**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**
**“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ**
**імені ІГОРЯ СІКОРСЬКОГО”**

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

**До захисту допущено:**
**В.о. Завідувач кафедри**
_____ Михайло НОВОТАРСЬКИЙ
(підпис)
“__” _____2025 р.

# Дипломний проєкт

**на здобуття ступеня бакалавра**
**за освітньо-професійною програмою “Інженерія програмного забезпечення комп’ютерних систем”**
**спеціальності 121 “Інженерія програмного забезпечення”**

на тему: Онлайн-платформа для управління курсами з елементами менторства та багаторівневого доступу

Виконав : студент __4__ курсу, групи _ІМ-11
(шифр групи)
Шевирьов Владислав Олегович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник асистент кафедри, доктор філософії Шульга М.В. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль) ас. кафедри, Гончаренко О. О. _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент __асистент кафедри БМІ Матвійчук О.В._____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2025 р.

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**
**“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ**
**імені ІГОРЯ СІКОРСЬКОГО”**

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“Інженерія програмного забезпечення комп’ютерних систем”

спеціальності 121 “Інженерія програмного забезпечення”

**ЗАТВЕРДЖУЮ**
**В.о. Завідувач кафедри**
_____ Михайло НОВОТАРСЬКИЙ
(підпис)
“__” _____2025 р.

**ЗАВДАННЯ**
на бакалаврський дипломний проєкт студента

Шевирьова Владислава Олеговича

1. Тема проєкту *Онлайн-платформа для управління курсами з елементами менторства та багаторівневого доступу*
керівник проєкту Шульга Максим Володимирович, асистент кафедри, доктор філософії
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 23 травня 2025 року №1705-с

2. Термін здачі студентом закінченого проєкту *03 червня 2025 р.*

3. Вихідні дані до проєкту *технічна документація, теоретичні дані.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)
Розділ 1. Постановка завдання та аналіз існуючих рішень.

Розділ 2. Огляд технологій для розробки.

Розділ 3. Деталі розробки системи.

Розділ 4. Дослідження та аналіз розробленої системи.

5. Перелік графічного матеріалу (з точним позначенням обов'язкових креслень) структурна схема системи (структурна схема), діаграма структури даних (функціональна схема), алгоритм дій програмного забезпечення (принципова схема).

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

| Розділ | Консультант | Підпис, дата | |
|---|---|---|---|
| | | Завдання видав | Завдання прийняв |
| Нормоконтроль | Гончаренко О.О. | | |
| | | | |

7. Дата видачі завдання 30 жовтня 2024 р.

Календарний план

| № п/п | Найменування етапів дипломного проєкту | Терміни виконання етапів проєкту | Примітки |
|---|---|---|---|
| 1. | *Затвердження теми проєкту* | 30.11.2024 | |
| *2.* | *Вивчення та аналіз завдання* | 01.11.2024 – 28.02.2025 | |
| *3.* | *Розробка архітектури та загальної структури системи* | 01.03.2025 – 10.04.2025 | |
| *4.* | *Програмна реалізація системи* | 11.04.2025 – 20.05.2025 | |
| *5.* | *Оформлення пояснювальної записки* | 24.05.2025 – 02.06.2025 | |
| 6. | *Захист програмного продукту* | 28.05.2025 | |
| 7. | *Передзахист* | 02.06.2025 | |
| *8.* | *Захист* | 19.06.2025 | |

Студент-дипломник _____ Владислав ШЕВИРЬОВ
(підпис)

Керівник проєкту _____ Максим ШУЛЬГА
(підпис)

## АНОТАЦІЯ

Даний дипломний проєкт зосереджений на розробці серверної частини для онлайн-платформи, призначеної для адміністрування навчальних курсів. Система спроектована з урахуванням сучасних вимог до освітніх платформ, зокрема, передбачає реалізацію функціоналу менторської підтримки слухачів та впровадження гнучкої системи багаторівневого розмежування прав доступу. У процесі роботи було здійснено аналіз вимог до подібних платформ та обґрунтовано вибір технологічного стеку для серверної реалізації. Результатом дипломного проєкту є готове до використання серверне API, яке надає необхідний набір інструментів для управління навчальним контентом, користувачами, їх ролями та взаємодіями в рамках освітнього процесу з елементами менторства.



## ANNOTATION

This thesis centers on the development of the server-side application for an online platform dedicated to the administration of educational courses. Engineered to align with contemporary standards for educational technologies, the system notably incorporates mentorship support for learners and a versatile multi-level access control mechanism. The project entailed a comprehensive analysis of requirements for analogous platforms, guiding the reasoned selection of the technology stack for the server-side implementation. The principal outcome of this thesis is a deployment-ready server API, furnishing the essential toolkit for managing learning content, user accounts, role-based permissions, and interactions within an educational framework enriched by mentorship capabilities.

| Довідки | Формат | *Значення* | *Найменування* | Кіл. листів | № екземпляра | *Додаток* |
|---|---|---|---|---|---|---|
| | | | Документація загальна | | | |
| | | | Знову розроблена | | | |
| | | | | | | |
| | ***А4*** | ***ІАЛЦ.467200.002 ТЗ*** | Онлайн-платформа для управління | 3 | | |
| | | | курсами з елементами менторства | | | |
| | | | та багаторівневого доступу | | | |
| | | | Технічне завдання | | | |
| | | | | | | |
| | ***А4*** | ***ІАЛЦ.467200.003 ПЗ*** | Онлайн-платформа для управління | 67 | | |
| | | | курсами з елементами менторства | | | |
| | | | та багаторівневого доступу | | | |
| | | | Пояснювальна записка | | | |
| | | | | | | |
| | ***А4*** | ***ІАЛЦ.467200.004 Д1*** | Онлайн-платформа для управління | 1 | | |
| | | | курсами з елементами менторства | | | |
| | | | та багаторівневого доступу | | | |
| | | | Архітектурна діаграма застосунку | | | |
| | | | (структурна схема) | | | |
| | | | | | | |
| | ***А4*** | ***ІАЛЦ.4672008.005 Д2*** | Онлайн-платформа для управління | 1 | | |
| | | | курсами з елементами менторства | | | |
| | | | та багаторівневого доступу | | | |
| | | | Діаграма доменних класів | | | |
| | | | (функціональна схема) | | | |
| | | | | | | |
| | ***А4*** | ***ІАЛЦ.467200.006 Д3*** | Онлайн-платформа для управління | 1 | | |
| | | | курсами з елементами менторства | | | |
| | | | та багаторівневого доступу | | | |
| | | | Алгоритм дії веб-застосунку | | | |
| | | | (принципова схема) | | | |
| | | | | | | |
| | ***А4*** | ***ІАЛЦ.467200.007 Д4*** | Онлайн-платформа для управління | 15 | | |
| | | | курсами з елементами менторства | | | |
| | | | та багаторівневого доступу | | | |
| | | | Текст програмного коду | | | |
| | | | | | | |

| | | | | | |
|---|---|---|---|---|---|
| | | | | | ***ІАЛЦ.467200.001 ОА*** |
| *Зм* | *Лист* | *№ докум.* | *Підп* | *Дата* | |
| ***Розроб.*** | | Шевирьов В.О. | | | Онлайн-платформа для управління курсами з елементами менторства та багаторівневого доступу<br>Опис альбому |
| ***Перев.*** | | Шульга М.В. | | | |

| Літ. | | | Арк уш | Аркушів |
|---|---|---|---|---|
| | | | 1 | 1 |

***НТУУ "КПІ" ФІОТ ІМ-11***

# ТЕХНІЧНЕ ЗАВДАННЯ
# ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Онлайн-платформа для управління курсами з елементами менторства та багаторівневого доступу*»

Київ – 2025

# ЗМІСТ



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | ІАЛЦ.467200.002 ТЗ | | |
| | | № докум. | Підпис | Дата | | | |
| Розробив | | Шевирьов В. О. | | | Онлайн-платформа для управління курсами з елементами менторства та багаторівневого доступу<br>Технічне завдання | Літ. | Аркуш 1 / Аркушів 3 |
| Перевірив | | Шульга М. В. | | | | | |
| | | Матвійчук О. В. | | | | НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІМ-11 | |
| Н. Контр. | | Гончаренко О. О. | | | | | |
| Затвердив | | | | | | | |

# 1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання поширюється на розробку серверної частини онлайн-платформи для управління навчальними курсами, що реалізує функціонал менторства та систему багаторівневого доступу, а також на подальшу підтримку та вдосконалення розробленої системи. Областю застосування цього застосунку є сфера корпоративного навчання, академічна освіта, ринок комерційних освітніх послуг.

# 2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки даного застосунку є завдання для виконання роботи кваліфікаційно-освітнього рівня «бакалавр інженерії програмного забезпечення», який був затверджений факультетом “Інформатики та обчислювальної техніки” кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний інститут ім. Ігоря Сікорського».

# 3. МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою та призначенням даної роботи є розробка серверної частини онлайн-платформи для управління навчальними курсами, що реалізує функціонал менторства та систему багаторівневого доступу.

# 4. ДЖЕРЕЛА РОЗРОБКИ

Даний дипломний проект базується на офіційних документах, інтернет-публікаціях по темі та науково-технічній літературі.

# 5. ТЕХНІЧНІ ВИМОГИ

## 5.1. Вимоги до розробленого продукту

Розроблена система має виконувати такі вимоги:

- Простий та логічно побудований інтерфейс системи.
- Надати можливість через інтерфейс передавати дані для управління курсами, користувачами та їх правами доступа.
- Надати можливість гнучко запитувати та отримувати необхідні дані.
- Надати документацію для розробленого застосунку.

## 5.2. Вимоги до програмного забезпечення

- ОС: Linux (64-bit), Windows 10 (64-bit) або macOS 11+
- Docker Engine: версія 20.10 або новіша
- Docker Compose: версія 2.2 або новіша.

## 5.3. Вимоги до апаратної частини

- ЦП не менше ніж 2 віртуальних ядра (vCPU).
- ROM не менше ніж 64 ГБ.
- RAM не менше ніж 4 ГБ.

# 6 ЕТАПИ РОЗРОБКИ

| Назва етапів виконання | Термін виконання |
|---|---|
| Затвердження теми роботи | 01.11.2024 – 30.11.2024 |
| Вивчення та аналіз завдання | 01.12.2024 – 28.02.2025 |
| Розробка архітектури та загальної структури системи | 01.03.2025 – 10.04.2025 |
| Розробка структур окремих частин системи | 11.04.2025 – 30.04.2025 |
| Програмна реалізація системи | 01.05.2025 – 20.05.2025 |
| Виправлення помилок | 21.05.2025 – 23.05.2025 |
| Оформлення пояснювальної записки | 24.05.2025 – 02.06.2025 |

# ПОЯСНЮВАЛЬНА ЗАПИСКА
# ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Онлайн-платформа для управління курсами з елементами менторства та багаторівневого доступу*»

Київ – 2025

ЗМІСТ

# ПЕРЕЛІК СКОРОЧЕНЬ

| | |
|---|---|
| JIT | (Just-In-Time Compilation) компіляція під час виконання |
| I/O | (Input/Output) ввод-вивід |
| RBAC | (Role-Based Access Control) модель доступу на основі ролей |
| DI | (Dependency Injection) механізм впровадження залежностей |
| ORM | (Object-Relational Mapping) об'єктно-реляційне відображення |
| JWT | (JSON Web Token) токен доступу |
| AWS | (Amazon Web Services) хмарна сервісна платформа |
| CQS | (Command-Query Separation) патерн програмування |
| DTO | (Data Transfer Objects) об'єкти передачі даних |

# ВСТУП

Сучасний етап розвитку освітніх технологій характеризується масовим переходом до онлайн-форматів навчання. Це, у свою чергу, висуває підвищені вимоги до функціональності та гнучкості програмних платформ, що використовуються для організації навчального процесу. Якщо раніше основний акцент робився на доставці контенту та базовому адмініструванні, то сьогодні актуалізується потреба в інструментах, що забезпечують глибшу взаємодію, персоналізацію та ефективне управління складними освітніми структурами.

Типові проблеми, з якими стикаються користувачі освітніх онлайн-платформ, включають:

- складність у наданні доступу до навчальних матеріалів та функціоналу для різних категорій користувачів;
- недостатня інтеграція механізмів зворотного зв'язку та менторської допомоги безпосередньо в навчальне середовище;

У даній роботі розглядається концепція та підходи до проектування онлайн-платформи, призначеної для ефективного управління навчальними курсами з інтеграцією двох ключових компонентів: елементів менторства та системи багаторівневого доступу. Під елементами менторства розуміється набір функціональних можливостей, що дозволяють організувати взаємодію між досвідченими наставниками (менторами) та студентами для перевірки завдань. Багаторівневий доступ, у свою чергу, передбачає створення гнучкої системи ролей та дозволів, яка чітко регламентує права кожного користувача на доступ до певних розділів платформи, навчальних матеріалів та адміністративних функцій. Існуючі на ринку платформи часто пропонують ці можливості в обмеженому вигляді. Ця робота зосереджена на розробці такого архітектурного рішення, де зазначені компоненти є невід'ємною частиною функціонуючої системи, спрямованої на підвищення якості освітнього процесу та зручності його адміністрування. Будуть розглянуті основні підходи до

структурування такої платформи, взаємодії її модулів та забезпечення ефективної реалізації заявленого функціоналу.

# РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

## 1.1 Розвиток онлайн-навчання та навчальних систем

### 1.1.1 Історія дистанційної освіти

Дистанційна освіта, як освітня парадигма, що дозволяє здобувати знання без необхідності фізичної присутності в навчальному закладі, має глибші історичні корені, ніж це зазвичай уявляється. Початкові форми такого навчання, що базувалися на кореспондентському обміні навчальними матеріалами за допомогою поштових служб, з'явилися ще у XVIII столітті. У XIX столітті технологічний прогрес, зокрема розвиток радіо та телебачення, надав нові інструменти для трансляції освітнього контенту, розширюючи аудиторію та географію навчання. Ці ранні етапи заклали фундаментальні принципи освітньої взаємодії на відстані [1].

Справжній же трансформаційний зсув у розвитку дистанційної освіти відбувся з поширенням персональних комп'ютерів та, особливо, з глобальним проникненням мережі Інтернет. Це відкрило епоху цифрового навчання, уможлививши створення інтерактивних онлайн-курсів, віртуальних навчальних середовищ та спеціалізованих програмних платформ для управління навчальним процесом. Доступність мультимедійного контенту, можливість миттєвого зворотного зв'язку та асинхронного навчання стали визначальними перевагами цього нового формату.

Безпрецедентним каталізатором для розвитку та масового впровадження технологій дистанційної освіти стала глобальна пандемія COVID-19, що розпочалася у 2019-2020 роках. Умови вимушеної ізоляції та карантинних обмежень призвели до стрімкого, практично одномоментного переходу освітніх установ усіх рівнів – від шкіл до університетів та корпоративних навчальних центрів – на дистанційний формат роботи. Цей період характеризувався не лише експоненційним зростанням попиту на відповідні технологічні рішення,

але й значним прискоренням інновацій у сфері освітніх технологій (EdTech), а також виявленням як сильних сторін, так і певних викликів цифрової освіти. Пік індустрії припав саме на цей час, коли дистанційне навчання з нішового явища перетворилося на глобальну освітню норму.

Наслідки цього періоду продовжують впливати на сучасний освітній ландшафт, закріпивши за дистанційною освітою статус невід'ємної складової глобальної системи освіти та стимулюючи подальший розвиток гібридних моделей навчання. Таким чином, еволюція дистанційної освіти від простих кореспондентських курсів до складних інтерактивних онлайн-екосистем демонструє її адаптивність та зростаючий потенціал у відповідь на технологічні зміни та суспільні виклики.

### 1.1.2 Роль онлайн-платформ у навчанні

Онлайн-платформи утвердилися як невід’ємний та багатогранний інструмент сучасної освітньої системи, кардинально трансформувавши традиційні підходи до здобуття та поширення знань. Їхня роль виходить далеко за межі простого сховища навчальних матеріалів, перетворюючись на комплексні цифрові середовища, що активно впливають на всі аспекти освітнього процесу.

Насамперед, онлайн-платформи відіграють ключову роль у розширенні доступу до освіти. Вони ефективно долають географічні бар'єри, надаючи можливість навчатися особам з віддалених регіонів або тим, хто має обмежену мобільність. Гнучкість, що забезпечується можливістю асинхронного навчання, дозволяє студентам формувати індивідуальний графік, поєднуючи освітні потреби з професійною діяльністю чи особистими обставинами. Це сприяє реалізації концепції безперервної освіти та відкриває шлях до здобуття нових знань і кваліфікацій для широкого кола людей, незалежно від віку чи попереднього освітнього рівня.

Іншим важливим аспектом є трансформація педагогічних підходів та збагачення арсеналу методів навчання. Онлайн-платформи надають викладачам потужні інструменти для створення мультимедійного контенту, що сприяє кращому засвоєнню матеріалу та підвищенню залученості студентів. Крім того, платформи пропонують різноманітні засоби оцінювання, забезпечуючи оперативний зворотний зв'язок.

Онлайн-платформи також суттєво впливають на оптимізацію адміністративних процесів в освітніх установах. Вони забезпечують централізоване управління навчальним контентом, спрощують процедури зарахування студентів, ведення обліку успішності та формування звітності. Автоматизація рутинних завдань дозволяє викладачам та адміністративному персоналу зосередитися на більш значущих аспектах педагогічної та організаційної роботи.

