    "axios": "0.27.2",
    "bcrypt": "5.0.1",
    "dotenv": "16.0.1",
    "fastify": "4.2.1",
    "guruhub-shared": "file:../shared/build",
    "jose": "4.8.3",
    "knex": "2.1.0",
    "objection": "3.0.1",
    "pg": "8.7.3",
    "socket.io": "4.5.2",
    "stripe": "10.7.0"
  }
}
```

```
    "guruhub-shared": "file:../shared/build",
    "history": "5.3.0",
    "react": "18.2.0",
    "react-datepicker": "4.8.0",
    "react-dom": "18.2.0",
    "react-hook-form": "7.33.1",
    "react-multi-carousel": "2.8.2",
    "react-redux": "8.0.2",
    "react-router-dom": "6.3.0",
    "react-scripts": "5.0.1",
    "react-select": "5.4.0",
    "react-stripe-checkout": "2.6.3",
    "react-table": "7.8.0",
    "react-toastify": "9.0.8",
    "socket.io-client": "4.5.2"
  },
  "devDependencies": {
    "@types/node": "18.0.5",
    "@types/react": "18.0.15",
    "@types/react-datepicker": "4.4.2",
    "@types/react-dom": "18.0.6",
    "@types/react-table": "7.7.12",
    "sass": "1.53.0",
    "stylelint-config-standard-scss": "5.0.0"
  },
  "eslintConfig": {
    "extends": [
      "react-app"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  }
}
```

Файл shared/package.json

```
{
  "name": "shared",
  "private": "true",
  "engines": {
    "node": ">=16 <17",
    "npm": ">=8.8 <9"
  },
  "scripts": {
    "lint:js": "npx eslint \"src/**/*.ts\"",
    "lint:type": "npx tsc --noEmit",
    "lint": "npm run lint:js && npm run lint:type",
    "build:ts": "tsc && tsc-alias -p tsconfig.json",
    "build": "npm run build:ts && cp package.json
package-lock.json build"
  },
  "devDependencies": {
    "@types/change-case": "2.3.1",
    "@types/debounce": "1.2.1",
    "@types/sanitize-html": "2.6.2",
    "@types/uuid": "8.3.4",
    "tsc-alias": "1.6.11"
  },
  "dependencies": {
    "change-case": "4.1.2",
    "date-fns": "2.29.2",
    "debounce": "1.2.1",
    "joi": "17.6.0",
    "sanitize-html": "2.7.1",
    "uuid": "8.3.2"
  },
  "version": "0.0.2"
}
```

Файл tsconfig.json

```
{
  "compilerOptions": {
    "target": "es2017",
    "lib": ["dom", "dom.iterable", "esnext"],
    "allowJs": true,
    "esModuleInterop": true,
    "forceConsistentCasingInFileNames": true,
    "strict": true,
    "resolveJsonModule": true,
    "noFallthroughCasesInSwitch": true,
    "noImplicitOverride": true,
    "skipLibCheck": true
  }
}
```

Файл backend/tsconfig.json

```
{
  "extends": "../tsconfig.json",
  "include": ["src"],
  "exclude": ["node_modules"],
  "compilerOptions": {
    "baseUrl": ".",
    "paths": {
      "~/*": ["./src/*"]
    },
    "outDir": "build",
    "module": "commonjs"
  }
```

}

Файл frontend/tsconfig.json

```
{
 "extends": "../tsconfig.json",
 "include": ["src"],
 "exclude": ["node_modules"],
 "compilerOptions": {
  "baseUrl": "src",
  "allowJs": true,
  "skipLibCheck": true,
  "esModuleInterop": true,
  "allowSyntheticDefaultImports": true,
  "forceConsistentCasingInFileNames": true,
  "noFallthroughCasesInSwitch": true,
  "module": "ESNext",
  "moduleResolution": "Node",
  "resolveJsonModule": true,
  "noEmit": true,
  "jsx": "react-jsx"
 }
}
```

Файл shared/tsconfig.json

```
{
 "extends": "../tsconfig.json",
 "include": ["src"],
 "exclude": ["node_modules"],
 "compilerOptions": {
  "baseUrl": ".",
  "paths": {
   "~/*": ["src/*"]
  },
  "outDir": "build",
  "declaration": true,
  "module": "commonjs"
 }
}
```

Файл shared/src/common/enums/api/api-path.enum.ts

```
enum ApiPath {
 AUTH = '/auth',
 BILLING = '/billing',
 CHATS = '/chats',
 INTERVIEWS = '/interviews',
 PERMISSIONS = '/permissions',
 GROUPS = '/groups',
 USERS = '/users',
 USER_DETAILS = '/user-details',
 CATEGORIES = '/categories',
 COURSES = '/courses',
 COURSE_MODULES = '/courses',
 MENTORS = '/mentors',
 TASKS = '/tasks',
 DASHBOARD = '/dashboard',
}
export { ApiPath };
```

Файл shared/src/common/enums/api/auth-api-path.enum.ts

```
enum AuthApiPath {
 ROOT = '/',
 SIGN_UP = '/sign-up',
 SIGN_IN = '/sign-in',
 SIGN_REDIRECT = '/sign-redirect',
 CURRENT_USER = '/current-user',
 GOOGLE_SIGN = '/google/sign',
 GOOGLE_CALLBACK = '/google/callback',
 GITHUB_SIGN = '/github/sign',
 GITHUB_CALLBACK = '/github/callback',
}
export { AuthApiPath };
```

Файл shared/src/common/enums/api/billing-api-path.enum.ts

```
enum BillingApiPath {
 BALANCE = '/balance',
 REPLENISH = '/replenish',
 WITHDRAW = '/withdraw',
 TRANSACTIONS = '/transactions',
}
export { BillingApiPath };
```

Файл shared/src/common/enums/api/categories-api-path.enum.ts

```
enum CategoriesApiPath {
 ROOT = '/',
 DASHBOARD = '/dashboard',
 $ID = '/:id',
}
export { CategoriesApiPath };
```

Файл shared/src/common/enums/api/chats-api-path.enum.ts

```
enum ChatsApiPath {
 ROOT = '/',
 $ID = '/:id',
 HAS_UNREAD_MESSAGES =
'/has-unread-messages',
 $ID_READ = '/:id/read',
 READ = '/read',
}
export { ChatsApiPath }
```

Файл shared/src/common/enums/api/course-modules-api-path.enum.ts

```
enum CourseModulesApiPath {
  ROOT = '/',
  COURSES_$ID_MODULES =
'/:courseId/modules',
  COURSES_$ID_MODULES_$ID =
'/:courseId/modules/:moduleId',
  COURSES_$ID_MENTEES_$ID =
'/:courseId/mentees/:menteeId',
  MENTEES = '/mentees',
}
export { CourseModulesApiPath };
```

Файл shared/src/common/enums/api/courses-api-path.enum.ts

```
enum CoursesApiPath {
  ROOT = '/',
  MODULES = '/modules',
  CATEGORY = '/category',
  CATEGORIES = '/categories',
  $ID = '/:id',
  $ID_MODULES = '/:id/modules',
  $ID_MENTORS = '/:id/mentors',
  $ID_MENTEES = '/:id/mentees',
  $ID_IS_MENTOR_CHECK =
'/:id/is-mentor-check',
  IS_MENTOR_CHECK = '/is-mentor-check',
  $ID_HAS_MENTOR_CHECK =
'/:id/has-mentor-check',
  HAS_MENTOR_CHECK = '/has-mentor-check',
  MENTORS = '/mentors',
  MENTEES = '/mentees',
  $ID_CATEGORY = '/:id/category',
  DASHBOARD = '/dashboard',
  MENTORING = '/mentoring',
  STUDYING = '/studying',
  $ID_MENTEES_$ID_IS_MENTOR_CHECK =
'/:courseId/mentees/:menteeId/is-mentor-check',
  POPULAR = '/popular',
}
export { CoursesApiPath };
```

Файл shared/src/common/enums/api/groups-api-path.enum.ts

```
enum GroupsApiPath {
  ROOT = '/',
  $ID = '/:id',
}
export { GroupsApiPath };
```

Файл shared/src/common/enums/api/interviews-api-path.enum.ts

```
enum InterviewsApiPath {
  ROOT = '/',
  $ID = '/:id',
  $ID_UPDATE_WITHOUT_INTERVIEWER =
'/:id/update-without-interviewer',
  UPDATE_WITHOUT_INTERVIEWER =
'/update-without-interviewer',
  NOTES = '/notes',
  INTERVIEWEE_USER_$ID_CATEGORIES =
'/interviewee/:intervieweeUserId/categories',

INTERVIEWEE_USER_$ID_ACTIVE_CATEG
ORIES =
'/interviewee/:intervieweeUserId/active/categories'
,
  INTERVIEWERS_CATEGORIES_$ID =
'/interviewers/categories/:categoryId',
  INTERVIEWERS = '/interviewers',
  INTERVIEWEE = '/interviewee',
  CATEGORIES = '/categories',
  $ID_OTHER = '/:id/other',
  OTHER = '/other',
  ACTIVE = '/active',
}
export { InterviewsApiPath };
```

Файл shared/src/common/enums/api/mentors-api-path.enum.ts

```
enum MentorsApiPath {
  ROOT = '/',
  COURSES = '/courses',
export { MentorsApiPath };
  MENTEES = '/mentees',
  COURSES_$ID_MENTEES_$ID =
'/courses/:courseId/mentees/:menteeId',
```

Файл shared/src/common/enums/api/permission-api-path.enum.ts

```
enum PermissionApiPath {
  ROOT = '/',
}
export { PermissionApiPath };
```

Файл shared/src/common/enums/api/tasks-api-path.enum.ts

```
enum TasksApiPath {
  TASKS_$ID = '/:taskId',
  MODULES_$ID_MENTEES_$ID =
'/modules/:moduleId/mentees/:menteeId',
  TASKS_$ID_NOTES = '/:taskId/notes',
```

```
  MODULES = '/modules',
  MENTEES = '/mentees',
  NOTES = '/notes',

COURSES_$ID_MODULES_$ID_MENTEES_$
ID =
'/courses/:courseId/modules/:moduleId/mentees/:
menteeId',
  COURSES = '/courses',
  COURSES_$ID_MENTEES_$ID =
'/courses/:courseId/mentees/:menteeId',
}
export { TasksApiPath };
```

Файл shared/src/common/enums/api/user-details-api-path.enum.ts

```
enum UserDetailsApiPath {
  ROOT = '/',
  USER_$ID_AVATAR = '/:userId/avatar',
  USER = '/user',
  AVATAR = '/avatar',
}
export { UserDetailsApiPath };
```

Файл shared/src/common/enums/api/users-api-path.enum.ts

```
enum UsersApiPath {
  ROOT = '/',
  $ID = '/:id',
}
export { UsersApiPath };
```

Файл shared/src/common/enums/case/case.enum.ts

```
enum StringCase {
  KEBAB_CASE = 'kebab',
  SNAKE_CASE = 'snake',
  CAMEL_CASE = 'camel',
}
export { StringCase };
```

Файл shared/src/common/enums/chat/chat-message-status.enum.ts

```
enum ChatMessageStatus {
  READ = 'read',
  UNREAD = 'unread',
}
export { ChatMessageStatus };
```

Файл shared/src/common/enums/chat/chat-validation-message.enum.ts

```
enum ChatValidationMessage {
  MESSAGE_REQUIRE = 'Message is required',
  MESSAGE_STRING = 'Message must be of
type string',
  RECEIVER_ID_REQUIRE = 'Receiver id is
required',
  RECEIVER_ID_INTEGER = 'Receiver id must
be of type integer',
  CHAT_ID_STRING = 'Chat id must be of type
string',
}
export { ChatValidationMessage };
```

Файл shared/src/common/enums/course/course-validation-message.enum.ts

```
import { CourseValidationRule } from
'./course-validation-rule.enum';
const CourseValidationMessage = {
  INVALID_URL:
    'URL is not valid. URL must be a Udemy
course link (https://www.udemy.com/24823,
https://www.udemy.com/java-tutorial)',
  URL_REQUIRE: 'URL is required',
  EMPTY_COURSE_ID: 'Course id can not be
empty',
  EMPTY_COURSE_CATEGORY_ID: 'Course
category can not be empty',
  CATEGORY_REQUIRE: 'Category id is
required',
  CATEGORY_INTEGER: 'Category id must be
of type integer',
  MENTOR_ID_REQUIRE: 'Mentor id is
required',
  MENTOR_ID_INTEGER: 'Mentor id must be of
type integer',
  MENTEE_ID_REQUIRE: 'Mentee id is
required',
  MENTEE_ID_INTEGER: 'Mentee id must be of
type integer',
  INVALID_STUDENTS_COUNT_NUMBER:
`Students count number should be not less than
${CourseValidationRule.STUDENTS_COUNT_
MIN_NUMBER}`,
} as const;
export { CourseValidationMessage };
```

Файл shared/src/common/enums/course/course-validation-rule.enum.ts

```
const CourseValidationRule = {
  STUDENTS_COUNT_MIN_NUMBER: 1,
} as const;
export { CourseValidationRule };
```

Файл shared/src/common/enums/exceptions/custom-exception-name.enum.ts

```
enum CustomExceptionName {
  HTTP_ERROR = 'HttpError',
  AUTH_ERROR = 'AuthError',
  BILLING_ERROR = 'BillingError',
  INVALID_CREDENTIALS =
'InvalidCredentials',
  INVALID_GROUP = 'InvalidGroup',
  INVALID_COURSE = 'InvalidCourse',
  COURSE_MODULE_ERROR =
'CourseModuleError',
  PERMISSION_ERROR = 'PermissionError',
  INTERVIEW_NOTE_ERROR =
'InterviewNoteError',
  STORAGE_ERROR = 'StorageError',
  USER_ERROR = 'UserError',
  INTERVIEW_ERROR = 'InterviewError',
  COURSES_TO_MENTORS_ERROR =
'CoursesToMentorsError',
  MENTEES_TO_MENTORS_ERROR =
'MenteesToMentorsError',
  INVALID_FILE = 'InvalidFileError',
  TASKS_ERROR = 'TasksError',
  USER_DETAILS_ERROR = 'UserDetailsError',
}
export { CustomExceptionName };
```

Фалй shared/src/common/enums/exceptions/exception-message.enum.ts

```
enum ExceptionMessage {
  UNKNOWN_ERROR = 'Unknown error
occurred.',
  BAD_CREDENTIALS = 'You have entered an
invalid email or password.',
  UNAUTHORIZED_USER = 'Unauthorized
user.',
  EMAIL_IS_ALREADY_TAKEN = 'This email
is already taken.',
  INVALID_GROUP_NAME = 'Group name is
invalid.',
  INVALID_GROUP_PERMISSIONS = 'Group
permissions are invalid.',
  INVALID_GROUP_USERS = 'Group users are
invalid.',
  INVALID_GROUP_ID = 'Group id is invalid.',
  INVALID_TOKEN = 'Token is invalid.',
  INVALID_URL_HOST = 'URL is invalid. Only
courses from Udemy are supported.',
  INVALID_COURSE_VENDOR = 'Course
vendor is invalid.',
  INVALID_COURSE_CATEGORY = 'Course
category is invalid.',

UDEMY_SERVER_RETURNED_AN_INVALID
_RESPONSE = 'Udemy server returned an invalid
response.',
  COURSE_NOT_FOUND = 'Course not found.',
  COURSE_EXIST = 'Course already exists.',
  UNKNOWN_INTERVIEW_NOTE_AUTHOR
= 'Unknown interview note author.',
  PERMISSION_LACK = 'You do not have
permission to access this resource.',
  STORAGE_NOT_FOUND = 'Storage was not
found.',
  INVALID_USER_REQUEST = 'User request is
invalid.',
  INTERVIEW_EXIST = 'Interview for this
category was either passed or is in progress.',
  INTERVIEW_DOES_NOT_EXIST = 'Interview
with that ID does not exist.',

INTERVIEWEE_CAN_NOT_UPDATE_OWN_I
NTERVIEW = 'Interviewee can not update own
interview.',
  ALREADY_MENTOR_FOR_COURSE = 'You
are already a mentor for this course.',
  ALREADY_HAVE_MENTOR_FOR_COURSE
= 'You already have a mentor for this course.',
  INVALID_FILE_TYPE = 'The uploaded file has
an invalid type. Only PNG, JPEG and SVG files
can be uploaded.',
  TASK_DOES_NOT_EXIST = 'Task does not
exist.',
  TASK_COMPLETED = 'Task is already
completed and cannot be updated.',
  USER_DETAILS_NOT_FOUND = 'User was
not found.',
  FILE_TOO_BIG = 'This file is too large to
upload. The maximum supported file size is:
1MB.',

UNABLE_TO_PERFORM_BILLING_OPERATI
ON = 'Unable to perform billing operation.',
  NOT_ENOUGH_FUNDS_TO_WITHDRAW =
'You do not have minimal sum of funds (1$) to
withdraw.',

NOT_ENOUGH_FUNDS_TO_PAY_FOR_MEN
TORS_SERVICES = 'You do not have enough
funds on your account to pay for mentors
services.',
  USER_CAN_NOT_BE_DELETED = 'This user
is forbidden to delete.',
  MENTOR_CAN_NOT_BE_DELETED = 'This
user is forbidden to delete because he is a mentor.',
```

```
  MENTEE_CAN_NOT_BE_DELETED = 'This user is forbidden to delete because he is a mentee.',
  INTERVIEWEE_CAN_NOT_BE_DELETED = 'This user is forbidden to delete because he has interviews.',
  PROTECTED_GROUP_UPDATE = 'This group is protected and cannot be updated.',
  PROTECTED_GROUP_DELETE = 'This group is protected and cannot be deleted.',
  MENTOR_CANT_BE_STUDENT = 'A course mentor cannot become a student of the course',
  STUDENT_CANT_BE_MENTOR = 'A course student cannot become a mentor of the course',
}
export { ExceptionMessage };
```

Файл shared/src/common/enums/file/content-type.enum.ts

```
enum ContentType {
  JSON = 'application/json',
  IMAGE = 'image',
  IMAGE_PNG = 'image/png',
  IMAGE_JPEG = 'image/jpeg',
  IMAGE_SVG = 'image/svg+xml',
  FORM_DATA = 'multipart/form-data',
}
export { ContentType };
```

Файл shared/src/common/enums/group/group-validation-message.enum.ts

```
import { GroupValidationRule } from './group-validation-rule.enum';
const GroupValidationMessage = {
  NAME_REQUIRE: 'Name is required',
  NAME_STRING: 'Name must be of type string',
  NAME_MIN_LENGTH: `Name length must be at least ${GroupValidationRule.NAME_MIN_LENGTH} characters long`,
  NAME_MAX_LENGTH: `Name length must be at most ${GroupValidationRule.NAME_MAX_LENGTH } characters long`,
  PERMISSION_IDS_REQUIRE: 'Array with permissions ids is required',
  PREMISSION_IDS_INTEGER: 'Permissions ids must be an array of integers',
  PERMISSION_IDS_MIN_LENGTH: `Array with permission ids must contain at least ${GroupValidationRule.PERMISSION_IDS_MIN_LENGTH} permission`,
  USER_IDS_REQUIRE: 'Array with users ids is required',
  USER_IDS_INTEGER: 'Users ids must be an array integers',
} as const;
export { GroupValidationMessage };
```

Файл shared/src/common/enums/group/group-validation-rule.enum.ts

```
const GroupValidationRule = {
  NAME_MIN_LENGTH: 2,
  NAME_MAX_LENGTH: 20,
  PERMISSION_IDS_MIN_LENGTH: 1,
} as const;
export { GroupValidationRule };
```

Файл shared/src/common/enums/http/http-code.enum.ts

```
enum HttpCode {
  OK = 200,
  CREATED = 201,
  BAD_REQUEST = 400,
  UNAUTHORIZED = 401,
  NOT_FOUND = 404,
  INTERNAL_SERVER_ERROR = 500,
  NO_CONTENT = 204,
  FORBIDDEN = 403,
  CONFLICT = 409,
}
export { HttpCode };
```

Файл shared/src/common/enums/http/http-header.enum.ts

```
enum HttpHeader {
  CONTENT_TYPE = 'content-type',
  AUTHORIZATION = 'authorization',
}
export { HttpHeader };
```

Файл shared/src/common/enums/http/http-method.enum.ts

```
enum HttpMethod {
  DELETE = 'DELETE',
  GET = 'GET',
  POST = 'POST',
  PUT = 'PUT',
  PATCH = 'PATCH',
}
export { HttpMethod };
```

Файл shared/src/common/enums/http/http-status-message.enum.ts

```
enum HttpStatusMessage {
  OK = 'OK',
  CREATED = 'Created',
  BAD_REQUEST = 'Bad Request',
```

```
  UNAUTHORIZED = 'Unauthorized',
  NOT_FOUND = 'Not Found',
  INTERNAL_SERVER_ERROR = 'Internal
Server Error',
  NO_CONTENT = 'No Content',
  FORBIDDEN = 'Forbidden',
}
export { HttpStatusMessage };
```

Файл shared/src/common/enums/interview/interview-status.enum.ts

```
enum InterviewStatus {
  PENDING = 'pending',
  IN_PROGRESS = 'in-progress',
  COMPLETED = 'completed',
  REJECTED = 'rejected',
  NEW = 'new',
  CANCELED = 'canceled',
}
export { InterviewStatus };
```

Файл shared/src/common/enums/interview/interview-validation-message.enum.ts

```
enum InterviewValidationMessage {
  INTERVIEWER_ID_INTEGER = 'Interviewer
id must be of type integer',
  STATUS_REQUIRE = 'Status is required',
  STATUS_STRING = 'Status must be of type
string',
  STATUS_VALID = 'Status must be one of the
following: Pending, New, In Progress, Completed,
Rejected, Canceled',
  INTERVIEWEE_ID_INTEGER = 'Interviewee
id must be of type integer',
  INTERVIEWEE_ID_REQUIRE = 'Interviewee
id is required',
  CATEGORY_ID_INTEGER = 'Category id must
be of type integer',
  CATEGORY_ID_REQUIRE = 'Category id id
required',
  NOTE_STRING = 'Note must be of type string',
  NOTE_REQUIRE = 'Note is required',
}
export { InterviewValidationMessage };
```

Файл shared/src/common/enums/mentees-to-mentors/mentees-to-mentors-status.enum.ts

```
enum MenteesToMentorsStatus {
  IN_PROGRESS = 'in-progress',
  COMPLETED = 'completed',
}
export { MenteesToMentorsStatus };
```

Файл shared/src/common/enums/mentor/mentor-validation-message.enum.ts

```
enum MentorValidationMessage {
  COURSE_ID_REQUIRE = 'Course id is
required',
  COURSE_ID_INTEGER = 'Course id must be of
type integer',
  USER_ID_REQUIRE = 'User id is required',
  USER_ID_INTEGER = 'User id must be of type
integer',
}
export { MentorValidationMessage };
```

Файл shared/src/common/enums/pagination/pagination-default-value.enum.ts

```
enum PaginationDefaultValue {
  DEFAULT_PAGE = 1,
  DEFAULT_COUNT = 5,
  DEFAULT_COUNT_BY_10 = 10,
  DEFAULT_COUNT_BY_8 = 8,
  DEFAULT_COUNT_BY_20 = 20,
}
export { PaginationDefaultValue };
```

Файл shared/src/common/enums/pagination/pagination-validation-message.enum.ts

```
import { PaginationValidationRule } from
'./pagination-validation-rule.enum';
const PaginationValidationMessage = {
  MIN_PAGE: `Page number should be not less
than ${PaginationValidationRule.MIN_PAGE}`,
  MIN_COUNT: `Count number should be not less
than
${PaginationValidationRule.MIN_COUNT}`,
} as const;
export { PaginationValidationMessage };
```

Файл shared/src/common/enums/pagination/pagination-validation-rule.enum.ts

```
enum PaginationValidationRule {
  MIN_PAGE = 1,
  MIN_COUNT = 1,
}
export { PaginationValidationRule };
```

Файл shared/src/common/enums/permissions/permission-key.enum.ts

```
enum PermissionKey {
  MANAGE_UAM = 'manage_uam',
  MANAGE_INTERVIEWS =
'manage_interviews',
  MANAGE_INTERVIEW = 'manage_interview',
  MANAGE_CATEGORIES =
'manage_categories',
```

```
  MANAGE_MENTORING =
'manage_mentoring',
```

Файл shared/src/common/enums/socket/socket-event.enum.ts

```
enum SocketEvent {
  CHAT_JOIN_ROOM = 'chat-join-room',
  CHAT_LEAVE_ROOM = 'chat-leave-room',
  CHAT_CREATE_MESSAGE =
'chat-create-message',
  CHAT_ADD_MESSAGE = 'chat-add-message',
  CONNECTION = 'connection',
}
export { SocketEvent };
```

Файл shared/src/common/enums/socket/socket-namespace.enum.ts

```
enum SocketNamespace {
  CHAT = '/chat',
}
export { SocketNamespace };
```

Файл shared/src/common/enums/sort/sort-order.enum.ts

```
enum SortOrder {
  ASC = 'asc',
  DESC = 'desc',
}
export { SortOrder };
```

Файл shared/src/common/enums/stripe/payment-unit.enum.ts

```
enum PaymentUnit {
  CENTS_IN_ONE_DOLLAR = 100,
}
export { PaymentUnit };
```

Файл shared/src/common/enums/task/task-status.enum.ts

```
enum TaskStatus {
  UNCOMPLETED = 'uncompleted',
  PENDING = 'pending',
  REJECTED = 'rejected',
  COMPLETED = 'completed',
}
export { TaskStatus };
```

Файл shared/src/common/enums/task-note/task-note-validation-message.enum.ts

```
enum TaskNoteValidationMessage {
  MESSAGE_EMPTY = 'Message cannot be
empty.',
  STATUS_EMPTY = 'Status cannot be empty.',
}
export { TaskNoteValidationMessage };
```

Файл shared/src/common/enums/transaction/transaction-status.enum.ts

```
enum TransactionStatus {
  PENDING = 'pending',
  HOLD = 'hold',
  FULFILLED = 'fulfilled',
  REJECTED = 'rejected',
}
export { TransactionStatus };
```

Файл shared/src/common/enums/transaction/transaction-type.enum.ts

```
enum TransactionType {
  SPENDING = 'spending',
  INCOME = 'income',
}
export { TransactionType };
```

Файл shared/src/common/enums/user/user-age.enum.ts

```
enum UserAge {
  MIN = 13,
  MAX = 125,
}
export { UserAge };
```

Файл shared/src/common/enums/user/user-gender.enum.ts

```
enum UserGender {
  MALE = 'male',
  FEMALE = 'female',
  OTHER = 'other',
}
export { UserGender };
```

Файл shared/src/common/enums/user/user-validation-message.enum.ts

```
import { UserValidationRule } from
'./user-validation-rule.enum';
const UserValidationMessage = {
  NAME_REQUIRE: 'Full name is required',
  NAME_MIN_LENGTH: `Full name must be at
least
```

```
${UserValidationRule.NAME_MIN_LENGTH}
characters long`,
  NAME_MAX_LENGTH: `Full name must be at
most
${UserValidationRule.NAME_MAX_LENGTH}
characters long`,
  NAME_WRONG: 'Full name must consist of
alphabetic characters',
  NAME_STRING: 'Full name must be of type
string',
  EMAIL_REQUIRE: 'Email is required',
  EMAIL_WRONG: 'Email is invalid',
  EMAIL_MIN_LENGTH: `Email must be at least
${UserValidationRule.EMAIL_MIN_LENGTH}
characters long`,
  EMAIL_MAX_LENGTH: `Email must be at
most
${UserValidationRule.EMAIL_MAX_LENGTH}
characters long`,
  EMAIL_STRING: 'Email must be of type string',
  PASSWORD_REQUIRE: 'Password is required',
  PASSWORD_MIN_LENGTH: `Password must
be at least
${UserValidationRule.PASSWORD_MIN_LENG
TH} characters long`,
  PASSWORD_MAX_LENGTH: `Password must
be at most
${UserValidationRule.PASSWORD_MAX_LEN
GTH} characters long`,
  PASSWORD_STRING: 'Password must be of
type string',
} as const;
export { UserValidationMessage };
```

Файл shared/src/common/enums/user/user-validation-rule.enum.ts

```
const UserValidationRule = {
  EMAIL_MIN_LENGTH: 5,
  EMAIL_MAX_LENGTH: 60,
  PASSWORD_MIN_LENGTH: 8,
  PASSWORD_MAX_LENGTH: 32,
  NAME_MIN_LENGTH: 3,
  NAME_MAX_LENGTH: 60,
  NAME_PATTERN: /^[ A-Za-z-'`]*$/,
} as const;
export { UserValidationRule };
```

Файл shared/src/common/enums/user-details/user-details-validation-message.enum.ts

```
import { UserDetailsValidationRule } from
'./user-details-validation-rule.enum';
const UserDetailsValidationMessage = {
  FULL_NAME_REQUIRE: 'Full name is
required',
  FULL_NAME_MIN_LENGTH: `Full name
must be at least
${UserDetailsValidationRule.FULL_NAME_MIN
_LENGTH} characters long`,
  FULL_NAME_MAX_LENGTH: `Full name
must be at most
${UserDetailsValidationRule.FULL_NAME_MA
X_LENGTH} characters long`,
  FULL_NAME_WRONG: 'Full name must
consist of alphabetic characters',
  FULL_NAME_STRING: 'Full name must be of
type string',
  GENDER_REQUIRE: 'Gender is required',
  GENDER_STRING: 'Gender must be of type
string',
  TELEGRAM_USERNAME_MIN_LENGTH:
`Telegram username must be at least
${UserDetailsValidationRule.TELEGRAM_USE
RNAME_MIN_LENGTH} characters long`,
  TELEGRAM_USERNAME_MAX_LENGTH:
`Telegram username must be at most
${UserDetailsValidationRule.TELEGRAM_USE
RNAME_MAX_LENGTH} characters long`,
  TELEGRAM_USERNAME_WRONG:
   'Telegram username could consist of alphabetic
characters, digits and underscores',
  TELEGRAM_USERNAME_STRING:
'Telegram username must be of type string',
} as const;
export { UserDetailsValidationMessage };
```

Файл shared/src/common/enums/user-details/user-details-validation-rule.enum.ts

```
const UserDetailsValidationRule = {
  FULL_NAME_MIN_LENGTH: 2,
  FULL_NAME_MAX_LENGTH: 40,
  FULL_NAME_PATTERN: /^[ A-Za-z-'`]*$/,
  TELEGRAM_USERNAME_MIN_LENGTH: 5,
  TELEGRAM_USERNAME_MAX_LENGTH:
32,
  TELEGRAM_USERNAME_PATTERN:
/^[A-Za-z0-9_]*$/,
} as const;
export { UserDetailsValidationRule };
```

Файл shared/src/common/types/billing/billing-replenish-params-dto.type.ts

```
import { BillingReplenishToken } from
'./billing-replenish-token.type';
type BillingReplenishParamsDto = {
  amountOfMoneyToReplenish: number;
  token: BillingReplenishToken;
};
export { type BillingReplenishParamsDto };
```

Файл shared/src/common/types/billing/billing-replenish-token.type.ts

```
type BillingReplenishToken = {
  id: string;
};
export { type BillingReplenishToken };
```

Файл shared/src/common/types/chat-message/chat-get-all-messages-request-dto.type.ts

```
type ChatGetAllMessagesRequestDto = {
  chatId: string;
};
export { type ChatGetAllMessagesRequestDto };
```

Файл shared/src/common/types/chat-message/chat-get-last-messages-request-dto.type.ts

```
type ChatGetLastMessagesRequestDto = {
  userId: number;
  chatOpponentId: number;
};
export { type ChatGetLastMessagesRequestDto };
```

Файл shared/src/common/types/chat-message/chat-message-create-request-body-dto.type.ts

```
type ChatMessageCreateRequestBodyDto = {
  message: string;
  receiverId: number;
  chatId: string | null;
};
export { type
ChatMessageCreateRequestBodyDto };
```

Файл shared/src/common/types/chat-message/chat-message-create-request-dto.type.ts

```
type ChatMessageCreateRequestDto = {
  receiverId: number;
  senderId: number;
  message: string;
  chatId: string | null;
};
export { type ChatMessageCreateRequestDto };
```

Файл shared/src/common/types/chat-message/chat-message-create-request-with-status-dto.type.ts

```
import { ChatMessageStatus } from
'~/common/enums/enums';
import { ChatMessageCreateRequestDto } from
'./chat-message-create-request-dto.type';
type ChatMessageCreateRequestWithStatusDto =
ChatMessageCreateRequestDto & {
  status: ChatMessageStatus;
};
export { type
ChatMessageCreateRequestWithStatusDto };
```

Файл shared/src/common/types/chat-message/chat-message-filtering-dto.type.ts

```
type ChatMessageFilteringDto = {
  fullName: string;
};
export { ChatMessageFilteringDto };
```

Файл shared/src/common/types/chat-message/chat-message-get-all-item-response-dto.type.ts