Водночас, ефективне використання онлайн-платформ вимагає від усіх учасників освітнього процесу розвитку відповідних цифрових компетентностей. Взаємодія з платформою не лише передбачає наявність базових навичок роботи з комп'ютером та Інтернетом, але й сприяє їхньому подальшому вдосконаленню, що є критично важливим в умовах сучасного інформаційного суспільства та ринку праці.

Таким чином, роль онлайн-платформ у навчанні є фундаментальною та багатоаспектною. Вони не просто технологічний засіб, а каталізатор освітніх інновацій, що розширює можливості, підвищує ефективність та якість навчального процесу, роблячи його більш гнучким, доступним та орієнтованим на потреби сучасного здобувача освіти. Подальший розвиток таких платформ, зокрема з інтеграцією більш складних функцій, як-от елементи наставництва та гнучкі системи доступу, є логічним кроком на шляху до створення ще досконаліших освітніх середовищ.

## 1.2 Вимоги до платформ управління курсами

Для забезпечення високої якості та ефективності освітнього процесу в цифровому середовищі, сучасні платформи управління курсами повинні відповідати комплексу чітко окреслених вимог. Ці вимоги формують основу для оцінки наявних рішень та проектування нових систем, охоплюючи як фундаментальні функціональні можливості, так і специфічні аспекти, що відображають актуальні освітні потреби. У наступних підрозділах буде проведено детальний аналіз цих ключових критеріїв, що визначають спроможність платформи підтримувати якісне та гнучке онлайн-навчання.

### 1.2.1 Ключові функції платформи

Ефективність та практична цінність будь-якої сучасної онлайн-платформи для навчання визначається насамперед її основними функціональними можливостями. Ці ключові функції є фундаментом, на якому будується весь освітній процес в цифровому середовищі, забезпечуючи його цілісність, керованість та інтерактивність. Розглянемо детальніше кожну з них.

Центральне місце займає функціонал, пов'язаний з роботою з матеріалами курсу. Це передбачає надання викладачам інтуїтивно зрозумілих інструментів для створення, завантаження, структурування та актуалізації навчального контенту в різноманітних форматах – від текстових лекцій та презентацій до мультимедійних файлів. Для студентів платформа повинна забезпечувати легкий та логічний доступ до всіх необхідних матеріалів, зручну навігацію по них. Якісна організація контенту, його чітке розмежування за модулями чи темами, а також можливість пошуку є критично важливими для ефективного самостійного навчання.

Наступною невід'ємною складовою є ведення навчального процесу. Ця функція охоплює інструменти для адміністрування та методичного супроводу навчання. Платформа має дозволяти створювати чітку структуру курсу,

визначати послідовність вивчення матеріалу. Важливим є механізм видачі індивідуальних завдань, їх збору, перевірки та оцінювання, а також надання конструктивного зворотного зв'язку. Відстеження прогресу кожного студента дозволяє викладачам моніторити процес навчання, а студентам – бачити власні досягнення та зони, що потребують додаткової уваги.

Третім ключовим елементом є забезпечення можливостей для спілкування та взаємодії. Ефективне навчання неможливе без комунікації, тому платформа повинна містити набір інструментів для її підтримки.

Четвертою фундаментальною функцією є аналіз успішності студентів. Платформа повинна надавати інструменти для проведення різних видів оцінювання знань. Для самих студентів доступ до аналітики власної успішності є важливим фактором мотивації та саморегуляції.

Отже, ці чотири групи функцій – робота з матеріалами, ведення навчального процесу, спілкування та взаємодія, а також аналіз успішності – складають кістяк будь-якої повноцінної освітньої онлайн-платформи. Їх узгоджена та якісна реалізація є запорукою створення ефективного та привабливого навчального середовища.

### 1.2.2 Налаштування прав доступу

Одним із фундаментальних аспектів безпеки, керованості та функціональної доцільності онлайн-платформи для навчання є ретельно продумана система налаштування прав доступу. Цей механізм визначає, які дії можуть виконувати різні категорії користувачів та до яких саме інформаційних ресурсів чи функціональних блоків системи вони матимуть доступ. Ефективне налаштування прав доступу є критично важливим для забезпечення конфіденційності даних, запобігання несанкціонованим змінам та створення впорядкованого робочого середовища для кожного учасника освітнього процесу.

В основі гнучкої системи налаштування прав лежить можливість гранулярного розподілу повноважень. Це означає, що адміністратор платформи або уповноважені особи повинні мати інструменти для детального визначення дозволів не лише на рівні загальних ролей (як-от "студент", "викладач", "ментор", "адміністратор"), але й, за потреби, для окремих користувачів або груп. Налаштування можуть стосуватися прав на перегляд, створення, редагування та видалення навчального контенту, доступу до певних курсів чи їхніх розділів, можливості взаємодіяти з іншими користувачами, переглядати персональні дані, адмініструвати оцінки, а також керувати налаштуваннями самої платформи.

Сучасні платформи часто реалізують модель доступу на основі ролей RBAC, де кожній ролі присвоюється певний набір дозволів, а користувачам призначаються відповідні ролі. Важливою вимогою є не лише наявність стандартного набору ролей, але й можливість їх модифікації або створення кастомних ролей для задоволення унікальних потреб конкретного навчального закладу чи освітнього проекту. Такий підхід забезпечує необхідний баланс між структурованістю, безпекою та адаптивністю системи управління доступом, що є запорукою її ефективного функціонування в різноманітних освітніх контекстах. Правильно сконфігуровані права доступу не лише захищають систему, але й оптимізують досвід користувача, надаючи йому лише ті інструменти та інформацію, які є релевантними для його завдань та повноважень.

## 1.3 Порівняльний аналіз існуючих рішень

Далі, у табл. 1.1, буде проаналізовано функціональні можливості, а також висвітлено ключові переваги та прогалини таких популярних онлайн-платформ, як Udemy та Coursera, особливо з точки зору реалізації ними функцій наставництва та забезпечення гнучкого багаторівневого доступу.

Таблиця 1.1. – Порівняльний аналіз функціоналу Coursera та Udemy

| Вимога | Coursera | Udemy |
|---|---|---|
| Управління контентом курсу | Так | Так |
| Доступ, навігація та організація матеріалів для студентів | Так | Так |
| Моніторинг прогресу та аналітика успішності | Так | Так |
| Структурування курсу та завдань. | Так | Так |
| Підтримка та налаштування різних ролей користувачів | Обмежено | Ні |
| Гнучкість конфігурації ролей та дозволів | Ні | Ні |

Як наочно демонструють узагальнені дані, ретельно представлені у таблиці 1.1, кожна з проаналізованих провідних онлайн-освітніх платформ, а саме Coursera та Udemy, характеризується власним специфічним набором функціональних переваг та, водночас, певними іманентними обмеженнями. Це особливо помітно при розгляді їх можливостей крізь призму вимог до сучасної системи управління курсами, яка б ефективно інтегрувала елементи менторства та підтримувала гнучкий багаторівневий доступ. Детальний розгляд показує, що жодне з існуючих рішень не може бути визнане абсолютно універсальним чи ідеальним для задоволення всього спектру потреб, окреслених у рамках

даного дипломного дослідження, що лише підкреслює актуальність та практичну значущість проектування спеціалізованої платформи.

Після всебічного та поглибленого аналізу представлених у таблиці аспектів, що охоплюють як беззаперечно сильні сторони реалізації певних функцій, так і потенційно слабкі місця або недостатню гнучкість в інших, стає цілком очевидною нагальна необхідність формування зваженого та обґрунтованого підходу до визначення архітектури та функціонального наповнення власної програмної розробки. Проведене дослідження існуючих популярних рішень слугує не стільки для здійснення прямого вибору на користь однієї з них як еталону, скільки для глибокого та критичного осмислення накопиченого в індустрії досвіду, ідентифікації передових практик та виявлення тих ключових аспектів, що вимагають найбільш пильної уваги при проектуванні інноваційної онлайн-платформи. Особливий інтерес у цьому контексті представляють механізми підтримки менторських програм та реалізація розгалуженої системи управління правами доступу, оскільки саме ці компоненти є визначальними для специфіки об'єкта розробки.

Таким чином, знання та висновки, здобуті в ході цього порівняльного аналізу, формують міцне теоретичне та практичне підґрунтя для подальших, більш конкретизованих етапів наукового пошуку та інженерної розробки. Вони дозволяють не лише констатувати поточний стан справ у сегменті освітніх онлайн-технологій, але й набагато чіткіше окреслити детальні специфікації та пріоритетні функціональні вимоги до проектованої платформи. Наприклад, стає зрозуміло, що особливої уваги при подальшій розробці заслуговує проектування інтуїтивно зрозумілих, але водночас функціонально багатих інструментів для забезпечення ефективної та плідної взаємодії між студентами та менторами. Не менш важливим є створення надійної та гнучкої системи адміністрування прав доступу, яка б органічно враховувала різноманітність потенційних ролей користувачів – від студентів та викладачів до менторів різного рівня та адміністративного персоналу платформи. Подальша деталізація

цих та інших суттєвих аспектів, а також обґрунтування обраних проектних рішень, будуть послідовно та розгорнуто представлені у наступних розділах даної дипломної роботи. Це, в свою чергу, має на меті створення програмного продукту, що не тільки відповідатиме сучасним освітнім трендам, але й пропонуватиме унікальні переваги в контексті персоналізованої освітньої підтримки та забезпечення диференційованого доступу до навчальних ресурсів та функціоналу платформи.

# ВИСНОВОК ДО РОЗДІЛУ 1

У рамках першого розділу даної дипломної роботи було здійснено комплексний та багатоаспектний аналіз предметної області, що охоплює теоретичні засади, еволюцію та сучасний стан систем онлайн-навчання, а також ключові вимоги до функціональності платформ управління курсами, з особливим акцентом на інтеграцію елементів менторства та забезпечення гнучкого багаторівневого доступу. Проведене дослідження дозволило сформувати глибоке розуміння контексту, в якому здійснюватиметься розробка проектованої системи.

Насамперед, було детально простежено еволюційний шлях розвитку освітніх технологій на відстані, починаючи від найперших форм кореспондентського навчання, що зародилися ще у XVIII столітті, через епоху використання радіо та телебачення як інструментів трансляції знань, і закінчуючи сучасною добою домінування Інтернет-технологій. Особливо було підкреслено визначальний вплив глобальної мережі та поширення персональних комп'ютерів на становлення цифрового навчання, що уможливило створення інтерактивних та мультимедійних освітніх продуктів. Було відзначено, що глобальна пандемія COVID-19 виступила безпрецедентним каталізатором, який не лише експоненційно прискорив масове впровадження дистанційних освітніх форматів на всіх рівнях, але й стимулював значні інновації, виявивши водночас як потужні переваги, так і певні системні виклики цифрової трансформації освіти. Цей період остаточно закріпив за дистанційною освітою статус невід'ємної та стратегічно важливої складової глобального освітнього простору.

Далі, було всебічно розглянуто фундаментальну та багатогранну роль, яку сучасні онлайн-платформи відіграють у навчальному процесі. Встановлено, що їх значення виходить далеко за межі простого зберігання та доставки навчального контенту, перетворюючи їх на комплексні цифрові екосистеми, що

кардинально впливають на доступність освіти, педагогічні методики та адміністративне управління. Було наголошено на здатності онлайн-платформ долати географічні та часові бар'єри, сприяти реалізації концепції безперервної освіти, збагачувати арсенал дидактичних інструментів викладачів та оптимізувати управлінські процеси в освітніх установах. Водночас, було акцентовано увагу на тому, що ефективне використання таких систем вимагає від усіх учасників освітнього процесу належного рівня цифрових компетентностей, розвиток яких також є одним із побічних, але важливих результатів взаємодії з освітніми технологіями. Саме на цьому тлі було обґрунтовано логічність подальшого розвитку таких платформ у напрямку інтеграції більш складних та персоналізованих функцій, таких як розширені елементи наставництва та адаптивні системи управління доступом.

Центральне місце в аналітичній частині першого розділу посіло формулювання та детальне обґрунтування комплексу чітко окреслених вимог, яким повинні відповідати сучасні платформи управління курсами для забезпечення високої якості, ефективності та безпеки освітнього процесу в цифровому середовищі. Ці вимоги були структуровані за двома ключовими напрямками: фундаментальні функціональні можливості та специфічні аспекти, пов'язані з управлінням доступом. До першої групи увійшли вимоги до інструментів для роботи з навчальними матеріалами, ведення навчального процесу, забезпечення комунікації та взаємодії, а також аналізу успішності студентів. У рамках другої групи було детально розглянуто вимоги до системи налаштування прав доступу, підкреслено важливість гранулярного розподілу повноважень, підтримки рольової моделі з можливістю визначення таких ключових ролей, як "студент", "ментор" та "адміністратор", а також налаштування прав на виконання конкретних дій та доступ до інформаційних ресурсів системи. Було наголошено, що саме ретельно продумана система доступу є критично важливою для забезпечення конфіденційності, безпеки даних та створення впорядкованого робочого середовища.

На основі сформульованого переліку вимог, у підрозділі 1.3 було здійснено порівняльний аналіз функціональних можливостей двох провідних існуючих рішень на ринку освітніх технологій – платформ Coursera та Udemy. Цей аналіз, результати якого узагальнені у таблиці 1.1 та детально прокоментовані у супровідному тексті, дозволив виявити як беззаперечні сильні сторони цих широко використовуваних систем, так і певні обмеження та функціональні прогалини, особливо коли йдеться про комплексну реалізацію розширених функцій менторства та надання справді гнучких, кастомізованих багаторівневих систем доступу, що є центральними елементами концепції даного дипломного дослідження. Було встановлено, що, незважаючи на їх значний функціональний потенціал, жодна з розглянутих платформ не забезпечує в повній мірі оптимального поєднання інструментарію для глибокої та систематичної менторської підтримки студентів з одночасним наданням адміністраторам достатньо гнучких та потужних механізмів для детального налаштування гранулярних прав доступу для широкого спектру категорій користувачів, включаючи чітко відокремлену та функціонально насичену роль ментора.

Таким чином, проведений у першому розділі всебічний аналіз предметної області, який включав вивчення історичного контексту та сучасних тенденцій розвитку онлайн-навчання, визначення ключової ролі сучасних онлайн-платформ, детальне формулювання функціональних та технічних вимог до них, а також критичну оцінку можливостей наявних на ринку популярних рішень, закладає міцний та обґрунтований теоретичний фундамент для подальшої роботи. Отримані результати та зроблені висновки дозволяють не лише констатувати актуальний стан справ у досліджуваній сфері, але й чітко сформулювати завдання та окреслити напрямки наступних етапів дослідження. Ці етапи будуть безпосередньо спрямовані на проектування архітектури та розробку детальної концепції онлайн-платформи для управління курсами, яка б максимально повно враховувала виявлені потреби в глибокій інтеграції

елементів менторської підтримки та забезпеченні гнучкого, адаптивного багаторівневого доступу. Кінцевою метою такої розробки є створення більш ефективного, персоналізованого, безпечного та мотивуючого навчального середовища, що відповідатиме найсучаснішим освітнім викликам та потребам усіх учасників освітнього процесу.

# РОЗДІЛ 2. ОГЛЯД ОБРАНИХ ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ

## 2.1 Мова програмування та середовище виконання

Вибір фундаментальних технологій, зокрема мови програмування та середовища виконання, є визначальним етапом на шляху створення будь-якого програмного забезпечення, оскільки він закладає підвалини для подальшої розробки, впливаючи на ефективність, надійність, масштабованість та довгострокову підтримку системи. Для серверної частини онлайн-платформи, що є об'єктом даної дипломної роботи, цей вибір набуває особливої ваги з огляду на очікуване навантаження, вимоги до швидкодії та необхідність забезпечення гнучкості для майбутнього розвитку функціоналу.

Далі буде представлено детальний аналіз та обґрунтування рішень щодо основних інструментів. Основна увага буде приділена аргументації вибору мови програмування TypeScript та середовища виконання Node.js. Буде проведено оцінку їхніх сучасних можливостей, ключових переваг для реалізації складних, високопродуктивних веб-застосунків та відповідності специфічним цілям і завданням даного проєкту, що спрямований на створення ефективного та надійного освітнього середовища.

### 2.1.1 Мова TypeScript

Вибір мови програмування є фундаментальним рішенням при розробці будь-якого програмного забезпечення, оскільки він суттєво впливає на процес розробки, якість коду, можливості масштабування та подальшу підтримку проєкту. Для реалізації серверної частини онлайн-платформи, що є об'єктом даної дипломної роботи, після ретельного аналізу було обрано мову TypeScript. Щоб повною мірою зрозуміти переваги цього вибору, необхідно спочатку розглянути мову JavaScript, на якій базується TypeScript.

JavaScript є легкою, інтерпретованою (JIT) мовою програмування, що підтримує множинні парадигми, включно з об'єктно-орієнтованою на основі прототипів, імперативною та функціональною. JavaScript характеризується динамічною типізацією, що означає визначення типів змінних під час виконання програми, а не на етапі компіляції [11]. Це забезпечує високу гнучкість, але водночас може призводити до помилок, пов'язаних з типами, які виявляються лише на стадії виконання, що ускладнює розробку та налагодження великих і складних систем.

TypeScript є мовою програмування з відкритим вихідним кодом, розробленою та підтримуваною компанією Microsoft. Ключовою особливістю TypeScript є те, що вона позиціонується як строгий синтаксичний суперсет (надмножина) JavaScript, який розширює останню шляхом додавання опціональної статичної типізації, а також інших можливостей [10].

Принцип роботи TypeScript полягає в тому, що написаний на ньому код не виконується безпосередньо браузерами чи середовищем Node.js. Натомість він проходить процес транспіляції за допомогою компілятора TypeScript[12]. У результаті цього процесу TypeScript-код перетворюється на еквівалентний, чистий JavaScript-код, який уже може бути виконаний у будь-якому середовищі, що підтримує JavaScript. Важливо зазначити, що TypeScript зберігає повну сумісність із JavaScript: будь-який валідний JavaScript-код є також валідним TypeScript-кодом. Це дозволяє поступово інтегрувати TypeScript у вже існуючі JavaScript-проєкти.

Застосування TypeScript у рамках розробки серверної частини онлайн-платформи з елементами менторства та багаторівневого доступу надає низку суттєвих переваг:

- **Підвищення надійності та якості коду:** Статична типізація дозволяє виявляти значну кількість помилок, пов'язаних з типами даних, ще на етапі компіляції, а не під час виконання програми. Це особливо критично для

серверних застосунків, де помилки можуть призвести до збоїв у роботі всієї системи та порушення цілісності даних. Для онлайн-платформи, що оперує даними користувачів, курсів, прогресу навчання та прав доступу, забезпечення коректності обробки цих даних є пріоритетом.

- **Покращена читабельність та супроводжуваність коду:** Явне визначення типів даних, інтерфейсів та структур робить код більш зрозумілим та самодокументованим. Це спрощує навігацію по кодовій базі, полегшує командну розробку та значно знижує поріг входження для нових розробників або для повернення до коду через тривалий час. В умовах розробки складної платформи, яка може розвиватися та доповнюватися новими функціями, цей аспект є надзвичайно важливим.
- **Ефективний рефакторинг та масштабування:** Завдяки статичній типізації та потужним інструментам аналізу коду, рефакторинг (зміна внутрішньої структури коду без зміни його зовнішньої поведінки) стає значно безпечнішим та контрольованішим. Компілятор TypeScript допомагає відстежити всі місця, де зміни типів або інтерфейсів можуть спричинити проблеми. Це дозволяє більш впевнено вносити зміни та розширювати функціональність платформи, забезпечуючи її масштабованість.
- **Розширена підтримка інструментів розробки (Developer Experience):** TypeScript значно покращує досвід розробника завдяки таким можливостям, як автодоповнення коду, статичний аналіз, навігація по коду та підказки щодо типів безпосередньо в інтегрованому середовищі розробки. Це прискорює процес написання коду та зменшує кількість механічних помилок.
- **Сумісність з сучасною екосистемою:** TypeScript ідеально інтегрується з Node.js та популярними фреймворками. Це забезпечує доступ до великої

кількості бібліотек та інструментів, які підтримують або написані на TypeScript, що спрощує розробку та використання передових практик.

Таким чином, вибір TypeScript як основної мови програмування для серверної частини онлайн-платформи є стратегічно обґрунтованим рішенням. Воно спрямоване на забезпечення високої якості, надійності та підтримуваності коду, що є критично важливим для розробки складного програмного продукту, призначеного для тривалої експлуатації та подальшого розвитку. Можливості статичної типізації та розширена інструментальна підтримка сприятимуть ефективній реалізації функціоналу, пов'язаного з управлінням курсами, менторством та багаторівневим доступом.

### 2.1.2 Середовище виконання Node JS

Після вибору TypeScript як основної мови програмування, наступним критично важливим кроком є визначення середовища виконання для серверної частини онлайн-платформи. Середовище виконання забезпечує необхідну інфраструктуру для інтерпретації та виконання коду поза межами веб-браузера, надаючи доступ до системних ресурсів, файлової системи, мережевих операцій та інших можливостей, необхідних для повноцінного серверного застосунку. Для даного проєкту було обрано Node.js, і цей вибір ґрунтується на його унікальних архітектурних особливостях та доведеній ефективності для розробки високопродуктивних мережевих застосунків.

Node.js є асинхронним, подієво-орієнтованим середовищем виконання JavaScript, побудованим на високопродуктивному JavaScript-рушії V8 від Google, тому самому, що використовується у веб-браузері Chrome. Node.js розширює можливості JavaScript, дозволяючи виконувати його на серверній стороні та створювати різноманітні типи застосунків, від простих скриптів до складних корпоративних систем [15].

Ключовою архітектурною особливістю Node.js, що визначає його продуктивність та масштабованість, є його однопотокова, неблокуюча модель

вводу-виводу (I/O), керована циклом подій (event loop). На відміну від традиційних багатопотокових серверних моделей, де кожен клієнтський запит може оброблятися в окремому потоці (що призводить до значних витрат ресурсів на управління потоками), Node.js використовує один основний потік для обробки всіх запитів. Коли надходить операція вводу-виводу (наприклад, читання файлу, запит до бази даних, мережевий запит), Node.js не блокує цей потік в очікуванні завершення операції. Замість цього, він реєструє функцію зворотного виклику (callback) і передає операцію на виконання операційній системі або спеціалізованим робочим потокам (worker threads) для асинхронної обробки. Основний потік продовжує обробляти інші події та запити. Коли асинхронна операція завершується, відповідна функція зворотного виклику додається до черги подій і виконується циклом подій, як тільки основний потік стає вільним. Такий підхід дозволяє Node.js ефективно обробляти велику кількість одночасних з'єднань з мінімальними накладними витратами ресурсів [16].

Для серверної частини онлайн-платформи, яка передбачає обробку численних запитів від користувачів, застосування Node.js надає такі значущі переваги:

- **Висока продуктивність для I/O-орієнтованих завдань:** Онлайн-платформи за своєю природою є системами, де домінують операції вводу-виводу. Асинхронна неблокуюча архітектура Node.js ідеально підходить для таких сценаріїв, забезпечуючи швидку обробку запитів та низьку затримку відповіді, що є критичним для позитивного досвіду користувачів.
- **Масштабованість:** Легкість та ефективність обробки великої кількості одночасних з'єднань робить Node.js чудовим вибором для побудови систем, які можуть обслуговувати зростаючу кількість користувачів без значного погіршення продуктивності. Можливість горизонтального

масштабування шляхом запуску декількох екземплярів Node.js-застосунку також є важливою перевагою.

- **Велика та активна екосистема:** Node.js має доступ до величезного репозиторію пакетів Node Package Manager, який є найбільшою екосистемою бібліотек з відкритим вихідним кодом у світі. Це дозволяє розробникам швидко знаходити та інтегрувати готові рішення для широкого спектру завдань, від роботи з базами даних та реалізації протоколів автентифікації до інструментів для тестування та розгортання, що значно прискорює процес розробки.
- **Підтримка сучасних веб-технологій:** Node.js активно розвивається та підтримує новітні веб-стандарти і протоколи, що робить його придатною платформою для створення сучасних API, які є основою взаємодії серверної частини онлайн-платформи з клієнтськими застосунками.

Враховуючи вищезазначені аспекти, зокрема високу продуктивність при обробці I/O-операцій, можливості масштабування та багату екосистему, Node.js є оптимальним середовищем виконання для розробки серверної частини проектованої онлайн-платформи. Його поєднання з TypeScript дозволяє створювати швидкі, надійні та легко підтримувані серверні застосунки.

## 2.2 Фреймворк та бібліотеки

Після визначення та обґрунтування фундаментальних компонентів технологічного стеку, а саме мови програмування TypeScript та середовища виконання Node.js, які закладають міцну основу для серверної розробки, наступним логічним і стратегічно важливим етапом є ретельний вибір архітектурного фреймворку та набору ключових бібліотек. Ці інструменти вищого рівня абстракції відіграють критичну роль у формуванні загальної структури серверного застосунку, ефективній та стандартизованій реалізації його основної бізнес-логіки, забезпеченні належного рівня безпеки, а також у

вирішенні низки специфічних технічних завдань, що неминуче виникають у процесі створення сучасної, багатофункціональної онлайн-платформи.

Якщо обрані мова програмування та середовище виконання надають базові будівельні блоки та інструментарій для виконання коду, то архітектурний фреймворк пропонує узгоджену концепцію, перевірені патерни проектування та методологію розробки. Це сприяє підвищенню продуктивності команди, покращенню якості коду за рахунок його стандартизації та полегшенню довгострокової підтримки і масштабування системи. Водночас, спеціалізовані бібліотеки дозволяють інтегрувати в проєкт надійні, перевірені часом та широкою спільнотою розробників рішення для таких важливих аспектів, як управління доступом користувачів, валідація та трансформація даних, конфігурування застосунку та захист від поширених веб-вразливостей, тим самим звільняючи розробників від необхідності "винаходити колесо".

Далі буде детально розглянуто та всебічно аргументовано вибір основного архітектурного фреймворку, який стане каркасом для серверної частини проектованої онлайн-платформи. Також буде обґрунтовано застосування низки ключових бібліотек, що спільно забезпечуватимуть реалізацію специфічного функціоналу, надійність роботи та безпеку системи.

### 2.2.1 Фреймворк Nest JS

Для реалізації серверної логіки онлайн-платформи, що розробляється в рамках даної дипломної роботи, після ретельного аналізу альтернатив було обрано NestJS – прогресивний фреймворк для платформи Node.js, призначений для створення ефективних, надійних та високомасштабованих серверних застосунків.

NestJS розроблений з урахуванням та для повноцінного використання можливостей TypeScript, що дозволяє максимально розкрити переваги статичної типізації для побудови складних систем. Фундаментальною особливістю NestJS є його архітектура, яка значною мірою натхненна

принципами, закладеними в Angular – популярному фреймворку для розробки клієнтських застосунків. Цей вплив проявляється у використанні таких концепцій, як модулі (Modules), контролери (Controllers), провайдери (Providers), зокрема сервіси (Services), та потужного механізму впровадження залежностей - Dependency Injection (DI).

Модульність є наріжним каменем архітектури NestJS. Кожен застосунок складається з одного або декількох модулів, що інкапсулюють пов'язану функціональність. Такий підхід сприяє чіткому розмежуванню відповідальності, покращує структуру проєкту, полегшує його розуміння, тестування окремих компонентів та повторне використання коду. Для комплексної онлайн-платформи модульна архітектура є критично важливою для забезпечення довгострокової підтримуваності та можливості поступового розширення.

Механізм впровадження залежностей в NestJS дозволяє створювати слабко зв'язані компоненти. Замість того, щоб класи самі створювали свої залежності, ці залежності "впроваджуються" в них ззовні контейнером DI. Це значно спрощує тестування (особливо модульне тестування шляхом підміни залежностей на "заглушки" – mocks) та підвищує гнучкість системи. Контролери відповідають за обробку вхідних HTTP-запитів та делегування бізнес-логіки відповідним сервісам (провайдерам), що реалізують патерн "Сервісний шар" (Service Layer) та інкапсулюють основну логіку застосунку. Таке чітке розділення відповідальностей сприяє чистоті архітектури.

NestJS активно використовує декоратори TypeScript – спеціальну форму синтаксичного цукру, що дозволяє додавати метадані до класів, методів або властивостей. Це робить код більш декларативним та читабельним.

Важливою перевагою NestJS є його гнучкість щодо базового HTTP-сервера. За замовчуванням він використовує перевірений часом Express.js, що забезпечує широку сумісність з існуючою екосистемою middleware. Однак, для проєктів, де пріоритетом є максимальна продуктивність, NestJS дозволяє легко

інтегрувати та використовувати Fastify – високопродуктивний HTTP-фреймворк, відомий своєю низькою ресурсоємністю. Ця можливість вибору надає розробникам гнучкість без необхідності відмовлятися від структури та переваг самого NestJS [8].

Порівняння NestJS з альтернативними Node.js фреймворками:

- **Express.js:** Будучи де-факто стандартом для багатьох Node.js проєктів, Express.js є мінімалістичним та неструктурованим фреймворком. Він надає базовий набір інструментів для веб-розробки, залишаючи вибір архітектури, структуру проєкту та інтеграцію додаткових інструментів повністю на розсуд розробника. Хоча така гнучкість може бути привабливою для невеликих проєктів, для складних систем, як проектована онлайн-платформа, відсутність чітких архітектурних настанов може призвести до неузгодженості коду, ускладнення підтримки та масштабування. NestJS, на відміну від Express.js, пропонує готову, продуману структуру, що базується на передових практиках, тим самим знижуючи поріг входження для розробки великих застосунків та забезпечуючи їхню кращу архітектурну цілісність [2].
- **Fastify:** Як вже зазначалося, Fastify відомий своєю винятковою продуктивністю та низькими накладними витратами. Він використовує JSON Schema для валідації та серіалізації даних, що сприяє швидкодії. Однак, Fastify, подібно до Express.js, є менш структурованим порівняно з NestJS. Хоча NestJS може використовувати Fastify як HTTP-провайдер, перевага NestJS полягає в тому, що він додає до продуктивності Fastify потужний архітектурний шар [9].

Для розробки серверної частини онлайн-платформи NestJS пропонує оптимальний баланс між структурованістю, гнучкістю та продуктивністю розробки, тому це найкращій вибір.

### 2.2.2 Об’єктно-реляційний мапер MikroORM

Ефективна взаємодія між об'єктно-орієнтованим кодом застосунку та реляційною базою даних є критично важливим аспектом розробки серверної частини. Для подолання розриву між цими двома парадигмами широко використовуються інструменти об'єктно-реляційного відображення (ORM). ORM дозволяє розробникам працювати з даними бази через звичні об'єктні моделі та абстрагуватися від написання значної частини SQL-запитів вручну. Для даного проєкту, після аналізу доступних рішень для екосистеми TypeScript/Node.js, було обрано MikroORM.

MikroORM — це сучасний ORM для TypeScript та JavaScript, що базується на патернах Data Mapper, Unit of Work та Identity Map. Його філософія полягає у наданні розробнику явного контролю над поведінкою ORM та забезпеченні чіткого розділення відповідальності між доменною моделлю та логікою персистентності [14].

Ключові патерни, реалізовані в MikroORM, надають суттєві переваги:

- **Data Mapper:** Цей патерн передбачає існування окремого шару, відповідального за перенесення даних між об'єктами в пам'яті та записами в базі даних. На відміну від патерну ActiveRecord (де об'єкти самі відповідають за свою персистентність), Data Mapper дозволяє доменним сутностям залишатися "чистими" об'єктами, незалежними від логіки збереження. Це сприяє кращому дотриманню принципу єдиної відповідальності та полегшує тестування доменної логіки.
- **Unit of Work:** Цей патерн відстежує всі зміни, що відбуваються з об'єктами сутностей протягом однієї бізнес-транзакції. Коли транзакція завершується, Unit of Work автоматично визначає змінені об'єкти та генерує мінімально необхідний набір SQL-запитів для синхронізації стану об'єктів у пам'яті зі станом даних у базі. Це дозволяє значно оптимізувати кількість запитів до бази даних, групуючи операції та уникаючи

надлишкових оновлень, що позитивно впливає на загальну продуктивність системи. Також Unit of Work відіграє ключову роль у забезпеченні атомарності операцій та консистентності даних.

- **Identity Map:** Цей патерн гарантує, що для кожного запису з бази даних, завантаженого в пам'ять протягом однієї сесії (або Unit of Work), існує лише один унікальний екземпляр об'єкта. Якщо застосунок повторно запитує ту саму сутність, Identity Map повертає вже існуючий екземпляр, замість того, щоб створювати новий та завантажувати дані з бази ще раз. Це запобігає проблемам з розсинхронізацією даних при роботі з кількома екземплярами одного й того ж об'єкта та зменшує кількість запитів до бази даних, що також сприяє підвищенню продуктивності.

Переваги MikroORM та порівняння з альтернативами:

- **Продуктивність:** Завдяки ефективній реалізації патернів Unit of Work та Identity Map, MikroORM демонструє високу продуктивність, особливо в сценаріях зі складними транзакціями та великою кількістю операцій читання/запису. Оптимізація запитів та мінімізація взаємодії з базою даних є його сильною стороною.
- **Підтримуваність та активний розвиток:** MikroORM є активно розроблюваним проєктом з регулярними релізами, що включають нові можливості та швидкі виправлення помилок. Це забезпечує надійну підтримку та впевненість у довгостроковій життєздатності обраного інструменту.
- **Порівняння з TypeORM:** TypeORM є одним із найпопулярніших ORM в екосистемі TypeScript, що підтримує як патерн ActiveRecord, так і елементи Data Mapper [19]. Він відомий своєю широкою функціональністю та зрілістю. Однак, MikroORM часто позиціонується як більш строгий у дотриманні патерну Data Mapper, що забезпечує чистішу доменну модель. Також, реалізація Unit of Work та Identity Map в MikroORM вважається

однією з його ключових переваг, що може призводити до кращої продуктивності в певних сценаріях порівняно з деякими підходами в TypeORM. Вибір між ними часто зводиться до переваг у стилі API та ступеня контролю над процесом персистентності.

• **Порівняння з Prisma:** Prisma представляє дещо інший підхід до взаємодії з базами даних, позиціонуючи себе як "ORM наступного покоління". Вона використовує декларативний schema-first підхід (визначення схеми даних у спеціальному файлі schema.prisma) та генерує типізований клієнт для виконання запитів (Prisma Client) [20]. Prisma забезпечує високий рівень типобезпеки та чудовий досвід розробника. Однак, Prisma Client не є ORM у класичному розумінні роботи з об'єктами-сутностями та їх станами, як це реалізовано в MikroORM чи TypeORM через патерни Data Mapper або ActiveRecord. MikroORM надає більш традиційний об'єктно-орієнтований підхід до управління персистентністю, що може бути перевагою для розробників, звиклих до таких патернів, та для проєктів зі складною доменною логікою, тісно пов'язаною зі станом сутностей. Реалізація Unit of Work та Identity Map в MikroORM також є відмінною рисою, що забезпечує контроль над транзакціями та оптимізацію на рівні об'єктів.