```
import { ChatMessageStatus } from
'~/common/enums/enums';
import { UsersGetResponseDto } from
'~/common/types/types';
type ChatMessageGetAllItemResponseDto = {
  id: number;
  receiver: UsersGetResponseDto;
  sender: UsersGetResponseDto;
  message: string;
  createdAt: string;
  chatId: string;
  status: ChatMessageStatus;
};
export { type
ChatMessageGetAllItemResponseDto };
```

Файл shared/src/common/types/chat-message/chat-message-get-all-last-response-dto.type.ts

```
import { ChatMessageGetAllItemResponseDto }
from
'./chat-message-get-all-item-response-dto.type';
type ChatMessageGetAllLastResponseDto = {
  items: ChatMessageGetAllItemResponseDto[];
};
export { type
ChatMessageGetAllLastResponseDto };
```

Файл shared/src/common/types/chat-message/chat-message-get-all-last-with-empty-chats-dto.type.ts

```
import { ChatMessageGetAllItemResponseDto }
from
'./chat-message-get-all-item-response-dto.type';
import { ChatMessageGetEmptyChatDto } from
'./chat-message-get-empty-chat-dto.type';
type ChatMessageGetAllLastWithEmptyChatsDto
= {
  emptyChats: ChatMessageGetEmptyChatDto[];
  items: ChatMessageGetAllItemResponseDto[];
};
```

```
export { type
ChatMessageGetAllLastWithEmptyChatsDto };
```

Файл shared/src/common/types/chat-message/chat-message-get-all-request-params-dto.type.ts

```
type ChatMessageGetAllRequestParamsDto = {
  id: string;
};
export { type
ChatMessageGetAllRequestParamsDto };
```

Файл shared/src/common/types/chat-message/chat-message-get-all-response-dto.type.ts

```
import { UsersGetResponseDto } from
'~/common/types/types';
import { ChatMessageGetAllItemResponseDto }
from
'./chat-message-get-all-item-response-dto.type';

type ChatMessageGetAllResponseDto = {
  items: ChatMessageGetAllItemResponseDto[];
  chatId: string;
  chatOpponent: UsersGetResponseDto;
};
export { type ChatMessageGetAllResponseDto };
```

Файл shared/src/common/types/chat-message/chat-message-get-empty-chat-dto.type.ts

```
import { UsersGetResponseDto } from
'~/common/types/types';
type ChatMessageGetEmptyChatDto = {
  receiver: UsersGetResponseDto;
  chatId: string;
};
export { type ChatMessageGetEmptyChatDto };
```

Файл shared/src/common/types/chat-message/chat-message-read-params.type.ts

```
type ChatMessageReadParams = {
  id: string;
};
export { type ChatMessageReadParams };
```

Фалй shared/src/common/types/course/course-check-is-mentor-for-mentee-request-params-dto.type.ts

```
type
CourseCheckIsMentorForMenteeRequestParamsD
to = {
  courseId: number;
  menteeId: number;
};
export { type
CourseCheckIsMentorForMenteeRequestParamsD
to };
```

Файл shared/src/common/types/course/course-check-is-mentor-request-params-dto.type.ts

```
type CourseCheckIsMentorRequestParamsDto = {
  id: number;
};
export { type
CourseCheckIsMentorRequestParamsDto };
```

Файл shared/src/common/types/course/course-create-request-dto.type.ts

```
type CourseCreateRequestDto = {
  url: string;
};
export { CourseCreateRequestDto };
```

Файл shared/src/common/types/course/course-filtering-dto.type.ts

```
type CourseFilteringDto = {
  categoryKey: string;
  title: string;
};
export { CourseFilteringDto };
```

Файл shared/src/common/types/course/course-filtering-with-pagination-dto.type.ts

```
import { EntityPaginationRequestQueryDto }
from
'~/common/types/pagination/entity-pagination-req
uest-query-dto.type';

type CourseFilteringWithPaginationDto = {
  categoryKey: string;
  title: string;
} & EntityPaginationRequestQueryDto;
export { CourseFilteringWithPaginationDto };
```

Файл shared/src/common/types/course/course-get-mentees-by-mentor-request-dto.type.ts

```
type CourseGetMenteesByMentorRequestDto = {
  courseId: number;
  mentorId: number;
};
export { CourseGetMenteesByMentorRequestDto
};
```

Файл shared/src/common/types/course/course-get-mentoring-dto.type.ts

```
type CourseGetMentoringDto = {
  id: number;
  title: string;
  studentsCount: number;
```

```
};
export { type CourseGetMentoringDto };
```

Файл shared/src/common/types/course/course-get-mentors-request-dto.type.ts

```
import { CourseMentorsFilteringDto } from
'~/common/types/types';
type CourseGetMentorsRequestDto = {
  courseId: number;
  filteringOpts: CourseMentorsFilteringDto;
};
export { CourseGetMentorsRequestDto };
```

Файл shared/src/common/types/course/course-get-request-dto.type.ts

```
type CourseGetRequestParamsDto = {
  id: number;
};
export { type CourseGetRequestParamsDto };
```

Файл shared/src/common/types/course/course-get-response-dto.type.ts

```
import {
  CourseCategoryWithPriceDto,
  VendorGetResponseDto,
} from '~/common/types/types';
type CourseGetResponseDto = {
  id: number;
  title: string;
  description: string;
  url: string;
  imageUrl: string | null;
  category: CourseCategoryWithPriceDto | null;
  vendor: VendorGetResponseDto;
  courseCategoryId: number;
};
export { type CourseGetResponseDto };
```

Файл shared/src/common/types/course/course-mentors-filtering-dto.type.ts

```
type CourseMentorsFilteringDto = {
  mentorName: string;
};
export { CourseMentorsFilteringDto };
```

Файл shared/src/common/types/course/course-select-mentor-request-dto.type.ts

```
type CourseSelectMentorRequestDto = {
  mentorId: number;
  menteeId: number;
};
export { type CourseSelectMentorRequestDto };
```

Файл shared/src/common/types/course/course-select-mentor-request-params-dto.type.ts

```
type CourseSelectMentorRequestParamsDto = {
  id: number;
};
export { type
CourseSelectMentorRequestParamsDto };
```

Файл shared/src/common/types/course/course-update-category-request-dto.type.ts

```
type CourseUpdateCategoryRequestDto = {
  newCategoryId: number;
};
export { type CourseUpdateCategoryRequestDto
};
```

Файл shared/src/common/types/course/course-update-mentoring-dto.type.ts

```
type CourseUpdateMentoringDto = {
  studentsCount: number;
  courseId: number;
};
export { type CourseUpdateMentoringDto };
```

Файл shared/src/common/types/course/course-update-request-params-dto.type.ts

```
type CourseUpdateRequestParamsDto = {
  id: number;
};
export { type CourseUpdateRequestParamsDto };
```

Файл shared/src/common/types/course-category/course-category-get-all-item-response-dto.type.ts

```
import { CategoryGetAllItemResponseDto } from
'./course-category-get-all-item-response-dto.type';
type CategoryGetAllResponseDto = {
  items: CategoryGetAllItemResponseDto[];
};
export { CategoryGetAllResponseDto };
```

Файл shared/src/common/types/course-category/course-category-get-by-id-request-params-dto.type.ts

```
type CourseCategoryGetByIdRequestParamsDto
= {
  id: number;
};
export { type
CourseCategoryGetByIdRequestParamsDto };
```

Файл shared/src/common/types/course-category/course-category-get-response-dto.type.ts

```
type CourseCategoryGetResponseDto = {
  id: number;
  name: string;
  key: string;
};
export { CourseCategoryGetResponseDto };
```

Файл shared/src/common/types/course-category/course-category-with-price-dto.type.ts

```
import {
CourseCategoryPriceGetAllItemResponseDto }
from '../types';
type CourseCategoryWithPriceDto = {
  id: number;
  key: string;
  name: string;
  price:
CourseCategoryPriceGetAllItemResponseDto;
};
export { type CourseCategoryWithPriceDto };
```

Файл

shared/src/common/types/course-category-price/course-category-price-get-all-item-response-dto.type.t

s

```
import { CategoryGetAllItemResponseDto } from
'../types';
type
CourseCategoryPriceGetAllItemResponseDto = {
  id: number;
  category: CategoryGetAllItemResponseDto;
  price: number;
};
export { type
CourseCategoryPriceGetAllItemResponseDto };
```

Файл

shared/src/common/types/course-category-price/course-category-price-get-all-response-dto.type.ts

```
import {
CourseCategoryPriceGetAllItemResponseDto }
from
'./course-category-price-get-all-item-response-dto.
type';
type CourseCategoryPriceGetAllResponseDto = {
  items:
CourseCategoryPriceGetAllItemResponseDto[];
};
export { type
CourseCategoryPriceGetAllResponseDto };
```

Файл shared/src/common/types/course-module/course-module-get-by-id-response-dto.type.ts

```
type CourseModuleGetByIdResponseDto = {
  id: number;
  title: string;
  description: string;
  courseId: number;
  courseTitle: string;
};
export { type CourseModuleGetByIdResponseDto
};
```

Файл shared/src/common/types/course-module/course-module-get-request-params-dto.type.ts

```
type CourseModuleGetRequestParamsDto = {
  courseId: number;
  moduleId: number;
};
export { type
CourseModuleGetRequestParamsDto };
```

Файл shared/src/common/types/course-module/course-modules-get-all-item-response-dto.type.ts

```
type CourseModulesGetAllItemResponseDto = {
  id: number;
  title: string;
  description: string | null;
  courseId: number;
};
export { type
CourseModulesGetAllItemResponseDto };
```

Файл shared/src/common/types/course-module/course-modules-get-all-request-params-dto.type.ts

```
type CourseModulesGetAllRequestParamsDto = {
  courseId: number;
};
export { type CourseModulesGetAllRequestParamsDto };
```

Файл shared/src/common/types/course-module/course-modules-get-all-response-dto.type.ts

```
import { CourseModulesGetAllItemResponseDto } from './course-modules-get-all-item-response-dto.type';
type CourseModulesGetAllResponseDto = {
```

```
  items: CourseModulesGetAllItemResponseDto[];
};
export { CourseModulesGetAllResponseDto };
```

Файл shared/src/common/types/courses-to-mentors/courses-to-mentors-request-dto.type.ts

```
type CoursesToMentorsRequestDto = {
  userId: number;
  courseId: number;
};
export { type CoursesToMentorsRequestDto };
```

Файл shared/src/common/types/courses-to-mentors/courses-to-mentors-response-dto.type.ts

```
type CoursesToMentorsResponseDto = {
  id: number;
  userId: number;
  courseId: number;
};
export { type CoursesToMentorsResponseDto };
```

Файл shared/src/common/types/file/file-get-response-dto.type.ts

```
import { ContentType } from '~/common/enums/enums';
type FileGetResponseDto = {
  id: number;
  url: string;
  contentType: ContentType;
};
export { type FileGetResponseDto };
```

Файл shared/src/common/types/groups/groups-configure-request-dto.type.ts

```
type GroupsConfigureRequestDto = {
  name: string;
  permissionIds: number[];
  userIds?: number[];
};
export { type GroupsConfigureRequestDto };
```

Файл shared/src/common/types/groups/groups-create-request-dto.type.ts

```
type GroupsCreateRequestDto = {
  name: string;
  permissionIds: number[];
  userIds?: number[];
};
export { type GroupsCreateRequestDto };
```

Файл shared/src/common/types/groups/groups-delete-request-dto.types.ts

```
type GroupsDeleteRequestParamDto = {
  id: number;
};
export { type GroupsDeleteRequestParamDto };
```

Файл shared/src/common/types/groups/groups-get-by-id-request-dto.type.ts

```
type GroupsGetByIdRequestDto = {
  id: number;
};
export { type GroupsGetByIdRequestDto };
```

Файл shared/src/common/types/groups/groups-get-by-id-response-dto.type.ts

```
type GroupsGetByIdResponseDto = {
  id: number;
  name: string;
  key: string;
  permissionIds: number[];
  userIds: number[];
};
export { type GroupsGetByIdResponseDto };
```

Файл shared/src/common/types/groups/groups-item-response-dto.type.ts

```
type GroupsItemResponseDto = {
  id: number;
  name: string;
  key: string;
  createdAt: string;
};
export { type GroupsItemResponseDto };
```

Файл shared/src/common/types/groups/groups-update-request-dto.ts

```
type GroupsUpdateRequestDto = {
  name: string;
  permissionIds: number[];
  userIds: number[];
};
export { type GroupsUpdateRequestDto };
```

Файл shared/src/common/types/groups/groups-update-request-params-dto.ts

```
type GroupsUpdateRequestParamsDto = {
  id: number;
};
export { GroupsUpdateRequestParamsDto };
```

Файл shared/src/common/types/groups-to-permissions/groups-to-permissions.ts

```
type GroupsToPermissionsResponseDto = {
  id: number;
  permissionId: number;
  groupId: number;
};
export { type GroupsToPermissionsResponseDto
};
```

Файл shared/src/common/types/http/http-error-dto.type.ts

```
import { HttpCode, HttpStatusMessage } from '~/common/enums/enums';
type HttpErrorDto = {
  statusCode: HttpCode;
  error: HttpStatusMessage;
  message: string;
};
export { type HttpErrorDto };
```

Файл shared/src/common/types/http/http-options.type.ts

```
import { ContentType, HttpMethod } from
'~/common/enums/enums';
type HttpOptions = {
  method: HttpMethod;
  contentType: ContentType;
  payload: BodyInit | null;
  hasAuth?: boolean;
  queryString?: Record<string, unknown>;
  headers?: Record<string, string>;
};
export { type HttpOptions };
```

Файл shared/src/common/types/interview/interviews-by-id-request-params-dto.type.ts

```
type InterviewsByIdRequestParamsDto = {
  id: number;
};
export { type InterviewsByIdRequestParamsDto };
```

Файл shared/src/common/types/interview/interviews-by-id-response-dto.type.ts

```
import { InterviewStatus } from
'~/common/enums/enums';
import { CategoryGetAllItemResponseDto,
UsersGetResponseDto } from '../types';
type InterviewsByIdResponseDto = {
  id: number;
  interviewDate: string;
  status: InterviewStatus;
  interviewee: UsersGetResponseDto;
  interviewer: UsersGetResponseDto | null;
  courseCategory:
CategoryGetAllItemResponseDto;
};
export { type InterviewsByIdResponseDto };
```

Файл shared/src/common/types/interview/interviews-by-interviewee-id-request-dto.type.ts

```
type InterviewsByIntervieweeIdRequestDto = {
  intervieweeUserId: number;
};
export { type InterviewsByIntervieweeIdRequestDto };
```

Файл shared/src/common/types/interview/interviews-create-request-body-dto.type.ts

```
type InterviewsCreateRequestBodyDto = {
  intervieweeUserId: number;
  categoryId: number;
};
export { type InterviewsCreateRequestBodyDto };
```

Файл shared/src/common/types/interview/interviews-get-all-item-response-dto.type.ts

```
import { InterviewStatus } from
'~/common/enums/enums';
import { CategoryGetAllItemResponseDto,
UsersGetResponseDto } from '../types';
type InterviewsGetAllItemResponseDto = {
  id: number;
  interviewDate: string;
  status: InterviewStatus;
  interviewee: UsersGetResponseDto;
  interviewer: UsersGetResponseDto | null;
  courseCategory:
CategoryGetAllItemResponseDto;
};
export { type InterviewsGetAllItemResponseDto
};
```

Файл shared/src/common/types/interview/interviews-get-by-id-request-dto.type.ts

```
type InterviewsGetInterviewersByCategoryRequestDto = {
  categoryId: number;
};
export { type InterviewsGetInterviewersByCategoryRequestDto };
```

Файл shared/src/common/types/interview/interviews-get-interviewer-response-dto.type.ts

```
import { UsersGetResponseDto } from '../types';
type InterviewsGetInterviewerResponseDto = {
  interviewer: UsersGetResponseDto;
};
export { type InterviewsGetInterviewerResponseDto };
```

Файл shared/src/common/types/interview/interviews-get-other-item-response-dto.type.ts

```
import { InterviewStatus } from
'~/common/enums/enums';
import { CategoryGetAllItemResponseDto,
UsersGetResponseDto } from '../types';
type InterviewsGetOtherItemResponseDto = {
  id: number;
  interviewDate: string;
  status: InterviewStatus;
  interviewee: UsersGetResponseDto;
  interviewer: UsersGetResponseDto;
  courseCategory:
CategoryGetAllItemResponseDto;
};
export { type
InterviewsGetOtherItemResponseDto };
```

Файл shared/src/common/types/interview/interviews-get-other-request-dto.type.ts

```
import { EntityPaginationRequestQueryDto } from '../types';
type InterviewsGetOtherRequestDto = {
  interviewId: number;
} & EntityPaginationRequestQueryDto;
export { type InterviewsGetOtherRequestDto };
```

Файл shared/src/common/types/interview/interviews-response-dto.type.ts

```
import { InterviewStatus } from '~/common/enums/enums';
type InterviewsResponseDto = {
  id: number;
  status: InterviewStatus;
  categoryId: number;
  intervieweeUserId: number;
};
export { type InterviewsResponseDto };
```

Файл shared/src/common/types/interview/interviews-update-request-dto.type.ts

```
import { InterviewStatus } from '~/common/enums/enums';
type InterviewsUpdateRequestDto = {
  interviewerUserId: number | null;
  status: InterviewStatus;
  interviewDate: string | null;
};
export { type InterviewsUpdateRequestDto };
```

Файл shared/src/common/types/interview/interviews-update-request-params-dto.type.ts

```
type InterviewsUpdateRequestParamsDto = {
  id: number;
};
export { InterviewsUpdateRequestParamsDto };
```

Файл shared/src/common/types/interview/interviews-update-without-interviewer-request-dto.type.ts

```
import { InterviewStatus } from '~/common/enums/enums';
type InterviewsUpdateWithoutInterviewerRequestDto = {
  status: InterviewStatus;
  interviewDate: string | null;
};
export { type InterviewsUpdateWithoutInterviewerRequestDto };
```

Файл shared/src/common/types/interview-note/interview-note-author-dto.type.ts

```
import { UserDetailsResponseDto } from '../types';
type InterviewNoteAuthor = {
  id: number;
  email: string;
  userDetails: UserDetailsResponseDto;
};
export { InterviewNoteAuthor };
```

Файл shared/src/common/types/interview-note/interview-note-create-request-dto.type.ts

```
type InterviewNoteCreateRequestDto = {
  note: string;
};
export { type InterviewNoteCreateRequestDto };
```

Файл shared/src/common/types/interview-note/interview-note-create-request-params-dto.type.ts

```
type InterviewNoteCreateRequestParamsDto = {
  id: number;
};
export { type InterviewNoteCreateRequestParamsDto };
```

Файл shared/src/common/types/interview-note/interview-note-get-all-item-response-dto.type.ts

```
import { InterviewNoteAuthor } from './interview-note-author-dto.type';
type InterviewNoteGetAllItemResponseDto = {
  id: number;
  note: string;
  author: InterviewNoteAuthor;
  createdAt: string;
};
export { type InterviewNoteGetAllItemResponseDto };
```

Файл shared/src/common/types/interview-note/interview-note-get-all-request-params-dto.type.ts

```
type InterviewNoteGetAllRequestParamsDto = {
  id: number;
};
export { type InterviewNoteGetAllRequestParamsDto };
```

Файл shared/src/common/types/interview-note/interview-note-get-all-response-dto.type.ts

```
import { InterviewNoteGetAllItemResponseDto } from './interview-note-get-all-item-response-dto.type';
type InterviewNoteGetAllResponseDto = {
  items: InterviewNoteGetAllItemResponseDto[];
};
export { type InterviewNoteGetAllResponseDto };
```

Файл shared/src/common/types/mentees-to-mentors/get-mentor-request-params-dto.type.ts

```
type GetMentorRequestParamsDto = {
  menteeId: number;
  courseId: number;
};
export { GetMentorRequestParamsDto };
```
Файл shared/src/common/types/mentees-to-mentors/mentees-to-mentors-request-dto.type.ts

```
type MenteesToMentorsRequestDto = {
  courseId: number;
  mentorId: number;
  menteeId: number;
};
export { type MenteesToMentorsRequestDto };
```
Файл shared/src/common/types/mentees-to-mentors/mentees-to-mentors-response-dto.type.ts

```
import { MenteesToMentorsStatus } from '~/common/enums/enums';
import { UsersGetResponseDto } from '../types';
type MenteesToMentorsResponseDto = {
  id: number;
  courseId: number;
  mentor: UsersGetResponseDto;
  menteeId: number;
  status: MenteesToMentorsStatus;
};
export { type MenteesToMentorsResponseDto };
```
Файл shared/src/common/types/null/deep-non-nullable.type.ts

```
type DeepNonNullable<T> = {
  [K in keyof T]: DeepNonNullable<NonNullable<T[K]>>;
};
export { type DeepNonNullable };
```
Файл shared/src/common/types/null/null.ts

```
export { type DeepNonNullable } from './deep-non-nullable.type';
```
Файл shared/src/common/types/pagination/entity-pagination-request-query-dto.type.ts

```
type EntityPaginationRequestQueryDto = {
  page: number;
  count: number;
};
export { EntityPaginationRequestQueryDto };
```
Файл shared/src/common/types/pagination/entity-pagination.type.ts

```
type EntityPagination<T> = {
  items: T[];
  total: number;
};
export { EntityPagination };
```
Файл shared/src/common/types/permission/check-permission-type.type.ts

```
type CheckPermisssionType = 'every' | 'oneOf';
export { type CheckPermisssionType };
```
Файл shared/src/common/types/permission/permissions-get-all-item-response-dto.type.ts

```
type PermissionsGetAllItemResponseDto = {
  id: number;
  key: string;
  name: string;
};
export { type PermissionsGetAllItemResponseDto };
```
Файл shared/src/common/types/permission/permissions-get-all-response-dto.type.ts

```
import { PermissionsGetAllItemResponseDto } from './permissions-get-all-item-response-dto.type';
```

```
type PermissionsGetAllResponseDto = {
  items: PermissionsGetAllItemResponseDto[];
};
export { type PermissionsGetAllResponseDto };
```

Файл shared/src/common/types/task/task-by-id-request-params-dto.type.ts

```
type TaskByIdRequestParamsDto = {
  taskId: number;
};
export { type TaskByIdRequestParamsDto };
```

Файл shared/src/common/types/task/task-create-request-dto.type.ts

```
type TaskCreateRequestDto = {
  moduleId: number;
  menteesToMentorsId: number;
};
export { type TaskCreateRequestDto };
```

Файл shared/src/common/types/task/task-get-by-mentee-id-and-module-id-request-dto.type.ts

```
type TaskGetByMenteeIdAndModuleId = {
  menteeId: number;
  moduleId: number;
};
export { type TaskGetByMenteeIdAndModuleId };
```

Файл shared/src/common/types/task/task-get-by-mentee-id-course-id-module-id-request-dto.type.ts

```
type TaskGetByMenteeIdCourseIdModuleIdRequestDto = {
  menteeId: number;
  courseId: number;
  moduleId: number;
};
export { type TaskGetByMenteeIdCourseIdModuleIdRequestDto };
```

Файл shared/src/common/types/task/task-get-item-response-dto.type.ts

```
import { TaskStatus } from '~/common/enums/enums';
type TaskGetItemReponseDto = {
  id: number;
  menteesToMentorsId: number;
  moduleId: number;
  status: TaskStatus;
};
export { type TaskGetItemReponseDto };
```

Файл shared/src/common/types/task/task-manipulate-request-arguments-dto.type.ts

```
import { TaskStatus } from
'~/common/enums/enums';
type TaskManipulateRequestArgumentsDto = {
  taskId: number;
  note: string;
  authorId: number;
  status: TaskStatus;
};
export { type
TaskManipulateRequestArgumentsDto };
```

Файл shared/src/common/types/task/task-with-module-response-dto.type.ts

```
import { TaskStatus } from '~/common/enums/enums';
import { CourseModuleGetByIdResponseDto } from '../types';
type TaskWithModuleResponseDto = {
  id: number;
  menteesToMentorsId: number;
  moduleId: number;
  status: TaskStatus;
  module: CourseModuleGetByIdResponseDto;
```

```
};
export { type TaskWithModuleResponseDto };
```

Файл shared/src/common/types/task/tasks-get-by-course-id-and-mentee-id-request-dto.type.ts

```
type TasksGetByCourseIdAndMenteeIdRequestDto = {
  courseId: number;
  menteeId: number;
};
export { type TasksGetByCourseIdAndMenteeIdRequestDto };
```

Файл shared/src/common/types/task-note/task-note-get-item-response-dto.type.ts

```
import { TaskStatus } from
'~/common/enums/enums';
import { UsersGetResponseDto } from '../types';
type TaskNoteGetItemResponseDto = {
  id: number;
  author: UsersGetResponseDto;
  note: string;
  createdAt: string;
  status: TaskStatus;
};
export { type TaskNoteGetItemResponseDto };
```

Файл shared/src/common/types/task-note/task-note-manipulate-request-body-dto.type.ts

```
import { TaskStatus } from '~/common/enums/enums';
type TaskNoteManipulateRequestBodyDto = {
  note: string;
  status: TaskStatus;
};
export { type TaskNoteManipulateRequestBodyDto };
```

Файл shared/src/common/types/transaction/transaction-create-arguments-dto.type.ts

```
type TransactionCreateArgumentsDto = {
  senderId: number;
  receiverId: number;
  amount: number;
};
export { type TransactionCreateArgumentsDto };
```

Файл shared/src/common/types/transaction/transaction-get-all-item-response-dto.type.ts

```
import { TransactionStatus } from '~/common/enums/enums';
import { UsersGetResponseDto } from '../types';
type TransactionGetAllItemResponseDto = {
  id: number;
  createdAt: string;
  amount: number;
  status: TransactionStatus;
  sender: UsersGetResponseDto;
  receiver: UsersGetResponseDto;
};
export { type TransactionGetAllItemResponseDto };
```

Файл shared/src/common/types/transaction/transaction-update-status-dto-type.ts

```
import { TransactionStatus } from '~/common/enums/enums';
type TransactionUpdateStatusDto = {
  transactionId: number;
  newStatus: TransactionStatus;
};
export { type TransactionUpdateStatusDto };
```

Файл shared/src/common/types/udemy/udemy-course-get-response-dto.type.ts

```
type UdemyCourseGetResponseDto = {
  id: number;
```

```
  title: string;
  description: string;
  url: string;
  image_480x270: string;
};
export { type UdemyCourseGetResponseDto };
```
Файл shared/src/common/types/udemy/udemy-courses-get-response-dto.type.ts

```
import { UdemyCourseGetResponseDto } from './udemy-course-get-response-dto.type';
type UdemyCoursesGetResponseDto = {
  count: number;
  results: UdemyCourseGetResponseDto[];
};
export { type UdemyCoursesGetResponseDto };
```
Файл shared/src/common/types/udemy/udemy-module-get-response-dto.type.ts

```
type UdemyModuleGetResponseDto = {
  title: string;
  sort_order: number;
  description: string | null;
  _class: string;
};
export { type UdemyModuleGetResponseDto };
```
Файл shared/src/common/types/udemy/udemy-modules-get-response-dto.type.ts

```
import { UdemyModuleGetResponseDto } from './udemy-module-get-response-dto.type';
type UdemyModulesGetResponseDto = {
  count: number;
  next: string | null;
  results: UdemyModuleGetResponseDto[];
};
export { type UdemyModulesGetResponseDto };
```
Файл shared/src/common/types/user/user-delete-request-dto.type.ts

```
type UsersDeleteRequestParamsDto = {
  id: number;
};
export { type UsersDeleteRequestParamsDto };
```
Файл shared/src/common/types/user/user-sign-in-request-dto.type.ts

```
type UserSignInRequestDto = {
  email: string;
  password: string;
};
export { type UserSignInRequestDto };
```
Файл shared/src/common/types/user/user-sign-in-response-dto.type.ts

```
import { UserWithPermissions } from './user-with-permissions.type';
type UserSignInResponseDto = {
  token: string;
  user: UserWithPermissions;
};
export { type UserSignInResponseDto };
```
Файл shared/src/common/types/user/user-sign-up-request-dto.type.ts

```
type UserSignUpRequestDto = {
  email: string;
  password: string;
  fullName: string;
```

```
};
export { type UserSignUpRequestDto };
```
Файл shared/src/common/types/user/user-sign-up-response-dto.type.ts

```
import { UserWithPermissions } from './user-with-permissions.type';
type UserSignUpResponseDto = {
  token: string;
  user: UserWithPermissions;
};
export { type UserSignUpResponseDto };
```
Файл shared/src/common/types/user/user-with-permissions.type.ts

```
import {
  PermissionsGetAllItemResponseDto,
  UserDetailsResponseDto,
} from '~/common/types/types';
type UserWithPermissions = {
  id: number;
  email: string;
  createdAt: string;
  userDetails: UserDetailsResponseDto;
  permissions: PermissionsGetAllItemResponseDto[];
};
export { type UserWithPermissions };
```
Файл shared/src/common/types/user/users-basic-info-dto.type.ts

```
type UsersBasicInfoDto = {
  id: number;
  email: string;
  createdAt: string;
};
export { type UsersBasicInfoDto };
```
Файл shared/src/common/types/user/users-get-response-dto.type.ts

```
import { UserDetailsResponseDto } from '~/common/types/types';
type UsersGetResponseDto = {
  id: number;
  email: string;
  createdAt: string;
  userDetails: UserDetailsResponseDto;
};
export { type UsersGetResponseDto };
```
Файл shared/src/common/types/user-details/user-details-item-response-dto.type.ts

```
import { UserGender } from
'~/common/enums/enums';
import { FileGetResponseDto } from '../types';
type UserDetailsItemResponseDto = {
  avatar: FileGetResponseDto | null;
  createdAt: string;
  dateOfBirth: string | null;
  fullName: string;
  gender: UserGender | null;
  id: number;
  userId: number;
  telegramUsername: string | null;
};
export { type UserDetailsItemResponseDto };
```
Файл shared/src/common/types/user-details/user-details-response-dto.type.ts

```
import { UserGender } from
'~/common/enums/enums';
import { FileGetResponseDto } from '../types';
type UserDetailsResponseDto = {
  id: number;
  fullName: string;
  gender: UserGender | null;
  avatar: FileGetResponseDto | null;
  dateOfBirth: string | null;
  telegramUsername: string | null;
  userId: number;
};
export { type UserDetailsResponseDto };
```
Файл shared/src/common/types/user-details/user-details-update-avatar-request-params-dto.type.ts

```
type UserDetailsUpdateAvatarRequestParamsDto = {
  userId: number;
};
export { type UserDetailsUpdateAvatarRequestParamsDto };
```

Файл shared/src/common/types/user-details/user-details-update-info-request-dto.type.ts

```
import { UserGender } from '~/common/enums/enums';
type UserDetailsUpdateInfoRequestDto = {
  fullName: string;
  gender: UserGender | null;
  dateOfBirth: string | null;
  telegramUsername: string | null;
};
export { type UserDetailsUpdateInfoRequestDto };
```

Файл shared/src/common/types/users-to-groups/users-to-groups.ts

```
type UsersToGroupsResponseDto = {
  id: number;
  userId: number;
  groupId: number;
};
export { type UsersToGroupsResponseDto };
```

Файл shared/src/common/types/validation/validation-schema.type.ts

```
export { type Schema as ValidationSchema } from 'joi';
```

Файл shared/src/common/types/vendor/vendor-get-response-dto.type.ts

```
type VendorGetResponseDto = {
  id: number;
  name: string;
  key: string;
};
export { VendorGetResponseDto };
```

Файл shared/src/exceptions/http-error/http-error.exception.ts

```
import { HttpCode } from
'~/common/enums/enums';
const DEFAULT_SERVER_ERROR = 'Network
Error';
type Constructor = {
  message?: string;
  status?: number;
  cause?: unknown;
};
class HttpError extends Error {
  public status: HttpCode;
  public constructor({
    message = DEFAULT_SERVER_ERROR,
    status =
HttpCode.INTERNAL_SERVER_ERROR,
    cause,
  }: Constructor = {}) {
    super(message);
    this.status = status;
    this.message = message;
    this.cause = cause as Error;
  }
}
export { HttpError };
```

Файл shared/src/exceptions/invalid-credentials-error/invalid-credentials-error.exception.ts

```
import { CustomExceptionName, ExceptionMessage } from '~/common/enums/enums';
class InvalidCredentialsError extends Error {
  public constructor(message = ExceptionMessage.BAD_CREDENTIALS) {
    super(message);
    this.name = CustomExceptionName.INVALID_CREDENTIALS;
  }
}
export { InvalidCredentialsError };
```

Файл shared/src/helpers/date/check-is-current-year/check-is-current-year.helper.ts

```
const checkIsCurrentYear = (date: Date): boolean => {
  const currentYear = new Date().getFullYear();
  const yearToCompare = date.getFullYear();
```