Для онлайн-платформи, що розробляється, з її потенційно складною моделлю даних, обраний підхід MikroORM з його чітким розділенням відповідальності, ефективним управлінням станом сутностей через Unit of Work та Identity Map, а також сильною типізацією завдяки TypeScript, є оптимальним. Це дозволить створити надійний, продуктивний та добре структурований шар доступу до даних, що є критично важливим для стабільної роботи та подальшого розвитку платформи.

### 2.2.3 Бібліотека CASL

Забезпечення надійного та гнучкого механізму авторизації є однією з критично важливих задач при розробці будь-якої багатокористувацької системи, особливо такої складної, як онлайн-платформа для управління курсами з елементами менторства та багаторівневим доступом. Авторизація визначає, які дії користувач може виконувати та до яких ресурсів він має доступ після успішної автентифікації. Для реалізації цієї функціональності в проєктованій системі було обрано бібліотеку CASL.

CASL – це ізоморфна бібліотека авторизації для JavaScript та TypeScript, призначена для декларативного визначення та перевірки прав доступу в застосунках. Вона дозволяє описувати дозволи у простій та зрозумілій формі, концентруючись на тому, що користувач може (can) або не може (cannot) робити [5]. Такий підхід сприяє створенню чіткої та легко підтримуваної логіки управління правами.

Застосування CASL для розробки онлайн-платформи надає такі суттєві переваги:

- **Декларативність:** Правила доступу визначаються у явній та легко зрозумілій формі, що спрощує їх аналіз, модифікацію та підтримку. Це особливо важливо для системи з різними ролями та потенційно складною ієрархією прав.
- **Гнучкість та гранулярність:** CASL дозволяє визначати дуже деталізовані права доступу, аж до окремих полів об'єктів та з урахуванням динамічних умов. Це є критично важливим для реалізації багаторівневого доступу, де різні користувачі повинні мати різні рівні взаємодії з одними й тими ж ресурсами.
- **Підтримка складних сценаріїв:** Завдяки умовній логіці, CASL ефективно вирішує завдання, пов'язані з контекстно-залежними дозволами, що часто зустрічаються в освітніх платформах.

- **Ізоморфність:** Можливість використання однакової логіки авторизації як на серверній стороні, так і на клієнтській, що дозволяє уніфікувати перевірку прав та покращити досвід користувача, наприклад, приховуючи недоступні елементи інтерфейсу.

Враховуючи необхідність реалізації складної та гнучкої системи розмежування прав для різних ролей користувачів онлайн-платформи, а також потребу в контролі доступу до специфічних даних та функцій, бібліотека CASL надає потужний та водночас зрозумілий інструментарій. Її декларативний підхід та можливість визначення умовних дозволів роблять її оптимальним вибором для забезпечення надійного та легко підтримуваного механізму авторизації.

### 2.2.4 Бібліотека aws-jwt-verify

Для забезпечення безпечної та надійної роботи онлайн-платформи для управління курсами критично важливим є впровадження ефективного механізму автентифікації користувачів. Автентифікація являє собою процес верифікації особистості користувача, що підтверджує його право на доступ до системи та її ресурсів. У даному проєкті для управління ідентифікацією користувачів та видачі токенів доступу було обрано хмарний сервіс AWS Cognito. Цей сервіс використовує відкритий стандарт JSON Web Tokens (JWT) для безпечного представлення інформації про автентифікованого користувача.

JSON Web Token (JWT) – це відкритий стандарт (RFC 7519) для створення токенів доступу [13]. JWT дозволяє безпечно передавати інформацію про користувача між клієнтом та сервером у вигляді компактного JSON-об'єкта. Цей об'єкт має цифровий підпис, що гарантує його справжність та цілісність. Кожен JWT складається з трьох основних частин: заголовка (де вказано тип токена та алгоритм шифрування), корисного навантаження (з інформацією про

користувача, його ролі та термін дії токена) та цифрового підпису (для перевірки автентичності).

Процес автентифікації з використанням AWS Cognito та JWT у проєктованій системі реалізовано наступним чином:

1. Користувач ініціює процес входу, надаючи свої облікові дані (наприклад, логін та пароль) або використовуючи конфігуровані федеративні провайдери ідентичності (наприклад, Google, Facebook) через інтерфейси, що взаємодіють з AWS Cognito.

2. AWS Cognito проводить перевірку наданих облікових даних. У разі успішної автентифікації Cognito генерує набір JWT (зазвичай токен доступу, токен ідентифікації та токен оновлення) та повертає їх клієнтському застосунку.

3. Клієнтський застосунок зберігає отримані токени (зокрема, токен доступу) та додає його до заголовків авторизації (Authorization header, зазвичай як Bearer token) кожного наступного запиту до захищених ресурсів серверної частини платформи, розробленої на NestJS.

4. Серверна частина, отримуючи запит, повинна перевірити валідність JWT. Для цієї мети в проєкті використовується спеціалізована бібліотека aws-jwt-verify. Ця бібліотека призначена для валідації JWT, виданих сервісом AWS Cognito.

Використання AWS Cognito як провайдера ідентичності та бібліотеки aws-jwt-verify для валідації токенів на серверній стороні в рамках даного проєкту на NestJS надає низку суттєвих переваг:

- **Надійна та специфічна валідація токенів:** Бібліотека aws-jwt-verify оптимізована для роботи з токенами від AWS Cognito. Вона автоматично обробляє отримання та кешування публічних ключів від AWS, що необхідно для криптографічної перевірки підпису токена. Це забезпечує

високий ступінь надійності валідації та мінімізує ризик використання невалідних або скомпрометованих токенів [3].

- **Безстанова архітектура серверної частини:** Оскільки AWS Cognito видає JWT, які є самодостатніми, серверна частина платформи може залишатися безстановою (stateless). Уся необхідна для ідентифікації та базової авторизації інформація міститься в самому токені. Це спрощує архітектуру, полегшує горизонтальне масштабування серверних компонентів та покращує загальну відмовостійкість системи.
- **Спрощена інтеграція в екосистему NestJS:** Бібліотека aws-jwt-verify легко інтегрується в архітектуру NestJS, зокрема для створення кастомних охоронців (Guards), які захищають маршрути API, та для отримання даних користувача з валідованого токена в контролерах та сервісах [4].

Таким чином, для розроблюваної онлайн-платформи для управління курсами, яка має підтримувати безпечну автентифікацію значної кількості користувачів, забезпечувати можливість масштабування та надійно ідентифікувати користувачів для подальшого розмежування їхніх прав доступу, вибір AWS Cognito як провайдера ідентичності та використання бібліотеки aws-jwt-verify для валідації токенів на боці серверного застосунку NestJS є технологічно обґрунтованим, сучасним та ефективним рішенням. Такий підхід дозволяє реалізувати надійну систему безпеки, делегувавши значну частину специфічних завдань з управління ідентифікацією спеціалізованому та перевіреному хмарному сервісу.

## 2.3 Інфраструктура

Після детального обґрунтування вибору технологій, що складають основу програмної реалізації серверної частини онлайн-платформи, логічним продовженням є розгляд інструментарію та методологій, які забезпечать її ефективне розгортання, надійну експлуатацію та гнучке управління.

Відповідно, у цьому розділі буде представлено аналіз та аргументацію щодо вибору ключових інфраструктурних компонентів та підходів, що застосовуватимуться у проєкті.

### 2.3.1 Сховища даних (PostgreSQL та Redis)

Для забезпечення надійності, цілісності та високої продуктивності при роботі з даними, у рамках даного проєкту було прийнято рішення використовувати комбінований підхід, що поєднує реляційну систему управління базами даних PostgreSQL та сховище даних в оперативній пам'яті Redis, кожне з яких виконує свої специфічні завдання.

PostgreSQL обрано як основну систему управління реляційною базою даних (СУРБД) для проєкту. PostgreSQL є потужною, об'єктно-реляційною СУБД з відкритим вихідним кодом, що завоювала репутацію завдяки своїй надійності, відповідності стандартам SQL, розширюваності та гарантіям цілісності даних через повну підтримку транзакцій ACID (Atomicity, Consistency, Isolation, Durability). Для онлайн-платформи, де цілісність даних про користувачів, їхній прогрес, оплати (якщо передбачені), результати оцінювання та структуру курсів є критично важливою, властивості ACID PostgreSQL забезпечують необхідний рівень надійності [17].

Ключові переваги PostgreSQL, що обґрунтовують його вибір для даного проєкту, включають:

- **Структуроване зберігання даних:** Реляційна модель ідеально підходить для зберігання структурованих.
- **Гарантія цілісності даних:** Завдяки механізмам зовнішніх ключів, обмежень цілісності (constraints) та підтримці ACID-транзакцій, PostgreSQL забезпечує високий рівень консистентності даних.
- **Розширені можливості SQL:** PostgreSQL підтримує складні SQL-запити, включаючи підзапити, об'єднання (joins), віконні функції та

агрегацію, що необхідно для формування аналітичних звітів, пошуку та фільтрації даних за різними критеріями.

- **Розширюваність:** Можливість створювати власні типи даних, функції та оператори, а також підтримка таких розширень, як PostGIS для геопросторових даних або повнотекстовий пошук, надає гнучкість для майбутнього розвитку платформи.
- **Надійність та стабільність:** PostgreSQL відомий своєю стабільністю в роботі під високими навантаженнями та здатністю обробляти великі обсяги даних, що важливо для масштабованої онлайн-платформи.

Redis (Remote Dictionary Server), у свою чергу, обрано як допоміжне високопродуктивне сховище даних типу "ключ-значення", що працює переважно в оперативній пам'яті. Завдяки своїй архітектурі, Redis забезпечує надзвичайно низьку затримку при операціях читання та запису, що робить його ідеальним інструментом для специфічних завдань, де швидкість доступу є пріоритетом [18].

Використання PostgreSQL як основного сховища для персистентних, структурованих та критично важливих даних, у поєднанні з Redis для кешування та швидкого доступу до часто використовуваних або тимчасових даних, формує збалансовану та ефективну стратегію управління даними. PostgreSQL забезпечує надійність та цілісність, тоді як Redis сприяє підвищенню загальної продуктивності та чутливості онлайн-платформи. Такий підхід дозволяє оптимально використовувати сильні сторони кожного з обраних сховищ даних для задоволення різноманітних вимог сучасного веб-застосунку.

### 2.3.2 AWS Cognito

У міру зростання вимог до безпеки, масштабованості та користувацького досвіду сучасних онлайн-платформ, критично важливим аспектом стає надійна

та гнучка система управління ідентифікацією користувачів та контролю доступу. Забезпечення безпечного зберігання облікових даних, реалізація різноманітних механізмів автентифікації, управління життєвим циклом користувачів, а також надання можливостей для федерації ідентичностей є фундаментальними завданнями, що безпосередньо впливають на довіру користувачів та загальну безпеку системи. Для комплексного вирішення цих завдань у проєктованій онлайн-платформі для управління курсами було обрано сервіс Amazon Web Services (AWS) Cognito.

AWS Cognito – це керований сервіс від Amazon Web Services, що надає інструменти для простого та безпечного додавання функцій реєстрації, входу та контролю доступу для веб- та мобільних додатків. Він дозволяє розробникам швидко інтегрувати потужні механізми управління ідентифікацією без необхідності розробки та підтримки власної складної інфраструктури. AWS Cognito складається з двох основних компонентів: пулів користувачів (User Pools) та пулів ідентичностей (Identity Pools) [3]. Для даного проєкту ключову роль відіграють саме пули користувачів AWS Cognito, які є захищеними каталогами користувачів, що забезпечують функції реєстрації, автентифікації, управління профілями та безпеки облікових записів.

Впровадження AWS Cognito для управління ідентифікацією користувачів на онлайн-платформі надає наступні ключові переваги:

- **Значне скорочення часу та зусиль на розробку:** Делегування складних завдань управління ідентичностями, таких як розробка форм реєстрації/входу, безпечне зберігання паролів, реалізація MFA, обробка відновлення паролів, керованому сервісу дозволяє команді розробників зосередитись на основній функціональності платформи.
- **Висока масштабованість та надійність:** Як керований сервіс AWS, Cognito автоматично масштабується для обробки великої кількості користувачів та запитів автентифікації, забезпечуючи високу доступність

та відмовостійкість, що є критичним для онлайн-платформ з потенційно значною аудиторією.

- **Покращений користувацький досвід:** Можливість входу через популярні соціальні мережі, прості та зрозумілі процеси реєстрації та відновлення доступу, а також надійний захист облікових записів сприяють підвищенню задоволеності та довіри користувачів.
- **Економічна ефективність:** У багатьох випадках використання керованого сервісу, такого як Cognito, з його моделлю оплати за фактичне використання (pay-as-you-go), є більш економічно вигідним, ніж розробка, розгортання та підтримка власного рішення для управління ідентифікацією, особливо враховуючи витрати на забезпечення безпеки та масштабованості.
- **Безшовна інтеграція з екосистемою AWS:** Cognito легко інтегрується з іншими сервісами AWS, такими як API Gateway (для авторизації запитів до API), AWS AppSync (для GraphQL API), AWS Lambda (для кастомної логіки), що дозволяє будувати комплексні та безпечні хмарні рішення.

Враховуючи необхідність забезпечення безпечного, гнучкого та масштабованого управління користувачами для онлайн-платформи, а також прагнення оптимізувати ресурси розробки, вибір AWS Cognito є стратегічно виправданим рішенням. Цей сервіс дозволяє реалізувати сучасну систему автентифікації та авторизації, яка відповідає високим стандартам безпеки, забезпечує зручність для користувачів та легко інтегрується в загальну архітектуру проєкту.

### 2.3.3 Docker

Забезпечення узгодженості середовищ на різних етапах життєвого циклу програмного забезпечення – від розробки та тестування до розгортання та експлуатації – є однією з ключових проблем сучасної інженерії програмного

забезпечення. Розбіжності в конфігураціях операційних систем, версіях бібліотек та інших залежностей часто призводять до непередбачуваної поведінки застосунку та ускладнюють процес його доставки. Технологія контейнеризації, і зокрема платформа Docker, пропонує ефективне вирішення цих проблем, забезпечуючи ізоляцію застосунків та їх залежностей.

Docker – це відкрита платформа для розробки, доставки та запуску застосунків у стандартизованих, ізольованих середовищах, що називаються контейнерами. Контейнеризація дозволяє "запакувати" застосунок разом з усіма його залежностями (бібліотеками, системними інструментами, кодом та середовищем виконання) в один легко переносний артефакт – образ (image). Цей образ потім може бути запущений як контейнер на будь-якій системі, де встановлено Docker, гарантуючи ідентичність поведінки застосунку незалежно від базової інфраструктури.

Основою для створення образів у Docker є Dockerfile – файл, що містить послідовність інструкцій для автоматизованого збирання образу. Ці інструкції визначають базовий образ копіювання коду застосунку, встановлення залежностей, налаштування портів та визначення команди для запуску застосунку при старті контейнера. Зібрані образи можуть зберігатися та розповсюджуватися через публічні або приватні репозиторії образів, такі як Docker Hub [7].

Для серверного застосунку онлайн-платформи використання Docker дозволить створити надійне, відтворюване та легко кероване середовище. Це спростить процес розробки, забезпечить стабільність роботи застосунку на різних етапах його життєвого циклу та закладе основу для майбутнього масштабування та ефективної експлуатації платформи.

# ВИСНОВОК ДО РОЗДІЛУ 2

У другому розділі даної дипломної роботи було проведено всебічний аналіз та детальне обґрунтування вибору технологічного стеку, що становить фундаментальну основу для розробки серверної частини проектованої онлайн-платформи для управління курсами з елементами менторства та багаторівневого доступу. Метою цього етапу було формування сучасного, надійного, масштабованого та легко підтримуваного набору інструментів, який би максимально відповідав специфічним вимогам та функціональним завданням системи.

Ключовим рішенням на рівні мови програмування та середовища виконання став вибір зв'язки TypeScript та Node.js. Застосування TypeScript дозволяє забезпечити високу якість та надійність коду завдяки статичній типізації, що є критично важливим для складних систем, якими є онлайн-платформи. Це сприяє ранньому виявленню помилок, покращує читабельність коду, полегшує рефакторинг та масштабування проєкту. У свою чергу, Node.js, з його асинхронною, подієво-орієнтованою архітектурою та високопродуктивним рушієм V8, забезпечує ефективну обробку великої кількості одночасних запитів, що є типовим для I/O-орієнтованих завдань онлайн-платформи, гарантуючи швидкість відгуку та можливість горизонтального масштабування.

В якості основного архітектурного фреймворку було обрано NestJS. Його модульна структура, заснована на принципах Angular, використання TypeScript "з коробки", потужний механізм впровадження залежностей (Dependency Injection) та підтримка сучасних патернів проектування, таких як Сервісний шар та Контролери, дозволяють створювати добре структуровані, легко тестовані та масштабовані серверні застосунки. Гнучкість NestJS у виборі базового HTTP-сервера (Express.js або Fastify) та його переваги у створенні чітких та надійних API, порівняно з більш мінімалістичними фреймворками,

роблять його оптимальним вибором для реалізації комплексної бізнес-логіки онлайн-платформи.