```
  return yearToCompare === currentYear;
};
export { checkIsCurrentYear };
```

Файл shared/src/helpers/date/check-is-today/check-is-today.ts

```
import { isToday } from 'date-fns';
const checkIsToday = (date: Date): boolean => {
  return isToday(date);
};
export { checkIsToday };
```

Файл shared/src/helpers/date/get-formatted-date/get-formatted-date.ts

```
import { format, formatDistance } from 'date-fns';
type FormatDate =
  | 'distance'
  | 'yyyy-MM-dd'
  | 'dd-MM-yyyy'
  | 'dd.MM.yyyy'
  | 'dd MMM yyyy'
  | 'HH:mm'
  | 'dd MMM'
  | 'HH:mm, dd.MM'
  | 'kk:mm, dd/MM/yyyy'
  | 'HH:mm dd.MM.yyyy';
const getFormattedDate = (date: string,
formatDate: FormatDate): string => {
  switch (formatDate) {
    case 'distance': {
      return formatDistance(new Date(date), new
Date());
    }
    case 'HH:mm':
    case 'dd MMM':
    case 'HH:mm, dd.MM':
    case 'yyyy-MM-dd':
    case 'dd-MM-yyyy':
    case 'dd.MM.yyyy':
    case 'dd MMM yyyy':
    case 'HH:mm dd.MM.yyyy':
    case 'kk:mm, dd/MM/yyyy': {
      return format(new Date(date), formatDate);
    }
  }
};
export { getFormattedDate };
```

Файл shared/src/helpers/date/subtract-years/subtract-years.helper.ts

```
import { subYears } from 'date-fns';
const subtractYears = (date: Date, amount: number): Date => {
  return subYears(date, amount);
};
export { subtractYears };
```

Файл shared/src/helpers/debounce/debounce/debounce.helper.ts

```
import { debounce as debounceCallback } from 'debounce';
const debounce = (
  cb: () => void,
  timeout: number,
): (() => void) & { clear(): void } => debounceCallback(cb, timeout);
export { debounce };
```

Файл shared/src/helpers/permissions/check-has-permission/check-has-permission.helper.ts

```
import { PermissionKey } from
'~/common/enums/enums';
import {
  CheckPermisssionType,
  PermissionsGetAllItemResponseDto,
} from '~/common/types/types';
type Args = {
  permissionKeys: PermissionKey[];
  userPermissions:
PermissionsGetAllItemResponseDto[];
  checkMode?: CheckPermisssionType;
};
const checkHasPermission = ({
  permissionKeys,
  userPermissions,
  checkMode = 'oneOf',
}: Args): boolean => {
  const isRequiringPermissions =
Boolean(permissionKeys.length);
  const userPermissionKeys =
userPermissions.map((item) => item.key);
  if (!isRequiringPermissions) {
    return true;
  }
```

```
  switch (checkMode) {
    case 'every': {
      return
permissionKeys.every((pagePermission) => {
        return
userPermissionKeys.includes(pagePermission);
      });
    }
    case 'oneOf': {
      return permissionKeys.some((pagePermission)
=> {
        return
userPermissionKeys.includes(pagePermission);
      });
    }
  }
};
export { checkHasPermission };
```

Файл shared/src/helpers/sanitize/sanitize-html/sanitize-html.helper.ts

```
import sanitizeHtml from 'sanitize-html';
const sanitizeHTML = (html: string): string => {
  return sanitizeHtml(html, {});
};
export { sanitizeHTML };
```

Файл shared/src/helpers/string/change-string-case/change-string-case.helper.ts

```
import { camelCase, Options, paramCase,
snakeCase } from 'change-case';
import { StringCase } from
'~/common/enums/case/case.enum';
const caseTypeToFn: Record<StringCase, typeof
paramCase> = {
  [StringCase.KEBAB_CASE]: paramCase,
  [StringCase.SNAKE_CASE]: snakeCase,
  [StringCase.CAMEL_CASE]: camelCase,
};
type Args = {
  stringToChange: string;
  caseType: StringCase;
  options?: Options;
};
const changeStringCase = (args: Args): string => {
  const getChangedStringCase =
caseTypeToFn[args.caseType];
  return
getChangedStringCase(args.stringToChange,
args.options);
};
export { changeStringCase };
```

Файл shared/src/helpers/typescript/get-name-of/get-name-of.ts

```
const getNameOf = <T>(name: keyof T): string => name.toString();
export { getNameOf };
```

Файл shared/src/helpers/uuid/create-uuid/create-uuid.helper.ts

```
import { v4 as uuid4 } from 'uuid';
const createUuid = (): string => {
  return uuid4();
};

export { createUuid };
```

Файл shared/src/validation-schemas/billing/billing-replenish-params.validation-schema.ts

```
import * as Joi from 'joi';
import { BillingReplenishParamsDto } from '~/common/types/types';
import { getNameOf } from '~/helpers/helpers';
const billingReplenishParams = Joi.object({
  [getNameOf<BillingReplenishParamsDto>('amountOfMoneyToReplenish')]:
    Joi.number().positive().required(),
  [getNameOf<BillingReplenishParamsDto>('token')]: Joi.object({
    id: Joi.string().required(),
  }).required(),
});
export { billingReplenishParams };
```

Файл shared/src/validation-schemas/chat-message/chat-message-create-params.validation-schema.ts

```
import * as Joi from 'joi';
import { ChatValidationMessage } from
'~/common/enums/enums';
import { ChatMessageCreateRequestBodyDto }
from '~/common/types/types';
import { getNameOf } from '~/helpers/helpers';
const chatMessageCreateArguments = Joi.object({
```

```
[getNameOf<ChatMessageCreateRequestBodyDto>('message')]: Joi.string()
    .required()
    .messages({
      'string.base': ChatValidationMessage.MESSAGE_STRING,
      'any.required': ChatValidationMessage.MESSAGE_REQUIRE,
    }),

[getNameOf<ChatMessageCreateRequestBodyDto>('receiverId')]: Joi.number()
    .integer()
    .required()
    .messages({
      'number.base': ChatValidationMessage.RECEIVER_ID_INTEGER,
      'any.base': ChatValidationMessage.RECEIVER_ID_REQUIRE,
    }),

[getNameOf<ChatMessageCreateRequestBodyDto>('chatId')]: Joi.string().messages(
    {
      'string.base': ChatValidationMessage.CHAT_ID_STRING,
    },
  ),
});
export { chatMessageCreateArguments };
```

Файл shared/src/validation-schemas/chat-message/chat-message-filtering.validation-schema.ts

```
import * as Joi from 'joi';
import { ChatMessageFilteringDto } from '~/common/types/types';
import { getNameOf } from '~/helpers/helpers';
const chatMessageFiltering = Joi.object({
  [getNameOf<ChatMessageFilteringDto>('fullName')]: Joi.string().allow(
    null,
    '',
  ),
});
export { chatMessageFiltering };
```

Файл shared/src/validation-schemas/chat-message/chat-message-get-all.validation-schema.ts

```
import * as Joi from 'joi';
import { ChatMessageGetAllRequestParamsDto } from '~/common/types/types';
import { getNameOf } from '~/helpers/helpers';
const chatMessageGetAllParams = Joi.object({
  [getNameOf<ChatMessageGetAllRequestParamsDto>('id')]: Joi.string().required(),
});
export { chatMessageGetAllParams };
```

Файл shared/src/validation-schemas/chat-message/chat-message-read-params.validation-schema.ts

```
import Joi from 'joi';
import { ChatMessageReadParams } from '~/common/types/types';
import { getNameOf } from '~/helpers/helpers';
const chatMessageReadParams = Joi.object({
  [getNameOf<ChatMessageReadParams>('id')]: Joi.string().uuid().required(),
});
export { chatMessageReadParams };
```

Файл

shared/src/validation-schemas/course/course-check-is-mentor-for-student-params.validation-schema.ts

```
import Joi from 'joi';
import { CourseCheckIsMentorForMenteeRequestParamsDto } from '~/common/types/types';
import { getNameOf } from '~/helpers/helpers';
const courseCheckIsMentorForStudentParams = Joi.object({
  [getNameOf<CourseCheckIsMentorForMenteeRequestParamsDto>('courseId')]:
    Joi.number().integer().required(),
```

```
  [getNameOf<CourseCheckIsMentorForMenteeRequestParamsDto>('menteeId')]:
    Joi.number().integer().required(),
});
export { courseCheckIsMentorForStudentParams };
```

Файл shared/src/validation-schemas/course/course-check-is-mentor-params.validation-schema.ts

```
import * as Joi from 'joi';
import { CourseCheckIsMentorRequestParamsDto } from '~/common/types/types';
import { getNameOf } from '~/helpers/helpers';
const courseCheckIsMentorParams = Joi.object({
  [getNameOf<CourseCheckIsMentorRequestParamsDto>('id')]: Joi.number()
    .integer()
    .required(),
});
export { courseCheckIsMentorParams };
```

Файл shared/src/validation-schemas/course/course-create.validation-schema.ts

```
import * as Joi from 'joi';
import { CourseValidationMessage } from
'~/common/enums/enums';
import { CourseCreateRequestDto } from
'~/common/types/types';
import { getNameOf } from '~/helpers/helpers';
const courseCreate = Joi.object({
  [getNameOf<CourseCreateRequestDto>('url')]:
Joi.string()
    .trim()
    .uri()
    .messages({
      'string.uri':
CourseValidationMessage.INVALID_URL,
      'any.required':
CourseValidationMessage.URL_REQUIRE,
    }),
});
export { courseCreate };
```

Файл shared/src/validation-schemas/course/course-filtering.validation-schema.ts

```
import * as Joi from 'joi';

import {
  PaginationValidationMessage,
  PaginationValidationRule,
} from '~/common/enums/enums';
import {
  CourseFilteringDto,
  EntityPaginationRequestQueryDto,
} from '~/common/types/types';
import { getNameOf } from '~/helpers/helpers';
const courseFiltering = Joi.object({
  [getNameOf<CourseFilteringDto>('title')]: Joi.string().allow(null, ''),
  [getNameOf<CourseFilteringDto>('categoryKey')]: Joi.string().allow(null, ''),
  [getNameOf<EntityPaginationRequestQueryDto>('count')]: Joi.number()
    .integer()
    .min(PaginationValidationRule.MIN_COUNT)
    .messages({
      'number.min': PaginationValidationMessage.MIN_COUNT,
    }),
  [getNameOf<EntityPaginationRequestQueryDto>('page')]: Joi.number()
    .integer()
    .min(PaginationValidationRule.MIN_PAGE)
    .messages({
      'number.min': PaginationValidationMessage.MIN_PAGE,
    }),
});
export { courseFiltering };
```

Файл shared/src/validation-schemas/course/course-get-params.validation-schema.ts

```
import * as Joi from 'joi';
import { CourseGetRequestParamsDto } from '~/common/types/types';
import { getNameOf } from '~/helpers/helpers';
const courseGetParams = Joi.object({
  [getNameOf<CourseGetRequestParamsDto>('id')]: Joi.number()
    .integer()
    .required(),
});
export { courseGetParams };
```

Файл shared/src/validation-schemas/course/course-mentoring-update-count.validation-schema.ts

```
import * as Joi from 'joi';
import {
  CourseValidationMessage,
  CourseValidationRule,
} from '~/common/enums/enums';
import { CourseUpdateMentoringDto } from '~/common/types/types';
import { getNameOf } from '~/helpers/helpers';
const courseMentoringUpdateCount = Joi.object({
  [getNameOf<CourseUpdateMentoringDto>('courseId')]: Joi.number()
    .integer()
    .required(),
  [getNameOf<CourseUpdateMentoringDto>('studentsCount')]: Joi.number()
    .integer()
    .min(CourseValidationRule.STUDENTS_COUNT_MIN_NUMBER)
    .required()
    .label('Students count')
    .messages({
      'number.min': CourseValidationMessage.INVALID_STUDENTS_COUNT_NUMBER,
    }),
});
export { courseMentoringUpdateCount };
```

Файл shared/src/validation-schemas/course/course-mentors-filtering.validation-schema.ts

```
import * as Joi from 'joi';
import { CourseMentorsFilteringDto } from '~/common/types/types';
import { getNameOf } from '~/helpers/helpers';
const courseMentorsFiltering = Joi.object({
  [getNameOf<CourseMentorsFilteringDto>('mentorName')]: Joi.string().allow(
    null,
    '',
  ),
});
export { courseMentorsFiltering };
```

Файл shared/src/validation-schemas/course/course-update-category.validation-schema.ts

```
import * as Joi from 'joi';
import { CourseValidationMessage } from '~/common/enums/enums';
import { CourseUpdateCategoryRequestDto } from '~/common/types/types';
import { getNameOf } from '~/helpers/helpers';
const courseUpdateCategory = Joi.object({
  [getNameOf<CourseUpdateCategoryRequestDto>('newCategoryId')]: Joi.number()
    .integer()
    .required()
    .messages({
      'number.empty': CourseValidationMessage.EMPTY_COURSE_CATEGORY_ID,
```

```
    'number.base': CourseValidationMessage.CATEGORY_INTEGER,
    'any.required': CourseValidationMessage.CATEGORY_REQUIRE,
  }),
});
export { courseUpdateCategory };
```

Файл shared/src/validation-schemas/course/course-update-params.validation-schema.ts

```
import * as Joi from 'joi';
import { CourseUpdateRequestParamsDto } from '~/common/types/types';
import { getNameOf } from '~/helpers/helpers';
const courseUpdateByIdParams = Joi.object({
 [getNameOf<CourseUpdateRequestParamsDto>('id')]: Joi.number()
  .integer()
  .required(),
});
export { courseUpdateByIdParams };
```

Файл

shared/src/validation-schemas/course-categories/course-category-get-by-id-request-params-dto.validat

ion-schema.ts

```
import * as Joi from 'joi';
import { CourseCategoryGetByIdRequestParamsDto } from '~/common/types/types';
import { getNameOf } from '~/helpers/helpers';
const courseCategoryGetByIdParams = Joi.object({
 [getNameOf<CourseCategoryGetByIdRequestParamsDto>('id')]: Joi.number()
  .integer()
  .required(),
});
export { courseCategoryGetByIdParams };
```

Файл shared/src/validation-schemas/course-mentor/course-mentor-create.validation-schema.ts

```
import Joi from 'joi';
import { CourseValidationMessage } from '~/common/enums/enums';
import { CourseSelectMentorRequestDto } from '~/common/types/types';
import { getNameOf } from '~/helpers/helpers';
const courseMentorCreate = Joi.object({
 [getNameOf<CourseSelectMentorRequestDto>('mentorId')]: Joi.number()
  .integer()
  .required()
  .messages({
    'number.base': CourseValidationMessage.MENTOR_ID_INTEGER,
    'any.required': CourseValidationMessage.MENTOR_ID_REQUIRE,
  }),
 [getNameOf<CourseSelectMentorRequestDto>('menteeId')]: Joi.number()
  .integer()
  .required()
  .messages({
    'number.base': CourseValidationMessage.MENTEE_ID_INTEGER,
    'any.required': CourseValidationMessage.MENTEE_ID_REQUIRE,
  }),
});
export { courseMentorCreate };
```

Файл shared/src/validation-schemas/course-module/course-module-get-params.validation-schema.ts

```
import * as Joi from 'joi';
```

```
import { CourseModuleGetRequestParamsDto } from '~/common/types/types';
import { getNameOf } from '~/helpers/helpers';
const courseModuleGetParams = Joi.object({
  [getNameOf<CourseModuleGetRequestParamsDto>('courseId')]: Joi.number()
    .integer()
    .required(),
  [getNameOf<CourseModuleGetRequestParamsDto>('moduleId')]: Joi.number()
    .integer()
    .required(),
});
export { courseModuleGetParams };
```

Файл

shared/src/validation-schemas/course-module/course-modules-get-all-params.validation-schema.ts

```
import * as Joi from 'joi';
import { CourseModulesGetAllRequestParamsDto } from '~/common/types/types';
import { getNameOf } from '~/helpers/helpers';
const courseModulesGetAllParams = Joi.object({
  [getNameOf<CourseModulesGetAllRequestParamsDto>('courseId')]: Joi.number()
    .integer()
    .required(),
});
export { courseModulesGetAllParams };
```

Файл shared/src/validation-schemas/group/group-configure-client.validation-schema.ts

```
import Joi from 'joi';
import {
  GroupValidationMessage,
  GroupValidationRule,
} from '~/common/enums/enums';
import { GroupsConfigureRequestDto } from '~/common/types/types';
import { getNameOf } from '~/helpers/helpers';
const groupConfigureClient = Joi.object({
  [getNameOf<GroupsConfigureRequestDto>('name')]: Joi.string()
    .trim()
    .min(GroupValidationRule.NAME_MIN_LENGTH)
    .max(GroupValidationRule.NAME_MAX_LENGTH)
    .required()
    .messages({
      'string.empty': GroupValidationMessage.NAME_REQUIRE,
      'string.min': GroupValidationMessage.NAME_MIN_LENGTH,
      'string.max': GroupValidationMessage.NAME_MAX_LENGTH,
      'string.base': GroupValidationMessage.NAME_STRING,
      'any.required': GroupValidationMessage.NAME_REQUIRE,
    }),
});
export { groupConfigureClient };
```

Файл shared/src/validation-schemas/group/group-create-client.validation-schema.ts

```
import Joi from 'joi';
import {
  GroupValidationMessage,
  GroupValidationRule,
} from '~/common/enums/enums';
import { GroupsConfigureRequestDto } from '~/common/types/types';
import { getNameOf } from '~/helpers/helpers';
const groupCreateClient = Joi.object({
```

```
  [getNameOf<GroupsConfigureRequestDto>('name')]: Joi.string()
    .trim()
    .min(GroupValidationRule.NAME_MIN_LENGTH)
    .max(GroupValidationRule.NAME_MAX_LENGTH)
    .required()
    .messages({
      'string.empty': GroupValidationMessage.NAME_REQUIRE,
      'string.min': GroupValidationMessage.NAME_MIN_LENGTH,
      'string.max': GroupValidationMessage.NAME_MAX_LENGTH,
      'string.base': GroupValidationMessage.NAME_STRING,
      'any.required': GroupValidationMessage.NAME_REQUIRE,
    }),
});
export { groupCreateClient };
```

Файл shared/src/validation-schemas/group/group-create.validation-schema.ts

```
import * as Joi from 'joi';
import { GroupValidationRule } from
'~/common/enums/enums';
import { GroupValidationMessage } from
'~/common/enums/group/group';
import { GroupsConfigureRequestDto } from
'~/common/types/types';
import { getNameOf } from '~/helpers/helpers';
const groupCreate = Joi.object({

[getNameOf<GroupsConfigureRequestDto>('nam
e')]: Joi.string()
    .trim()

.min(GroupValidationRule.NAME_MIN_LENGT
H)

.max(GroupValidationRule.NAME_MAX_LENG
TH)
    .required()
    .messages({
      'string.empty':
GroupValidationMessage.NAME_REQUIRE,
      'string.min':
GroupValidationMessage.NAME_MIN_LENGTH
,
      'string.max':
GroupValidationMessage.NAME_MAX_LENGT
H,
      'string.base':
GroupValidationMessage.NAME_STRING,
      'any.required':
GroupValidationMessage.NAME_REQUIRE,
    }),

[getNameOf<GroupsConfigureRequestDto>('per
missionIds')]: Joi.array()
    .items(Joi.number().integer())

.min(GroupValidationRule.PERMISSION_IDS_
MIN_LENGTH)
    .required()
    .messages({
      'array.empty':
GroupValidationMessage.PERMISSION_IDS_RE
QUIRE,
      'array.min':
GroupValidationMessage.PERMISSION_IDS_MI
N_LENGTH,
      'array.base':
GroupValidationMessage.PREMISSION_IDS_IN
TEGER,
      'any.required':
GroupValidationMessage.PERMISSION_IDS_RE
QUIRE,
    }),

[getNameOf<GroupsConfigureRequestDto>('userI
ds')]: Joi.array().items(
    Joi.number().integer().messages({
      'array.base':
GroupValidationMessage.USER_IDS_INTEGER,
    }),
  ),
});
export { groupCreate };
```

Файл shared/src/validation-schemas/group/group-delete.validation-schema.ts

```
import * as Joi from 'joi';
import { GroupsDeleteRequestParamDto } from '~/common/types/types';
import { getNameOf } from '~/helpers/helpers';
const groupDelete = Joi.object({
  [getNameOf<GroupsDeleteRequestParamDto>('id')]: Joi.string()
    .trim()
```

```
    .required(),
});
export { groupDelete };
```

Файл shared/src/validation-schemas/group/group-get-by-id.validation-schema.ts

```
import * as Joi from 'joi';
import { GroupsGetByIdRequestDto } from '~/common/types/types';
import { getNameOf } from '~/helpers/helpers';
const groupGetById = Joi.object({
  [getNameOf<GroupsGetByIdRequestDto>('id')]: Joi.string().trim().required(),
});
export { groupGetById };
```

Файл shared/src/validation-schemas/group/group-update-params.validation-schema.ts

```
import * as Joi from 'joi';
import { GroupsUpdateRequestParamsDto } from '~/common/types/types';
import { getNameOf } from '~/helpers/helpers';
const groupUpdateParams = Joi.object({
  [getNameOf<GroupsUpdateRequestParamsDto>('id')]: Joi.number()
    .integer()
    .required(),
});
export { groupUpdateParams };
```

Файл shared/src/validation-schemas/group/group-update.validation-schema.ts

```
import * as Joi from 'joi';
import { GroupValidationRule } from '~/common/enums/enums';
import { GroupValidationMessage } from '~/common/enums/group/group';
import { GroupsUpdateRequestDto } from '~/common/types/types';
import { getNameOf } from '~/helpers/helpers';
const groupUpdate = Joi.object({
  [getNameOf<GroupsUpdateRequestDto>('name')]: Joi.string()
    .trim()
    .min(GroupValidationRule.NAME_MIN_LENGTH)
    .max(GroupValidationRule.NAME_MAX_LENGTH)
    .required()
    .messages({
      'string.empty': GroupValidationMessage.NAME_REQUIRE,
      'string.min': GroupValidationMessage.NAME_MIN_LENGTH,
      'string.max': GroupValidationMessage.NAME_MAX_LENGTH,
      'string.base': GroupValidationMessage.NAME_STRING,
      'any.required': GroupValidationMessage.NAME_REQUIRE,
    }),
  [getNameOf<GroupsUpdateRequestDto>('permissionIds')]: Joi.array()
    .items(Joi.number())
    .min(GroupValidationRule.PERMISSION_IDS_MIN_LENGTH)
    .required()
    .messages({
      'array.empty': GroupValidationMessage.PERMISSION_IDS_REQUIRE,
      'array.min': GroupValidationMessage.PERMISSION_IDS_MIN_LENGTH,
      'array.base': GroupValidationMessage.PREMISSION_IDS_INTEGER,
      'any.required': GroupValidationMessage.PERMISSION_IDS_REQUIRE,
    }),
  [getNameOf<GroupsUpdateRequestDto>('userIds')]: Joi.array()
    .items(Joi.number())
    .required()
    .messages({
```

```
    'array.base': GroupValidationMessage.USER_IDS_INTEGER,
    'any.required': GroupValidationMessage.USER_IDS_REQUIRE,
  }),
});
export { groupUpdate };
```

Файл shared/src/validation-schemas/interview/interview-by-id-params.validation-schema.ts

```
import Joi from 'joi';
import { InterviewsByIdRequestParamsDto } from '~/common/types/types';
import { getNameOf } from '~/helpers/helpers';
const interviewByIdParams = Joi.object({
  [getNameOf<InterviewsByIdRequestParamsDto>('id')]: Joi.number()
    .integer()
    .required(),
});
export { interviewByIdParams };
```

Файл shared/src/validation-schemas/interview/interview-by-interviewee-id.validation-schema.ts

```
import Joi from 'joi';
import { InterviewsByIntervieweeIdRequestDto } from '~/common/types/types';
import { getNameOf } from '~/helpers/helpers';
const interviewByIntervieweeId = Joi.object({
  [getNameOf<InterviewsByIntervieweeIdRequestDto>('intervieweeUserId')]:
    Joi.number().integer().required(),
});

export { interviewByIntervieweeId };
```

Файл shared/src/validation-schemas/interview/interview-create.validation-schema.ts

```
import Joi from 'joi';
import { InterviewValidationMessage } from '~/common/enums/enums';
import { InterviewsCreateRequestBodyDto } from '~/common/types/types';
import { getNameOf } from '~/helpers/helpers';
const interviewCreate = Joi.object({
  [getNameOf<InterviewsCreateRequestBodyDto>('categoryId')]: Joi.number()
    .integer()
    .required()
    .messages({
      'number.base': InterviewValidationMessage.CATEGORY_ID_INTEGER,
      'any.required': InterviewValidationMessage.CATEGORY_ID_REQUIRE,
    }),
  [getNameOf<InterviewsCreateRequestBodyDto>('intervieweeUserId')]: Joi.number()
    .integer()
    .required()
    .messages({
      'number.base': InterviewValidationMessage.INTERVIEWER_ID_INTEGER,
      'any.required': InterviewValidationMessage.INTERVIEWEE_ID_REQUIRE,
    }),
});
export { interviewCreate };
```

Файл

shared/src/validation-schemas/interview/interview-get-interviewers-by-category.validation-schema.ts

```
import Joi from 'joi';
import { InterviewsGetInterviewersByCategoryRequestDto } from '~/common/types/types';
import { getNameOf } from '~/helpers/helpers';
const interviewGetInterviewersByCategory = Joi.object({
```

```
  [getNameOf<InterviewsGetInterviewersByCategoryRequestDto>('categoryId')]:
    Joi.number().integer().required(),
});
export { interviewGetInterviewersByCategory };
```

Файл shared/src/validation-schemas/interview/interview-update-params.validation-schema.ts

```
import * as Joi from 'joi';
import { GroupsUpdateRequestParamsDto } from '~/common/types/types';
import { getNameOf } from '~/helpers/helpers';
const interviewUpdateParams = Joi.object({
  [getNameOf<GroupsUpdateRequestParamsDto>('id')]: Joi.number()
    .integer()
    .required(),
});
export { interviewUpdateParams };
```

Файл

shared/src/validation-schemas/interview/interview-update-without-interviewer-validation-schema.ts

```
import * as Joi from 'joi';
import { InterviewStatus } from '~/common/enums/enums';
import { InterviewsUpdateRequestDto } from '~/common/types/types';
import { getNameOf } from '~/helpers/helpers';
const interviewUpdateWithoutInterviewer = Joi.object({
  [getNameOf<InterviewsUpdateRequestDto>('status')]: Joi.string()
    .required()
    .valid(...Object.values(InterviewStatus)),
  [getNameOf<InterviewsUpdateRequestDto>('interviewDate')]:
    Joi.date().allow(null),
});
export { interviewUpdateWithoutInterviewer };
```

Файл shared/src/validation-schemas/interview/interview-update.validation-schema.ts

```
import * as Joi from 'joi';
import {
  InterviewStatus,
  InterviewValidationMessage,
} from '~/common/enums/enums';
import { InterviewsUpdateRequestDto } from '~/common/types/types';
import { getNameOf } from '~/helpers/helpers';
const interviewUpdate = Joi.object({
  [getNameOf<InterviewsUpdateRequestDto>('interviewerUserId')]: Joi.number()
    .integer()
    .allow(null)
    .messages({
      'number.base': InterviewValidationMessage.INTERVIEWER_ID_INTEGER,
    }),
  [getNameOf<InterviewsUpdateRequestDto>('status')]: Joi.string()
    .required()
    .valid(...Object.values(InterviewStatus))
    .messages({
      'string.base': InterviewValidationMessage.STATUS_STRING,
      'any.required': InterviewValidationMessage.STATUS_REQUIRE,
      'any.only': InterviewValidationMessage.STATUS_VALID,
    }),
  [getNameOf<InterviewsUpdateRequestDto>('interviewDate')]:
    Joi.date().allow(null),
});
```

```
export { interviewUpdate };
```
Файл

shared/src/validation-schemas/interview-note/interview-note-create-params.validation-schema.ts

```
import * as Joi from 'joi';
import { InterviewNoteCreateRequestParamsDto } from '~/common/types/types';
import { getNameOf } from '~/helpers/helpers';
const interviewNotesCreateParams = Joi.object({
  [getNameOf<InterviewNoteCreateRequestParamsDto>('id')]: Joi.number()
    .integer()
    .required(),
});
export { interviewNotesCreateParams };
```
Файл shared/src/validation-schemas/interview-note/interview-note-create.validation-schema.ts

```
import * as Joi from 'joi';
import { InterviewValidationMessage } from '~/common/enums/enums';
import { InterviewNoteCreateRequestDto } from '~/common/types/types';
import { getNameOf } from '~/helpers/helpers';
const interviewNotesCreateArguments = Joi.object({
  [getNameOf<InterviewNoteCreateRequestDto>('note')]: Joi.string()
    .required()
    .messages({
      'string.base': InterviewValidationMessage.NOTE_STRING,
      'any.required': InterviewValidationMessage.NOTE_REQUIRE,
    }),
});
export { interviewNotesCreateArguments };
```

Файл shared/src/validation-schemas/interview-note/interview-note-get-all.validation-schema.ts

```
import * as Joi from 'joi';
import { InterviewNoteGetAllRequestParamsDto } from '~/common/types/types';
import { getNameOf } from '~/helpers/helpers';
const interviewNotesGetAllParams = Joi.object({
  [getNameOf<InterviewNoteGetAllRequestParamsDto>('id')]: Joi.number()
    .integer()
    .required(),
});
export { interviewNotesGetAllParams };
```
Файл shared/src/validation-schemas/mentor/get-mentor.validation-schema.ts

```
import * as Joi from 'joi';
import { GetMentorRequestParamsDto } from '~/common/types/types';
import { getNameOf } from '~/helpers/helpers';
const getMentor = Joi.object({
  [getNameOf<GetMentorRequestParamsDto>('menteeId')]: Joi.number()
    .integer()
    .required(),
  [getNameOf<GetMentorRequestParamsDto>('courseId')]: Joi.number()
    .integer()
    .required(),
});
export { getMentor };
```
Файл shared/src/validation-schemas/mentor/mentor-create-body.validation-schema.ts

```
import Joi from 'joi';
```

```
import { MentorValidationMessage } from '~/common/enums/enums';
import { CoursesToMentorsRequestDto } from '~/common/types/types';
import { getNameOf } from '~/helpers/helpers';
const mentorCreateBody = Joi.object({
  [getNameOf<CoursesToMentorsRequestDto>('courseId')]: Joi.number()
    .integer()
    .required()
    .messages({
      'number.base': MentorValidationMessage.COURSE_ID_INTEGER,
      'any.required': MentorValidationMessage.COURSE_ID_REQUIRE,
    }),
  [getNameOf<CoursesToMentorsRequestDto>('userId')]: Joi.number()
    .integer()
    .required()
    .messages({
      'number.base': MentorValidationMessage.USER_ID_INTEGER,
      'any.required': MentorValidationMessage.USER_ID_REQUIRE,
    }),
});
export { mentorCreateBody };
```

Файл shared/src/validation-schemas/pagination/pagination.validation.ts

```
import * as Joi from 'joi';
import { PaginationValidationRule } from '~/common/enums/enums';
import { EntityPaginationRequestQueryDto } from '~/common/types/types';
import { getNameOf } from '~/helpers/helpers';

const pagination = Joi.object({
  [getNameOf<EntityPaginationRequestQueryDto>('count')]: Joi.number()
    .integer()
    .min(PaginationValidationRule.MIN_COUNT),
  [getNameOf<EntityPaginationRequestQueryDto>('page')]: Joi.number()
    .integer()
    .min(PaginationValidationRule.MIN_PAGE),
});
export { pagination };
```

Файл shared/src/validation-schemas/task/task-by-mentee-id-course-id-module-id-params.validation-schema.ts

```
import Joi from 'joi';
import { TaskGetByMenteeIdCourseIdModuleIdRequestDto } from '~/common/types/types';
import { getNameOf } from '~/helpers/helpers';
const taskByMenteeIdCourseIdModuleIdParams = Joi.object({
  [getNameOf<TaskGetByMenteeIdCourseIdModuleIdRequestDto>('courseId')]:
    Joi.number().integer().required(),
  [getNameOf<TaskGetByMenteeIdCourseIdModuleIdRequestDto>('menteeId')]:
    Joi.number().integer().required(),
  [getNameOf<TaskGetByMenteeIdCourseIdModuleIdRequestDto>('moduleId')]:
    Joi.number().integer().required(),
});
export { taskByMenteeIdCourseIdModuleIdParams };
```

Файл shared/src/validation-schemas/task/task-by-mentee-id-module-id.validation-schema.ts

```
import Joi from 'joi';
import { TaskGetByMenteeIdAndModuleId } from '~/common/types/types';
import { getNameOf } from '~/helpers/helpers';
const taskByMenteeIdAndModuleId = Joi.object({
```

```
  [getNameOf<TaskGetByMenteeIdAndModuleId>('moduleId')]: Joi.number()
    .integer()
    .required(),
  [getNameOf<TaskGetByMenteeIdAndModuleId>('menteeId')]: Joi.number()
    .integer()
    .required(),
});
export { taskByMenteeIdAndModuleId };
```

Файл shared/src/validation-schemas/task/tasks-by-course-id-and-mentee-id.validation-schema.ts

```
import Joi from 'joi';
import { TasksGetByCourseIdAndMenteeIdRequestDto } from '~/common/types/types';
import { getNameOf } from '~/helpers/helpers';
const tasksByCourseIdAndMenteeId = Joi.object({
  [getNameOf<TasksGetByCourseIdAndMenteeIdRequestDto>('courseId')]: Joi.number()
    .integer()
    .required(),
  [getNameOf<TasksGetByCourseIdAndMenteeIdRequestDto>('menteeId')]: Joi.number()
    .integer()
    .required(),
});
export { tasksByCourseIdAndMenteeId };
```

Файл shared/src/validation-schemas/task/tasks-by-id-params.validation-schema.ts

```
import Joi from 'joi';
import { TaskByIdRequestParamsDto } from '~/common/types/types';
import { getNameOf } from '~/helpers/helpers';
const tasksByIdParams = Joi.object({
  [getNameOf<TaskByIdRequestParamsDto>('taskId')]: Joi.number()
    .integer()
    .required(),
});
export { tasksByIdParams };
```

Файл shared/src/validation-schemas/task/tasks-manipulate-request-body.validation-schema.ts

```
import Joi from 'joi';
import { TaskNoteValidationMessage } from '~/common/enums/enums';
import { TaskNoteManipulateRequestBodyDto } from '~/common/types/types';
import { getNameOf } from '~/helpers/helpers';
const tasksManipulateRequestBody = Joi.object({
  [getNameOf<TaskNoteManipulateRequestBodyDto>('note')]: Joi.string()
    .required()
    .messages({ 'string.empty': TaskNoteValidationMessage.MESSAGE_EMPTY }),
  [getNameOf<TaskNoteManipulateRequestBodyDto>('status')]: Joi.string()
    .required()
    .messages({ 'string.empty': TaskNoteValidationMessage.STATUS_EMPTY }),
});
export { tasksManipulateRequestBody };
```

Файл shared/src/validation-schemas/user/user-delete.validation-schema.ts

```
import * as Joi from 'joi';
import { UsersDeleteRequestParamsDto } from '~/common/types/types';
import { getNameOf } from '~/helpers/helpers';
const userDelete = Joi.object({
  [getNameOf<UsersDeleteRequestParamsDto>('id')]: Joi.number()
    .integer()
    .required(),
});
```

```
export { userDelete };
```

Файл shared/src/validation-schemas/user/user-sign-in.validation-schema.ts

```
import * as Joi from 'joi';
import {
  UserValidationMessage,
  UserValidationRule,
} from '~/common/enums/enums';
import { UserSignInRequestDto } from '~/common/types/types';
import { getNameOf } from '~/helpers/helpers';
const userSignIn = Joi.object({
  [getNameOf<UserSignInRequestDto>('email')]: Joi.string()
    .trim()
    .min(UserValidationRule.EMAIL_MIN_LENGTH)
    .max(UserValidationRule.EMAIL_MAX_LENGTH)
    .email({ tlds: { allow: false } })
    .required()
    .messages({
      'string.email': UserValidationMessage.EMAIL_WRONG,
      'string.empty': UserValidationMessage.EMAIL_REQUIRE,
      'string.min': UserValidationMessage.EMAIL_MIN_LENGTH,
      'string.max': UserValidationMessage.EMAIL_MAX_LENGTH,
      'string.base': UserValidationMessage.EMAIL_STRING,
      'any.required': UserValidationMessage.EMAIL_REQUIRE,
    }),
  [getNameOf<UserSignInRequestDto>('password')]: Joi.string()
    .trim()
    .min(UserValidationRule.PASSWORD_MIN_LENGTH)
    .max(UserValidationRule.PASSWORD_MAX_LENGTH)
    .required()
    .messages({
      'string.empty': UserValidationMessage.PASSWORD_REQUIRE,
      'string.min': UserValidationMessage.PASSWORD_MIN_LENGTH,
      'string.max': UserValidationMessage.PASSWORD_MAX_LENGTH,
      'string.base': UserValidationMessage.PASSWORD_STRING,
      'any.required': UserValidationMessage.PASSWORD_REQUIRE,
    }),
});
export { userSignIn };
```

Файл shared/src/validation-schemas/user/user-sign-up.validation-schema.ts

```
import * as Joi from 'joi';
import {
  UserValidationMessage,
  UserValidationRule,
} from '~/common/enums/enums';
import { UserSignUpRequestDto } from '~/common/types/types';
import { getNameOf } from '~/helpers/helpers';
const userSignUp = Joi.object({
  [getNameOf<UserSignUpRequestDto>('email')]: Joi.string()
    .trim()
    .min(UserValidationRule.EMAIL_MIN_LENGTH)
    .max(UserValidationRule.EMAIL_MAX_LENGTH)
    .email({ tlds: { allow: false } })
    .min(UserValidationRule.EMAIL_MIN_LENGTH)
    .max(UserValidationRule.EMAIL_MAX_LENGTH)
    .required()
```

```
    .messages({
      'string.email': UserValidationMessage.EMAIL_WRONG,
      'string.empty': UserValidationMessage.EMAIL_REQUIRE,
      'string.min': UserValidationMessage.EMAIL_MIN_LENGTH,
      'string.max': UserValidationMessage.EMAIL_MAX_LENGTH,
      'string.base': UserValidationMessage.EMAIL_STRING,
      'any.required': UserValidationMessage.EMAIL_REQUIRE,
    }),
  [getNameOf<UserSignUpRequestDto>('password')]: Joi.string()
    .trim()
    .min(UserValidationRule.PASSWORD_MIN_LENGTH)
    .max(UserValidationRule.PASSWORD_MAX_LENGTH)
    .required()
    .messages({
      'string.empty': UserValidationMessage.PASSWORD_REQUIRE,
      'string.min': UserValidationMessage.PASSWORD_MIN_LENGTH,
      'string.max': UserValidationMessage.PASSWORD_MAX_LENGTH,
      'string.base': UserValidationMessage.PASSWORD_STRING,
      'any.required': UserValidationMessage.PASSWORD_REQUIRE,
    }),
  [getNameOf<UserSignUpRequestDto>('fullName')]: Joi.string()
    .trim()
    .min(UserValidationRule.NAME_MIN_LENGTH)
    .max(UserValidationRule.NAME_MAX_LENGTH)
    .required()
    .pattern(UserValidationRule.NAME_PATTERN)
    .messages({
      'string.empty': UserValidationMessage.NAME_REQUIRE,
      'string.min': UserValidationMessage.NAME_MIN_LENGTH,
      'string.max': UserValidationMessage.NAME_MAX_LENGTH,
      'string.pattern.base': UserValidationMessage.NAME_WRONG,
      'string.base': UserValidationMessage.NAME_STRING,
      'any.required': UserValidationMessage.NAME_REQUIRE,
    }),
});
export { userSignUp };
```

Файл shared/src/validation-schemas/user-details/user-details-update-info.validation-schema.ts

```
import * as Joi from 'joi';
import {
  UserDetailsValidationMessage,
  UserDetailsValidationRule,
} from '~/common/enums/enums';
import { UserDetailsUpdateInfoRequestDto } from '~/common/types/types';
import { getNameOf } from '~/helpers/helpers';
const userDetailsUpdateInfo = Joi.object({
  [getNameOf<UserDetailsUpdateInfoRequestDto>('fullName')]: Joi.string()
    .trim()
    .min(UserDetailsValidationRule.FULL_NAME_MIN_LENGTH)
    .max(UserDetailsValidationRule.FULL_NAME_MAX_LENGTH)
    .pattern(UserDetailsValidationRule.FULL_NAME_PATTERN)
    .required()
    .messages({
      'string.empty': UserDetailsValidationMessage.FULL_NAME_REQUIRE,
      'string.min': UserDetailsValidationMessage.FULL_NAME_MIN_LENGTH,
      'string.max': UserDetailsValidationMessage.FULL_NAME_MAX_LENGTH,
```

```
      'string.pattern.base': UserDetailsValidationMessage.FULL_NAME_WRONG,
      'string.base': UserDetailsValidationMessage.FULL_NAME_STRING,
      'any.required': UserDetailsValidationMessage.FULL_NAME_REQUIRE,
    }),
  [getNameOf<UserDetailsUpdateInfoRequestDto>('gender')]: Joi.string()
    .trim()
    .required()
    .messages({
      'string.empty': UserDetailsValidationMessage.GENDER_REQUIRE,
      'string.base': UserDetailsValidationMessage.GENDER_STRING,
      'any.required': UserDetailsValidationMessage.GENDER_REQUIRE,
    }),
  [getNameOf<UserDetailsUpdateInfoRequestDto>('dateOfBirth')]:
    Joi.date().allow(null),
  [getNameOf<UserDetailsUpdateInfoRequestDto>('telegramUsername')]: Joi.string()
    .trim()
    .min(UserDetailsValidationRule.TELEGRAM_USERNAME_MIN_LENGTH)
    .max(UserDetailsValidationRule.TELEGRAM_USERNAME_MAX_LENGTH)
    .pattern(UserDetailsValidationRule.TELEGRAM_USERNAME_PATTERN)
    .messages({
      'string.min': UserDetailsValidationMessage.TELEGRAM_USERNAME_MIN_LENGTH,
      'string.max': UserDetailsValidationMessage.TELEGRAM_USERNAME_MAX_LENGTH,
      'string.pattern.base':
        UserDetailsValidationMessage.TELEGRAM_USERNAME_WRONG,
      'string.base': UserDetailsValidationMessage.TELEGRAM_USERNAME_STRING,
    })
    .allow(null, ''),
});
export { userDetailsUpdateInfo };
```

Файл shared/src/validation-schemas/user-details/user-details-update-params.validation-schema.ts

```
import Joi from 'joi';
import { UserDetailsUpdateAvatarRequestParamsDto } from '~/common/types/types';
import { getNameOf } from '~/helpers/helpers';
const userDetailsUpdateParams = Joi.object({
  [getNameOf<UserDetailsUpdateAvatarRequestParamsDto>('userId')]: Joi.number()
    .integer()
    .required(),
});
export { userDetailsUpdateParams };
```

Файл backend/src/api/auth/auth.api.ts

```
import { FastifyPluginAsync, FastifyRequest }
from 'fastify';
import { AuthApiPath, HttpCode, HttpMethod }
from '~/common/enums/enums';
import {
  UserSignInRequestDto,
  UserSignUpRequestDto,
} from '~/common/types/types';
import {
  auth as authService,
  http as httpService,
  user as userService,
} from '~/services/services';
import {
  userSignIn as userSignInValidationSchema,
  userSignUp as userSignUpValidationSchema,
} from '~/validation-schemas/validation-schemas';
type Options = {
  services: {
    auth: typeof authService;
    user: typeof userService;
    http: typeof httpService;
  };
  frontendURL: string;
};
const initAuthApi: FastifyPluginAsync<Options>
= async (fastify, opts) => {
  const {
    services: { auth: authService, http: httpService
  },
```

```
    frontendURL,
  } = opts;

  fastify.route({
    method: HttpMethod.POST,
    url: AuthApiPath.SIGN_UP,
    schema: {
      body: userSignUpValidationSchema,
    },
    async handler(req: FastifyRequest<{ Body:
UserSignUpRequestDto }>, rep) {
      const user = await
authService.signUp(req.body);
      return
rep.status(HttpCode.CREATED).send(user);
    },
  });
  fastify.route({
    method: HttpMethod.POST,
    url: AuthApiPath.SIGN_IN,
    schema: {
      body: userSignInValidationSchema,
    },
    async handler(req: FastifyRequest<{ Body:
UserSignInRequestDto }>, rep) {
      const user = await
authService.signIn(req.body);
      return rep.status(HttpCode.OK).send(user);
    },
  });
  fastify.route({
    method: HttpMethod.GET,
    url: AuthApiPath.CURRENT_USER,
    async handler(req, rep) {
      const [, token] =
req.headers?.authorization?.split(' ') ?? [];
      const user = await
authService.getCurrentUser(token);
      return rep.status(HttpCode.OK).send(user);
    },
  });
  fastify.route({
    method: HttpMethod.GET,
    url: AuthApiPath.GOOGLE_CALLBACK,
    async handler(req, rep) {
      const {
        token: { access_token: accessToken,
token_type: tokenType },
      } = await
this.googleOAuth2.getAccessTokenFromAuthoriz
ationCodeFlow(req);
      const { name: fullName, email } = await
httpService.load<
        Record<'name' | 'email', string>
>('https://www.googleapis.com/oauth2/v2/userinfo
', {
        headers: { Authorization: `${tokenType}
${accessToken}` },
        method: HttpMethod.GET,
      });
      const { token } = await
authService.signOAuth({ fullName, email });
      return rep.redirect(

`${frontendURL}${AuthApiPath.SIGN_REDIRE
CT}?token=${token}`,
      );
    },
  });
  fastify.route({
    method: HttpMethod.GET,
    url: AuthApiPath.GITHUB_CALLBACK,
    async handler(req, rep) {
      const {
        token: { access_token: accessToken,
token_type: tokenType },
      } = await
this.githubOAuth2.getAccessTokenFromAuthoriz
ationCodeFlow(req);
      const userEmails = await httpService.load<
        (Record<'primary', boolean> &
Record<'email', string>)[]
      >('https://api.github.com/user/emails', {
        headers: {
          'User-Agent':
'platformatic-authentication-server',
          Accept: 'application/vnd.github+json',
          Authorization: `${tokenType}
${accessToken}`,
          'X-GitHub-Api-Version': '2022-11-28',
        },
      });
      const { email } = userEmails.find(
        ({ primary }) => primary,
      ) as NonNullable<typeof
userEmails>[number];
      const { name: fullName } = await
httpService.load<Record<'name', string>>(
        'https://api.github.com/user',
        {
          headers: {
            'User-Agent':
'platformatic-authentication-server',
            Accept: 'application/vnd.github+json',
            Authorization: `${tokenType}
${accessToken}`,
            'X-GitHub-Api-Version': '2022-11-28',
          },
        },
```

```
    );
    const { token } = await
authService.signOAuth({ fullName, email });
    return
rep.redirect(`${frontendURL}${AuthApiPath.SIG
N_REDIRECT}?token=${token}`);
```

Файл backend/src/api/billing/billing.api.ts

```
import { FastifyPluginAsync, FastifyRequest }
from 'fastify';
import {
 BillingApiPath,
 HttpCode,
 HttpMethod,
 PaginationDefaultValue,
} from '~/common/enums/enums';
import {
 BillingReplenishParamsDto,
 EntityPaginationRequestQueryDto,
} from '~/common/types/types';
import {
 billing as billingService,
 user as userService,
} from '~/services/services';
import {
 billingReplenishParams as
billingReplenishParamsValidationSchema,
 pagination as paginationValidationSchema,
} from '~/validation-schemas/validation-schemas';
type Options = {
 services: {
  billing: typeof billingService;
  user: typeof userService;
 };
};
const initBillingApi:
FastifyPluginAsync<Options> = async (fastify,
opts) => {
 const { billing: billingService, user: userService
} = opts.services;
 fastify.route({
  method: HttpMethod.GET,
  url: BillingApiPath.BALANCE,
  async handler(req, rep) {
   const { id } = req.user;
   const userBalance = await
userService.getByIdMoneyBalance(id);
   return
rep.status(HttpCode.OK).send(userBalance);
  },
 });
 fastify.route({
  method: HttpMethod.GET,
  url: BillingApiPath.TRANSACTIONS,
  schema: { querystring:
paginationValidationSchema },
  async handler(
   req: FastifyRequest<{ Querystring:
EntityPaginationRequestQueryDto }>,
   res,
  ) {
   const { id } = req.user;
   const {
    count =
PaginationDefaultValue.DEFAULT_COUNT,
    page =
PaginationDefaultValue.DEFAULT_PAGE,
   } = req.query;
   const userTransactions = await
billingService.getTransactionsByUserId(
    id,
    {
     count,
     page,
    },
   );
   return
res.status(HttpCode.OK).send(userTransactions);
  },
 });
 fastify.route({
  method: HttpMethod.PATCH,
  url: BillingApiPath.REPLENISH,
  schema: { body:
billingReplenishParamsValidationSchema },
  async handler(
   req: FastifyRequest<{ Body:
BillingReplenishParamsDto }>,
   rep,
  ) {
   const { id } = req.user;
   const { amountOfMoneyToReplenish, token }
= req.body;
   const userMoneyBalance = await
billingService.replenish({
    userId: id,
    amountOfMoneyToReplenish,
    token,
   });
   return
rep.status(HttpCode.OK).send(userMoneyBalance
);
  },
 });
```

```
  fastify.route({
    method: HttpMethod.PATCH,
    url: BillingApiPath.WITHDRAW,
    async handler(req, rep) {
      const { id } = req.user;
      const userMoneyBalance = await
billingService.withdraw(id);
```

Файл backend/src/api/categories/categories.api.ts

```
import { FastifyPluginAsync, FastifyRequest }
from 'fastify';
import { CategoriesApiPath, HttpCode,
HttpMethod } from '~/common/enums/enums';
import {
CourseCategoryGetByIdRequestParamsDto }
from '~/common/types/types';
import { courseCategory as
courseCategoryService } from
'~/services/services';
import { courseCategoryGetByIdParams as
courseCategoryGetByIdParamsValidationSchema
} from '~/validation-schemas/validation-schemas';
type Options = {
  services: {
    courseCategory: typeof courseCategoryService;
  };
};
const initCategoriesApi:
FastifyPluginAsync<Options> = async (
  fastify,
  opts,
) => {
  const { courseCategory: courseCategoryService
} = opts.services;
  fastify.route({
    method: HttpMethod.GET,
    url: CategoriesApiPath.ROOT,
    async handler(_req, rep) {
      const categories = await
courseCategoryService.getAll();
      return
rep.status(HttpCode.OK).send(categories);
```

Файл backend/src/api/chats/chats.api.ts

```
import { FastifyPluginAsync, FastifyRequest }
from 'fastify';
import { ChatsApiPath, HttpCode, HttpMethod }
from '~/common/enums/enums';
import {
  ChatMessageCreateRequestBodyDto,
  ChatMessageFilteringDto,
  ChatMessageGetAllRequestParamsDto,
  ChatMessageReadParams,
} from '~/common/types/types';
import { chatMessage as chatMessageService }
from '~/services/services';
```

```
      return
rep.status(HttpCode.OK).send(userMoneyBalance
);
    },
  });
};
export { initBillingApi };
```

```
    },
  });
  fastify.route({
    method: HttpMethod.GET,
    url: CategoriesApiPath.DASHBOARD,
    async handler(_req, rep) {
      const categoriesWithCourses =
        await
courseCategoryService.getAllWithCourses();
      return
rep.status(HttpCode.OK).send(categoriesWithCou
rses);
    },
  });
  fastify.route({
    method: HttpMethod.GET,
    url: CategoriesApiPath.$ID,
    schema: { params:
courseCategoryGetByIdParamsValidationSchema
},
    async handler(
      req: FastifyRequest<{ Params:
CourseCategoryGetByIdRequestParamsDto }>,
      rep,
    ) {
      const { id } = req.params;
      const category = await
courseCategoryService.getById(id);
      return
rep.status(HttpCode.OK).send(category);
    },
  });
};
export { initCategoriesApi };
```

```
import {
  chatMessageCreateArguments as
chatMessageCreateArgumentsValidationSchema,
  chatMessageFiltering as
chatMessageFilteringValidationSchema,
  chatMessageGetAllParams as
chatMessageGetAllParamsValidationSchema,
  chatMessageReadParams as
chatMessageReadParamsValidationSchema,
} from '~/validation-schemas/validation-schemas';
type Options = {
  services: {
```

```
    chatMessage: typeof chatMessageService;
  };
};
const initChatsApi: FastifyPluginAsync<Options>
= async (fastify, opts) => {
  const { chatMessage: chatMessageService } =
opts.services;
  fastify.route({
    method: HttpMethod.GET,
    url: ChatsApiPath.ROOT,
    schema: {
      querystring:
chatMessageFilteringValidationSchema,
    },
    async handler(
      req: FastifyRequest<{
        Querystring: ChatMessageFilteringDto;
      }>,
      rep,
    ) {
      const { id } = req.user;
      const { fullName } = req.query;
      const allChatsLastMessagesMessagesDto =
        await
chatMessageService.getAllLastMessages(id, {
fullName });
      const emptyChatsDto = await
chatMessageService.getAllEmptyChats({
        userId: id,
        fullName,
        lastMessagesInChats:
allChatsLastMessagesMessagesDto,
      });
      return rep.status(HttpCode.OK).send({
        items: allChatsLastMessagesMessagesDto,
        emptyChats: emptyChatsDto,
      });
    },
  });
  fastify.route({
    method: HttpMethod.GET,
    url: ChatsApiPath.$ID,
    schema: {
      params:
chatMessageGetAllParamsValidationSchema,
    },
    async handler(
      req: FastifyRequest<{ Params:
ChatMessageGetAllRequestParamsDto }>,
      rep,
    ) {
      const { id: userId } = req.user;
      const { id } = req.params;
      const { items } = await
chatMessageService.getAll({
        chatId: id,
      });
      let chatOpponent = null;
      const [currentChatMessage] = items;
      if (currentChatMessage) {
        chatOpponent =
          currentChatMessage.sender.id === userId
            ? currentChatMessage.receiver
            : currentChatMessage.sender;
      }
      return rep.status(HttpCode.OK).send({ items,
chatId: id, chatOpponent });
    },
  });
  fastify.route({
    method: HttpMethod.POST,
    url: ChatsApiPath.ROOT,
    schema: { body:
chatMessageCreateArgumentsValidationSchema
},
    async handler(
      req: FastifyRequest<{ Body:
ChatMessageCreateRequestBodyDto }>,
      rep,
    ) {
      const { id: userId } = req.user;
      const { message, chatId, receiverId } =
req.body;
      const newChatMessage = await
chatMessageService.create({
        senderId: userId,
        receiverId,
        message,
        chatId,
      });
      return
rep.status(HttpCode.CREATED).send(newChatM
essage);
    },
  });
  fastify.route({
    method: HttpMethod.GET,
    url:
ChatsApiPath.HAS_UNREAD_MESSAGES,
    async handler(req, rep) {
      const { id } = req.user;

      const hasUnreadMessages = await
chatMessageService.checkHasUnreadMessages(
        id,
      );
      return
rep.status(HttpCode.OK).send(hasUnreadMessage
s);
    },
  });
  fastify.route({
```

```
    method: HttpMethod.PATCH,
    url: ChatsApiPath.$ID_READ,
    schema: { params:
chatMessageReadParamsValidationSchema },
    async handler(req: FastifyRequest<{ Params:
ChatMessageReadParams }>, rep) {
      const { id } = req.params;
      const { id: userId } = req.user;
      const hasUnreadMessages = await
chatMessageService.readMessages(
        userId,
        id,
      );
      rep.status(HttpCode.OK).send(hasUnreadMessage
s);
    },
  });
};
export { initChatsApi };
```

Файл backend/src/api/course-modules/course-modules.api.ts

```
import { FastifyPluginAsync, FastifyRequest } from 'fastify';
import {
  CourseModulesApiPath,
  HttpCode,
  HttpMethod,
} from '~/common/enums/enums';
import {
  CourseModuleGetRequestParamsDto,
  CourseModulesGetAllRequestParamsDto,
} from '~/common/types/types';
import { courseModule as courseModuleService } from '~/services/services';
import {
  courseModuleGetParams as courseModuleGetParamsValidationSchema,
  courseModulesGetAllParams as courseModuleGetAllParamsValidationSchema,
} from '~/validation-schemas/validation-schemas';
type Options = {
  services: {
    courseModule: typeof courseModuleService;
  };
};
const initCourseModulesApi: FastifyPluginAsync<Options> = async (
  fastify,
  opts,
) => {
  const { courseModule: courseModuleService } = opts.services;
  fastify.route({
    method: HttpMethod.GET,
    url: CourseModulesApiPath.COURSES_$ID_MODULES_$ID,
    schema: { params: courseModuleGetParamsValidationSchema },
    async handler(
      req: FastifyRequest<{ Params: CourseModuleGetRequestParamsDto }>,
      rep,
    ) {
      const { courseId, moduleId } = req.params;
      const module = await courseModuleService.getById({ courseId, moduleId });
      rep.status(HttpCode.OK).send(module);
    },
  });
  fastify.route({
    method: HttpMethod.GET,
    url: CourseModulesApiPath.COURSES_$ID_MODULES,
    schema: { params: courseModuleGetAllParamsValidationSchema },
    async handler(
      req: FastifyRequest<{ Params: CourseModulesGetAllRequestParamsDto }>,
```

```
    rep,
   ) {
    const { courseId } = req.params;
    const items = await courseModuleService.getModulesByCourseId(courseId);
    rep.status(HttpCode.OK).send({ items });
   },
  });
};
export { initCourseModulesApi };
```

Файл backend/src/api/courses/courses.api.ts

```
import { FastifyPluginAsync, FastifyRequest }
from 'fastify';
import {
  CoursesApiPath,
  HttpCode,
  HttpMethod,
  PaginationDefaultValue,
  PermissionKey,
} from '~/common/enums/enums';
import {

CourseCheckIsMentorForMenteeRequestParamsD
to,
  CourseCheckIsMentorRequestParamsDto,
  CourseCreateRequestDto,
  CourseFilteringWithPaginationDto,
  CourseGetRequestParamsDto,
  CourseMentorsFilteringDto,
  CourseSelectMentorRequestDto,
  CourseSelectMentorRequestParamsDto,
  CourseUpdateCategoryRequestDto,
  CourseUpdateMentoringDto,
  EntityPaginationRequestQueryDto,
} from '~/common/types/types';
import { checkHasPermissions } from
'~/hooks/hooks';
import {
  course as courseService,
  mentor as mentorService,
} from '~/services/services';
import {
  courseCheckIsMentorForStudentParams as
courseCheckIsMentorForStudentParamsValidation
Schema,
  courseCheckIsMentorParams as
courseCheckIsMentorParamsValidationSchema,
  courseCreate as courseCreateValidationSchema,
  courseFiltering as
courseFilteringValidationSchema,
  courseGetParams as
courseGetParamsValidationSchema,
  courseMentorCreate as
courseMentorCreateBodyValidationSchema,
  courseMentoringUpdateCount as
courseMentoringUpdateCountValidationSchema,
  courseMentorsFiltering as
courseMentorsFilteringValidationSchema,
  courseUpdateByIdParams as
courseUpdateParamsValidationSchema,
  courseUpdateCategory as
courseUpdateCategoryValidationSchema,
  pagination as paginationValidationSchema,
} from '~/validation-schemas/validation-schemas';
type Options = {
  services: {
    course: typeof courseService;
    mentor: typeof mentorService;
  };
};
const initCoursesApi:
FastifyPluginAsync<Options> = async (fastify,
opts) => {
  const { course: courseService, mentor:
mentorService } = opts.services;

  fastify.route({
    method: HttpMethod.GET,
    url: CoursesApiPath.DASHBOARD,
    schema: {
      querystring: courseFilteringValidationSchema,
    },
    async handler(
      req: FastifyRequest<{
        Querystring:
CourseFilteringWithPaginationDto;
      }>,
      rep,
    ) {
      const courses = await
courseService.getAllWithCategories(req.query);
      return rep.status(HttpCode.OK).send(courses);
    },
  });
  fastify.route({
    method: HttpMethod.GET,
    url: CoursesApiPath.STUDYING,
    schema: { querystring:
paginationValidationSchema },
    async handler(
```

```
    req: FastifyRequest<{ Querystring:
EntityPaginationRequestQueryDto }>,
    res,
  ) {
    const courses = await
courseService.getAllCoursesStudying(
      req.user.id,
      req.query,
    );
    return res.status(HttpCode.OK).send(courses);
  },
 });
 fastify.route({
  method: HttpMethod.GET,
  url: CoursesApiPath.MENTORING,
  schema: { querystring:
paginationValidationSchema },
  async handler(
    req: FastifyRequest<{ Querystring:
EntityPaginationRequestQueryDto }>,
    res,
  ) {
    const courses = await
courseService.getAllCoursesMentoring(
      req.user.id,
      req.query,
    );
    return res.status(HttpCode.OK).send(courses);
  },
 });
 fastify.route({
  method: HttpMethod.PATCH,
  url: CoursesApiPath.MENTORING,
  schema: {
    body:
courseMentoringUpdateCountValidationSchema,
  },
  async handler(
    req: FastifyRequest<{
      Body: CourseUpdateMentoringDto;
    }>,
    res,
  ) {
    const result = await
courseService.updateStudentsCount(
      req.user.id,
      req.body,
    );
    return res.status(HttpCode.OK).send(result);
  },
 });
 fastify.route({
  method: HttpMethod.GET,
  url: CoursesApiPath.ROOT,
  schema: { querystring:
paginationValidationSchema },
  async handler(
    req: FastifyRequest<{ Querystring:
EntityPaginationRequestQueryDto }>,
    res,
  ) {
    const {
      count =
PaginationDefaultValue.DEFAULT_COUNT_BY
_10,
      page =
PaginationDefaultValue.DEFAULT_PAGE,
    } = req.query;
    const courseWithCategories = await
courseService.getAll({
      count,
      page,
    });
    return
res.status(HttpCode.OK).send(courseWithCategori
es);
  },
 });
 fastify.route({
  method: HttpMethod.POST,
  url: CoursesApiPath.ROOT,
  schema: {
    body: courseCreateValidationSchema,
  },
  async handler(req: FastifyRequest<{ Body:
CourseCreateRequestDto }>, rep) {
    const { url } = req.body;
    const course = await
courseService.createByUrl(url);
    return
rep.status(HttpCode.CREATED).send(course);
  },
 });
 fastify.route({
  method: HttpMethod.GET,
  url: CoursesApiPath.$ID,
  schema: { params:
courseGetParamsValidationSchema },
  async handler(
    req: FastifyRequest<{ Params:
CourseGetRequestParamsDto }>,
    rep,
  ) {
    const { id } = req.params;
    const course = await
courseService.getById(id);
    return rep.status(HttpCode.OK).send(course);
  },
 });
 fastify.route({
  method: HttpMethod.GET,
  url: CoursesApiPath.$ID_MENTORS,
```

```
    schema: {
      params: courseGetParamsValidationSchema,
      querystring:
courseMentorsFilteringValidationSchema,
    },
    async handler(
      req: FastifyRequest<{
        Params: CourseGetRequestParamsDto;
        Querystring: CourseMentorsFilteringDto;
      }>,
      rep,
    ) {
      const { id } = req.params;
      const { mentorName } = req.query;
      const mentors = await
courseService.getMentorsByCourseId({
        courseId: id,
        filteringOpts: { mentorName },
      });
      rep.status(HttpCode.OK).send(mentors);
    },
  });
  fastify.route({
    method: HttpMethod.GET,
    url: CoursesApiPath.$ID_MENTEES,
    schema: {
      params: courseGetParamsValidationSchema,
    },
    async handler(
      req: FastifyRequest<{
        Params: CourseGetRequestParamsDto;
      }>,
      rep,
    ) {
      const { id } = req.params;
      const mentees = await
courseService.getMenteesByCourseIdAndMentorI
d({
        mentorId: req.user.id,
        courseId: id,
      });
      rep.status(HttpCode.OK).send(mentees);
    },
  });
  fastify.route({
    method: HttpMethod.GET,
    url:
CoursesApiPath.$ID_IS_MENTOR_CHECK,
    schema: {
      params:
courseCheckIsMentorParamsValidationSchema,
    },
    async handler(
      req: FastifyRequest<{
        Params:
CourseCheckIsMentorRequestParamsDto;
      }>,
      rep,
    ) {
      const { id } = req.params;
      const { user } = req;
      const isMentor = await
mentorService.checkIsMentor({
        courseId: id,
        userId: user.id,
      });
      rep.status(HttpCode.OK).send(isMentor);
    },
  });
  fastify.route({
    method: HttpMethod.GET,
    url:
CoursesApiPath.$ID_HAS_MENTOR_CHECK,
    schema: {
      params:
courseCheckIsMentorParamsValidationSchema,
    },
    async handler(
      req: FastifyRequest<{
        Params:
CourseCheckIsMentorRequestParamsDto;
      }>,
      rep,
    ) {
      const { id } = req.params;
      const { user } = req;
      const hasMentor = await
mentorService.checkHasMentor({
        courseId: id,
        userId: user.id,
      });
      rep.status(HttpCode.OK).send(hasMentor);
    },
  });
  fastify.route({
    method: HttpMethod.PATCH,
    url: CoursesApiPath.$ID_CATEGORY,
    schema: {
      params:
courseUpdateParamsValidationSchema,
      body:
courseUpdateCategoryValidationSchema,
    },
    preHandler: checkHasPermissions('every',
PermissionKey.MANAGE_CATEGORIES),
    async handler(
      req: FastifyRequest<{
        Params: CourseGetRequestParamsDto;
        Body: CourseUpdateCategoryRequestDto;
      }>,
      rep,
    ) {
```