Для ефективної взаємодії з реляційною базою даних було обрано об'єктно-реляційний мапер MikroORM. Його прихильність до патернів Data Mapper, Unit of Work та Identity Map забезпечує чітке розділення відповідальності, оптимізацію кількості запитів до бази даних, консистентність даних та високу продуктивність. Порівняно з іншими популярними ORM, MikroORM вирізняється строгим дотриманням цих патернів та активним розвитком, що гарантує надійність шару доступу до даних для такої сутності, як PostgreSQL, обраної як основна СУРБД завдяки її надійності, ACID-сумісності та розширеним можливостям SQL. Доповнює стратегію роботи з даними Redis, високопродуктивне сховище типу "ключ-значення" в оперативній пам'яті, яке використовуватиметься для кешування часто запитуваних даних та управління сеансами, що суттєво підвищить загальну швидкодію платформи.

Забезпечення безпеки та контролю доступу є пріоритетним завданням у розробці онлайн-платформи. Для реалізації гнучкої та гранулярної системи авторизації була обрана бібліотека CASL. Вона дозволяє декларативно визначати права доступу на рівні окремих дій, ресурсів системи та специфічних умов, що є ідеальним рішенням для впровадження багаторівневого доступу для різних ролей користувачів, таких як студент, викладач, ментор та адміністратор. Автентифікація користувачів платформи реалізована з використанням сервісу AWS Cognito, який виступає як централізований провайдер ідентичності. AWS Cognito відповідає за процеси реєстрації, входу користувачів та управління їхніми обліковими записами, а також генерує JSON Web Tokens (JWT) для підтвердження їхньої особи. Подальша перевірка (валідація) цих JWT на серверній частині застосунку, розробленій на NestJS, здійснюється за допомогою спеціалізованої бібліотеки aws-jwt-verify. Такий підхід забезпечує безпечний, безстановий та масштабований механізм перевірки ідентичності, що

відповідає сучасним стандартам безпеки. Для моніторингу активності користувачів та відстеження ключових подій, пов'язаних із безпекою та управлінням ідентичностями, що є важливим аспектом загальної спостережуваності системи, використовуються вбудовані можливості сервісу AWS Cognito. Cognito надає доступ до детальних журналів аудиту (audit logs) та може бути інтегрований з іншими сервісами AWS, такими як AWS CloudTrail та Amazon CloudWatch. Це дозволяє збирати та аналізувати дані про операції автентифікації, зміни в облікових записах користувачів та інші значущі події безпеки, надаючи важливу інформацію для діагностики інцидентів та аналізу поведінки користувачів у контексті їх взаємодії з системою автентифікації.

Нарешті, для стандартизації середовищ розробки, тестування та розгортання, а також для забезпечення портативності та ізоляції застосунку, буде використана технологія контейнеризації Docker.

Таким чином, обраний у другому розділі технологічний стек є комплексним, збалансованим та сучасним рішенням. Кожен компонент був ретельно проаналізований та обґрунтований з точки зору його відповідності специфічним вимогам проєкту розробки онлайн-платформи для управління курсами. Синергія цих технологій дозволить створити надійну, продуктивну, безпечну та масштабовану серверну частину, що стане міцним фундаментом для подальшої реалізації функціоналу системи, детально описаного в наступних розділах дипломної роботи.

# РОЗДІЛ 3. ДЕТАЛІ РОЗРОБКИ ПЛАТФОРМИ

Даний розділ присвячено опису процесу практичної розробки та реалізації онлайн-платформи для управління курсами, яка інтегрує функціональні можливості для організації менторства та забезпечує систему багаторівневого доступу для різних категорій користувачів. Метою цього розділу є послідовне та структуроване висвітлення ключових етапів створення програмного продукту, починаючи від архітектурного проєктування і завершуючи підготовкою застосунку до розгортання.

У ході викладення матеріалу буде розкрито підхід до архітектурного проєктування системи, базуючись на принципах "Чистої архітектури" для забезпечення гнучкості, тестованості та керованості кодової бази. Буде висвітлено процес розробки основних функціональних модулів, відповідальних за реалізацію ключових можливостей платформи, а також створення спеціалізованих системних бібліотек, що інкапсулюють наскрізні завдання, такі як реалізація патерну Command-Query Separation (CQS), управління контекстом запиту та механізми атрибутивної авторизації.

Окрему увагу буде приділено питанням інтеграції обраного технологічного стеку, налаштування інфраструктурних компонентів, управління конфігураціями застосунку та підготовці його до розгортання. Важливим аспектом цього етапу є контейнеризація онлайн-платформи за допомогою Docker, що забезпечує портативність, відтворюваність та стабільність середовища виконання.

Таким чином, цей розділ надасть загальне уявлення про ключові архітектурні рішення, технологічні аспекти та етапи реалізації онлайн-платформи, слугуючи основою для розуміння її внутрішньої структури та функціонування. Опис буде вестися на концептуальному рівні, ілюструючи загальні підходи без надмірного заглиблення в деталі коду окремих дрібних функцій, які можуть бути представлені в наступних підрозділах або додатках.

## 3.1 Реалізація архітектурних шарів застосунку

В основі розробки онлайн-платформи лежить застосування принципів "Чистої архітектури". Такий підхід спрямований на створення гнучкої, тестованої та легко підтримуваної системи шляхом чіткого розділення відповідальностей між окремими шарами. Кожен шар має свої специфічні завдання та залежить лише від внутрішніх шарів, що забезпечує низьку зв'язність компонентів та високу когезію. У цьому підрозділі буде розглянуто ключові аспекти реалізації кожного з основних шарів архітектури на прикладі розроблюваної платформи.

### 3.1.1 Проєктування доменного шару

Доменний шар є центральним елементом "Чистої архітектури", інкапсулюючи основні бізнес-правила та логіку системи, незалежно від деталей реалізації інших шарів. Фундаментом цього шару є доменні сутності (Entities) та об'єкти-значення (Value Objects), які моделюють ключові концепції предметної області.

У системі сутності були спроєктовані для представлення об'єктів, що мають унікальну ідентичність, яка зберігалася протягом їхнього життєвого циклу, навіть якщо їхні атрибути змінювалися. Вони також інкапсулювали поведінку, пов'язану з бізнес-логікою. Прикладом такої сутності слугував клас User (рис. 3.1), що представляв користувача системи. Сутності, що визначені як корінь агрегату (Aggregate Root), як-от User, успадковували клас AggregateRoot, який розширював Entity для управління доменними подіями, що виникали внаслідок зміни стану агрегату.

```typescript
export class User extends AggregateRoot<Uuid, UserProps> {
  public constructor(id: Uuid, props: UserProps) {
    super(id, props);
  }

  public get email(): Email {
    return this.props.email;
  }

  public get name(): Name {
    return this.props.name;
  }

  public get version(): number {
    return this.props.version;
  }

  public static create(input: CreateUserProps): User {
    const props = {
      email: input.email,
      name: input.name,
      version: 0,
    };

    return new User(input.id, props);
  }
}
```

Рисунок 3.1 - Код сутності User

Об'єкти-значення використовувалися для опису атрибутів сутностей або інших аспектів домену, де важливим було саме значення, а не ідентичність. Вони були реалізовані як незмінні (immutable) після створення та не мали власного ідентифікатора в контексті порівняння – два об'єкти-значення вважалися рівними, якщо їхні атрибути були однакові. У модулі користувачів прикладом об'єкта-значення став клас Email (рис. 3.2).

```typescript
export class Email extends ValueObject<string> {
  private constructor(value: string) {
    super(value);
  }

  public static create(value: string): Email {
    const EMAIL_REGEX = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;

    if (!EMAIL_REGEX.test(value)) {
      throw new Error(`Invalid email format. Received: "${value}"`);
    }

    return new Email(value);
  }
}
```

Рисунок 3.2 - Клас Email

Подібні класи інкапсулювали логіку валідації відповідних даних (наприклад, перевірку формату електронної пошти або довжини імені) безпосередньо при їх створенні. Усі об'єкти-значення в проєкті успадковували базовий клас ValueObject з директорії core/domain (рис. 3.3), який забезпечив загальну структуру та механізм порівняння.

```
export abstract class ValueObject<T> {
  readonly #value: T;

  protected constructor(value: T) {
    this.#value = value;
    Object.freeze(this);
  }

  public get value(): T {
    return this.#value;
  }

  public equals(other: ValueObject<T>): boolean {
    return isEqual(this.#value, other.value);
  }
}
```

Рисунок 3.3 - Базовий клас ValueObject

Таким чином, реалізоване розділення на сутності та об'єкти-значення дозволило чітко структурувати доменну модель, інкапсулювати бізнес-правила та валідацію даних безпосередньо в об'єктах, до яких вони належать, що сприяло підвищенню надійності та зрозумілості коду.

### 3.1.2 Розробка шару застосунку

Шар застосунку (Application Layer) виступає в ролі координатора, що реалізує сценарії використання системи (use cases) та керує потоками даних між доменним шаром та зовнішніми інтерфейсами. Важливою характеристикою цього шару є те, що він не містить власної складної бізнес-логіки, а делегує її виконання доменним об'єктам, таким як сутності та сервіси домену. Це

забезпечує чітке дотримання принципу єдиної відповідальності та сприяє низькій зв'язності системи. У даному проєкті для структурування логіки шару застосунку було обрано підхід на основі патерну Command Query Separation (CQS). Такий підхід передбачає чітке розділення операцій, що змінюють стан системи (команди), від операцій, призначених для отримання даних (запити) [6]. Команди, як-от CreateUserCommand інкапсулюють дані, необхідні для виконання певної мутаційної дії. Запити, наприклад FindMeQuery, визначають критерії для отримання інформації без зміни стану системи.

Для кожної команди та кожного запиту було створено відповідний обробник (handler). Наприклад, CreateUserHandler (рис. 3.4) відповідає за логіку створення нового користувача, тоді як FindMeHandler (рис. 3.5) реалізує отримання даних поточного автентифікованого користувача. Ці обробники містять логіку, специфічну для конкретного сценарію використання, та взаємодіють з доменним шаром через абстрактні інтерфейси, відомі як порти (Ports).

```
@CommandHandler(CreateUserCommand)
export class CreateUserHandler implements ICommandHandler<CreateUserCommand> {
  public constructor(
    @Inject(USERS_REPOSITORY) private readonly repository: IUsersRepository,
  ) {}

  @Transactional()
  public async execute(command: CreateUserCommand): Promise<void> {
    const id = Uuid.create(command.id);
    const name = Name.create(command.name);
    const email = Email.create(command.email);
    const userExists = await this.repository.findByEmail(email);

    if (userExists) return;

    const user = User.create({ email, id, name });
    await this.repository.save(user);
  }
}
```

Рисунок 3.4 - Обробник CreateUserHandler

```typescript
@QueryHandler(FindMeQuery)
export class FindMeQueryHandler implements IQueryHandler<FindMeQuery> {
  public constructor(
    @Inject(CLS_SERVICE) private readonly clsService: IClsService,
    @Inject(USER_DAO) private readonly userDao: IUserDao,
  ) {}

  public async execute(_query: FindMeQuery): Promise<UserModel> {
    const context = this.clsService.get<AuthContext>(AUTH_CONTEXT_KEY);

    if (!context) throw new UnauthorizedException();

    return await this.userDao.findOneByIdOrFail(context.identity.id);
  }
}
```

Рисунок 3.5 - Обробник FindMeHandler

Порти визначають контракти, які повинні бути реалізовані інфраструктурним шаром для забезпечення необхідної функціональності, наприклад, для доступу до даних. Прикладами таких портів у модулі користувачів є IUsersRepository, що описує операції для роботи з агрегатом користувача, та IUserDao, що надає інтерфейс для отримання даних користувачів у вигляді простих структур даних. Такий підхід дозволяє шару застосунку залишатися незалежним від конкретних технологій зберігання даних, що підвищує його тестованість та гнучкість. Диспетчеризація команд та запитів до відповідних обробників здійснюється за допомогою спеціалізованих сервісів, реалізованих у бібліотеці CQS.

### 3.1.3 Побудова інфраструктурного шару

Інфраструктурний шар був реалізований для забезпечення взаємодії системи із зовнішнім світом та надання конкретних реалізацій для абстракцій, визначених у шарі застосунку та ядрі системи. Цей шар містить технологічно-специфічні компоненти, такі як механізми роботи з базою даних, інтеграції з іншими сервісами та інструменти для логування або обробки конфігурацій. Ключовим аспектом інфраструктурного шару стала організація персистентності даних. У проєкті було обрано систему управління базами

даних PostgreSQL, а для взаємодії з нею використано об'єктно-реляційний мапер (ORM) MikroORM. Це дозволило абстрагуватися від деталей SQL-запитів та працювати з даними на рівні об'єктів.

Інтеграція із зовнішніми сервісами автентифікації, зокрема з AWS Cognito, також була частиною цього шару. Механізм перевірки JWT, отриманих від Cognito, було реалізовано з використанням бібліотеки aws-jwt-verify, що гарантувало безпечну автентифікацію користувачів. Відповідні налаштування для підключення до сервісів та бази даних були винесені в конфігураційні файли.

Таким чином, інфраструктурний шар забезпечив необхідну технічну основу для функціонування бізнес-логіки, ізолюючи її від деталей конкретних технологій.

### 3.1.4 Формування шару представлення (інтерфейсів)

Шар представлення (Interfaces Layer або Presenters) виконував роль основної точки входу для взаємодії зовнішніх клієнтів із системою. Його ключовим завданням була адаптація запитів, що надходили від користувачів або інших програмних систем, до формату, зрозумілого шару застосунку, а також перетворення результатів виконання операцій у відповідь, зручну для споживача. Це забезпечувало належний рівень абстракції та ізоляції внутрішньої логіки системи від специфіки зовнішніх інтерфейсів. У розробленій онлайн-платформі цей шар був представлений переважно HTTP контролерами, які були розроблені з використанням фреймворку NestJS. Прикладом такого компонента може слугувати UsersController (рис. 3.6), який обробляв HTTP-запити, пов'язані з функціоналом користувачів, наприклад, отримання інформації про поточного автентифікованого користувача.

```
@ApiTags('User')
@ApiBearerAuth()
@Controller({ path: '/users', version: '1' })
export class UsersController {
  public constructor(
    @Inject(QUERY_DISPATCHER)
    private readonly queryDispatcher: IQueryDispatcher,
  ) {}

  @Get('me')
  @HttpCode(HttpStatus.OK)
  @ApiOperation({ summary: 'Create user account.' })
  @ApiUnauthorizedResponse()
  @ApiOkResponse({ type: UserResponseDto })
  @ApiInternalServerErrorResponse()
  public async findMe(): Promise<UserResponseDto> {
    return this.queryDispatcher.execute(new FindMeQuery());
  }
}
```

Рисунок 3.6 - Компонент UsersController

Для структурування, валідації та передачі даних між клієнтом і сервером, а також між шаром представлення та шаром застосунку, активно використовувалися об'єкти передачі даних (DTO – Data Transfer Objects). Наприклад, UserCreatedEventDto (рис. 3.7) використовувався для валідації даних, що надходили при створенні користувача через веб-хук, забезпечуючи їх відповідність очікуваному формату та наявність необхідних полів. Для повернення даних клієнту, наприклад, інформації про профіль користувача, був розроблений UserResponseDto.

```
export class UserCreatedEventDto {
  @IsUUID()
  public readonly id: string;

  @IsNotEmpty()
  @MaxLength(100)
  @MinLength(2)
  @IsString()
  public readonly name: string;

  @IsEmail()
  public readonly email: string;
}
```

Рисунок 3.7 - Код UserCreatedEventDto

Таким чином, шар представлення відповідав за ефективну та безпечну комунікацію системи із зовнішнім світом.

## 3.2 Розробка інфраструктурних компонентів

Успішна експлуатація будь-якого програмного продукту значною мірою залежить від належної підготовки його інфраструктури та забезпечення стабільного й відтворюваного середовища для розгортання. Цей підрозділ присвячено розгляду ключових інфраструктурних рішень, прийнятих у проєкті онлайн-платформи. Буде коротко висвітлено деякі загальні механізми, що підтримують функціонування та взаємодію компонентів системи, після чого основну увагу буде приділено завершальному та критично важливому етапу підготовки застосунку до розгортання – його контейнеризації.