```
    const { id } = req.params;
    const { newCategoryId } = req.body;
    const course = await
courseService.updateCategory(id,
newCategoryId);
    return rep.status(HttpCode.OK).send(course);
   },
  });
  fastify.route({
   method: HttpMethod.POST,
   url: CoursesApiPath.$ID_MENTORS,
   schema: {
    params: courseGetParamsValidationSchema,
    body:
courseMentorCreateBodyValidationSchema,
   },
   async handler(
    req: FastifyRequest<{
     Params:
CourseSelectMentorRequestParamsDto;
     Body: CourseSelectMentorRequestDto;
    }>,
    rep,
   ) {
    const { mentorId, menteeId } = req.body;
    const { id } = req.params;
    const chooseMentor = await
mentorService.chooseMentor({
     courseId: id,
     mentorId,
     menteeId,
    });
    return
rep.status(HttpCode.CREATED).send(chooseMen
tor);
   },
  });
  fastify.route({
   method: HttpMethod.PUT,
   url: CoursesApiPath.$ID_MENTORS,
   schema: {
    params: courseGetParamsValidationSchema,
    body:
courseMentorCreateBodyValidationSchema,
   },
   async handler(
    req: FastifyRequest<{
     Params:
CourseSelectMentorRequestParamsDto;
     Body: CourseSelectMentorRequestDto;
    }>,
    rep,
   ) {
    const { mentorId, menteeId } = req.body;
    const { id } = req.params;
    const changeMentor = await
mentorService.changeMentor({
     courseId: id,
     mentorId,
     menteeId,
    });
    return
rep.status(HttpCode.OK).send(changeMentor);
   },
  });
  fastify.route({
   method: HttpMethod.GET,
   url:
CoursesApiPath.$ID_MENTEES_$ID_IS_MENT
OR_CHECK,
   schema: { params:
courseCheckIsMentorForStudentParamsValidation
Schema },
   async handler(
    req: FastifyRequest<{
     Params:
CourseCheckIsMentorForMenteeRequestParamsD
to;
    }>,
    rep,
   ) {
    const { courseId, menteeId } = req.params;
    const { id: mentorId } = req.user;
    const isMentorForMentee = await
mentorService.checkIsMentorForMentee({
     courseId,
     menteeId,
     mentorId,
    });

rep.status(HttpCode.OK).send(isMentorForMente
e);
   },
  });
  fastify.route({
   method: HttpMethod.GET,
   url: CoursesApiPath.POPULAR,
   async handler(_req, rep) {
    const courses = await
courseService.getPopular();
    rep.status(HttpCode.OK).send(courses);
   },
  });
};
export { initCoursesApi };
```

Файл backend/src/api/groups/groups.api.ts

```
import { FastifyPluginAsync, FastifyRequest }
from 'fastify';
import {
  GroupsApiPath,
  HttpCode,
  HttpMethod,
  PaginationDefaultValue,
  PermissionKey,
} from '~/common/enums/enums';
import {
  EntityPaginationRequestQueryDto,
  GroupsConfigureRequestDto,
  GroupsDeleteRequestParamDto,
  GroupsUpdateRequestDto,
  GroupsUpdateRequestParamsDto,
} from '~/common/types/types';
import { checkHasPermissions } from
'~/hooks/hooks';
import { group as groupService } from
'~/services/services';
import {
  groupCreate as groupCreateValidationSchema,
  groupDelete as groupsDeleteValidationSchema,
  groupGetById as
groupGetByIdValidationSchema,
  groupUpdate as groupUpdateValidationSchema,
  groupUpdateParams as
groupUpdateParamsValidationSchema,
  pagination as paginationQueryValidationSchema,
} from '~/validation-schemas/validation-schemas';
type Options = {
  services: {
    group: typeof groupService;
  };
};
const initGroupsApi:
FastifyPluginAsync<Options> = async (fastify,
opts) => {
  const { group: groupService } = opts.services;
  fastify.route({
    method: HttpMethod.POST,
    url: GroupsApiPath.ROOT,
    schema: {
      body: groupCreateValidationSchema,
    },
    preHandler: checkHasPermissions('every',
PermissionKey.MANAGE_UAM),
    async handler(
      req: FastifyRequest<{ Body:
GroupsConfigureRequestDto }>,
      rep,
    ) {
      const group = await
groupService.create(req.body);
      return
rep.status(HttpCode.CREATED).send(group);
    },
  });
  fastify.route({
    method: HttpMethod.GET,
    url: GroupsApiPath.ROOT,
    schema: {
      querystring:
paginationQueryValidationSchema,
    },
    preHandler: checkHasPermissions('every',
PermissionKey.MANAGE_UAM),
    async handler(
      req: FastifyRequest<{ Querystring:
EntityPaginationRequestQueryDto }>,
      rep,
    ) {
      const {
        page =
PaginationDefaultValue.DEFAULT_PAGE,
        count =
PaginationDefaultValue.DEFAULT_COUNT,
      } = req.query;
      const groups = await groupService.getAll({
        page,
        count,
      });
      return rep.status(HttpCode.OK).send(groups);
    },
  });
  fastify.route({
    method: HttpMethod.GET,
    url: GroupsApiPath.$ID,
    schema: { params:
groupGetByIdValidationSchema },
    preHandler: checkHasPermissions('every',
PermissionKey.MANAGE_UAM),
    async handler(req: FastifyRequest<{ Params: {
id: string } }>, rep) {
      const { id } = req.params;
      const group = await
groupService.getById(Number(id));
      return rep.status(HttpCode.OK).send(group);
    },
  });
  fastify.route({
    method: HttpMethod.PUT,
    url: GroupsApiPath.$ID,
    schema: {
      body: groupUpdateValidationSchema,
      params:
groupUpdateParamsValidationSchema,
    },
    preHandler: checkHasPermissions('every',
PermissionKey.MANAGE_UAM),
    async handler(
      req: FastifyRequest<{
```

```
      Body: GroupsUpdateRequestDto;
      Params: GroupsUpdateRequestParamsDto;
    }>,
    rep,
  ) {
    const group = await groupService.update({
      id: req.params.id,
      groupsRequestDto: req.body,
    });
    return rep.status(HttpCode.OK).send(group);
  },
 });
 fastify.route({
  method: HttpMethod.DELETE,
  url: GroupsApiPath.$ID,
  schema: { params:
groupsDeleteValidationSchema },
  preHandler: checkHasPermissions('every',
PermissionKey.MANAGE_UAM),
  async handler(
    req: FastifyRequest<{ Params:
GroupsDeleteRequestParamDto }>,
    rep,
  ) {
    const { id } = req.params;
    const isDeleted = await
groupService.delete(Number(id));
    return rep
      .status(isDeleted ? HttpCode.OK :
HttpCode.NOT_FOUND)
      .send(isDeleted);
  },
 });
};
export { initGroupsApi };
```

Файл backend/src/api/interviews/interviews.api.ts

```
import { FastifyPluginAsync, FastifyRequest }
from 'fastify';
import {
 HttpCode,
 HttpMethod,
 InterviewsApiPath,
 InterviewStatus,
 PaginationDefaultValue,
 PermissionKey,
} from '~/common/enums/enums';
import {
 EntityPaginationRequestQueryDto,
 InterviewNoteCreateRequestDto,
 InterviewNoteCreateRequestParamsDto,
 InterviewsByIdRequestParamsDto,
 InterviewsByIntervieweeIdRequestDto,
 InterviewsCreateRequestBodyDto,

InterviewsGetInterviewersByCategoryRequestDto
,
 InterviewsUpdateRequestDto,
 InterviewsUpdateRequestParamsDto,

InterviewsUpdateWithoutInterviewerRequestDto,
} from '~/common/types/types';
import { checkHasPermissions } from
'~/hooks/hooks';
import { interview as interviewService } from
'~/services/services';
import {
 interviewByIdParams as
interviewByIdParamsValidationSchema,
 interviewByIntervieweeId as
interviewByIntervieweeIdValidationSchema,
 interviewCreate as
interviewCreateValidationSchema,
 interviewGetInterviewersByCategory as
interviewGetInterviewersByCategoryValidationSc
hema,
 interviewNotesCreateArguments as
interviewNotesCreateArgumentsValidationSchem
a,
 interviewNotesCreateParams as
interviewNotesCreateParamsValidationSchema,
 interviewNotesGetAllParams as
interviewNotesGetAllParamsValidationSchema,
 interviewUpdate as
interviewUpdateValidationSchema,
 interviewUpdateParams as
interviewUpdateParamsValidationSchema,
 interviewUpdateWithoutInterviewer as
interviewUpdateWithoutInterviewerValidationSch
ema,
 pagination as paginationValidationSchema,
} from '~/validation-schemas/validation-schemas';
type Options = {
 services: {
  interview: typeof interviewService;
 };
};
const initInterviewsApi:
FastifyPluginAsync<Options> = async (
 fastify,
 opts,
) => {
 const { interview: interviewService } =
opts.services;
 fastify.route({
  method: HttpMethod.GET,
  url: InterviewsApiPath.ROOT,
  schema: { querystring:
paginationValidationSchema },
```

```
    preHandler: checkHasPermissions(
      'oneOf',
      PermissionKey.MANAGE_INTERVIEWS,
      PermissionKey.MANAGE_INTERVIEW,
    ),
    async handler(
      req: FastifyRequest<{ Querystring:
EntityPaginationRequestQueryDto }>,
      res,
    ) {
      const { id, permissions } = req.user;
      const {
        count =
PaginationDefaultValue.DEFAULT_COUNT,
        page =
PaginationDefaultValue.DEFAULT_PAGE,
      } = req.query;
      const interviews = await
interviewService.getAll({
        userId: id,
        permissions,
        count,
        page,
      });
      return
res.status(HttpCode.OK).send(interviews);
    },
  });
  fastify.route({
    method: HttpMethod.GET,
    url: InterviewsApiPath.$ID,
    preHandler: checkHasPermissions(
      'oneOf',
      PermissionKey.MANAGE_INTERVIEW,
      PermissionKey.MANAGE_INTERVIEWS,
    ),
    async handler(req: FastifyRequest<{ Params: {
id: number } }>, res) {
      const { id } = req.params;
      const interview = await
interviewService.getById(id);
      return
res.status(HttpCode.OK).send(interview);
    },
  });
  fastify.route({
    method: HttpMethod.PUT,
    url: InterviewsApiPath.$ID,
    schema: {
      body: interviewUpdateValidationSchema,
      params:
interviewUpdateParamsValidationSchema,
    },
    preHandler: checkHasPermissions('oneOf',
PermissionKey.MANAGE_INTERVIEWS),
    async handler(
      req: FastifyRequest<{
        Body: InterviewsUpdateRequestDto;
        Params:
InterviewsUpdateRequestParamsDto;
      }>,
      rep,
    ) {
      const interview = await
interviewService.update({
        id: req.params.id,
        interviewUpdateInfoRequestDto: req.body,
        user: req.user,
      });
      return
rep.status(HttpCode.OK).send(interview);
    },
  });
  fastify.route({
    method: HttpMethod.PUT,
    url:
InterviewsApiPath.$ID_UPDATE_WITHOUT_IN
TERVIEWER,
    schema: {
      body:
interviewUpdateWithoutInterviewerValidationSch
ema,
      params:
interviewUpdateParamsValidationSchema,
    },
    preHandler: checkHasPermissions(
      'oneOf',
      PermissionKey.MANAGE_INTERVIEW,
      PermissionKey.MANAGE_INTERVIEWS,
    ),
    async handler(
      req: FastifyRequest<{
        Body:
InterviewsUpdateWithoutInterviewerRequestDto;
        Params:
InterviewsUpdateRequestParamsDto;
      }>,
      rep,
    ) {
      const interview = await
interviewService.updateWithoutInterviewer({
        id: req.params.id,
        interviewUpdateInfoRequestDto: req.body,
      });
      return
rep.status(HttpCode.OK).send(interview);
    },
  });
  fastify.route({
    method: HttpMethod.POST,
    url: InterviewsApiPath.ROOT,
```

```
    schema: { body:
interviewCreateValidationSchema },
    async handler(
      req: FastifyRequest<{ Body:
InterviewsCreateRequestBodyDto }>,
      rep,
    ) {
      const { categoryId, interviewerUserId } =
req.body;
      const interview = await
interviewService.create({
        categoryId,
        interviewerUserId,
        status: InterviewStatus.NEW,
      });

rep.status(HttpCode.CREATED).send(interview);
    },
  });
  fastify.route({
    method: HttpMethod.GET,
    url:
InterviewsApiPath.INTERVIEWEE_USER_$ID_
CATEGORIES,
    schema: { params:
interviewByIntervieweeIdValidationSchema },
    async handler(
      req: FastifyRequest<{ Params:
InterviewsByIntervieweeIdRequestDto }>,
      rep,
    ) {
      const { intervieweeUserId } = req.params;
      const categoryIds =
        await
interviewService.getPassedInterviewsCategoryIds
ByUserId(
          intervieweeUserId,
        );
      rep.status(HttpCode.OK).send(categoryIds);
    },
  });
  fastify.route({
    method: HttpMethod.GET,
    url:
InterviewsApiPath.INTERVIEWEE_USER_$ID_
ACTIVE_CATEGORIES,
    schema: { params:
interviewByIntervieweeIdValidationSchema },
    async handler(
      req: FastifyRequest<{ Params:
InterviewsByIntervieweeIdRequestDto }>,
      rep,
    ) {
      const { intervieweeUserId } = req.params;
      const categoryIds =
        await
interviewService.getActiveInterviewsCategoryIds
ByUserId(
          intervieweeUserId,
        );
      rep.status(HttpCode.OK).send(categoryIds);
    },
  });
  fastify.route({
    method: HttpMethod.GET,
    url:
InterviewsApiPath.INTERVIEWERS_CATEGOR
IES_$ID,
    schema: { params:
interviewGetInterviewersByCategoryValidationSc
hema },
    preHandler: checkHasPermissions('oneOf',
PermissionKey.MANAGE_INTERVIEWS),
    async handler(
      req: FastifyRequest<{
        Params:
InterviewsGetInterviewersByCategoryRequestDto
;
      }>,
      rep,
    ) {
      const { categoryId } = req.params;
      const interviewers = await
interviewService.getInterviewersByCategoryId(
        categoryId,
      );
      rep.status(HttpCode.OK).send(interviewers);
    },
  });
  fastify.route({
    method: HttpMethod.GET,
    url:
`${InterviewsApiPath.$ID}${InterviewsApiPath.
NOTES}`,
    schema: { params:
interviewNotesGetAllParamsValidationSchema },
    preHandler: checkHasPermissions(
      'oneOf',
      PermissionKey.MANAGE_INTERVIEW,
      PermissionKey.MANAGE_INTERVIEWS,
    ),
    async handler(
      req: FastifyRequest<{ Params:
InterviewNoteCreateRequestParamsDto }>,
      rep,
    ) {
      const { id: interviewId } = req.params;
      const notesDto = await
interviewService.getAllNotes(interviewId);
      return
rep.status(HttpCode.OK).send(notesDto);
```

```
    },
  });
  fastify.route({
    method: HttpMethod.POST,
    url:
`${InterviewsApiPath.$ID}${InterviewsApiPath.
NOTES}`,
    schema: {
      params:
interviewNotesCreateParamsValidationSchema,
      body:
interviewNotesCreateArgumentsValidationSchem
a,
    },
    preHandler: checkHasPermissions(
      'oneOf',
      PermissionKey.MANAGE_INTERVIEW,
      PermissionKey.MANAGE_INTERVIEWS,
    ),
    async handler(
      req: FastifyRequest<{
        Params:
InterviewNoteCreateRequestParamsDto;
        Body: InterviewNoteCreateRequestDto;
      }>,
      rep,
    ) {
      const { id: authorId } = req.user;
      const { id: interviewId } = req.params;
      const { note } = req.body;
      const newNote = await
interviewService.createNote({
        note,
        interviewId,
        authorId,
      });
      return
rep.status(HttpCode.CREATED).send(newNote);
    },
  });
```

Файл backend/src/api/mentors/mentors.api.ts

```
import { FastifyPluginAsync, FastifyRequest }
from 'fastify';
import { HttpCode, HttpMethod, MentorsApiPath
} from '~/common/enums/enums';
import {
  CoursesToMentorsRequestDto,
  GetMentorRequestParamsDto,
} from '~/common/types/types';
import { mentor as mentorService } from
'~/services/services';
import {
  getMentor as getMentorValidationSchema,
  mentorCreateBody as
mentorCreateBodyValidationSchema,
} from '~/validation-schemas/validation-schemas';
type Options = {
  services: {
    mentor: typeof mentorService;
  };
};
const initMentorsApi:
FastifyPluginAsync<Options> = async (fastify,
opts) => {
  const { mentor: mentorService } = opts.services;
  fastify.route({
    method: HttpMethod.POST,
    url: MentorsApiPath.ROOT,
```

```
    schema: { body:
mentorCreateBodyValidationSchema },
    async handler(
      req: FastifyRequest<{ Body:
CoursesToMentorsRequestDto }>,
      rep,
    ) {
      const { courseId, userId } = req.body;

      const addedToCourseMentor = await
mentorService.addMentorToCourse({
        courseId,
        userId,
      });

rep.status(HttpCode.CREATED).send(addedToCo
urseMentor);
    },
  });
  fastify.route({
    method: HttpMethod.GET,
    url:
MentorsApiPath.COURSES_$ID_MENTEES_$I
D,
    schema: { params: getMentorValidationSchema
},
    async handler(
      req: FastifyRequest<{ Params:
GetMentorRequestParamsDto }>,
      rep,
    ) {
      const { menteeId, courseId } = req.params;
      const menteeToMentor = await
mentorService.getMentor({
        courseId,
        menteeId,
      });
      return
rep.status(HttpCode.OK).send(menteeToMentor);
    },
  });
};
export { initMentorsApi }
```

Файл backend/src/api/permissions/permissions.api.ts

```
import { FastifyPluginAsync, FastifyRequest }
from 'fastify';
import {
  HttpCode,
  HttpMethod,
  PaginationDefaultValue,
  PermissionApiPath,
} from '~/common/enums/enums';
import { EntityPaginationRequestQueryDto }
from '~/common/types/types';
import { permission as permissionService } from
'~/services/services';
import { pagination as
paginationValidationSchema } from
'~/validation-schemas/validation-schemas';
type Options = {
  services: {
    permission: typeof permissionService;
  };
};
const initPermissionsApi:
FastifyPluginAsync<Options> = async (
  fastify,
  opts,
) => {
  const { permission: permissionService } =
opts.services;
  fastify.route({
    method: HttpMethod.GET,
    url: PermissionApiPath.ROOT,
    schema: { querystring:
paginationValidationSchema },
    async handler(
      req: FastifyRequest<{ Querystring:
EntityPaginationRequestQueryDto }>,
      rep,
    ) {
      const {
        page =
PaginationDefaultValue.DEFAULT_PAGE,
        count =
PaginationDefaultValue.DEFAULT_COUNT,
      } = req.query;
      const permissions = await
permissionService.getAll({ page, count });
      return
rep.status(HttpCode.OK).send(permissions);
    },
  });
};
export { initPermissionsApi };
```

Файл backend/src/api/tasks/tasks.api.ts

```
import { FastifyPluginAsync, FastifyRequest }
from 'fastify';
import {
  HttpCode,
  HttpMethod,
  PaginationDefaultValue,
  TasksApiPath,
} from '~/common/enums/enums';
```

```
import {
  EntityPaginationRequestQueryDto,
  TaskByIdRequestParamsDto,
  TaskGetByMenteeIdAndModuleId,

TaskGetByMenteeIdCourseIdModuleIdRequestDt
o,
  TaskNoteManipulateRequestBodyDto,
  TasksGetByCourseIdAndMenteeIdRequestDto,
} from '~/common/types/types';
import { task as taskService } from
'~/services/services';
import {
  pagination as paginationValidationSchema,
  taskByMenteeIdAndModuleId as
taskByMenteeIdAndModuleIdValidationSchema,
  taskByMenteeIdCourseIdModuleIdParams as
taskByMenteeIdCourseIdModuleIdParamsValidati
onSchema,
  tasksByCourseIdAndMenteeId as
tasksByCourseIdAndMenteeIdValidationSchema,
  tasksByIdParams as
tasksByIdParamsValidationSchema,
  tasksManipulateRequestBody as
tasksManipulateRequestBodyValidationSchema,
} from '~/validation-schemas/validation-schemas';
type Options = {
  services: {
    task: typeof taskService;
  };
};
const initTasksApi: FastifyPluginAsync<Options>
= async (fastify, opts) => {
  const { task: taskService } = opts.services;
  fastify.route({
    method: HttpMethod.POST,
    url: TasksApiPath.TASKS_$ID,
    schema: {
      params: tasksByIdParamsValidationSchema,
      body:
tasksManipulateRequestBodyValidationSchema,
    },
    async handler(
      req: FastifyRequest<{
        Body: TaskNoteManipulateRequestBodyDto;
        Params: TaskByIdRequestParamsDto;
      }>,
      rep,
    ) {
      const { note, status } = req.body;
      const { taskId } = req.params;
      const { id: authorId } = req.user;
      const newNote = await
taskService.manipulate({
        note,
        authorId,
        taskId,
        status,
      });

rep.status(HttpCode.CREATED).send(newNote);
    },
  });
  fastify.route({
    method: HttpMethod.GET,
    url:
TasksApiPath.MODULES_$ID_MENTEES_$ID,
    schema: { params:
taskByMenteeIdAndModuleIdValidationSchema
},
    async handler(
      req: FastifyRequest<{ Params:
TaskGetByMenteeIdAndModuleId }>,
      rep,
    ) {
      const { menteeId, moduleId } = req.params;
      const task = await
taskService.getByMenteeIdAndModuleId({
        menteeId,
        moduleId,
      });
      rep.status(HttpCode.OK).send(task);
    },
  });
  fastify.route({
    method: HttpMethod.GET,
    url: TasksApiPath.TASKS_$ID_NOTES,
    schema: {
      params: tasksByIdParamsValidationSchema,
      querystring: paginationValidationSchema,
    },
    async handler(
      req: FastifyRequest<{
        Params: TaskByIdRequestParamsDto;
        Querystring:
EntityPaginationRequestQueryDto;
      }>,
      rep,
    ) {
      const { taskId } = req.params;
      const {
        count =
PaginationDefaultValue.DEFAULT_COUNT,
        page =
PaginationDefaultValue.DEFAULT_PAGE,
      } = req.query;
      const notes = await taskService.getAllNotes({
taskId, count, page });

      rep.status(HttpCode.OK).send(notes);
    },
  });
```

```
  fastify.route({
    method: HttpMethod.GET,
    url:
TasksApiPath.COURSES_$ID_MODULES_$ID_
MENTEES_$ID,
    schema: { params:
taskByMenteeIdCourseIdModuleIdParamsValidati
onSchema },
    async handler(
      req: FastifyRequest<{
        Params:
TaskGetByMenteeIdCourseIdModuleIdRequestDt
o;
      }>,
      rep,
    ) {
      const { courseId, menteeId, moduleId } =
req.params;
      const task = await
taskService.getByMenteeIdCourseIdModuleId({
        courseId,
        menteeId,
        moduleId,
      });
      rep.status(HttpCode.OK).send(task);
    },
  });
  fastify.route({
    method: HttpMethod.GET,
    url:
TasksApiPath.COURSES_$ID_MENTEES_$ID,
    schema: { params:
tasksByCourseIdAndMenteeIdValidationSchema
},
    async handler(
      req: FastifyRequest<{ Params:
TasksGetByCourseIdAndMenteeIdRequestDto }>,
      rep,
    ) {
      const { courseId, menteeId } = req.params;
      const tasks = await
taskService.getAllByCourseIdAndMenteeId({
        courseId,
        menteeId,
      });

      rep.status(HttpCode.OK).send(tasks);
    },
  });
};
export { initTasksApi };
```

Файл backend/src/api/user-details/user-details.api.ts

```
import { FastifyPluginAsync, FastifyRequest }
from 'fastify';
import { HttpCode, HttpMethod,
UserDetailsApiPath } from
'~/common/enums/enums';
import {
  UserDetailsUpdateAvatarRequestParamsDto,
  UserDetailsUpdateInfoRequestDto,
} from '~/common/types/types';
import { userDetails as userDetailsService } from
'~/services/services';
import {
  userDetailsUpdateInfo as
userDetailsUpdateInfoValidationSchema,
  userDetailsUpdateParams as
userDetailsUpdateParamsValidationSchema,
} from '~/validation-schemas/validation-schemas';
type Options = {
  services: {
    userDetails: typeof userDetailsService;
  };
};
const initUserDetailsApi:
FastifyPluginAsync<Options> = async (
  fastify,
  opts,
) => {
  const { userDetails: userDetailsService } =
opts.services;
  fastify.route({
    method: HttpMethod.GET,
    url: UserDetailsApiPath.ROOT,
    async handler(req, rep) {
      const userDetails = await
userDetailsService.getByUserId(req.user.id);
      return
rep.status(HttpCode.OK).send(userDetails);
    },
  });
  fastify.route({
    method: HttpMethod.PUT,
    url: UserDetailsApiPath.ROOT,
    schema: {
      body:
userDetailsUpdateInfoValidationSchema,
    },
    async handler(
      req: FastifyRequest<{ Body:
UserDetailsUpdateInfoRequestDto }>,
      rep,
    ) {
```

```
      const userDetails = await
userDetailsService.update(
        req.user.id,
        req.body,
      );
      return rep.send(userDetails);
    },
  });
  fastify.route({
    method: HttpMethod.PUT,
    url: UserDetailsApiPath.USER_$ID_AVATAR,
    schema: { params:
userDetailsUpdateParamsValidationSchema },
    async handler(
      req: FastifyRequest<{ Params:
UserDetailsUpdateAvatarRequestParamsDto }>,
      rep,
    ) {
      const { fileBuffer } = req;
      const { userId } = req.params;
      const updatedUserDetails = await
userDetailsService.uploadAvatar(
        userId,
        fileBuffer,
      );

rep.status(HttpCode.OK).send(updatedUserDetails
);
    },
  });
};
export { initUserDetailsApi };
```

Файл backend/src/api/users/users.api.ts

```
import { FastifyPluginAsync, FastifyRequest }
from 'fastify';
import {
  HttpCode,
  HttpMethod,
  PaginationDefaultValue,
  PermissionKey,
  UsersApiPath,
} from '~/common/enums/enums';
import {
  EntityPaginationRequestQueryDto,
  UsersDeleteRequestParamsDto,
} from '~/common/types/types';
import { checkHasPermissions } from
'~/hooks/hooks';
import { user as userService } from
'~/services/services';
import {
  pagination as paginationQueryValidationSchema,
  userDelete as
userDeleteRequestParamsValidationSchema,
} from '~/validation-schemas/validation-schemas';
type Options = {
  services: {
    user: typeof userService;
  };
};
const initUsersApi: FastifyPluginAsync<Options>
= async (fastify, opts) => {
  const { user: userService } = opts.services;
  fastify.route({
    method: HttpMethod.GET,
    url: UsersApiPath.ROOT,
    schema: {
      querystring:
paginationQueryValidationSchema,
    },
    async handler(
      req: FastifyRequest<{
        Querystring:
EntityPaginationRequestQueryDto;
      }>,
      rep,
    ) {
      const {
        page =
PaginationDefaultValue.DEFAULT_PAGE,
        count =
PaginationDefaultValue.DEFAULT_COUNT,
      } = req.query;
      const users = await userService.getAll({
        page,
        count,
      });
      return rep.status(HttpCode.OK).send(users);
    },
  });
  fastify.route({
    method: HttpMethod.DELETE,
    url: UsersApiPath.$ID,
    schema: { params:
userDeleteRequestParamsValidationSchema },
    preHandler: checkHasPermissions('every',
PermissionKey.MANAGE_UAM),
    async handler(
      req: FastifyRequest<{ Params:
UsersDeleteRequestParamsDto }>,
      rep,
    ) {
      const { id } = req.params;
      const isDeleted = await
userService.delete(req.user, Number(id));
      return rep
```

```
      .status(isDeleted ? HttpCode.OK :
HttpCode.NOT_FOUND)
      .send(isDeleted);
```

```
    },
  });
};
export { initUsersApi };
```

Файл backend/src/common/constants/allowed-image-extensions.constants.ts

```
import { ContentType } from '~/common/enums/enums';
const ALLOWED_IMAGE_EXTENSIONS: ContentType[] = [
  ContentType.IMAGE_JPEG,
  ContentType.IMAGE_PNG,
  ContentType.IMAGE_SVG,
];
export { ALLOWED_IMAGE_EXTENSIONS };
```

Файл backend/src/common/constants/api.constants.ts

```
import {
  ApiPath,
  AuthApiPath,
  CourseModulesApiPath,
  CoursesApiPath,
  ENV,
  HttpMethod,
  InterviewsApiPath,
  MentorsApiPath,
} from '~/common/enums/enums';
import { WhiteRoute } from '~/common/types/types';
const WHITE_ROUTES: WhiteRoute[] = [
  {
    route: `${ENV.API.V1_PREFIX}${ApiPath.AUTH}${AuthApiPath.GOOGLE_SIGN}`,
    methods: [HttpMethod.GET],
  },
  {
    route: `${ENV.API.V1_PREFIX}${ApiPath.AUTH}${AuthApiPath.GOOGLE_CALLBACK}`,
    methods: [HttpMethod.GET],
  },
  {
    route: `${ENV.API.V1_PREFIX}${ApiPath.AUTH}${AuthApiPath.GITHUB_SIGN}`,
    methods: [HttpMethod.GET],
  },
  {
    route: `${ENV.API.V1_PREFIX}${ApiPath.AUTH}${AuthApiPath.GITHUB_CALLBACK}`,
    methods: [HttpMethod.GET],
  },
  {
    route: `${ENV.API.V1_PREFIX}${ApiPath.AUTH}${AuthApiPath.SIGN_IN}`,
    methods: [HttpMethod.POST],
  },
  {
    route: `${ENV.API.V1_PREFIX}${ApiPath.AUTH}${AuthApiPath.SIGN_UP}`,
    methods: [HttpMethod.POST],
  },
  {
    route: `${ENV.API.V1_PREFIX}${ApiPath.CATEGORIES}`,
    methods: [HttpMethod.GET],
  },
```

```
  {
    route: `${ENV.API.V1_PREFIX}${ApiPath.CATEGORIES}${CoursesApiPath.$ID}`,
    methods: [HttpMethod.GET],
  },
  {
    route: `${ENV.API.V1_PREFIX}${ApiPath.COURSES}${ApiPath.DASHBOARD}`,
    methods: [HttpMethod.GET],
  },
  {
    route: `${ENV.API.V1_PREFIX}${ApiPath.CATEGORIES}${ApiPath.DASHBOARD}`,
    methods: [HttpMethod.GET],
  },
  {
    route: `${ENV.API.V1_PREFIX}${ApiPath.COURSES}${CoursesApiPath.ROOT}`,
    methods: [HttpMethod.GET],
  },
  {
    route: `${ENV.API.V1_PREFIX}${ApiPath.COURSES}${CoursesApiPath.$ID}`,
    methods: [HttpMethod.GET],
  },
  {
    route: `${ENV.API.V1_PREFIX}${ApiPath.COURSES}${CoursesApiPath.POPULAR}`,
    methods: [HttpMethod.GET],
  },
  {
    route:
`${ENV.API.V1_PREFIX}${ApiPath.COURSE_MODULES}${CourseModulesApiPath.COURSES_$ID_
MODULES}`,
    methods: [HttpMethod.GET],
  },
  {
    route:
`${ENV.API.V1_PREFIX}${ApiPath.INTERVIEWS}${InterviewsApiPath.INTERVIEWEE_USER_$ID_
CATEGORIES}`,
    methods: [HttpMethod.GET],
  },
  {
    route: `${ENV.API.V1_PREFIX}${ApiPath.MENTORS}${MentorsApiPath.ROOT}`,
    methods: [HttpMethod.GET],
  },
];
export { WHITE_ROUTES };
```

Файл backend/src/common/constants/user.constants.ts

```
const USER_PASSWORD_SALT_ROUNDS = 10;
export { USER_PASSWORD_SALT_ROUNDS };
```

Файл backend/src/common/enums/api/controller-hook.enum.ts

```
const ControllerHook = {
  ON_REQUEST: 'onRequest',
  PRE_PARSING: 'preParsing',
  PRE_VALIDATION: 'preValidation',
  PRE_HANDLER: 'preHandler',
  HANDLER: 'handler',
  PRE_SERIALIZATION: 'preSerialization',
  ON_ERROR: 'onError',
  ON_SEND: 'onSend',
```

```
  ON_RESPONSE: 'onResponse',
  ON_TIMEOUT: 'onTimeout',
} as const;
export { ControllerHook };
```

Файл backend/src/common/enums/app/app-environment.enum.ts

```
enum AppEnvironment {
  DEVELOPMENT = 'development',
  PRODUCTION = 'production',
}
export { AppEnvironment };
```

Файл backend/src/common/enums/app/env.enum.ts

```
import { config } from 'dotenv';
import { BillingApiVersion } from
'~/common/types/types';
import { AppEnvironment } from
'./app-environment.enum';
config();
const {
  NODE_ENV,
  PORT,
  HOST,
  DATABASE_URL,
  DB_POOL_MIN,
  DB_POOL_MAX,
  DB_DIALECT,
  SECRET_KEY,
  FRONTEND_URL,
  UDEMY_CLIENT_ID,
  UDEMY_CLIENT_SECRET,
  UDEMY_BASE_URL,
  EDX_CLIENT_ID,
  EDX_CLIENT_SECRET,
  EDX_BASE_URL,
  GOOGLE_CLIENT_ID,
  GOOGLE_CLIENT_SECRET,
  GITHUB_CLIENT_ID,
  GITHUB_CLIENT_SECRET,
  AWS_ACCESS_KEY_ID,
  AWS_SECRET_ACCESS_KEY,
  AWS_REGION,
  AWS_USERS_FILES_BUCKET_NAME,
  STRIPE_SECRET_KEY,
  STRIPE_API_VERSION,
} = process.env;
const ENV = {
  APP: {
    NODE_ENV: <AppEnvironment>NODE_ENV,
    SERVER_PORT: Number(PORT),
    SERVER_HOST: HOST,
    BASE_URL: `http://${HOST}:${PORT}`,
    FRONTEND_URL: FRONTEND_URL as
string,
  },
  DB: {
    CONNECTION_STRING: DATABASE_URL,
    POOL_MIN: Number(DB_POOL_MIN),
    POOL_MAX: Number(DB_POOL_MAX),
    DIALECT: DB_DIALECT,
  },
  API: {
    V1_PREFIX: '/api/v1',
  },
  JWT: {
    SECRET: SECRET_KEY as string,
    EXPIRES_IN: '24h',
    ALG: 'HS256',
  },
  GOOGLE: {
    CLIENT_ID: GOOGLE_CLIENT_ID as string,
    CLIENT_SECRET:
GOOGLE_CLIENT_SECRET as string,
  },
  GITHUB: {
    CLIENT_ID: GITHUB_CLIENT_ID as string,
    CLIENT_SECRET:
GITHUB_CLIENT_SECRET as string,
  },
  UDEMY: {
    BASE_URL: UDEMY_BASE_URL as string,
    CLIENT_ID: UDEMY_CLIENT_ID as string,
    CLIENT_SECRET:
UDEMY_CLIENT_SECRET as string,
  },
  EDX: {
    BASE_URL: EDX_BASE_URL as string,
    CLIENT_ID: EDX_CLIENT_ID as string,
    CLIENT_SECRET: EDX_CLIENT_SECRET
as string,
  },
  AWS: {
    ACCESS_KEY_ID: AWS_ACCESS_KEY_ID
as string,
    SECRET_ACCESS_KEY:
AWS_SECRET_ACCESS_KEY as string,
    REGION: AWS_REGION as string,
    USERS_FILES_BUCKET_NAME:
AWS_USERS_FILES_BUCKET_NAME as
string,
  },
```

```
  STRIPE: {
    SECRET_KEY: STRIPE_SECRET_KEY as string,
    API_VERSION: STRIPE_API_VERSION as BillingApiVersion,
  },
};
export { ENV };
```

Файл backend/src/common/enums/app/log-level.enum.ts

```
enum LogLevel {
  DEBUG = 'debug',
  ERROR = 'error',
  FATAL = 'fatal',
  INFO = 'info',
  TRACE = 'trace',
  WARN = 'warn',
}
export { LogLevel };
```

Файл backend/src/common/enums/course/course-host.enum.ts

```
enum CourseHost {
  W_UDEMY = 'www.udemy.com',
  UDEMY = 'udemy.com',
  EDX = 'www.edx.org',
}
export { CourseHost };
```

Файл backend/src/common/enums/db/db-table-name.enum.ts

```
enum DbTableName {
  USERS = 'users',
  USER_DETAILS = 'user_details',
  PERMISSIONS = 'permissions',
  MIGRATIONS = 'migrations',
  CHAT_MESSAGES = 'chat_messages',
  COURSE_CATEGORIES = 'course_categories',
  COURSE_CATEGORIES_PRICES = 'course_categories_prices',
  COURSES = 'courses',
  COURSE_MODULES = 'course_modules',
  GROUPS_TO_PERMISSIONS = 'groups_to_permissions',
  USERS_TO_GROUPS = 'users_to_groups',
  GROUPS = 'groups',
  VENDORS = 'vendors',
  INTERVIEWS = 'interviews',
  INTERVIEW_NOTES = 'interview_notes',
  FILES = 'files',
  COURSES_TO_MENTORS = 'courses_to_mentors',
  MENTEES_TO_MENTORS = 'mentees_to_mentors',
  TASKS = 'tasks',
  TASK_NOTES = 'task_notes',
  TRANSACTIONS = 'transactions',
}
export { DbTableName };
```

Файл backend/src/common/enums/file/file-size-bytes-value.enum.ts

```
enum FileSizeBytesValue {
  ONE_MB = 1000000,
  FIVE_MB = 1024 * 1024 * 5,
}
export { FileSizeBytesValue };
```

Файл backend/src/common/enums/group/protected-group-key.enum.ts

```
enum ProtectedGroupKey {
  MENTORS = 'mentors',
}
```

```
export { ProtectedGroupKey };
```

Файл backend/src/common/types/api/white-route.type.ts

```
import { HttpMethod } from '~/common/enums/enums';
type WhiteRoute = {
  route: string;
  methods: HttpMethod[];
};
export { type WhiteRoute };
```

Файл backend/src/common/types/billing/billing-api-version.type.ts

```
type BillingApiVersion = '2022-08-01';
export { type BillingApiVersion };
```

Файл backend/src/common/types/billing/billing-init-hold-student-payment-arguments-dto.type.ts

```
type BillingInitHoldStudentPaymentArgumentsDto = {
  menteeId: number;
  mentorId: number;
  rawPriceOfStudying: number;
};
export { type BillingInitHoldStudentPaymentArgumentsDto };
```

Файл backend/src/common/types/billing/billing-replenish-arguments-dto.type.ts

```
import { BillingReplenishToken } from './billing';
type BillingReplenishArgumentsDto = {
  userId: number;
  amountOfMoneyToReplenish: number;
  token: BillingReplenishToken;
};
export { type BillingReplenishArgumentsDto };
```

Файл backend/src/common/types/chat-message/chat-message-get-empty-chats-request-dto.type.ts

```
import { ChatMessageGetAllItemResponseDto } from '~/common/types/types';
type ChatMessageGetEmptyChatsRequestDto = {
  userId: number;
  fullName: string;
  lastMessagesInChats: ChatMessageGetAllItemResponseDto[];
};
export { type ChatMessageGetEmptyChatsRequestDto };
```

Файл backend/src/common/types/common/id-container/id-container.type.ts

```
type IdContainer = {
  id: number;
};
export { IdContainer };
```

Файл

backend/src/common/types/common/numerical-value-container/numerical-value-conteinter.type.ts

```
type NumericalValueContainer = {
  value: number;
};
export { type NumericalValueContainer };
```

Файл backend/src/common/types/course/course-all-mentors-dto.ts

```
import { UsersGetResponseDto } from '~/common/types/types';
type CourseAllMentorsDto = {
  mentors: UsersGetResponseDto[];
};
```

```
export { type CourseAllMentorsDto };
```

Файл backend/src/common/types/course/course-create-arguments-dto.type.ts

```
import { VendorKey } from '~/common/enums/enums';
type CourseCreateArgumentsDto = {
  title: string;
  description: string;
  url: string;
  vendorKey: VendorKey;
  originalId: string;
  imageUrl: string | null;
};
export { type CourseCreateArgumentsDto };
```

Файл backend/src/common/types/course/course-create-request-arguments-dto.ts

```
type CourseCreateRequestArgumentsDto = {
  title: string;
  description: string;
  url: string;
  vendorId: number;
  courseCategoryId?: number;
  originalId: string;
  imageUrl: string | null;
};
export { type CourseCreateRequestArgumentsDto };
```

Файл backend/src/common/types/course/course-get-by-id-and-vendor-key-arguments-dto.type.ts

```
import { VendorKey } from '~/common/enums/enums';
type CourseGetByIdAndVendorKeyArgumentsDto = {
  originalId: string;
  vendorKey: VendorKey;
};
export { type CourseGetByIdAndVendorKeyArgumentsDto };
```

Файл backend/src/common/types/course/get-all-mentees-dto.ts

```
import { UsersGetResponseDto } from '~/common/types/types';
type GetAllMenteesDto = {
  mentees: UsersGetResponseDto[];
};
export { type GetAllMenteesDto };
```

Файл backend/src/common/types/course-module/course-module-create-arguments-dto.type.ts

```
type CourseModuleCreateArgumentsDto = {
  title: string;
  courseId: number;
  description: string | null;
};
export { type CourseModuleCreateArgumentsDto };
```

Файл backend/src/common/types/edx/edx-course-get-response-dto.type.ts

```
type EdxCourseGetResponseDto = {
  course_id: string;
  name: string;
  description: string;
};
export { type EdxCourseGetResponseDto };
```

Файл backend/src/common/types/encryption/encryption-data.type.ts

```
type EncryptionData = {
  data: string;
  salt: string;
  passwordHash: string;
};
export { type EncryptionData };
```
Файл backend/src/common/types/file/file-get-url-request-dto.type.ts

```
type FileGetUrlRequestDto = {
  bucket: string;
  fileName: string;
};
export { type FileGetUrlRequestDto };
```
Файл backend/src/common/types/file/file-table-insert-request-dto.type.ts

```
import { ContentType } from '~/common/enums/enums';
type FileTableInsertRequestDto = {
  url: string;
  contentType: ContentType;
};
export { type FileTableInsertRequestDto };
```
Файл backend/src/common/types/file/file-upload-request-dto.type.ts

```
import { ContentType } from '~/common/enums/enums';
type FileUploadRequestDto = {
  bucket: string;
  file: Buffer;
  fileName: string;
  contentType: ContentType;
};
export { type FileUploadRequestDto };
```
Файл backend/src/common/types/groups/groups-with-permission-ids-dto.type.ts

```
type GroupsWithPermissionIdsDto = {
  id: number;
  name: string;
  key: string;
  permissionIds: number[];
};
export { type GroupsWithPermissionIdsDto };
```
Файл backend/src/common/types/groups/groups-with-permissions-dto.type.ts

```
import { Group, Permission } from '~/data/models/models';
type GroupsWithPermissionsDto = Group & {
  permissions: Permission[];
};
export { GroupsWithPermissionsDto };
```
Файл backend/src/common/types/interview/interviews-create-request-dto.type.ts

```
import { InterviewStatus } from '~/common/enums/enums';
type InterviewsCreateRequestDto = {
  status: InterviewStatus;
  intervieweeUserId: number;
  categoryId: number;
};
export { type InterviewsCreateRequestDto };
```
Файл backend/src/common/types/interview/interviews-get-all-request-dto.type.ts

```
import {
```

```
  EntityPaginationRequestQueryDto,
  PermissionsGetAllItemResponseDto,
} from '~/common/types/types';
type InterviewsGetAllRequestDto = {
  userId: number;
  permissions: PermissionsGetAllItemResponseDto[];
} & EntityPaginationRequestQueryDto;
export { type InterviewsGetAllRequestDto };
```

Файл backend/src/common/types/interview/interviews-get-by-user-id-request-dto.type.ts

```
import { EntityPaginationRequestQueryDto } from '~/common/types/types';
type InterviewsGetByUserIdRequestDto = {
  userId: number;
} & EntityPaginationRequestQueryDto;
export { type InterviewsGetByUserIdRequestDto };
```

Файл backend/src/common/types/interview/interviews-get-other-request-arguments-dto.type.ts

```
import { EntityPaginationRequestQueryDto } from '~/common/types/types';
type InterviewsGetOtherRequestArgumentsDto = {
  intervieweeUserId: number;
  interviewId: number;
} & EntityPaginationRequestQueryDto;
export { type InterviewsGetOtherRequestArgumentsDto };
```

Файл backend/src/common/types/interview-note/interview-note-create-request-arguments-dto.type.ts

```
type InterviewNoteCreateRequestArgumentsDto = {
  note: string;
  interviewId: number;
  authorId: number;
};
export { type InterviewNoteCreateRequestArgumentsDto };
```

Файл

backend/src/common/types/mentees-to-mentors/mentees-to-mentors-change-status-request-dto.type.ts

```
import { MenteesToMentorsStatus } from '~/common/enums/enums';
type MenteesToMentorsChangeStatusRequestDto = {
  id: number;
  status: MenteesToMentorsStatus;
};
export { type MenteesToMentorsChangeStatusRequestDto };
```

Файл backend/src/common/types/stripe/stripe-replenish-arguments-dto.type.ts

```
import { BillingReplenishToken } from '../types';
type StripeReplenishArgumentsDto = {
  amount: number;
  token: BillingReplenishToken;
};
export { type StripeReplenishArgumentsDto };
```

Файл backend/src/common/types/task-note/task-note-create-arguments-dto.type.ts

```
import { TaskStatus } from '~/common/enums/enums';
type TaskNoteCreateArgumentsDto = {
  note: string;
  taskId: number;
  authorId: number;
  status: TaskStatus;
};
export { type TaskNoteCreateArgumentsDto };
```

Файл backend/src/common/types/task-note/task-note-get-all-arguments-dto.type.ts

```
import { EntityPaginationRequestQueryDto } from '~/common/types/types';
type TaskNoteGetAllArgumentsDto = {
  taskId: number;
} & EntityPaginationRequestQueryDto;
export { type TaskNoteGetAllArgumentsDto };
```

Файл backend/src/common/types/token/token-payload.type.ts

```
type TokenPayload = {
  userId: number;
};
export { type TokenPayload };
```

Файл backend/src/common/types/user/user-count-request-dto.type.ts

```
type UserCountRequestDto = {
  id: number;
  count: number;
};
export { type UserCountRequestDto };
```

Файл backend/src/common/types/user/user-sign-oauth-request-dto.type.ts

```
import { UserSignUpRequestDto } from 'guruhub-shared/common/types/types';
type UserSignOAuthRequestDto = Omit<UserSignUpRequestDto, 'password'> &
  Partial<Record<'googleId' | 'githubId', number>>;
export { type UserSignOAuthRequestDto };
```

Файл backend/src/common/types/user/users-by-email-response-dto.type.ts

```
import { UserDetailsResponseDto } from '~/common/types/types';
type UsersByEmailResponseDto = {
  id: number;
  email: string;
  passwordHash: string;
  passwordSalt: string;
  createdAt: string;
  userDetails: UserDetailsResponseDto;
};
export { type UsersByEmailResponseDto };
```

Файл backend/src/data/models/abstract/abstract.model.ts

```
import { Model } from 'objection';
class Abstract extends Model {
  public 'id': number;
  public 'createdAt': string;
  public 'updatedAt': string;
  public override $beforeInsert(): void {
    this.createdAt = new Date().toISOString();
    this.updatedAt = new Date().toISOString();
  }
  public override $beforeUpdate(): void {
    this.updatedAt = new Date().toISOString();
  }
}

export { Abstract };
```

Файл backend/src/data/models/chat-message/chat-message.model.ts

```
import { Model, RelationMappings } from
'objection';
import { ChatMessageStatus, DbTableName }
from '~/common/enums/enums';
import { Abstract } from
'../abstract/abstract.model';
import { User } from '../models';
class ChatMessage extends Abstract {
  public 'message': string;
  public 'chatId': string;
  public 'senderId': number;
  public 'receiverId': number;
  public 'status': ChatMessageStatus;
  public static override get relationMappings():
RelationMappings {
    return {
      sender: {
        relation: Model.BelongsToOneRelation,
        modelClass: User,
        join: {
          from:
`${DbTableName.CHAT_MESSAGES}.senderId`
,
          to: `${DbTableName.USERS}.id`,
        },
      },
      receiver: {
        relation: Model.BelongsToOneRelation,
        modelClass: User,
        join: {
          from:
`${DbTableName.CHAT_MESSAGES}.receiverI
d`,
          to: `${DbTableName.USERS}.id`,
        },
      },
    };
  }
  public static override get tableName(): string {
    return DbTableName.CHAT_MESSAGES;
  }
}
export { ChatMessage };
```

Файл

backend/src/data/models/course/course.model.t

s

```
import { Model, RelationMappings } from
'objection';
import { DbTableName } from
'~/common/enums/enums';
import { Abstract } from
'../abstract/abstract.model';
import { CourseCategory, User, Vendor } from
'../models';
class Course extends Abstract {
  public 'title': string;
  public 'description': string;
  public 'url': string;
  public 'imageUrl': string | null;
  public 'vendorId': number;
  public 'courseCategoryId': number;
  public 'originalId': string;
  public static override get relationMappings():
RelationMappings {
    return {
      vendor: {
        relation: Model.HasOneRelation,
        modelClass: Vendor,
        join: {
          from:
`${DbTableName.COURSES}.vendorId`,
          to: `${DbTableName.VENDORS}.id`,
        },
      },
      mentors: {
        relation: Model.ManyToManyRelation,
        modelClass: User,
        join: {
          from: `${DbTableName.COURSES}.id`,
          through: {
            from:
`${DbTableName.COURSES_TO_MENTORS}.c
ourseId`,
            to:
`${DbTableName.COURSES_TO_MENTORS}.u
serId`,
            extra: {
              studentsCount: 'studentsCount',
            },
          },
          to: `${DbTableName.USERS}.id`,
        },
      },
      mentees: {
        relation: Model.ManyToManyRelation,
        modelClass: User,
        join: {
          from: `${DbTableName.COURSES}.id`,
          through: {
            from:
`${DbTableName.MENTEES_TO_MENTORS}.c
ourseId`,
            to:
`${DbTableName.MENTEES_TO_MENTORS}.
menteeId`,
          },
          to: `${DbTableName.USERS}.id`,
```

```
      },
    },
    mentorsWithMentees: {
      relation: Model.ManyToManyRelation,
      modelClass: User,
      join: {
        from: `${DbTableName.COURSES}.id`,
        through: {
          from:
`${DbTableName.MENTEES_TO_MENTORS}.c
ourseId`,
          to:
`${DbTableName.MENTEES_TO_MENTORS}.
mentorId`,
        },
        to: `${DbTableName.USERS}.id`,
      },
      },
      category: {
        relation: Model.HasOneRelation,
        modelClass: CourseCategory,
        join: {
          from:
`${DbTableName.COURSES}.courseCategoryId`,
          to:
`${DbTableName.COURSE_CATEGORIES}.id`,
        },
      },
    };
  }
  public static override get tableName(): string {
    return DbTableName.COURSES;
  }
}
export { Course };
```

Файл backend/src/data/models/course-categories-price/course-categories-price.model.ts

```
import { Model, RelationMappings } from
'objection';
import { DbTableName } from
'~/common/enums/enums';
import { Abstract } from
'../abstract/abstract.model';
import { CourseCategory } from '../models';
class CourseCategoryPrice extends Abstract {
  public 'categoryId': number;
  public 'price': number;
  public static override get relationMappings():
RelationMappings {
    return {
      category: {
        relation: Model.BelongsToOneRelation,
        modelClass: CourseCategory,
        join: {
          from:
`${DbTableName.COURSE_CATEGORIES_PRI
CES}.categoryId`,
          to:
`${DbTableName.COURSE_CATEGORIES}.id`,
        },
      },
    };
  }
  public static override get tableName(): string {
    return
DbTableName.COURSE_CATEGORIES_PRICE
S;
  }
}
export { CourseCategoryPrice };
```

Файл backend/src/data/models/course-category/course-category.model.ts

```
import { Model, RelationMappings } from
'objection';
import { DbTableName } from
'~/common/enums/enums';
import { Abstract } from
'../abstract/abstract.model';
import { Course, CourseCategoryPrice, Interview
} from '../models';
class CourseCategory extends Abstract {
  public 'name': string;
  public 'key': string;
  public static override get relationMappings():
RelationMappings {
    return {
      interviews: {
        relation: Model.HasManyRelation,
        modelClass: Interview,
        join: {
          from:
`${DbTableName.COURSE_CATEGORIES}.id`,
          to:
`${DbTableName.INTERVIEWS}.categoryId`,
        },
      },
      courses: {
        relation: Model.HasManyRelation,
        modelClass: Course,
        join: {
          from:
`${DbTableName.COURSE_CATEGORIES}.id`,
          to:
`${DbTableName.COURSES}.courseCategoryId`,
        },
      },
      price: {
        relation: Model.HasOneRelation,
```

```
      modelClass: CourseCategoryPrice,
      join: {
        from:
`${DbTableName.COURSE_CATEGORIES}.id`,
        to:
`${DbTableName.COURSE_CATEGORIES_PRI
CES}.categoryId`,
      },
    },
  };
 }
 public static override get tableName(): string {
  return
DbTableName.COURSE_CATEGORIES;
 }
}
export { CourseCategory };
```

Файл backend/src/data/models/course-module/course-module.model.ts

```
import { Model, RelationMappings } from
'objection';
import { Abstract } from
'src/data/models/abstract/abstract.model';
import { Course } from 'src/data/models/models';
import { DbTableName } from
'~/common/enums/enums';
class CourseModule extends Abstract {
 public 'title': string;
 public 'description': string | null;
 public 'courseId': number;
 public static override get relationMappings():
RelationMappings {
  return {
   course: {
    relation: Model.BelongsToOneRelation,
    modelClass: Course,
    join: {
     from:
`${DbTableName.COURSE_MODULES}.courseI
d`,
      to: `${DbTableName.COURSES}.id`,
    },
   },
  };
 }
 public static override get tableName(): string {
  return DbTableName.COURSE_MODULES;
 }
}
export { CourseModule };
```

Файл backend/src/data/models/courses-to-mentors/courses-to-mentors.model.ts

```
import { DbTableName } from
'~/common/enums/enums';
import { Abstract } from
'../abstract/abstract.model';
class CoursesToMentors extends Abstract {
 public 'userId': number;
 public 'courseId': number;
 public 'studentsCount': number;
 public static override get tableName(): string {
  return
DbTableName.COURSES_TO_MENTORS;
 }
}
export { CoursesToMentors };
```

Файл backend/src/data/models/file/file.model.ts

```
import { ContentType, DbTableName } from
'~/common/enums/enums';
import { Abstract } from
'../abstract/abstract.model';
class File extends Abstract {
 public 'url': string;
 public 'contentType': ContentType;
 public static override get tableName(): string {
  return DbTableName.FILES;
 }
}
export { File };
```

Файл backend/src/data/models/group/group.model.ts

```
import { Model, RelationMappings } from
'objection';
import { DbTableName } from
'~/common/enums/enums';
import { Abstract } from
'../abstract/abstract.model';
import { Permission } from '../models';
class Group extends Abstract {
 public 'name': string;
 public 'key': string;
 public static override get relationMappings():
RelationMappings {
  return {
   permissions: {
    relation: Model.ManyToManyRelation,
    modelClass: Permission,
    join: {
     from: `${DbTableName.GROUPS}.id`,
     through: {
      from:
`${DbTableName.GROUPS_TO_PERMISSIONS
}.groupId`,
```

```
          to:
`${DbTableName.GROUPS_TO_PERMISSIONS
}.permissionId`,
        },
        to: `${DbTableName.PERMISSIONS}.id`,
      },
    },
  };
 }
 public static override get tableName(): string {
  return DbTableName.GROUPS;
 }
}
export { Group };
```

Файл backend/src/data/models/groups-to-permissions/groups-to-permissions.model.ts

```
import { DbTableName } from '~/common/enums/enums';
import { Abstract } from '../abstract/abstract.model';
class GroupsToPermissions extends Abstract {
 public 'permissionId': number;
 public 'groupId': number;
 public static override get tableName(): string {
  return DbTableName.GROUPS_TO_PERMISSIONS;
 }
}
export { GroupsToPermissions };
```

Файл backend/src/data/models/interveiw-note/interview-note.model.ts

```
import { Model, RelationMappings } from
'objection';
import { DbTableName } from
'~/common/enums/enums';
import { Abstract } from
'../abstract/abstract.model';
import { Interview, User } from '../models';
class InterviewNote extends Abstract {
 public 'note': string;
 public 'interviewId': number;
 public 'authorId': number;
 public static override get relationMappings():
RelationMappings {
  return {
   interview: {
    relation: Model.BelongsToOneRelation,
    modelClass: Interview,
    join: {
     from:
`${DbTableName.INTERVIEW_NOTES}.intervie
wId`,
      to: `${DbTableName.INTERVIEWS}.id`,
     },
    },
    author: {
     relation: Model.BelongsToOneRelation,
     modelClass: User,
     join: {
      from:
`${DbTableName.INTERVIEW_NOTES}.authorI
d`,
      to: `${DbTableName.USERS}.id`,
     },
    },
   };
 }
 public static override get tableName(): string {
  return DbTableName.INTERVIEW_NOTES;
 }
}
export { InterviewNote };
```

Файл backend/src/data/models/interview/interview.model.ts

```
import { Model, RelationMappings } from
'objection';
import { DbTableName, InterviewStatus } from
'~/common/enums/enums';
import { Abstract } from
'../abstract/abstract.model';
import { CourseCategory } from
'../course-category/course-category.model';
import { User } from '../models';
class Interview extends Abstract {
 public 'interviewDate': string | null;
 public 'status': InterviewStatus;
 public 'categoryId': number;
 public 'intervieweeUserId': number;
 public 'interviewerUserId': number | null;
 public static override get relationMappings():
RelationMappings {
  return {
   courseCategory: {
    relation: Model.BelongsToOneRelation,
    modelClass: CourseCategory,
    join: {
     from:
`${DbTableName.INTERVIEWS}.categoryId`,
      to:
`${DbTableName.COURSE_CATEGORIES}.id`,
```

```
      },
    },
    interviewee: {
      relation: Model.BelongsToOneRelation,
      modelClass: User,
      join: {
        from:
`${DbTableName.INTERVIEWS}.intervieweeUse
rId`,
        to: `${DbTableName.USERS}.id`,
      },
    },
    interviewer: {
      relation: Model.BelongsToOneRelation,
      modelClass: User,
      join: {
        from:
`${DbTableName.INTERVIEWS}.interviewerUse
rId`,
        to: `${DbTableName.USERS}.id`,
      },
    },
  };
 }
 public static override get tableName(): string {
  return DbTableName.INTERVIEWS;
 }
}
export { Interview };
```

Файл backend/src/data/models/permission/permission.model.ts

```
import { DbTableName, PermissionKey } from
'~/common/enums/enums';
import { Abstract } from
'../abstract/abstract.model';
class Permission extends Abstract {
 public 'name': string;
 public 'key': PermissionKey;
 public static override get tableName(): string {
  return DbTableName.PERMISSIONS;
 }
}
export { Permission }
```

Файл backend/src/data/models/task/task.model.ts

```
import { Model, RelationMappings } from
'objection';
import { DbTableName, TaskStatus } from
'~/common/enums/enums';
import { Abstract } from
'../abstract/abstract.model';
import { CourseModule, MenteesToMentors }
from '../models';
class Task extends Abstract {
 public 'status': TaskStatus;
 public 'menteesToMentorsId': number;
 public 'moduleId': number;
 public static override get relationMappings():
RelationMappings {
  return {
    menteesToMentors: {
      relation: Model.BelongsToOneRelation,
      modelClass: MenteesToMentors,
      join: {
        from:
`${DbTableName.TASKS}.menteesToMentorsId`,
        to:
`${DbTableName.MENTEES_TO_MENTORS}.i
d`,
      },
    },
    module: {
      relation: Model.BelongsToOneRelation,
      modelClass: CourseModule,
      join: {
        from:
`${DbTableName.TASKS}.moduleId`,
        to:
`${DbTableName.COURSE_MODULES}.id`,
      },
    },
  };
 }
 public static override get tableName(): string {
  return DbTableName.TASKS;
 }
}
export { Task };
```

Файл backend/src/data/repositories/course-category/course-category.repository.ts

```
import { CourseCategory as CourseCategoryM }
from '~/data/models/models';
type Constructor = {
 CourseCategoryModel: typeof
CourseCategoryM;
};
class CourseCategory {
 #CourseCategoryModel: typeof
CourseCategoryM;
 public constructor({ CourseCategoryModel }:
Constructor) {
  this.#CourseCategoryModel =
CourseCategoryModel;
 }
 public getAll(): Promise<CourseCategoryM[]> {
  return
this.#CourseCategoryModel.query().execute();
 }
```

```
  public getAllWithCourses():
Promise<CourseCategoryM[]> {
    return this.#CourseCategoryModel
      .query()
      .select('courseCategories.id', 'key', 'name')
      .distinct('courseCategories.id')
      .innerJoin('courses', 'courseCategories.id',
'courses.courseCategoryId')
      .execute();
  }
  public async getByKey(key: string):
Promise<CourseCategoryM | null> {
    const courseCategory = await
this.#CourseCategoryModel
      .query()
      .select()
      .where({ key })
      .first();
    return courseCategory ?? null;
  }
  public async getById(id: number):
Promise<CourseCategoryM | null> {
    const courseCategory = await
this.#CourseCategoryModel
      .query()
      .select()
      .findById(id);

    return courseCategory ?? null;
  }
}
export { CourseCategory };
```

Файл backend/src/data/repositories/course-category-price/course-category-price.repository.ts

```
import {
CourseCategoryPriceGetAllItemResponseDto }
from '~/common/types/types';
import { CourseCategoryPrice as
CourseCategoryPriceM } from
'~/data/models/models';
type Constructor = {
  CourseCategoryPriceModel: typeof
CourseCategoryPriceM;
};
class CourseCategoryPrice {
  #CourseCategoryPriceModel: typeof
CourseCategoryPriceM;
  public constructor({
    CourseCategoryPriceModel:
CourseCategoryModel,
  }: Constructor) {
    this.#CourseCategoryPriceModel =
CourseCategoryModel;
  }
  public getAll():
Promise<CourseCategoryPriceGetAllItemRespons
eDto[]> {
    return this.#CourseCategoryPriceModel
      .query()
      .withGraphJoined('category')

.castTo<CourseCategoryPriceGetAllItemResponse
Dto[]>()
      .execute();
  }
  public async getById(
    id: number,
  ):
Promise<CourseCategoryPriceGetAllItemRespons
eDto | null> {
    const courseCategoryPrice = await
this.#CourseCategoryPriceModel
      .query()
      .findById(id)
      .withGraphJoined('category')

.castTo<CourseCategoryPriceGetAllItemResponse
Dto>();
    return courseCategoryPrice ?? null;
  }
  public async getByCategoryId(
    categoryId: number,
  ):
Promise<CourseCategoryPriceGetAllItemRespons
eDto | null> {
    const courseCategoryPrice = await
this.#CourseCategoryPriceModel
      .query()
      .select()
      .where('categoryId', categoryId)
      .first()

.castTo<CourseCategoryPriceGetAllItemResponse
Dto>();
    return courseCategoryPrice ?? null;
  }
}
export { CourseCategoryPrice };
```

Файл backend/src/data/repositories/course-module/course-module.repository.ts

```
import {
  CourseModuleCreateArgumentsDto,
  CourseModuleGetByIdResponseDto,
  CourseModuleGetRequestParamsDto,
  CourseModulesGetAllItemResponseDto,
  CourseModulesGetAllRequestParamsDto,
```

```
  NumericalValueContainer,
} from '~/common/types/types';
import { CourseModule as ModuleM } from
'~/data/models/models';
type Constructor = {
  ModuleModel: typeof ModuleM;
};
class CourseModule {
  #ModuleModel: typeof ModuleM;
  public constructor({ ModuleModel }:
Constructor) {
    this.#ModuleModel = ModuleModel;
  }
  public create(
    courseModule:
CourseModuleCreateArgumentsDto,
  ): Promise<ModuleM> {
    const { title, description, courseId } =
courseModule;
    return this.#ModuleModel
      .query()
      .insert({
        title,
        description,
        courseId,
      })
      .execute();
  }
  public async getById({
    courseId,
    moduleId,
  }: CourseModuleGetRequestParamsDto):
Promise<CourseModuleGetByIdResponseDto |
null> {
    const module = await this.#ModuleModel
      .query()
      .where({
        courseId,
      })
      .andWhere('course_modules.id', moduleId)
      .joinRelated('course')
      .select('course_modules.*', 'course.title as
courseTitle')
      .first()