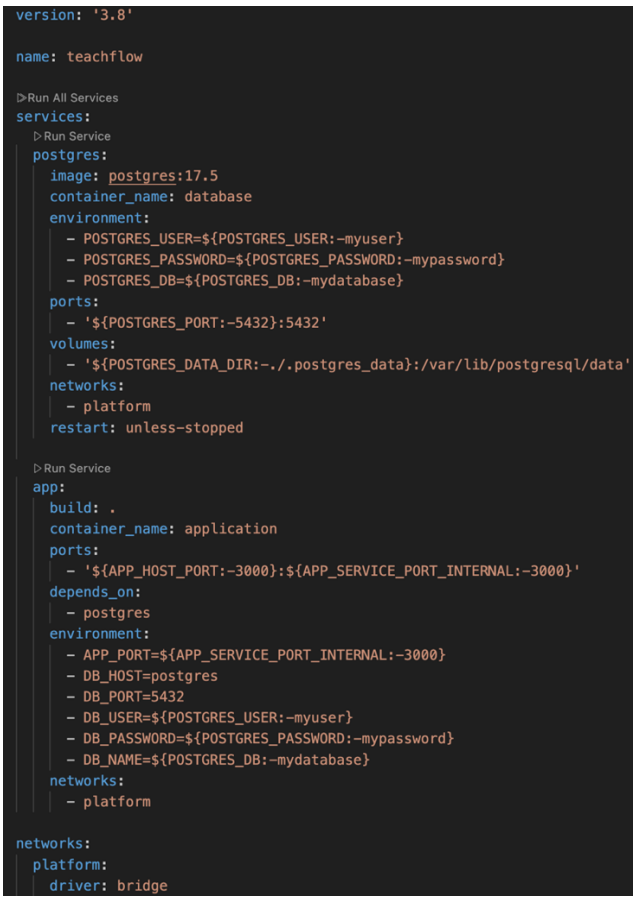
### 3.2.1 Компоненти конфігурації

У рамках підготовки інфраструктури онлайн-платформи було приділено значну увагу розробці фундаментальних компонентів, відповідальних за управління конфігураціями системи та реалізацію загальних механізмів, що

забезпечують її узгоджене та безпечне функціонування. Для управління налаштуваннями застосунку було впроваджено централізований підхід. Це дозволило структурувати конфігураційні параметри, відокремивши їх від основної логіки програми, та забезпечити можливість їх гнучкого завантаження та використання в різних частинах системи. Особливу увагу було приділено безпеці чутливих даних, таких як параметри доступу до бази даних чи зовнішніх сервісів, для чого було передбачено механізми їх завантаження зі змінних оточення, що є стандартною практикою для сучасних застосунків.

### 3.2.2 Контейнеризація застосунку для забезпечення портативності та розгортання

Для забезпечення портативності, консистентності середовища виконання та спрощення процесу розгортання онлайн-платформи було прийнято рішення про її контейнеризацію з використанням технології Docker. Цей підхід дозволяє інкапсулювати застосунок та всі його залежності в ізольовані контейнери, які можуть бути легко перенесені та запущені на будь-якій системі, що підтримує Docker, мінімізуючи проблеми, пов'язані з відмінностями в конфігурації середовищ. Ключовим елементом процесу контейнеризації стала розробка файлу docker-compose.yml (рис. 3.8).



```yaml
version: '3.8'

name: teachflow

▷Run All Services
services:
  ▷Run Service
  postgres:
    image: postgres:17.5
    container_name: database
    environment:
      - POSTGRES_USER=${POSTGRES_USER:-myuser}
      - POSTGRES_PASSWORD=${POSTGRES_PASSWORD:-mypassword}
      - POSTGRES_DB=${POSTGRES_DB:-mydatabase}
    ports:
      - '${POSTGRES_PORT:-5432}:5432'
    volumes:
      - '${POSTGRES_DATA_DIR:-./.postgres_data}:/var/lib/postgresql/data'
    networks:
      - platform
    restart: unless-stopped

  ▷Run Service
  app:
    build: .
    container_name: application
    ports:
      - '${APP_HOST_PORT:-3000}:${APP_SERVICE_PORT_INTERNAL:-3000}'
    depends_on:
      - postgres
    environment:
      - APP_PORT=${APP_SERVICE_PORT_INTERNAL:-3000}
      - DB_HOST=postgres
      - DB_PORT=5432
      - DB_USER=${POSTGRES_USER:-myuser}
      - DB_PASSWORD=${POSTGRES_PASSWORD:-mypassword}
      - DB_NAME=${POSTGRES_DB:-mydatabase}
    networks:
      - platform

networks:
  platform:
    driver: bridge
```

Рисунок 3.8 - Файл docker-compose.yml

Цей файл визначає мультиконтейнерний застосунок, що складається з декількох сервісів: основного серверного застосунку (app) та бази даних PostgreSQL (postgres).

У конфігурації сервісу postgres було вказано використання офіційного образу postgres:17.5. Були налаштовані змінні оточення для визначення користувача, пароля та назви бази даних. Для забезпечення персистентності даних PostgreSQL було налаштовано том (volume), який зберігає дані бази поза контейнером на хост-машині. Обидва сервіси були об'єднані в одну віртуальну мережу platform для забезпечення їх взаємодії.

Сервіс app, що представляє основний застосунок платформи на NestJS, було налаштовано таким чином, щоб він збирався на основі Dockerfile, розташованого в корені проєкту. Важливим аспектом стало налаштування залежності сервісу app від сервісу postgres, що гарантує запуск контейнера з

базою даних перед запуском основного застосунку. Змінні оточення для підключення до бази даних (хост, порт, користувач, пароль, назва БД) передавалися в контейнер app, причому в якості хоста бази даних вказувалася назва сервісу postgres, що дозволяло застосунку звертатися до бази даних через внутрішню мережу Docker.

Використання Docker Compose значно спростило процес локальної розробки та тестування, дозволивши швидко розгортати все необхідне оточення однією командою. Це також створило міцну основу для подальшого розгортання платформи в продуктивному середовищі, забезпечуючи відтворюваність та ізоляцію її компонентів.

# ВИСНОВОК ДО РОЗДІЛУ 3

У розділі було детально розглянуто процес практичної розробки та реалізації програмного забезпечення онлайн-платформи для управління курсами з елементами менторства та багаторівневого доступу. Було послідовно висвітлено ключові етапи створення системи, починаючи від закладання архітектурних основ і завершуючи підготовкою застосунку до розгортання.

У ході виконання робіт, описаних у цьому розділі, було створено функціональний програмний прототип онлайн-платформи, який реалізує основні заплановані можливості та архітектурні рішення. Розроблена система є підготовленою для переходу до наступних етапів життєвого циклу, таких як комплексне тестування, розгортання в продуктивному середовищі та подальша оцінка її ефективності.

# РОЗДІЛ 4. ОГЛЯД РОЗРОБЛЕНОЇ ПЛАТФОРМИ

Попередні розділи даної роботи були присвячені теоретичному обґрунтуванню обраних підходів, аналізу предметної області, а також детальному опису процесу проєктування та практичної реалізації онлайн-платформи для управління курсами з елементами менторства та багаторівневого доступу. Були розглянуті ключові архітектурні рішення, технологічний стек та етапи створення основних функціональних компонентів системи. Метою даного розділу є демонстрація та перевірка працездатності розробленої онлайн-платформи. У цьому розділі буде представлено огляд реалізованого функціоналу через призму взаємодії користувача із системою. Будуть продемонстровані основні сценарії використання. Огляд буде супроводжуватися ілюстративними матеріалами, такими як знімки екрана інтерфейсів платформи, що демонструють виконання конкретних операцій та взаємодію з різними компонентами системи.

## 4.1 Реєстрація користувачів

Для створення нового облікового запису користувач повинен надіслати POST-запит на ендпоінт /v1/auth/register. Запит має містити тіло у форматі JSON з необхідними даними для реєстрації, такими як адреса електронної пошти та пароль. На рисунку 4.1 продемонстровано приклад такого запиту, виконаного за допомогою утиліти curl.

```
curl --location 'http://localhost:3000/v1/auth/register' \
--header 'Content-Type: application/json' \
--data-raw '{
    "email": "vladyslav.shevyrov@gmail.com",
    "password": "VeryStrong_pass1"
}'
```

Рисунок 4.1 - Приклад запиту на реєстрацію нового користувача

У разі успішної реєстрації система, як правило, повертає HTTP-відповідь зі статусом 201 Created або 200 OK та може опціонально повертати дані створеного користувача або токени доступу. Якщо користувач з такою електронною поштою вже існує або надані дані не відповідають вимогам валідації, система поверне відповідний код помилки (наприклад, 400 Bad Request або 409 Conflict).

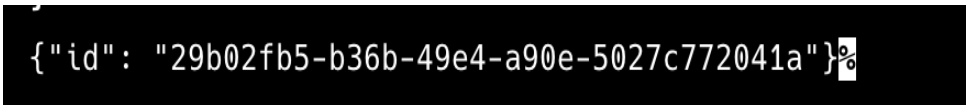



Рисунок 4.2 - Приклад відповіді серверу у разі успішної реєстрації

## 4.2 Автентифікація користувача

```
curl --location 'http://localhost:3000/v1/auth/login' \
--header 'Content-Type: application/json' \
--data-raw '{
    "email": "vladyslav.shevyrov@gmail.com",
    "password": "VeryStrong_pass1"
}'
```

Рисунок 4.3 - Приклад запиту автентифікації існуючого користувача

У разі успішної автентифікації сервер повертає відповідь, що зазвичай містить JWT (JSON Web Tokens) – токен доступу (access token).

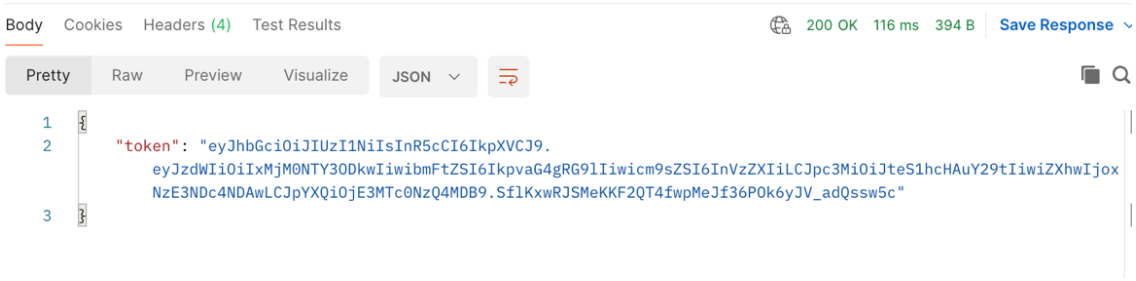



Рисунок 4.4 - Приклад успішної віпдовіді на запит автентифікації користувача

## 4.3 Перевірка обмежень для різних ролей користувачів

Одним із ключових сценаріїв є спроба доступу до вмісту курсу користувачем, який не має на це відповідних прав. Наприклад, користувач з роллю "студент" намагається переглянути список уроків курсу, до якого він не зареєстрований або який є приватним.

```
curl --location 'http://localhost:3000/v1/courses/
    0358a3a5-6f54-468b-89f9-ea2f56af1bef/lessons' \
--header 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
    eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwicm9sZSI6InVzZXIiLCJ
    pc3MiOiJteS1hcHAuY29tIiwiZXhwIjoxNzE3NDc4NDAwLCJpYXQiOjE3MTc0NzQ4MDB9.
    SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c'
```

Рисунок 4.5 - Приклад запиту студента до уроків закритого курсу

Оскільки користувач не має необхідних прав (наприклад, не є зарахованим на курс, або курс не є публічним), система повинна відмовити в доступі. На рисунку 4.6 продемонстровано відповідь сервера. Як і очікувалося, система повертає HTTP-статус 403 Forbidden, що підтверджує коректну роботу механізму розмежування прав.

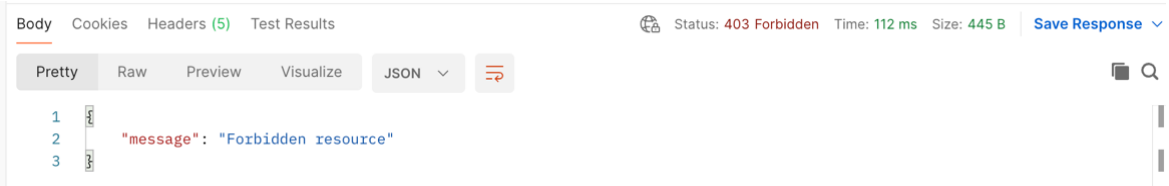



Рисунок 4.6 - Відповідь системи на запит студента до уроків закритого курсу

## 4.4 Диференційований доступ до ресурсів платформи

Платформа передбачає наявність курсів та уроків на різних стадіях готовності, зокрема, у стані чернетки (isPublished: false) та опублікованих (isPublished: true). Доступ до цих ресурсів також має бути диференційованим залежно від ролі користувача.

Користувач з роллю "адміністратор" або "паблішер" має бачити повний список усіх курсів, включаючи чернетки, для можливості їх редагування та публікації.

```
curl --location 'http://localhost:3000/v1/courses' \
--header 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
    eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwicm9sZSI6InVzZXIiLCJ
    pc3MiOiJteS1hcHAuY29tIiwiZXhwIjoxNzE3NDc4NDAwLCJpYXQiOjE3MTc0NzQ4MDB9.
    SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c'
```

Рисунок 4.7 - Приклад запиту адміністратора на /v1/courses

На рисунку 4.7 показано такий запит. У відповідь (рис. 4.8) система повертає повний перелік курсів, незалежно від їхнього статусу публікації, що відповідає очікуваній поведінці для привілейованих ролей.

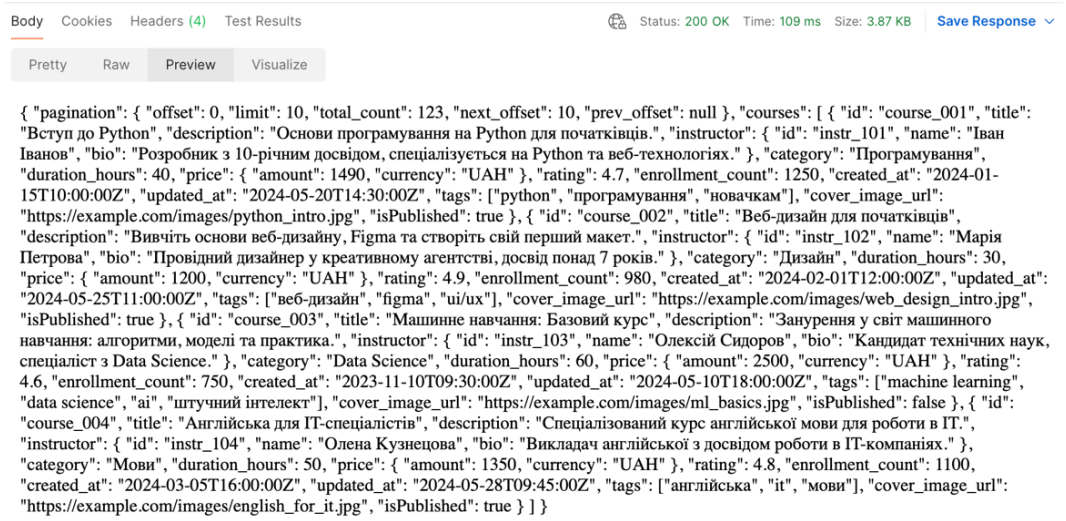



Рисунок 4.8 - відповідь системи адміністратору зі списком всіх курсів

Користувач з роллю "студент" при запиті списку курсів повинен отримувати лише ті курси, які є опублікованими.

```
curl --location 'http://localhost:3000/v1/courses' \
--header 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
    eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwicm9sZSI6InVzZXIiLCJ
    pc3MiOiJteS1hcHAuY29tIiwiZXhwIjoxNzE3NDc4NDAwLCJpYXQiOjE3MTc0NzQ4MDB9.
    SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c'
```

Рисунок 4.9 - запит студента на отримання списку курсів

У відповіді системи (рис. 4.10) присутні лише ті курси, що мають статус isPublished: true. Чернетки курсів для студента приховані.

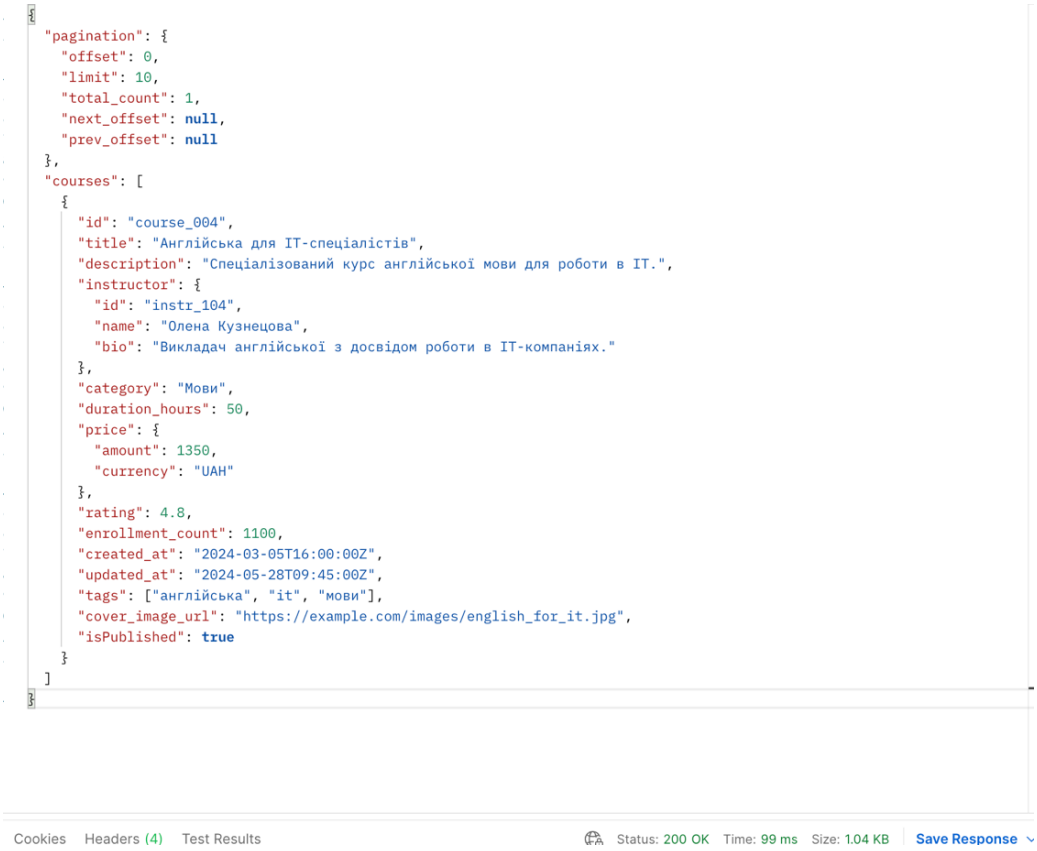



Рисунок 4.10 - відповідь системи студенту зі списком лише доступних та опублікованих курсів

Щодо уроків, студент повинен бачити лише уроки тих курсів, до яких він має доступ (на які він зарахований або які є публічними та доступними для всіх). Навіть якщо курс опублікований, але студент не має до нього доступу, уроки цього курсу не повинні бути йому доступні. Якщо ж студент має доступ до курсу, він бачить усі опубліковані уроки цього курсу. Адміністратор або паблішер, в свою чергу, при перегляді конкретного курсу (навіть чернетки) бачить усі його уроки, незалежно від їхнього статусу публікації, для можливості управління ними.

Ці сценарії демонструють, що система авторизації коректно обробляє різні рівні доступу та статуси публікації контенту, забезпечуючи як

адміністративні потреби управління платформою, так і захист контенту від неавторизованого доступу з боку звичайних користувачів.

# ВИСНОВОК ДО РОЗДІЛУ 4

У даному розділі було проведено огляд та демонстрацію функціональних можливостей розробленої онлайн-платформи для управління курсами з елементами менторства та багаторівневого доступу. Було перевірено працездатність ключових сценаріїв взаємодії користувачів із системою, що дозволило оцінити відповідність реалізованого продукту поставленим на етапі проєктування вимогам. Зокрема, було продемонстровано процес реєстрації нових користувачів та їх подальшої автентифікації в системі. На прикладі відповідного запиту та очікуваних відповідей сервера було підтверджено коректну роботу механізмів створення облікових записів. Також було розглянуто процес авторизації, який передбачає отримання JWT для доступу до захищених ресурсів платформи. Важливою частиною огляду стала перевірка реалізації механізму авторизації та розмежування прав доступу. Проведений огляд та демонстрація працездатності ключових функцій розробленої онлайн-платформи дозволяють зробити висновок, що система функціонує відповідно до основних вимог, закладених у проєкті. Реалізовані механізми реєстрації, автентифікації, авторизації та розмежування доступу забезпечують базовий рівень безпеки та контролю, необхідний для подальшого тестування та потенційної експлуатації платформи.

# ВИСНОВКИ

У ході виконання даного дипломного проєкту було розроблено серверну частину онлайн-платформи для управління курсами, що включає функціонал менторської підтримки та систему багаторівневого розмежування прав доступу. Ця розробка була спрямована на вирішення актуальних проблем сучасних освітніх онлайн-середовищ, зокрема, потреби в більш гнучких інструментах адміністрування та персоналізованій взаємодії між учасниками навчального процесу.

На першому етапі роботи було проведено комплексний аналіз предметної області, що охопив історію розвитку дистанційної освіти, роль сучасних онлайн-платформ, а також сформульовано ключові вимоги до проектованої системи. Було здійснено порівняльний аналіз існуючих рішень, таких як Coursera та Udemy, що дозволило виявити їхні сильні сторони та функціональні обмеження, особливо в контексті реалізації менторства та гнучкого управління доступом. Результати цього аналізу обґрунтували необхідність створення спеціалізованої платформи та заклали теоретичну основу для подальшої розробки.

На другому етапі було обґрунтовано вибір технологічного стеку для серверної реалізації. Було обрано мову програмування TypeScript та середовище виконання Node.js як основу, фреймворк NestJS для структурування застосунку, MikroORM для взаємодії з базою даних PostgreSQL, Redis для кешування, AWS Cognito для управління ідентифікацією та бібліотеку aws-jwt-verify для валідації токенів, CASL для реалізації авторизації, а також Docker для контейнеризації. Кожен технологічний вибір був аргументований з точки зору його переваг для створення надійної, масштабованої та легко підтримуваної системи.

Третій розділ був присвячений деталям практичної реалізації платформи. Було продемонстровано застосування принципів "Чистої архітектури" через

послідовну розробку доменного, прикладного, інфраструктурного шарів та шару представлення, з акцентом на ролі ключових інтерфейсів ядра (core). Описано створення системних бібліотек (libs) для реалізації патерну CQS, управління контекстом запиту (CLS) та авторизації (CASL). Також було висвітлено налаштування інфраструктурних компонентів та процес контейнеризації застосунку за допомогою Docker та Docker Compose.

У четвертому розділі було проведено огляд розробленої платформи та перевірку її працездатності. Було продемонстровано успішне виконання ключових сценаріїв, таких як реєстрація та автентифікація користувачів, а також коректна робота механізмів авторизації та розмежування прав доступу для різних ролей. Це підтвердило, що система функціонує відповідно до закладених вимог.

Результатом виконання дипломного проєкту є функціональна серверна частина онлайн-платформи, що реалізує управління навчальним контентом, користувачами та їх ролями, а також забезпечує елементи менторства та багаторівневий доступ. Розроблена система готова для подальшого тестування, розгортання та потенційної експлуатації в освітніх установах чи корпоративному секторі. Проєкт демонструє застосування сучасних підходів до проєктування та розробки програмного забезпечення, що дозволило створити гнучке та масштабоване рішення.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Учасники проектів Вікімедіа. Дистанційне навчання – Вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Дистанційне_навчання.
2. Athreya aka Maneshwar. Express.js vs nest.js. DEV Community. URL: https://dev.to/lovestaco/nodejs-vs-nestjs-a-tale-of-two-frameworks-lp2.
3. AWS cognito. URL: https://docs.aws.amazon.com/cognito/.
4. Awslabs/aws-jwt-verify: JS library for verifying JWTs signed by Amazon Cognito, and any OIDC-compatible IDP that signs JWTs with RS256, RS384, RS512, ES256, ES384, ES512, Ed25519 and Ed448. GitHub. URL: https://github.com/awslabs/aws-jwt-verify.
5. CASL. Isomorphic Authorization JavaScript library. CASL.js. URL: https://casl.js.org/v6/en/guide/intro.
6. Command query separation. martinfowler.com. URL: https://martinfowler.com/bliki/CommandQuerySeparation.html.
7. Docker hub. Docker Documentation. URL: https://docs.docker.com/docker-hub/.
8. Documentation | NestJS - A progressive Node.js framework. Documentation | NestJS - A progressive Node.js framework. URL: https://docs.nestjs.com/.
9. Fast and low overhead web framework, for node.js | fastify. Fast and low overhead web framework, for Node.js | Fastify. URL: https://fastify.dev/.
10. Handbook - The TypeScript Handbook. TypeScript: JavaScript With Syntax For Types. URL: https://www.typescriptlang.org/docs/handbook/intro.html.
11. JavaScript | MDN. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript.
12. JavaScript with syntax for types. TypeScript: JavaScript With Syntax For Types. URL: https://www.typescriptlang.org/.
13. JSON web token introduction - jwt.io. JSON Web Tokens - jwt.io. URL: https://jwt.io/introduction.

14. MikroORM: TypeScript ORM for Node.js based on Data Mapper, Unit of Work and Identity Map patterns. | MikroORM. MikroORM: TypeScript ORM for Node.js based on Data Mapper, Unit of Work and Identity Map patterns. | MikroORM. URL: https://mikro-orm.io/.
15. Node.js – about node.js®. Node.js – Run JavaScript Everywhere. URL: https://nodejs.org/en/about.
16. Node.js – the node.js event loop. Node.js – Run JavaScript Everywhere. URL: https://nodejs.org/en/learn/asynchronous-work/event-loop-timers-and-nexttick.
17. PostgreSQL: documentation. PostgreSQL: The world's most advanced open source database. URL: https://www.postgresql.org/docs/.
18. Redis docs. Docs. URL: https://redis.io/documentation.
19. TypeORM - Amazing ORM for TypeScript and JavaScript (ES7, ES6, ES5). Supports MySQL, PostgreSQL, MariaDB, SQLite, MS SQL Server, Oracle, WebSQL databases. Works in NodeJS, Browser, Ionic, Cordova and Electron platforms. URL: https://typeorm.io/.
20. What is prisma ORM? (overview) | prisma documentation. Prisma | Instant Postgres plus an ORM for simpler db workflows. URL: https://www.prisma.io/docs/orm/overview/introduction/what-is-prisma.

# ДОДАТОК 1

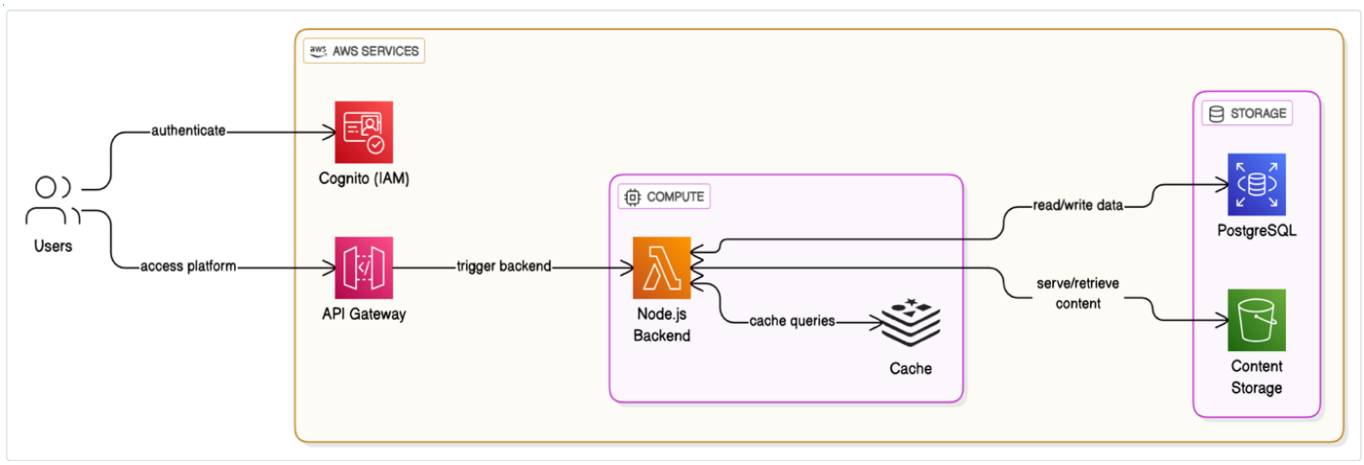
Онлайн-платформа для управління курсами з елементами менторства та багаторівневого доступу

## Архітектурна діаграма застосунку (структурна схема)

ІАЛЦ.467200.004 Д1

Аркушів 1

**Київ 2025 р**




| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | ІАЛЦ.467200.004 Д1 | | |
| | | № докум. | Підпис | Дата | | | |
| Розробив | | Шевирьов В. О. | | | **Онлайн-платформа для управління курсами з елементами менторства та багаторівневого доступу** | Літ. | Аркуш 1 / Аркушів 1 |
| Перевірив | | Шульга М. В. | | | | | |
| Реценз. | | Матвійчук О. В. | | | | НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІМ-11 | |
| Н. Контр. | | Гончаренко О. О. | | | Архітектурна діаграма застосунку | | |
| Затвердив | | | | | | | |

# ДОДАТОК 2

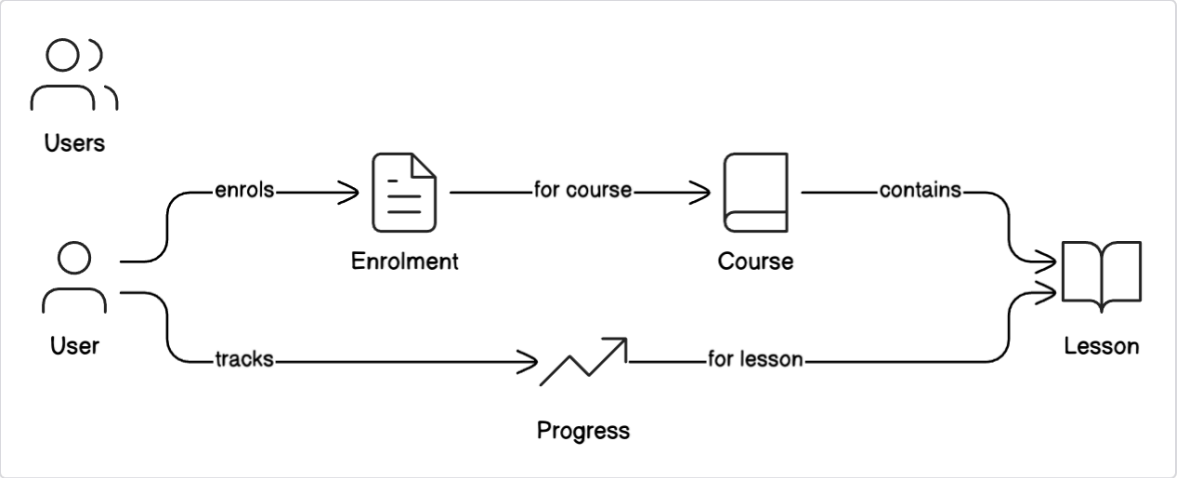
Еволюційні алгоритми глобальної пошукової оптимізації

## Діаграма доменних класів (діаграма класів)

ІАЛЦ.467200.005 Д2

Аркушів 1

**Київ 2025 р**




| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | ІАЛЦ.467200.005 Д2 | | |
| | | № докум. | Підпис | Дата | | | |
| Розробив | | Шевирьов В. О. | | | **Онлайн-платформа для управління курсами з елементами менторства та багаторівневого доступу** | Літ. | Аркуш 1 / Аркушів 1 |
| Перевірив | | Шульга М. В. | | | | | |
| Реценз. | | Матвійчук О. В. | | | Діаграма доменних класів | НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІМ-11 | |
| Н. Контр. | | Гончаренко О. О. | | | | | |
| Затвердив | | | | | | | |

# ДОДАТОК 3

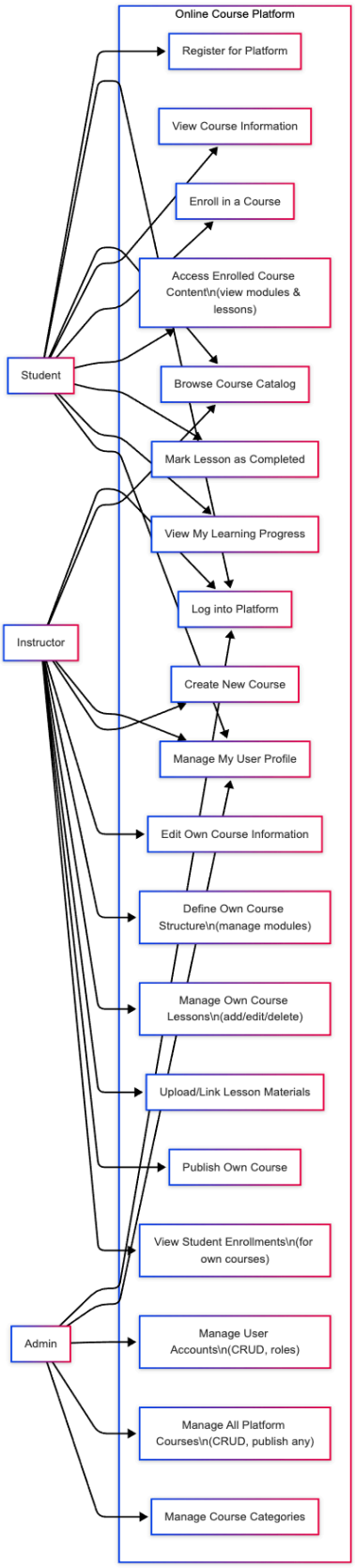
Онлайн-платформа для управління курсами з елементами менторства та багаторівневого доступу

## Алгоритм дії веб-застосунку (функціональна схема)

ІАЛЦ.467200.006 Д3

Аркушів 1

**Київ 2025 р**




| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | ІАЛЦ.467200.006 Д3 | | |
| | | № докум. | Підпис | Дата | | | |
| Розробив | | Шевирьов В. О | | | **Онлайн-платформа для управління курсами з елементами менторства та багаторівневого доступу** Алгоритм дії веб-застосунку | Літ. | Аркуш 1 / Аркушів 1 |
| Перевірив | | Шульгаа М. В. | | | | | |
| Реценз. | | Матвійчук О. В. | | | | НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІМ-11 | |
| Н. Контр. | | Гончаренко О. О. | | | | | |
| Затвердив | | | | | | | |

# ДОДАТОК 4

Онлайн-платформа для управління курсами з елементами менторства та багаторівневого доступу

Текст програмного коду

ІАЛЦ.467200.007 Д4

Аркушів 10

**Київ 2025 р**

```
import type { Event } from './event';
import { Entity } from './entity';

export abstract class AggregateRoot<T, Y> extends Entity<T, Y> {
#events: Event[] = [];

public pull(): ReadonlyArray<Event> {
const events = this.#events;
this.#events = [];
return events;
}

protected record(event: Event): void {
this.#events.push(event);
}
}

import { RESULT_KEY } from '../consts/result-key';

export abstract class Command<T = unknown> {
public readonly [RESULT_KEY]: T;
}

export abstract class Entity<Id, Props> {
readonly #id: Id;
readonly #props: Props;

public get id(): Id {
return this.#id;
}

public get props(): Props {
return this.#props
this.#props = props;
}
}
```