.castTo<CourseModuleGetByIdResponseDto>();
    return module ?? null;
  }
  public async getAllByCourseId({
    courseId,
  }: CourseModulesGetAllRequestParamsDto):
Promise<
    CourseModulesGetAllItemResponseDto[]
  > {
    const modules = await this.#ModuleModel
      .query()
      .where({ courseId })
      .returning('*')

.castTo<CourseModulesGetAllItemResponseDto[]
>();
    return modules ?? [];
  }
  public getAllCourseModulesCountByCourseId({
    courseId,
  }: CourseModulesGetAllRequestParamsDto):
Promise<NumericalValueContainer> {
    return this.#ModuleModel
      .query()
      .select()
      .where({ courseId })
      .count('* as value')
      .first()
      .castTo<NumericalValueContainer>()
      .execute();
  }
}
export { CourseModule };
```

Файл backend/src/data/repositories/courses-to-mentors/courses-to-mentors.repository.ts

```
import {
  CoursesToMentorsRequestDto,
  CourseUpdateMentoringDto,
} from '~/common/types/types';
import { CoursesToMentors as
CoursesToMentorsM } from
'~/data/models/models';
type Constructor = {
  CoursesToMentorsModel: typeof
CoursesToMentorsM;
};
class CoursesToMentors {
  #CoursesToMentorsModel: typeof
CoursesToMentorsM;
  private static RECORD_EXISTS_CHECK = 1;
  public constructor({ CoursesToMentorsModel }:
Constructor) {
    this.#CoursesToMentorsModel =
CoursesToMentorsModel;
  }
  public createMentorToCourse({
    courseId,
    userId,
  }: CoursesToMentorsRequestDto):
Promise<CoursesToMentorsM> {
    return this.#CoursesToMentorsModel
      .query()
      .insert({ courseId, userId })
      .execute();
  }
```

```
  public updateStudentsCount(
    userId: number,
    { courseId, studentsCount }:
CourseUpdateMentoringDto,
  ): Promise<number> {
    return this.#CoursesToMentorsModel
      .query()
      .findOne({
        courseId,
        userId,
      })
      .patch({
        studentsCount,
      })
      .execute();
  }
  public async checkIsMentor({
    courseId,
    userId,
  }: CoursesToMentorsRequestDto):
Promise<boolean> {
    const courseToMentor = await
this.#CoursesToMentorsModel
      .query()
.select(CoursesToMentors.RECORD_EXISTS_C
HECK)
      .where({ courseId })
      .andWhere({ userId })
      .first();
    return Boolean(courseToMentor);
  }
  public async
checkIsMentorForAnyCourse(userId: number):
Promise<boolean> {
    const courseToMentor = await
this.#CoursesToMentorsModel
      .query()
.select(CoursesToMentors.RECORD_EXISTS_C
HECK)
      .where({ userId })
      .first();
    return Boolean(courseToMentor);
  }
}
export { CoursesToMentors }
```

Файл backend/src/data/repositories/file/file.repository.ts

```
import { FileTableInsertRequestDto } from
'~/common/types/types';
import { File as FileM } from
'~/data/models/models';
type Constructor = {
  FileModel: typeof FileM;
};
class File {
  #FileModel: typeof FileM;
  public constructor({ FileModel }: Constructor) {
    this.#FileModel = FileModel;
  }
  public create({
    contentType,
    url,
  }: FileTableInsertRequestDto): Promise<FileM>
{
    return this.#FileModel.query().insert({
contentType, url }).execute();
  }
}
export { File };
```

Файл backend/src/data/repositories/group/group.repository.ts

```
import { SortOrder } from
'~/common/enums/enums';
import {
  EntityPagination,
  EntityPaginationRequestQueryDto,
  GroupsWithPermissionIdsDto,
  GroupsWithPermissionsDto,
} from '~/common/types/types';
import { Group as GroupM } from
'~/data/models/models';

type Constructor = {
  GroupModel: typeof GroupM;
};
class Group {
  #GroupModel: typeof GroupM;
  public constructor({ GroupModel }: Constructor)
{
    this.#GroupModel = GroupModel;
  }
  public async getById(id: number):
Promise<GroupsWithPermissionIdsDto | null> {
    const group = await this.#GroupModel
      .query()
      .where('groups.id', id)
      .select('groups.*')
      .withGraphJoined('permissions')
      .castTo<GroupsWithPermissionsDto>()
      .first()
      .then((data) => {
        if (!data) {
          return null;
        }
        return {
          ...data,
```

```
        permissionIds:
data.permissions.map((permission) =>
permission.id),
      };
    });
  return group ?? null;
 }
 public async getAll({
  page,
  count,
 }: EntityPaginationRequestQueryDto):
Promise<EntityPagination<GroupM>> {
  const result = await this.#GroupModel
    .query()
    .orderBy('id', SortOrder.ASC)
    .page(page, count);
  return {
    items: result.results,
    total: result.total,
  };
 }
 public async create(group: { name: string; key:
string }): Promise<GroupM> {
  const { name, key } = group;
  return this.#GroupModel.query().insert({
    name,
    key,
  });
 }
 public async getByName(name: string):
Promise<GroupM | null> {
  const group = await this.#GroupModel
    .query()
    .select()
    .where({ name })
    .first();
  return group ?? null;
 }
 public update(group: {
  id: number;
  name: string;
  key: string;
 }): Promise<GroupM> {
  const { id, name, key } = group;
  return this.#GroupModel
    .query()
    .patchAndFetchById(id, {
      name,
      key,
    })
    .execute();
 }
 public delete(groupId: number):
Promise<number> {
  return
this.#GroupModel.query().delete().where({ id:
groupId }).execute();
 }
 public async getByKey(key: string):
Promise<GroupM | null> {
  const group = await
this.#GroupModel.query().where({ key }).first();

  return group ?? null;
 }
}
export { Group };
```

Файл backend/src/data/repositories/groups-to-permissions/groups-to-permissions.repository.ts

```
import { GroupsToPermissions as
GroupsToPermissionsM } from
'~/data/models/models';
type Constructor = {
 GroupsToPermissionsModel: typeof
GroupsToPermissionsM;
};
class GroupsToPermissions {
 #GroupsToPermissionsModel: typeof
GroupsToPermissionsM;
 public constructor({ GroupsToPermissionsModel
}: Constructor) {
  this.#GroupsToPermissionsModel =
GroupsToPermissionsModel;
 }
 public async create(groupsToPermissions: {
  groupId: number;
  permissionId: number;
 }): Promise<GroupsToPermissionsM> {
  const { groupId, permissionId } =
groupsToPermissions;
  return
this.#GroupsToPermissionsModel.query().insert({
    groupId,
    permissionId,
  });
 }
 public async update(groupsToPermissions: {
  groupId: number;
  permissionIds: number[];
 }): Promise<void> {
  const { groupId, permissionIds } =
groupsToPermissions;
  await this.#GroupsToPermissionsModel
    .query()
    .where({ groupId })
    .whereNotIn('permissionId', permissionIds)
    .delete()
    .execute();
```

```
    await Promise.all(
      permissionIds.map((permissionId: number) =>
{
        return this.#GroupsToPermissionsModel
          .query()
          .insert({
            groupId,
            permissionId,
          })
          .onConflict(['permissionId', 'groupId'])
          .ignore();
      }),
    );
  }
}
export { GroupsToPermissions };
```

Файл backend/src/data/repositories/permission/permission.repository.ts

```
import { SortOrder } from
'~/common/enums/enums';
import {
  EntityPagination,
  EntityPaginationRequestQueryDto,
} from '~/common/types/types';
import { Permission as PermissionM } from
'~/data/models/models';
type Constructor = {
  PermissionModel: typeof PermissionM;
};
class Permission {
  #PermissionModel: typeof PermissionM;
  public constructor({ PermissionModel }:
Constructor) {
    this.#PermissionModel = PermissionModel;
  }
  public async getAll({
    page,
    count,
  }: EntityPaginationRequestQueryDto):
Promise<EntityPagination<PermissionM>> {
    const result = await this.#PermissionModel
      .query()
      .orderBy('id', SortOrder.ASC)
      .page(page, count);
    return {
      items: result.results,
      total: result.total,
    };
  }
  public async getByIds(ids: number[]):
Promise<PermissionM[]> {
    const permissions = await
this.#PermissionModel.query().findByIds(ids);
    return permissions;
  }
}
export { Permission }
```

Файл backend/src/data/repositories/task/task.repository.ts

```
import { TaskStatus } from
'~/common/enums/enums';
import {
  TaskCreateRequestDto,
  TaskGetByMenteeIdAndModuleId,

TaskGetByMenteeIdCourseIdModuleIdRequestDt
o,
  TasksGetByCourseIdAndMenteeIdRequestDto,
  TaskWithModuleResponseDto,
} from '~/common/types/types';
import { Task as TaskM } from
'~/data/models/models';
type Constructor = {
  TaskModel: typeof TaskM;
};

class Task {
  #TaskModel: typeof TaskM;
  public constructor({ TaskModel }: Constructor) {
    this.#TaskModel = TaskModel;
  }
  public updateStatus(taskId: number, status:
TaskStatus): Promise<TaskM> {
    return this.#TaskModel
      .query()
      .patchAndFetchById(taskId, { status })
      .execute();
  }
  public async getById(id: number):
Promise<TaskM | null> {
    const task = await
this.#TaskModel.query().findById(id);
    return task ?? null;
  }
  public async getByMenteeIdAndModuleId({
    moduleId,
    menteeId,
  }: TaskGetByMenteeIdAndModuleId):
Promise<TaskM | null> {
    const task = await this.#TaskModel
      .query()
      .where('tasks.moduleId', moduleId)
      .andWhere('menteesToMentors.menteeId',
menteeId)
```

```
      .withGraphJoined('menteesToMentors')
      .first();
    return task ?? null;
  }
  public async
getByMenteeIdCourseIdModuleId({
    courseId,
    menteeId,
    moduleId,
  }:
TaskGetByMenteeIdCourseIdModuleIdRequestDt
o): Promise<TaskM | null> {
    const task = await this.#TaskModel
      .query()
      .where('tasks.moduleId', moduleId)
      .andWhere('menteesToMentors.menteeId',
menteeId)
      .andWhere('menteesToMentors.courseId',
courseId)
      .withGraphJoined('menteesToMentors')
      .first();
    return task ?? null;
  }
  public getAllByCourseIdAndMenteeId({
    courseId,
    menteeId,
  }:
TasksGetByCourseIdAndMenteeIdRequestDto):
Promise<
    TaskWithModuleResponseDto[]
  > {
    return this.#TaskModel
      .query()
      .withGraphJoined('[menteesToMentors,
module]')
      .where('menteesToMentors.courseId',
courseId)
      .andWhere('menteesToMentors.menteeId',
menteeId)
      .castTo<TaskWithModuleResponseDto[]>()
      .execute();
  }
  public async
hasUncompletedModulesByMenteesToMentorsId(
    menteesToMentorsId: number,
  ): Promise<boolean> {
    const firstUncompleted = await
this.#TaskModel
      .query()
      .findOne({ menteesToMentorsId })
      .whereNot('status', TaskStatus.COMPLETED);
    return Boolean(firstUncompleted);
  }
  public createTask({
    menteesToMentorsId,
    moduleId,
  }: TaskCreateRequestDto): Promise<TaskM> {
    return this.#TaskModel
      .query()
      .insert({
        menteesToMentorsId,
        moduleId,
      })
      .execute();
  }
}
export { Task };
```

Файл backend/src/data/repositories/transaction/transaction.repository.ts

```
import { Page } from 'objection';
import { SortOrder, TransactionStatus } from
'~/common/enums/enums';
import {
  EntityPagination,
  EntityPaginationRequestQueryDto,
  TransactionCreateArgumentsDto,
  TransactionGetAllItemResponseDto,
  TransactionUpdateStatusDto,
} from '~/common/types/types';
import { Transaction as TransactionM } from
'~/data/models/models';

type Constructor = {
  TransactionModel: typeof TransactionM;
};
const INITIAL_TRANSACTION_STATUS =
TransactionStatus.PENDING;
class Transaction {
  #TransactionModel: typeof TransactionM;
  public constructor({ TransactionModel }:
Constructor) {
    this.#TransactionModel = TransactionModel;
  }
  public getById(id: number):
Promise<TransactionGetAllItemResponseDto> {
    return this.#TransactionModel
      .query()
      .findById(id)
      .withGraphJoined(
        '[sender(withoutPassword).[userDetails],
receiver(withoutPassword).[userDetails]]',
      )

.castTo<TransactionGetAllItemResponseDto>()
      .execute();
  }
  public async getTransactionsByUserId(
```

```
    userId: number,
    pagination: EntityPaginationRequestQueryDto,
  ):
Promise<EntityPagination<TransactionGetAllItem
ResponseDto>> {
    const { count, page } = pagination;
    const { results, total } = await
this.#TransactionModel
      .query()
      .where('senderId', userId)
      .orWhere('receiverId', userId)
      .withGraphJoined(
        '[sender(withoutPassword).[userDetails],
receiver(withoutPassword).[userDetails]]',
      )
      .orderBy('createdAt', SortOrder.DESC)
      .page(page, count)
      .castTo<Page<TransactionM &
TransactionGetAllItemResponseDto>>()
      .execute();
    return {
      items: results,
      total,
    };
  }
  public getHoldBySenderAndReceiverId(
    senderId: number,
    receiverId: number,
  ): Promise<TransactionGetAllItemResponseDto>
{
    return this.#TransactionModel
      .query()
      .where('senderId', senderId)
      .andWhere('receiverId', receiverId)
      .andWhere('status', TransactionStatus.HOLD)
      .first()
      .withGraphJoined(
        '[sender(withoutPassword).[userDetails],
receiver(withoutPassword).[userDetails]]',
      )

.castTo<TransactionGetAllItemResponseDto>()
      .execute();
  }
  public create({
    senderId,
    receiverId,
    amount,
  }: TransactionCreateArgumentsDto):
Promise<TransactionGetAllItemResponseDto> {
    return this.#TransactionModel
      .query()
      .insert({
        senderId,
        receiverId,
        amount,
        status: INITIAL_TRANSACTION_STATUS,
      })
      .withGraphFetched(
        '[sender(withoutPassword).[userDetails],
receiver(withoutPassword).[userDetails]]',
      )

.castTo<TransactionGetAllItemResponseDto>()
      .execute();
  }
  public updateStatus({
    transactionId,
    newStatus,
  }: TransactionUpdateStatusDto):
Promise<TransactionGetAllItemResponseDto> {
    return this.#TransactionModel
      .query()
      .patchAndFetchById(transactionId, {
        status: newStatus,
      })
      .withGraphFetched(
        '[sender(withoutPassword).[userDetails],
receiver(withoutPassword).[userDetails]]',
      )

.castTo<TransactionGetAllItemResponseDto>()
      .execute();
  }
}
export { Transaction };
```

Файл backend/src/documentation/documentation.yaml

```
openapi: 3.0.0
info:
  title: GuruHub API
  version: 1.0.0
servers:
  - url: /api/v1
paths:
  # Auth
  /auth/sign-up:
    $ref: './endpoints/auth/user-sign-up.yaml'
  /auth/sign-in:
    $ref: './endpoints/auth/user-sign-in.yaml'
  /auth/current-user:
    $ref: './endpoints/auth/current-user.yaml'
  # Billing
  /billing/balance:
```

```
    $ref: './endpoints/billing/billing-balance.yaml'
  /billing/transactions:
    $ref:
'./endpoints/billing/billing-transactions.yaml'
  /billing/replenish:
    $ref: './endpoints/billing/billing-replenish.yaml'
  /billing/withdraw:
    $ref: './endpoints/billing/billing-withdraw.yaml'
  # Users
  /users:
    $ref: './endpoints/users/users-get-all.yaml'
  /users/:id:
    $ref: './endpoints/users/users-delete.yaml'
  # Groups
  /groups:
    $ref:
'./endpoints/groups/groups-basic-operations.yaml'
  /groups/:id:
    $ref:
'./endpoints/groups/groups-manipulate.yaml'
  # Chat Messages
  /chats:
    $ref:
'./endpoints/chat-messages/chat-messages-root-op
erations.yaml'
  /chats/:chatId:
    $ref:
'./endpoints/chat-messages/chat-get-all-messages.y
aml'
  /chats/has-unread-messages:
    $ref:
'./endpoints/chat-messages/chat-has-unread-messa
ges.yaml'
  # Courses
  /courses:
    $ref: './endpoints/courses/courses.yaml'
  /courses/:id:
    $ref:
'./endpoints/courses/courses-by-id-operations.yaml
'
  /courses/:id/category:
    $ref:
'./endpoints/courses/course-update-category-by-id.
yaml'
  /courses/:id/mentors:
    $ref: './endpoints/courses/courses-mentor.yaml'
  /courses/:id/is-mentor-check:
    $ref:
'./endpoints/courses/course-check-is-mentor.yaml'
  /courses/:id/has-mentor-check:
    $ref:
'./endpoints/courses/course-check-has-mentor.yam
l'

/courses/:courseId/mentees/:menteeId/is-mentor-c
heck:
    $ref:
'./endpoints/courses/course-check-is-mentor-for-m
entee.yaml'
  /courses/popular:
    $ref:
'./endpoints/courses/courses-get-popular.yaml'
  # Courses Modules
  /courses/:courseId/modules:
    $ref:
'./endpoints/courses-modules/courses-modules-get
-all-by-course-id.yaml'
  /courses/:courseId/modules/:moduleId:
    $ref:
'./endpoints/courses-modules/courses-modules-get
-by-course-id-and-module-id.yaml'
  # Categories
  /categories:
    $ref: './endpoints/categories/categories.yaml'
  /categories/:id:
    $ref:
'./endpoints/categories/categories-get-by-id.yaml'
  # Permissions
  /permissions:
    $ref:
'./endpoints/permissions/permissions-get-all.yaml'
  # Interviews
  /interviews:
    $ref: './endpoints/interviews/interviews.yaml'
  /interviews/{id}:
    $ref:
'./endpoints/interviews/interview-manipulate.yaml'
  /interviews/:id/notes:
    $ref:
'./endpoints/interviews/interview-notes-operations.
yaml'

/interviews/interviewee/{intervieweeUserId}/cate
gories:
    $ref:
'./endpoints/interviews/interviews-interviewee-get-
pending-or-passed-category-ids.yaml'

/interviews/interviewee/{intervieweeUserId}/activ
e/categories:
    $ref:
'./endpoints/interviews/interviews-interviewee-get-
active-category-ids.yaml'
  /interviews/:id/update-without-interviewer:
    $ref:
'./endpoints/interviews/interview-update-without-i
nterviewer.yaml'
  # Mentors
  /mentors:
    $ref: './endpoints/mentors/mentors.yaml'

/mentors/courses/{courseId}/mentees/{menteeId}:
```

```
    $ref: './endpoints/mentors/mentor-get.yaml'
  # User Details
  /user-details:
    $ref:
'./endpoints/user-details/user-details-manipulate.ya
ml'
  # Tasks
  /tasks/:id:
    $ref: './endpoints/tasks/tasks-manipulate.yaml'
  /tasks/modules/:moduleId/mentees/:menteeId:
    $ref:
'./endpoints/tasks/tasks-get-by-mentee-id-and-mod
ule-id.yaml'
  /tasks/:id/notes:
    $ref: './endpoints/tasks/tasks-get-notes.yaml'

/tasks/courses/:courseId/modules/:moduleId/mente
es/:menteeId:
    $ref:
'./endpoints/tasks/tasks-get-by-course-id-module-i
d-mentee-id.yaml'
  /tasks/courses/:courseId/mentees/:menteeId:
    $ref:
'./endpoints/tasks/tasks-get-by-course-id-mentee-i
d.yaml'
components:
  schemas:
    UserSignUpRequestDto:
      $ref:
'./schemas/auth/user-sign-up-request-dto.yaml'
    UserSignUpResponseDto:
      $ref:
'./schemas/auth/user-sign-up-response-dto.yaml'
    UsersGetAllResponseDto:
      $ref:
'./schemas/users/users-get-all-response-dto.yaml'
    UserSignInRequestDto:
      $ref:
'./schemas/auth/user-sign-in-request-dto.yaml'
    UserSignInResponseDto:
      $ref:
'./schemas/auth/user-sign-in-response-dto.yaml'
    UsersDeleteRequestDto:
      $ref:
'./schemas/users/users-delete-request-dto.yaml'
    BillingReplenishRequestDto:
      $ref:
'./schemas/billing/billing-replenish-request-dto.ya
ml'
    BillingGetAllTransactionsResponseDto:
      $ref:
'./schemas/billing/billing-get-all-transactions-respo
nse-dto.yaml'
    CurrentUserResponseDto:
      $ref:
'./schemas/auth/current-user-response-dto.yaml'
    GroupsGetAllResponseDto:
      $ref:
'./schemas/groups/groups-get-all-response-dto.ya
ml'
    GroupsCreateRequestDto:
      $ref:
'./schemas/groups/groups-create-request-dto.yaml'
    GroupsUpdateRequestDto:
      $ref:
'./schemas/groups/groups-update-request-dto.yaml'
    ChatCreateMessageRequestDto:
      $ref:
'./schemas/chat-messages/chat-create-message-req
uest-dto.yaml'
    ChatGetAllMessagesResponseDto:
      $ref:
'./schemas/chat-messages/chat-get-all-messages-re
sponse-dto.yaml'
    ChatCreateMessageResponseDto:
      $ref:
'./schemas/chat-messages/chat-get-message-respon
se-dto.yaml'
    CourseCreateRequestDto:
      $ref:
'./schemas/courses/course-add-request-dto.yaml'
    CourseGetByIdRequestDto:
      $ref:
'./schemas/courses/course-get-by-id-request-dto.ya
ml'
    CourseGetByIdResponseDto:
      $ref:
'./schemas/courses/course-get-by-id-response-dto.
yaml'
    CoursesAddMentorRequestDto:
      $ref:
'./schemas/courses/courses-add-mentor-request-dt
o.yaml'
    CoursesAddMentorResponseDto:
      $ref:
'./schemas/courses/courses-add-mentor-response-d
to.yaml'
    CoursesUpdateCategoryRequestDto:
      $ref:
'./schemas/courses/course-update-category-request
-dto.yaml'
    CoursesModulesGetAllByCourseIdRequestDto:
      $ref:
'./schemas/courses-modules/courses-modules-get-
all-by-course-id-request-dto.yaml'

CoursesModulesGetAllByCourseIdResponseDto:
      $ref:
'./schemas/courses-modules/courses-modules-get-
all-by-course-id-response-dto.yaml'
```

```
CoursesModulesGetByCourseIdAndModuleIdRequestDto:
      $ref:
'./schemas/courses-modules/courses-modules-get-by-course-id-and-module-id-request-dto.yaml'

CoursesModulesGetByCourseIdAndModuleIdResponseDto:
      $ref:
'./schemas/courses-modules/courses-modules-get-all-by-course-id-response-dto.yaml'
    CategoriesGetAllResponseDto:
      $ref:
'./schemas/categories/categories-get-all-response-dto.yaml'
    CategoriesGetByIdRequestDto:
      $ref:
'./schemas/categories/category-get-by-id-request-dto.yaml'
    CategoriesGetByIdResponseDto:
      $ref:
'./schemas/categories/category-get-by-id-response-dto.yaml'
    PermissionsGetAllResponseDto:
      $ref:
'./schemas/permissions/permissions-get-all-response-dto.yaml'
    InterviewsGetAllResponseDto:
      $ref:
'./schemas/interviews/interview-get-all-response-dto.yaml'
    InterviewsGetByIdResponseDto:
      $ref:
'./schemas/interviews/interview-response-dto.yaml
'
```

Файл backend/src/plugins/file/file.plugin.ts

```
import { MultipartFile } from '@fastify/multipart';
import { FastifyPluginAsync, FastifyRequest }
from 'fastify';
import fp from 'fastify-plugin';
import {
  ControllerHook,
  ExceptionMessage,
  HttpCode,
} from '~/common/enums/enums';
import { FilesError, InvalidFilesError } from
'~/exceptions/exceptions';

type Options = {
  allowedExtensions: string[];
};
```

```
    InterviewsGetAllNotesRequestDto:
      $ref:
'./schemas/interviews/interviews-get-all-notes-request-dto.yaml'
    InterviewsGetAllNotesResponseDto:
      $ref:
'./schemas/interviews/interviews-get-all-notes-response-dto.yaml'
    InterviewsCreateNoteRequestDto:
      $ref:
'./schemas/interviews/interviews-create-note-request-dto.yaml'
    InterviewsCreateNoteResponseDto:
      $ref:
'./schemas/interviews/interviews-create-note-response-dto.yaml'

InterviewsGetPendingOrPassedCategoryIdsDto:
      $ref:
'./schemas/interviews/interviews-interviewee-get-pending-or-passed-category-ids-response.yaml'
    UserDetailsResponseDto:
      $ref:
'./schemas/user-details/user-details-response-dto.yaml'
    UserDetailsUpdateInfoRequestDto:
      $ref:
'./schemas/user-details/user-details-update-request-dto.yaml'
    MenteesToMentorsResponseDto:
      $ref:
'./schemas/mentors/mentor-get-response-dto.yaml'
  securitySchemes:
    bearerAuth:
      type: http
      scheme: bearer
      bearerFormat: JWT
```

```
const upload: FastifyPluginAsync<Options> =
async (fastify, opts) => {
  const { allowedExtensions } = opts;

  fastify.decorateRequest('fileBuffer', null);

  fastify.addHook(
    ControllerHook.PRE_VALIDATION,
    async (
      request: FastifyRequest<{ Body: { file:
MultipartFile } }>,
      reply,
    ) => {
      try {
        if (!request.isMultipart()) {
          return;
        }
```

```
      const { file } = request.body;
      const isAllowedExtension =
allowedExtensions.some(
        (extension) => extension ===
file.mimetype,
      );
      if (!isAllowedExtension) {
        throw new InvalidFilesError();
      }
      if (file.file.truncated) {
        throw new FilesError({
          message:
ExceptionMessage.FILE_TOO_BIG,
        });
      }
      const fileBuffer = await file.toBuffer();
      request.fileBuffer = fileBuffer;
    } catch (err) {

reply.status(HttpCode.BAD_REQUEST).send(err)
;
    }
  },
 );
};
const file = fp(upload);
export { file };
```

Файл backend/src/plugins/github-oauth/github-oauth.plugin.ts

```
import fastifyOauth2 from '@fastify/oauth2';
import { FastifyPluginAsync } from 'fastify';
import fp from 'fastify-plugin';
import { ApiPath, AuthApiPath } from
'~/common/enums/enums';
type Options = {
  clientId: string;
  clientSecret: string;
  baseURL: string;
};
const githubOAuthFn:
FastifyPluginAsync<Options> = async (fastify,
opts) => {
  const { clientId, clientSecret, baseURL } = opts;

  fastify.register(fastifyOauth2, {
    name: 'githubOAuth2',
    scope: ['read:user', 'user:email'],
    credentials: {
      client: {
        id: clientId,
        secret: clientSecret,
      },
      auth:
fastifyOauth2.GITHUB_CONFIGURATION,
    },
    startRedirectPath:
`${ApiPath.AUTH}${AuthApiPath.GITHUB_SI
GN}`,
    callbackUri:
`${baseURL}${ApiPath.AUTH}${AuthApiPath.
GITHUB_CALLBACK}`,
    callbackUriParams: {
      access_type: 'offline',
    },
  });
};
const githubOAuth = fp(githubOAuthFn);
export { githubOAuth };
```

Файл backend/src/plugins/google-oauth/google-oauth.plugin.ts

```
import fastifyOauth2 from '@fastify/oauth2';
import { FastifyPluginAsync } from 'fastify';
import fp from 'fastify-plugin';
import { ApiPath, AuthApiPath } from
'~/common/enums/enums';
type Options = {
  clientId: string;
  clientSecret: string;
  baseURL: string;
};
const googleOAuthFn:
FastifyPluginAsync<Options> = async (fastify,
opts) => {
  const { clientId, clientSecret, baseURL } = opts;

  fastify.register(fastifyOauth2, {
    name: 'googleOAuth2',
    scope: ['profile', 'email'],
    credentials: {
      client: {
        id: clientId,
        secret: clientSecret,
      },
      auth:
fastifyOauth2.GOOGLE_CONFIGURATION,
    },
    startRedirectPath:
`${ApiPath.AUTH}${AuthApiPath.GOOGLE_SI
GN}`,
    callbackUri:
`${baseURL}${ApiPath.AUTH}${AuthApiPath.
GOOGLE_CALLBACK}`,
```

```
    callbackUriParams: {
      access_type: 'offline',
    },
```

Файл backend/src/services/auth/auth.service.ts

```
import { ExceptionMessage, HttpCode } from
'~/common/enums/enums';
import {
  UserSignInRequestDto,
  UserSignInResponseDto,
  UserSignOAuthRequestDto,
  UserSignUpRequestDto,
  UserSignUpResponseDto,
  UserWithPermissions,
} from '~/common/types/types';
import { AuthError } from
'~/exceptions/exceptions';
import {
  encrypt as encryptServ,
  token as tokenServ,
  user as userServ,
} from '~/services/services';
type Constructor = {
  userService: typeof userServ;
  tokenService: typeof tokenServ;
  encryptService: typeof encryptServ;
};
class Auth {
  #userService: typeof userServ;
  #tokenService: typeof tokenServ;
  #encryptService: typeof encryptServ;
  public constructor({
    userService,
    encryptService,
    tokenService,
  }: Constructor) {
    this.#userService = userService;
    this.#encryptService = encryptService;
    this.#tokenService = tokenService;
  }
  public async signOAuth(
    userRequestDto: UserSignOAuthRequestDto,
  ): Promise<UserSignUpResponseDto> {
    const { email, fullName, googleId, githubId } =
userRequestDto;
    const user = await this.#userService.upsert({
      email,
      fullName,
      googleId,
      githubId,
      password: String(googleId ?? githubId),
    });
    const token = await this.#tokenService.create({
userId: user.id });
    return {
      user,
      token,
    };
  }
  public async signUp(
    userRequestDto: UserSignUpRequestDto,
  ): Promise<UserSignUpResponseDto> {
    const { email } = userRequestDto;
    const userByEmail = await
this.#userService.getByEmail(email);
    if (userByEmail) {
      throw new AuthError({
        message:
ExceptionMessage.EMAIL_IS_ALREADY_TAK
EN,
        status: HttpCode.CONFLICT,
      });
    }
    const user = await
this.#userService.create(userRequestDto);
    const token = await this.#tokenService.create({
userId: user.id });
    return {
      user,
      token,
    };
  }
  public async verifySignIn(
    signInUserDto: UserSignInRequestDto,
  ): Promise<UserWithPermissions> {
    const user = await
this.#userService.getByEmail(signInUserDto.emai
l);
    if (!user) {
      throw new AuthError({
        status: HttpCode.BAD_REQUEST,
        message:
ExceptionMessage.BAD_CREDENTIALS,
      });
    }
    const encryptionData = {
      data: signInUserDto.password,
      salt: user.passwordSalt,
      passwordHash: user.passwordHash,
    };
    const isPasswordValid = await
this.#encryptService.compare(encryptionData);
    if (!isPasswordValid) {
      throw new AuthError({
        status: HttpCode.BAD_REQUEST,
  });
};
const googleOAuth = fp(googleOAuthFn);
export { googleOAuth };
```

```
        message:
ExceptionMessage.BAD_CREDENTIALS,
      });
    }
    const permissions = await
this.#userService.getUserPermissions(user.id);
    return {
      id: user.id,
      email: user.email,
      userDetails: user.userDetails,
      createdAt: user.createdAt,
      permissions,
    };
  }
  public async signIn(
    userRequestDto: UserSignInRequestDto,
  ): Promise<UserSignInResponseDto> {
    const user = await
this.verifySignIn(userRequestDto);
    const token = await this.#tokenService.create({
userId: user.id });

    return {
      user,
      token,
    };
  }
  public async getCurrentUser(
    token: string,
  ): Promise<UserWithPermissions | null> {
    try {
      const { userId } = await
this.#tokenService.decode(token);
      const user = await
this.#userService.getById(userId);
      return user;
    } catch {
      throw new AuthError({
        status: HttpCode.UNAUTHORIZED,
        message:
ExceptionMessage.UNAUTHORIZED_USER,
      });
    }
  }
}
export { Auth };
```

Файл backend/src/services/aws/file/file.service.ts

```
import {
  PutObjectCommand,
  S3Client,
  S3ServiceException,
} from '@aws-sdk/client-s3';

import { ExceptionMessage } from
'~/common/enums/enums';
import {
  FileGetResponseDto,
  FileGetUrlRequestDto,
  FileUploadRequestDto,
} from '~/common/types/types';
import { file as fileRep } from
'~/data/repositories/repositories';
import { FilesError } from
'~/exceptions/exceptions';
type Constructor = {
  region: string;
  accessKeyId: string;
  secretAccessKey: string;
  fileRepository: typeof fileRep;
};
class File {
  #storage: S3Client;
  #fileRepository: typeof fileRep;
  public constructor({
    accessKeyId,
    secretAccessKey,
    region,
    fileRepository,
  }: Constructor) {
    this.#storage = new S3Client({
      region,
      credentials: {
        accessKeyId,
        secretAccessKey,
      },
    });
    this.#fileRepository = fileRepository;
  }
  public async uploadFile({
    file,
    fileName,
    bucket,
    contentType,
  }: FileUploadRequestDto):
Promise<FileGetResponseDto> {
    try {
      await this.#storage.send(
        new PutObjectCommand({ Bucket: bucket,
Key: fileName, Body: file }),
      );
      const fileUrl = this.getFileUrl({ bucket,
fileName });
      return this.#fileRepository.create({
        contentType,
        url: fileUrl,
      });
    } catch (err) {
```

```
      this.throwError(err);
    }
  }
  private getFileUrl({ bucket, fileName }:
FileGetUrlRequestDto): string {
    return
`https://${bucket}.s3.amazonaws.com/${fileName
}`;
  }
  private throwError(err: unknown): never {
    if (err instanceof S3ServiceException) {
      throw new FilesError({
        message: err.message as ExceptionMessage,
        status: err.$response?.statusCode,
      });
    }
    throw new FilesError();
  }
}
export { File };
```

Файл backend/src/services/billing/billing.service.ts

```
import StripeApi from 'stripe';
import {
  ExceptionMessage,
  PaymentUnit,
  TransactionStatus,
} from '~/common/enums/enums';
import {
  BillingApiVersion,
  BillingInitHoldStudentPaymentArgumentsDto,
  BillingReplenishArgumentsDto,
  EntityPagination,
  EntityPaginationRequestQueryDto,
  StripeReplenishArgumentsDto,
  TransactionCreateArgumentsDto,
  TransactionGetAllItemResponseDto,
} from '~/common/types/types';
import { BillingError } from
'~/exceptions/exceptions';
import {
  transaction as transactionServ,
  user as userServ,
  userDetails as userDetailsServ,
} from '~/services/services';
type Constructor = {
  secretKey: string;
  apiVersion: BillingApiVersion;
  transactionService: typeof transactionServ;
  userService: typeof userServ;
  userDetailsService: typeof userDetailsServ;
};
class Billing {
  #transactionService: typeof transactionServ;
  #userService: typeof userServ;
  #userDetailsService: typeof userDetailsServ;
  #stripe: StripeApi;
  private static BILLING_CURRENCY = 'usd';
  private static
DEFAULT_STUDYING_PRICE_COEFFICIENT
= 0.5;
  private static
NEW_USER_BALANCE_AFTER_WITHDRAW
= 0;
  public constructor({
    transactionService,
    userService,
    userDetailsService,
    secretKey,
    apiVersion,
  }: Constructor) {
    this.#stripe = new StripeApi(secretKey, {
      apiVersion,
    });
    this.#transactionService = transactionService;
    this.#userService = userService;
    this.#userDetailsService = userDetailsService;
  }
  public async replenish({
    userId,
    amountOfMoneyToReplenish,
    token,
  }: BillingReplenishArgumentsDto):
Promise<number> {
    const userBalance = await
this.#userService.getByIdMoneyBalance(userId);
    await this.initReplenish({
      amount: amountOfMoneyToReplenish,
      token,
    });
    const newBalance = userBalance +
amountOfMoneyToReplenish;
    return
this.#userDetailsService.updateMoneyBalance(use
rId, newBalance);
  }
  public async withdraw(userId: number):
Promise<number> {
    const userBalance = await
this.#userService.getByIdMoneyBalance(userId);
    await this.initWithdraw(userBalance);
    return
this.#userDetailsService.updateMoneyBalance(
      userId,

Billing.NEW_USER_BALANCE_AFTER_WITH
DRAW,
    );
```

```
  }
  public getTransactionsByUserId(
    userId: number,
    pagination: EntityPaginationRequestQueryDto,
  ):
Promise<EntityPagination<TransactionGetAllItem
ResponseDto>> {
    return
this.#transactionService.getTransactionsByUserId(
userId, pagination);
  }
  public async initHoldStudentPayment({
    menteeId,
    mentorId,
    rawPriceOfStudying,
  }:
BillingInitHoldStudentPaymentArgumentsDto):
Promise<void> {
    const menteeBalance = await
this.#userService.getByIdMoneyBalance(menteeI
d);
    const priceOfStudying =
      rawPriceOfStudying *
Billing.DEFAULT_STUDYING_PRICE_COEFFI
CIENT;
    if (menteeBalance < priceOfStudying) {
      const symbolsAmount = 2;
      const requiredBalance = `You need to add $${(
        priceOfStudying - menteeBalance
      ).toFixed(symbolsAmount)} to your balance.`;
      throw new BillingError({
        message:
`${ExceptionMessage.NOT_ENOUGH_FUNDS_
TO_PAY_FOR_MENTORS_SERVICES}
${requiredBalance}`,
      });
    }
    const newMenteeBalance = menteeBalance -
priceOfStudying;

    await
this.#userDetailsService.updateMoneyBalance(
      menteeId,
      newMenteeBalance,
    );
    const transaction = await
this.makeTransaction({
      senderId: menteeId,
      receiverId: mentorId,
      amount: priceOfStudying,
    });
    await this.holdTransaction(transaction.id);
  }
  public
getHoldTransactionBySenderAndReceiverId(
    senderId: number,
    receiverId: number,
  ): Promise<TransactionGetAllItemResponseDto>
{
    return
this.#transactionService.getHoldBySenderAndRec
eiverId(
      senderId,
      receiverId,
    );
  }
  public initReplenish({
    amount,
    token,
  }: StripeReplenishArgumentsDto):
Promise<StripeApi.Charge> {
    try {
      return this.#stripe.charges.create({
        source: token.id,
        amount: amount *
PaymentUnit.CENTS_IN_ONE_DOLLAR,
        currency: Billing.BILLING_CURRENCY,
      });
    } catch (err) {
      this.throwError(err);
    }
  }
  public initWithdraw(
    amount: number,
  ):
Promise<StripeApi.Response<StripeApi.Payout>
> {
    try {
      return this.#stripe.payouts.create({
        amount: amount *
PaymentUnit.CENTS_IN_ONE_DOLLAR,
        currency: Billing.BILLING_CURRENCY,
      });
    } catch (err) {
      this.throwError(err);
    }
  }
  public makeTransaction(
    transactionCreateBody:
TransactionCreateArgumentsDto,
  ): Promise<TransactionGetAllItemResponseDto>
{
    return
this.#transactionService.create(transactionCreateB
ody);
  }
  public holdTransaction(
    transactionId: number,
  ): Promise<TransactionGetAllItemResponseDto>
{
    return this.#transactionService.updateStatus({
      transactionId,
```

```
      newStatus: TransactionStatus.HOLD,
    });
  }
  public fulfillTransaction(
    transactionId: number,
  ): Promise<TransactionGetAllItemResponseDto>
{
    return this.#transactionService.updateStatus({
      transactionId,
      newStatus: TransactionStatus.FULFILLED,
    });
  }
  public rejectTransaction(
    transactionId: number,
  ): Promise<TransactionGetAllItemResponseDto>
{
    return this.#transactionService.updateStatus({
      transactionId,
      newStatus: TransactionStatus.REJECTED,
    });
  }
  private throwError(err: unknown): never {
    if (err instanceof StripeApi.errors.StripeError) {
      throw new BillingError({
        message: err.message,
        status: err.statusCode,
      });
    }
    throw new BillingError({ cause: err });
  }
}
export { Billing };
```

Файл backend/src/services/course/course.service.ts

```
import { CourseHost, ExceptionMessage,
VendorKey } from '~/common/enums/enums';
import {
  CourseCreateArgumentsDto,
  CourseFilteringWithPaginationDto,
  CourseGetByIdAndVendorKeyArgumentsDto,
  CourseGetMenteesByMentorRequestDto,
  CourseGetMentoringDto,
  CourseGetMentorsRequestDto,
  CourseGetResponseDto,
  CourseUpdateMentoringDto,
  EntityPagination,
  EntityPaginationRequestQueryDto,
  UsersGetResponseDto,
} from '~/common/types/types';
import { course as courseRep } from
'~/data/repositories/repositories';
import { CoursesError } from
'~/exceptions/exceptions';
import { convertPageToZeroIndexed,
sanitizeHTML } from '~/helpers/helpers';
import {
  courseCategory as courseCategoryServ,
  courseModule as courseModuleServ,
  coursesToMentors as coursesToMentorsServ,
  edx as edxServ,
  udemy as udemyServ,
  vendor as vendorServ,
} from '~/services/services';
type Constructor = {
  courseRepository: typeof courseRep;
  courseModuleService: typeof
courseModuleServ;
  vendorService: typeof vendorServ;
  udemyService: typeof udemyServ;
  edxService: typeof edxServ;
  courseCategoryService: typeof
courseCategoryServ;
  coursesToMentorsService: typeof
coursesToMentorsServ;
};
class Course {
  #courseRepository: typeof courseRep;
  #courseModuleService: typeof
courseModuleServ;
  #vendorService: typeof vendorServ;
  #udemyService: typeof udemyServ;
  #edxService: typeof edxServ;
  #courseCategoryService: typeof
courseCategoryServ;
  #coursesToMentorsService: typeof
coursesToMentorsServ;
  public constructor({
    courseRepository,
    courseModuleService,
    vendorService,
    udemyService,
    edxService,
    courseCategoryService,
    coursesToMentorsService,
  }: Constructor) {
    this.#courseRepository = courseRepository;
    this.#courseModuleService =
courseModuleService;
    this.#vendorService = vendorService;
    this.#udemyService = udemyService;
    this.#edxService = edxService;
    this.#courseCategoryService =
courseCategoryService;
    this.#coursesToMentorsService =
coursesToMentorsService;
  }
  public async getAllWithCategories(
```

```
    filteringAndPagination:
CourseFilteringWithPaginationDto,
  ):
Promise<EntityPagination<CourseGetResponseDt
o>> {
    const { categoryKey, title, page, count } =
filteringAndPagination;
    const categoryId = await
this.getCategoryIdByKey(categoryKey);
    const zeroIndexPage =
convertPageToZeroIndexed(page);
    return
this.#courseRepository.getAllWithCategories({
      categoryId,
      title,
      page: zeroIndexPage,
      count,
    });
  }
  public getAll(
    args: EntityPaginationRequestQueryDto,
  ):
Promise<EntityPagination<CourseGetResponseDt
o>> {
    const { page, count } = args;
    const zeroIndexPage =
convertPageToZeroIndexed(page);
    return this.#courseRepository.getAll({
      page: zeroIndexPage,
      count,
    });
  }
  public getAllCoursesStudying(
    userId: number,
    pagination: EntityPaginationRequestQueryDto,
  ):
Promise<EntityPagination<CourseGetResponseDt
o>> {
    const { page, count } = pagination;
    const zeroIndexPage =
convertPageToZeroIndexed(page);
    return
this.#courseRepository.getAllCoursesStudying(us
erId, {
      count,
      page: zeroIndexPage,
    });
  }
  public getAllCoursesMentoring(
    userId: number,
    pagination: EntityPaginationRequestQueryDto,
  ):
Promise<EntityPagination<CourseGetMentoringD
to>> {
    const { page, count } = pagination;
    const zeroIndexPage =
convertPageToZeroIndexed(page);
    return
this.#courseRepository.getAllCoursesMentoring(u
serId, {
      count,
      page: zeroIndexPage,
    });
  }
  public async create(
    courseRequestDto:
CourseCreateArgumentsDto,
  ): Promise<CourseGetResponseDto> {
    const { description, title, url, vendorKey,
originalId, imageUrl } =
      courseRequestDto;
    const vendor = await
this.#vendorService.getByKey(vendorKey);
    if (!vendor) {
      throw new CoursesError({
        message:
ExceptionMessage.INVALID_COURSE_VENDO
R,
      });
    }
    const courseByOriginalIdAndVendor = await
this.getByOriginalIdAndVendorKey({
      originalId,
      vendorKey,
    });
    if (courseByOriginalIdAndVendor) {
      throw new CoursesError({
        message:
ExceptionMessage.COURSE_EXIST,
      });
    }
    const course = await
this.#courseRepository.create({
      description: sanitizeHTML(description),
      title,
      url,
      vendorId: vendor.id,
      originalId,
      imageUrl,
    });
    return {
      ...course,
      category: null,
      vendor,
    };
  }
  public async createByUrl(url: string):
Promise<CourseGetResponseDto | null> {
    const urlObject = new URL(url);
    const { host } = urlObject;
    switch (host) {
```

```
      case CourseHost.UDEMY:
      case CourseHost.W_UDEMY: {
        const courseData = await
this.#udemyService.getCourseByUrl(urlObject);
        const { description, title, url, id,
image_480x270 } = courseData;
        const course = await this.create({
          description,
          title,
          url,
          vendorKey: VendorKey.UDEMY,
          originalId: id.toString(),
          imageUrl: image_480x270,
        });
        await
this.#courseModuleService.createModulesByCour
seId(id, course.id);
        return course;
      }
      case CourseHost.EDX: {
        const courseData = await
this.#edxService.getCourseByUrl(urlObject);
        const { description, name, course_id } =
courseData;
        const course = await this.create({
          description,
          title: name,
          url,
          vendorKey: VendorKey.EDX,
          originalId: course_id.toString(),
          imageUrl: null,
        });
        return course;
      }
      default: {
        throw new CoursesError({
          message:
ExceptionMessage.INVALID_URL_HOST,
        });
      }
    }
  }
  public async getCategoryIdByKey(categoryKey:
string): Promise<number | null> {
    if (!categoryKey) {
      return null;
    }
    const category = await
this.#courseCategoryService.getByKey(categoryK
ey);
    if (!category) {
      throw new CoursesError({
        message:
ExceptionMessage.INVALID_COURSE_CATEG
ORY,
      });
    }
    return category.id;
  }
  public async getByOriginalIdAndVendorKey({
    originalId,
    vendorKey,
  }:
CourseGetByIdAndVendorKeyArgumentsDto):
Promise<CourseGetResponseDto | null> {
    const course = await
this.#courseRepository.getByOriginalIdAndVendo
rKey({
      originalId,
      vendorKey,
    });
    return course ?? null;
  }
  public async getById(courseId: number):
Promise<CourseGetResponseDto | null> {
    const course = await
this.#courseRepository.getById(courseId);
    return course ?? null;
  }

  public async getMentorsByCourseId({
    courseId,
    filteringOpts,
  }: CourseGetMentorsRequestDto):
Promise<UsersGetResponseDto[]> {
    const mentorsWithMenteesCount =
      await
this.#courseRepository.getMentorsWithMenteesC
ount(courseId);
    const mentorsWithMenteesMaxCount =
      await
this.#courseRepository.getMentorsWithMenteesM
axCount(courseId);
    const filteredMentorIds =
mentorsWithMenteesMaxCount.map(
      (mentorWithMaxCount) => {
        const mentorWithCountFiltered =
mentorsWithMenteesCount.find(
          (mentorWithCount) => mentorWithCount.id
=== mentorWithMaxCount.id,
        );
        if (!mentorWithCountFiltered) {
          return mentorWithMaxCount.id;
        }
        if (mentorWithCountFiltered.count <
mentorWithMaxCount.count) {
          return mentorWithCountFiltered.id;
        }
        return -1;
      },
    );
```

```
    return
this.#courseRepository.getMentorsByCourseId(filt
eredMentorIds, {
      courseId,
      filteringOpts,
    });
  }
  public getMenteesByCourseIdAndMentorId({
    mentorId,
    courseId,
  }: CourseGetMenteesByMentorRequestDto):
Promise<UsersGetResponseDto[]> {
    return
this.#courseRepository.getMenteesByCourseIdAn
dMentorId({
      courseId,
      mentorId,
    });
  }

  public updateStudentsCount(
    userId: number,
    data: CourseUpdateMentoringDto,
  ): Promise<number> {
    return
this.#coursesToMentorsService.updateStudentsCo
unt(userId, data);
  }
  public updateCategory(
    courseId: number,
    newCategoryId: number,
  ): Promise<CourseGetResponseDto> {
    return
this.#courseRepository.updateCategory(courseId,
newCategoryId);
  }
  public getPopular():
Promise<CourseGetResponseDto[]> {
    return this.#courseRepository.getPopular();
  }
}
export { Course };
```

Файл backend/src/services/course-category/course-category.service.ts

```
import {
  CategoryGetAllResponseDto,
  CourseCategoryGetResponseDto,
  CourseCategoryPriceGetAllItemResponseDto,
  CourseCategoryPriceGetAllResponseDto,
} from '~/common/types/types';
import {
  courseCategory as courseCategoryRep,
  courseCategoryPrice as courseCategoryPriceRep,
} from '~/data/repositories/repositories';
type Constructor = {
  courseCategoryRepository: typeof
courseCategoryRep;
  courseCategoryPriceRepository: typeof
courseCategoryPriceRep;
};
class CourseCategory {
  #courseCategoryRepository: typeof
courseCategoryRep;

  #courseCategoryPriceRepository: typeof
courseCategoryPriceRep;
  public constructor({
    courseCategoryRepository,
    courseCategoryPriceRepository,
  }: Constructor) {
    this.#courseCategoryRepository =
courseCategoryRepository;
    this.#courseCategoryPriceRepository =
courseCategoryPriceRepository;
  }

  public async getAll():
Promise<CategoryGetAllResponseDto> {
    const categories = await
this.#courseCategoryRepository.getAll();
    return {
      items: categories.map((category) => ({
        id: category.id,
        key: category.key,
        name: category.name,
      })),
    };
  }
  public async getAllWithCourses():
Promise<CategoryGetAllResponseDto> {
    const categories = await
this.#courseCategoryRepository.getAllWithCourse
s();
    return {
      items: categories,
    };
  }
  public getByKey(key: string):
Promise<CourseCategoryGetResponseDto | null>
{
    return
this.#courseCategoryRepository.getByKey(key);
  }
  public getById(id: number):
Promise<CourseCategoryGetResponseDto | null>
{
    return
this.#courseCategoryRepository.getById(id);
```

```
  }
  public async getAllPriceDtos():
Promise<CourseCategoryPriceGetAllResponseDt
o> {
    const categoryPriceDtos =
      await
this.#courseCategoryPriceRepository.getAll();
    return { items: categoryPriceDtos };
  }
  public getPriceDtoById(
    id: number,
  ):
Promise<CourseCategoryPriceGetAllItemRespons
eDto | null> {
    return
this.#courseCategoryPriceRepository.getById(id);
  }
  public getPriceDtoByCategoryId(
    categoryId: number,
  ):
Promise<CourseCategoryPriceGetAllItemRespons
eDto | null> {
    return
this.#courseCategoryPriceRepository.getByCatego
ryId(categoryId);
  }
}
export { CourseCategory };
```

Файл backend/src/services/course-module/course-module.service.ts

```
import { ExceptionMessage } from
'~/common/enums/enums';
import {
  CourseModuleCreateArgumentsDto,
  CourseModuleGetByIdResponseDto,
  CourseModuleGetRequestParamsDto,
  CourseModulesGetAllItemResponseDto,
} from '~/common/types/types';
import { courseModule as moduleRep } from
'~/data/repositories/repositories';
import { CoursesModulesError } from
'~/exceptions/exceptions';
import { sanitizeHTML } from '~/helpers/helpers';
import { udemy as udemyServ } from
'~/services/services';
type Constructor = {
  moduleRepository: typeof moduleRep;
  udemyService: typeof udemyServ;
};
class CourseModule {
  #moduleRepository: typeof moduleRep;
  #udemyService: typeof udemyServ;
  public constructor({ moduleRepository,
udemyService }: Constructor) {
    this.#moduleRepository = moduleRepository;
    this.#udemyService = udemyService;
  }
  public create(
    moduleRequestDto:
CourseModuleCreateArgumentsDto,
  ):
Promise<CourseModulesGetAllItemResponseDto
> {
    const { title, description, courseId } =
moduleRequestDto;
    return this.#moduleRepository.create({
      title,
      description,
      courseId,
    });
  }
  public async getModulesByCourseId(
    courseId: number,
  ):
Promise<CourseModulesGetAllItemResponseDto
[]> {
    const modules = await
this.#moduleRepository.getAllByCourseId({
courseId });
    return modules;
  }
  public async getCourseModulesCount(courseId:
number): Promise<number> {
    const countDto =
      await
this.#moduleRepository.getAllCourseModulesCou
ntByCourseId({
        courseId,
      });
    return countDto.value;
  }
  public async createModulesByCourseId(
    serviceCourseId: number,
    dbCourseId: number,
  ): Promise<void> {
    const courseData = await
this.#udemyService.getModulesByCourseId(
      serviceCourseId,
    );
    if (!courseData.length) {
      throw new CoursesModulesError({
        message:
ExceptionMessage.UDEMY_SERVER_RETURN
ED_AN_INVALID_RESPONSE,
      });
    }

    for (const courseModule of courseData) {
```

```
    await this.create({
      ...courseModule,
      description: courseModule.description
        ? sanitizeHTML(courseModule.description)
        : null,
      courseId: dbCourseId,
    });
  }
 }
 public getById({
   courseId,
   moduleId,
 }: CourseModuleGetRequestParamsDto):
Promise<CourseModuleGetByIdResponseDto |
null> {
   return this.#moduleRepository.getById({
courseId, moduleId });
 }
}
export { CourseModule };
```

Файл backend/src/services/groups-to-permissions/groups-to-permissions.service.ts

```
import { GroupsToPermissionsResponseDto }
from '~/common/types/types';
import { groupsToPermissions as
groupsToPermissionsRep } from
'~/data/repositories/repositories';
type Constructor = {
 groupsToPermissionsRepository: typeof
groupsToPermissionsRep;
};
class GroupsToPermissions {
 #groupsToPermissionsRepository: typeof
groupsToPermissionsRep;
 public constructor({
groupsToPermissionsRepository }: Constructor) {
   this.#groupsToPermissionsRepository =
groupsToPermissionsRepository;
 }
 public async
createGroupsToPermissions(groupsToPermissions
: {
   groupId: number;
   permissionId: number;
 }):
Promise<GroupsToPermissionsResponseDto> {
   const model = await
this.#groupsToPermissionsRepository.create(
     groupsToPermissions,
   );
   return {
     id: model.id,
     groupId: model.groupId,
     permissionId: model.permissionId,
   };
 }
 public async
updateGroupsToPermissions(groupsToPermission
s: {
   groupId: number;
   permissionIds: number[];
 }): Promise<void> {
   const { groupId, permissionIds } =
groupsToPermissions;
   return
this.#groupsToPermissionsRepository.update({
     groupId,
     permissionIds,
   });
 }
}
export { GroupsToPermissions };
```

Файл backend/src/services/http/http.service.ts

```
import axios, {
 AxiosError,
 AxiosInstance,
 AxiosRequestHeaders,
 AxiosResponse,
} from 'axios';
import { HttpMethod } from
'~/common/enums/enums';
import { HttpOptions } from
'~/common/types/types';
import { HttpError } from
'~/exceptions/exceptions';
class Http {
 #http: AxiosInstance;
 public constructor() {
   this.#http = axios.create();
 }
 public async load<T = unknown>(
   url: string,
   options: Partial<HttpOptions> = {},
 ): Promise<T> {
   const { headers = {}, method =
HttpMethod.GET } = options;
   const res = this.#http
     .request({
       url,
       method,
```

```
      headers: headers as AxiosRequestHeaders |
undefined,
    })
    .then(this.getData<T>)
    .catch(this.throwError);
  return res;
 }
 private getData<T = unknown>(res:
AxiosResponse): T {
  return res.data;
 }
 private throwError(err: AxiosError): never {
  if (err.response) {
   throw new HttpError({
    status: err.response.status,
    message: err.message,
   });
  }
  throw err;
 }
}
export { Http };
```

Файл backend/src/services/token/token.service.ts

```
import {
 decodeJwt,
 generateSecret,
 JWTPayload,
 jwtVerify,
 SignJWT,
} from 'jose';
import { TokenPayload } from
'~/common/types/types';
type Constructor = {
 alg: string;
 expiresIn: string;
};
class Token {
 #alg: string;
 #expiresIn: string;
 public constructor({ alg, expiresIn }:
Constructor) {
  this.#alg = alg;
  this.#expiresIn = expiresIn;
 }
 public async create(data: TokenPayload):
Promise<string> {
  const secretKey = await
generateSecret(this.#alg);
  return new SignJWT(data)
   .setProtectedHeader({ alg: this.#alg })
   .setExpirationTime(this.#expiresIn)
   .sign(secretKey);
 }
 public async verify<T>(token: string):
Promise<JWTPayload & T> {
  const secretKey = await
generateSecret(this.#alg);
  const { payload } = await jwtVerify(token,
secretKey);
  return payload as JWTPayload & T;
 }
 public async decode<T>(token: string):
Promise<JWTPayload & T> {
  const data = decodeJwt(token);
  return data as JWTPayload & T;
 }
}
export { Token };
```

Файл backend/src/services/transaction/transaction.service.ts

```
import {
 EntityPagination,
 EntityPaginationRequestQueryDto,
 TransactionCreateArgumentsDto,
 TransactionGetAllItemResponseDto,
 TransactionUpdateStatusDto,
} from '~/common/types/types';
import { transaction as transactionRep } from
'~/data/repositories/repositories';
import { convertPageToZeroIndexed } from
'~/helpers/helpers';
type Constructor = {
 transactionRepository: typeof transactionRep;
};
class Transaction {
 #transactionRepository: typeof transactionRep;
 public constructor({ transactionRepository }:
Constructor) {
  this.#transactionRepository =
transactionRepository;
 }
 public getById(id: number):
Promise<TransactionGetAllItemResponseDto> {
  return this.#transactionRepository.getById(id);
 }
 public getHoldBySenderAndReceiverId(
  senderId: number,
  receiverId: number,
 ): Promise<TransactionGetAllItemResponseDto>
{
  return
this.#transactionRepository.getHoldBySenderAnd
ReceiverId(
```

```
      senderId,
      receiverId,
    );
  }
  public getTransactionsByUserId(
    userId: number,
    { count, page }:
EntityPaginationRequestQueryDto,
  ):
Promise<EntityPagination<TransactionGetAllItem
ResponseDto>> {
    const zeroIndexPage =
convertPageToZeroIndexed(page);
    return
this.#transactionRepository.getTransactionsByUse
rId(userId, {
      count,
      page: zeroIndexPage,
    });
  }
  public create(
    transactionCreateBody:
TransactionCreateArgumentsDto,
  ): Promise<TransactionGetAllItemResponseDto>
{
    return
this.#transactionRepository.create(transactionCrea
teBody);
  }
  public updateStatus(
    transactionUpdateStatusBody:
TransactionUpdateStatusDto,
  ): Promise<TransactionGetAllItemResponseDto>
{
    return this.#transactionRepository.updateStatus(
      transactionUpdateStatusBody,
    );
  }
}
export { Transaction };
```

Файл backend/src/services/udemy/udemy.service.ts

```
import { HttpMethod } from
'~/common/enums/enums';
import {
  UdemyCourseGetResponseDto,
  UdemyModuleGetResponseDto,
  UdemyModulesGetResponseDto,
} from '~/common/types/types';
import { UdemyError } from
'~/exceptions/exceptions';
import { http as httpServ } from
'~/services/services';
type Constructor = {
  httpService: typeof httpServ;
  baseUrl: string;
  clientId: string;
  clientSecret: string;
};
class Udemy {
  #MODULE_API_PAGE_SIZE = 100;
  #INITIAL_PAGE = 1;
  #MODULE_TYPE = 'chapter';
  #authorizationToken: string;
  #baseUrl: string;
  #httpService: typeof httpServ;
  #clientId: string;

  #clientSecret: string;
  public constructor({
    httpService,
    baseUrl,
    clientId,
    clientSecret,
  }: Constructor) {
    this.#authorizationToken = this.getToken();
    this.#baseUrl = baseUrl;
    this.#httpService = httpService;
    this.#clientId = clientId;
    this.#clientSecret = clientSecret;
  }
  public async getCourseByUrl(url: URL):
Promise<UdemyCourseGetResponseDto> {
    try {
      const courseIdOrSlug = url.pathname
        .split('/')
        .filter(Boolean)
        .pop() as string;
      const headers = this.getHeaders();
      const res = await
this.#httpService.load<UdemyCourseGetResponse
Dto>(
        this.getCourseRequestUrl(courseIdOrSlug),
        { headers, method: HttpMethod.GET },
      );
      return res;
    } catch {
      throw new UdemyError();
    }
  }
  public async getModulesByCourseId(
    courseId: number,
  ): Promise<UdemyModuleGetResponseDto[]> {
    const modules = await
this.fetchAllCourseModules(
      this.#INITIAL_PAGE,
      courseId,
    );

    return modules;
```

```
  }
  private async fetchAllCourseModules(
    page: number,
    courseId: number,
  ): Promise<UdemyModuleGetResponseDto[]> {
    const modules:
UdemyModuleGetResponseDto[] = [];
    let fetchedModulesData:
UdemyModulesGetResponseDto =
      await this.fetchModulesPage(
        this.getModuleRequestUrl(courseId, page),
        this.getHeaders(),
      );
    while (fetchedModulesData.next) {
      fetchedModulesData.results.forEach((item) =>
{
        if (item._class === this.#MODULE_TYPE)
{
          modules.push(item);
        }
      });
      fetchedModulesData = await
this.fetchModulesPage(
        fetchedModulesData.next,
        this.getHeaders(),
      );
    }
    fetchedModulesData.results.forEach((item) =>
{
      if (item._class === this.#MODULE_TYPE) {
        modules.push(item);
      }
    });
    return modules;
  }
  private async fetchModulesPage(
    requestUrl: string,
    headers: Record<string, string>,
  ): Promise<UdemyModulesGetResponseDto> {
    const fetchedModules =
      await
this.#httpService.load<UdemyModulesGetRespon
seDto>(requestUrl, {
        headers,
        method: HttpMethod.GET,
      });
    return fetchedModules;
  }
  private getCourseRequestUrl(courseIdOrSlug:
string): string {
    return `${
      this.#baseUrl

}courses/${courseIdOrSlug}?fields[course]=title,d
escription,url,image_480x270`;
  }
  private getModuleRequestUrl(courseId: number,
page: number): string {
    return `${
      this.#baseUrl

}courses/${courseId}/public-curriculum-items/?pa
ge=${page}&page_size=${
      this.#MODULE_API_PAGE_SIZE
    }`;
  }
  private getHeaders(): Record<string, string> {
    const headers = {
      Authorization: `Basic
${this.#authorizationToken}`,
    };
    return headers;
  }
  private getToken(): string {
    return Buffer.from(
      `${this.#clientId}:${this.#clientSecret}`,
      'utf-8',
    ).toString('base64');
  }
}
export { Udemy };
```

Файл backend/src/server.ts

```
import fastifyCookie from '@fastify/cookie';
import fastifyCors from '@fastify/cors';
import fastifyMultipart from '@fastify/multipart';
import fastifyStatic from '@fastify/static';
import fastifySwagger from '@fastify/swagger';
import Fastify from 'fastify';
import Knex from 'knex';
import path from 'node:path';
import { Model } from 'objection';
import { initApi } from '~/api/api';
import { ENV, FileSizeBytesValue } from
'~/common/enums/enums';
import { socket as socketService } from
'~/services/services';
import knexConfig from '../knexfile';
const app = Fastify({
  logger: {
    transport: {
      target: 'pino-pretty',
    },
  },
});
socketService.initializeIo(app.server);
Model.knex(Knex(knexConfig[ENV.APP.NODE_
ENV]));
```

```
app.register(fastifyCors);
app.register(fastifyMultipart, {
  limits: {
    fileSize: FileSizeBytesValue.FIVE_MB,
  },
  attachFieldsToBody: true,
  throwFileSizeLimit: false,
});
app.register(fastifyCookie);
app.register(initApi, {
  prefix: ENV.API.V1_PREFIX,
  baseURL:
`${ENV.APP.BASE_URL}${ENV.API.V1_PREF
IX}`,
  frontendURL: ENV.APP.FRONTEND_URL,
});
const staticPath = path.join(__dirname, '../public');
app.register(fastifyStatic, {
  root: staticPath,
  prefix: '/',
});
app.setNotFoundHandler((_req, res) => {
  res.sendFile('index.html', staticPath);
});
app.register(fastifySwagger, {
  mode: 'static',
  prefix: '/documentation',
  exposeRoute: true,
  specification: {
    path: path.resolve(__dirname,
'./documentation/documentation.yaml'),
    baseDir: path.resolve(__dirname,
'./documentation'),
  },
});
app.listen(
  { port: ENV.APP.SERVER_PORT, host:
ENV.APP.SERVER_HOST },
  (err, address) => {
    if (err) {
      app.log.error(err);
    }
    app.log.info(`Listening on: ${address};
Environment: ${ENV.APP.NODE_ENV}`);
  },
);
```