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | ІАЛЦ.467200.007 Д4 | | | |
| | | № докум. | Підпис | Дата | | | | |
| Розробив | | Шевирьов В. О | | | **Онлайн-платформа для управління курсами з елементами менторства та багаторівневого доступу** Алгоритм дії веб-застосунку | Літ. | Аркуш | Аркушів |
| Перевірив | | Шульгаа М. В. | | | | | 1 | 10 |
| Реценз. | | Матвійчук О. В. | | | | НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІМ-11 | | |
| Н. Контр. | | Гончаренко О. О. | | | | | | |
| Затвердив | | | | | | | | |

```
export abstract class Event {}

import { RESULT_KEY } from '../consts/result-key';
export abstract class Query<T = unknown> {
  public readonly [RESULT_KEY]: T;
}

import * as isEqual from 'fast-deep-equal/es6';
export abstract class ValueObject<T> {
  readonly #value: T;

  public constructor(value: T) {
    this.validate(value);
    this.#value = value;
  }

  public get value(): T {
    return this.#value;
  }

  protected abstract validate(value: T): void;

  public equals(other: ValueObject<T>): boolean {
    return isEqual(this.#value, other.value);
  }
}

import 'reflect-metadata';
import type { Command } from '../classes/command';
import {
  COMMAND_HANDLER_METADATA_KEY,
  COMMAND_METADATA_KEY,
} from '../consts/metadata-keys';
import { randomUUID } from 'node:crypto';

/**
 * Decorator that marks a class as a command handler. A command handler
 * handles commands (actions) executed by your application code.
 *
 * The decorated class must implement the `ICommandHandler` interface.
 *
 * @param command command *type* to be handled by this handler.
 * @param options injectable options passed on to the "@Injectable" decorator.
 *
 *
 * @publicApi
 */
export const CommandHandler = (
  command: Command | (new (...args: unknown[]) => Command),
): ClassDecorator => {
  return (target: Function) => {
    if (!Reflect.hasOwnMetadata(COMMAND_METADATA_KEY, command)) {
      const metadataValue = { id: randomUUID() };
      Reflect.defineMetadata(COMMAND_METADATA_KEY, metadataValue, command);
    }

    Reflect.defineMetadata(COMMAND_HANDLER_METADATA_KEY, command, target);
  };
};

import 'reflect-metadata';
import { randomUUID } from 'crypto';
import type { Event } from '../classes/event';
```

```
import {
  EVENT_METADATA_KEY,
  EVENTS_HANDLER_METADATA_KEY,
} from '../consts/metadata-keys';

/**
 * Decorator that marks a class as a event handler. An event handler
 * handles events executed by your application code.
 *
 * The decorated class must implement the `IEventsHandler` interface.
 *
 * @param events one or more event *types* to be handled by this handler.
 * @param options injectable options passed on to the "@Injectable" decorator.
 *
 *
 * @publicApi
 */
export const EventsHandler = (
  ...events: (Event | (new (...args: unknown[]) => Event))[]
): ClassDecorator => {
  return (target: Function) => {
    events.forEach((event) => {
      if (!Reflect.hasOwnMetadata(EVENT_METADATA_KEY, event)) {
        const metadataValue = { id: randomUUID() };
        Reflect.defineMetadata(EVENT_METADATA_KEY, metadataValue, event);
      }
    });

    Reflect.defineMetadata(EVENTS_HANDLER_METADATA_KEY, events, target);
  };
};

import 'reflect-metadata';
import { Query } from '../classes/query';
import {
  QUERY_HANDLER_METADATA_KEY,
  QUERY_METADATA_KEY,
} from '../consts/metadata-keys';
import { randomUUID } from 'node:crypto';

/**
 * Decorator that marks a class as a query handler. A query handler
 * handles queries executed by your application code.
 *
 * The decorated class must implement the `IQueryHandler` interface.
 *
 * @param query query *type* to be handled by this handler.
 * @param options injectable options passed on to the "@Injectable" decorator.
 *
 *
 * @publicApi
 */
export const QueryHandler = (query: Query): ClassDecorator => {
  return (target: Function) => {
    if (!Reflect.hasOwnMetadata(QUERY_METADATA_KEY, query)) {
      const metadataValue = { id: randomUUID() };
      Reflect.defineMetadata(QUERY_METADATA_KEY, metadataValue, query);
    }

    Reflect.defineMetadata(QUERY_HANDLER_METADATA_KEY, query, target);
  };
};
```

```
import 'reflect-metadata';
import { Injectable, Logger, LoggerService, Type } from '@nestjs/common';
import { ModuleRef } from '@nestjs/core';
import { InstanceWrapper } from '@nestjs/core/injector/instance-wrapper';
import { Command } from '../classes';
import { ICommandDispatcher, ICommandHandler } from '../interfaces';
import {
  COMMAND_HANDLER_METADATA_KEY,
  COMMAND_METADATA_KEY,
} from '../consts/metadata-keys';
import { CommandMetadata } from '../interfaces/command/command-metadata.interface';
import {
  CommandDispatcherIsFrozenException,
  CommandHandlerNotFoundException,
  InvalidCommandHandlerException,
} from '../exceptions';

@Injectable()
export class CommandDispatcher implements ICommandDispatcher {
  #isFrozen: boolean;
  readonly #logger: LoggerService;
  readonly #handlers: Map<string, <R>(command: Command<R>) => Promise<R>>;

  public constructor(private readonly moduleRef: ModuleRef) {
    this.#isFrozen = false;
    this.#handlers = new Map();
    this.#logger = new Logger(CommandDispatcher.name);
  }

  #bind(handler: InstanceWrapper<ICommandHandler>): void {
    const typeRef = handler.metatype as Type<ICommandHandler>;
    const commandId = this.#reflectCommandId(typeRef);

    if (this.#handlers.has(commandId)) {
      this.#logger.warn(
        `Command handler [${typeRef.name}] is already registered. Overriding previously registered handler.`,
      );
    }

    const options = { strict: false };
    const instance = this.moduleRef.get(typeRef, options);

    this.#handlers.set(commandId, <R>(command: Command<R>): Promise<R> => {
      return instance.execute(command) as Promise<R>;
    });
  }

  #reflectCommandId(handler: Type<ICommandHandler>): string {
    const command = Reflect.getMetadata(
      COMMAND_HANDLER_METADATA_KEY,
      handler,
    ) as Type<Command>;

    const metadata = Reflect.getMetadata(COMMAND_METADATA_KEY, command) as
      | CommandMetadata
      | undefined;

    if (!metadata) {
      throw new InvalidCommandHandlerException();
    }

    return metadata.id;
```

```
  }

  #getCommandId(command: Command): string {
    const metadata = Reflect.getMetadata(
      COMMAND_METADATA_KEY,
      command.constructor,
    ) as CommandMetadata | undefined;

    if (!metadata) {
      throw new CommandHandlerNotFoundException(command.constructor.name);
    }

    return metadata.id;
  }

  public register(
    handlers: Array<InstanceWrapper<ICommandHandler>> = [],
  ): void {
    if (this.#isFrozen) {
      throw new CommandDispatcherIsFrozenException();
    }

    handlers.forEach((handler) => this.#bind(handler));
    this.#isFrozen = true;
  }

  public execute<R>(command: Command<R>): Promise<R> {
    const commandId = this.#getCommandId(command);
    const handler = this.#handlers.get(commandId);

    if (!handler) {
      throw new CommandHandlerNotFoundException(command.constructor.name);
    }

    return handler(command);
  }
}

import { Injectable, Logger, LoggerService, Type } from '@nestjs/common';
import 'reflect-metadata';
import { IEventsDispatcher, IEventsHandler } from '../interfaces';
import { ModuleRef } from '@nestjs/core';
import { InstanceWrapper } from '@nestjs/core/injector/instance-wrapper';
import {
  EVENT_METADATA_KEY,
  EVENTS_HANDLER_METADATA_KEY,
} from '../consts/metadata-keys';
import { EventMetadata } from '../interfaces/event/event-metadata.interface';
import {
  EventsDispatcherIsFrozenException,
  EventsHandlerNotFoundException,
  InvalidEventsHandlerException,
} from '../exceptions';

@Injectable()
export class EventsDispatcher implements IEventsDispatcher {
  #isFrozen: boolean;
  readonly #logger: LoggerService;
  readonly #handlers: Map<string, (event: Event) => unknown>;

  public constructor(private readonly moduleRef: ModuleRef) {
    this.#isFrozen = false;
    this.#handlers = new Map();
```

```
    this.#logger = new Logger(EventsDispatcher.name);
  }

  #bind(handler: InstanceWrapper<IEventsHandler>): void {
    const typeRef = handler.metatype as Type<IEventsHandler>;
    const eventId = this.#reflectEventId(typeRef);
    const options = { strict: false };
    const instance = this.moduleRef.get(typeRef, options);

    this.#handlers.set(eventId, (event: Event) => {
      return instance.handle(event);
    });
  }

  #reflectEventId(handler: Type<IEventsHandler>): string {
    const event = Reflect.getMetadata(
      EVENTS_HANDLER_METADATA_KEY,
      handler,
    ) as Type<Event>;

    const metadata = Reflect.getMetadata(EVENT_METADATA_KEY, event) as
      | EventMetadata
      | undefined;

    if (!metadata) {
      throw new InvalidEventsHandlerException();
    }

    return metadata.id;
  }

  #getEventId(event: Event): string {
    const metadata = Reflect.getMetadata(
      EVENT_METADATA_KEY,
      event.constructor,
    ) as EventMetadata | undefined;

    if (!metadata) {
      throw new EventsHandlerNotFoundException(event.constructor.name);
    }

    return metadata.id;
  }

  public register(handlers: Array<InstanceWrapper<IEventsHandler>> = []): void {
    if (this.#isFrozen) {
      throw new EventsDispatcherIsFrozenException();
    }

    handlers.forEach((handler) => this.#bind(handler));
    this.#isFrozen = true;
  }

  public async executeAll(events: Array<Event>): Promise<void> {
    const executions = events.map((event) => {
      const eventId = this.#getEventId(event);
      const handler = this.#handlers.get(eventId);

      if (!handler) {
        throw new EventsHandlerNotFoundException(event.constructor.name);
      }

      return handler(event);
```

```
    });

    await Promise.all(executions);
  }
}

import 'reflect-metadata';
import { Injectable, Logger, LoggerService, Type } from '@nestjs/common';
import { ModuleRef } from '@nestjs/core';
import { InstanceWrapper } from '@nestjs/core/injector/instance-wrapper';
import { IQueryDispatcher, IQueryHandler } from '../interfaces';
import { Query } from '../classes';
import {
  QUERY_HANDLER_METADATA_KEY,
  QUERY_METADATA_KEY,
} from '../consts/metadata-keys';
import { QueryMetadata } from '../interfaces/query/query-metadata.interface';
import {
  InvalidQueryHandlerException,
  QueryDispatcherIsFrozenException,
  QueryHandlerNotFoundException,
} from '../exceptions';

@Injectable()
export class QueryDispatcher implements IQueryDispatcher {
  #isFrozen: boolean;
  readonly #logger: LoggerService;
  readonly #handlers: Map<string, <R>(query: Query<R>) => Promise<R>>;

  public constructor(private readonly moduleRef: ModuleRef) {
    this.#isFrozen = false;
    this.#handlers = new Map();
    this.#logger = new Logger(QueryDispatcher.name);
  }

  #bind(handler: InstanceWrapper<IQueryHandler>): void {
    const typeRef = handler.metatype as Type<IQueryHandler>;
    const queryId = this.#reflectQueryId(typeRef);

    if (this.#handlers.has(queryId)) {
      this.#logger.warn(
        `Query handler [${typeRef.name}] is already registered. Overriding previously registered handler.`,
      );
    }

    const options = { strict: false };
    const instance = this.moduleRef.get(typeRef, options);

    this.#handlers.set(queryId, <R>(query: Query<R>): Promise<R> => {
      return instance.execute(query) as Promise<R>;
    });
  }

  #reflectQueryId(handler: Type<IQueryHandler>): string {
    const query = Reflect.getMetadata(
      QUERY_HANDLER_METADATA_KEY,
      handler,
    ) as Type<Query>;

    const metadata = Reflect.getMetadata(QUERY_METADATA_KEY, query) as
      | QueryMetadata
      | undefined;
```

```
    if (!metadata) {
      throw new InvalidQueryHandlerException();
    }

    return metadata.id;
  }

  #getQueryId(query: Query): string {
    const metadata = Reflect.getMetadata(
      QUERY_METADATA_KEY,
      query.constructor,
    ) as QueryMetadata | undefined;

    if (!metadata) {
      throw new QueryHandlerNotFoundException(query.constructor.name);
    }

    return metadata.id;
  }

  public register(handlers: Array<InstanceWrapper<IQueryHandler>> = []): void {
    if (this.#isFrozen) {
      throw new QueryDispatcherIsFrozenException();
    }

    handlers.forEach((handler) => this.#bind(handler));
    this.#isFrozen = true;
  }

  public execute<R>(query: Query<R>): Promise<R> {
    const queryId = this.#getQueryId(query);
    const handler = this.#handlers.get(queryId);

    if (!handler) {
      throw new QueryHandlerNotFoundException(query.constructor.name);
    }

    return handler(query);
  }
}

import { Injectable } from '@nestjs/common';
import { InstanceWrapper } from '@nestjs/core/injector/instance-wrapper';
import { ModulesContainer } from '@nestjs/core/injector/modules-container';

import { ICommandHandler } from '../interfaces/command/command-handler.interface';
import {
  COMMAND_HANDLER_METADATA_KEY,
  EVENTS_HANDLER_METADATA_KEY,
  QUERY_HANDLER_METADATA_KEY,
} from '../consts/metadata-keys';
import { IQueryHandler } from '../interfaces/query/query-handler.interface';
import { IEventsHandler } from '../interfaces/event/events-handler.interface';

@Injectable()
export class ExplorerService {
  public constructor(private readonly container: ModulesContainer) {}

  public explore(): ProvidersIntrospectionResult {
    const queries = this.#getByKey<IQueryHandler>(QUERY_HANDLER_METADATA_KEY);
    const events = this.#getByKey<IEventsHandler>(EVENTS_HANDLER_METADATA_KEY);
    const commands = this.#getByKey<ICommandHandler>(
      COMMAND_HANDLER_METADATA_KEY,
```

```
    );

    return { commands, queries, events };
  }

  #getByKey<T>(metadataKey: string): InstanceWrapper<T>[] {
    return [...this.container.values()]
      .flatMap((moduleRef) => [...moduleRef.providers.values()])
      .filter((wrapper): wrapper is InstanceWrapper<T> =>
        this.#hasMetadata(wrapper, metadataKey),
      );
  }

  #hasMetadata<T>(wrapper: InstanceWrapper<T>, metadataKey: string): boolean {
    const target = wrapper.instance?.constructor;
    return !!target && Reflect.hasMetadata(metadataKey, target);
  }
}
```

```
import { UserIdentity, AuthContext } from '../interfaces';
import { AuthContextNotFoundException } from '../exceptions';
import { ClsService } from 'nestjs-cls';
import { Injectable } from '@nestjs/common';
import { USER_IDENTITY_KEY } from '../consts/cls-keys';

@Injectable()
export class UserIdentityContext implements AuthContext {
  constructor(private readonly cls: ClsService) {}

  public getOrFail(): UserIdentity {
    const identity = this.cls.get<UserIdentity>(USER_IDENTITY_KEY);

    if (!identity) {
      throw new AuthContextNotFoundException();
    }

    return identity;
  }
}
```

```
import {
  CanActivate,
  ExecutionContext,
  Injectable,
  UnauthorizedException,
} from '@nestjs/common';
import { Reflector } from '@nestjs/core';
import { JwtService } from '@nestjs/jwt';
import { Request } from 'express';
import { AuthedRequest, JwtTokenPayload } from '../interfaces';
import { IS_PUBLIC_KEY } from '../consts/metadata-keys';

@Injectable()
export class AuthGuard implements CanActivate {
  public constructor(
    private readonly jwtService: JwtService,
    private readonly reflector: Reflector,
  ) {}

  public async canActivate(context: ExecutionContext): Promise<boolean> {
    const targets = [context.getHandler(), context.getClass()];

    const isPublic = this.reflector.getAllAndOverride<boolean>(
```

```
      IS_PUBLIC_KEY,
      targets,
    );

    if (isPublic) return true;

    const request = context.switchToHttp().getRequest<Request>();
    const token = this.#extractTokenFromHeader(request);

    if (!token) throw new UnauthorizedException();

    try {
      const payload = await this.jwtService.verifyAsync<JwtTokenPayload>(token);

      (request as AuthedRequest).auth = payload;
    } catch {
      throw new UnauthorizedException();
    }
    return true;
  }

  #extractTokenFromHeader(request: Request): string | undefined {
    const [type, token] = request.headers.authorization?.split(' ') ?? [];
    return type === 'Bearer' ? token : undefined;
  }
}

import {
  CallHandler,
  ExecutionContext,
  Injectable,
  NestInterceptor,
} from '@nestjs/common';
import { Observable } from 'rxjs';
import { USER_IDENTITY_KEY } from '../consts/cls-keys';
import { createMongoAbility } from '@casl/ability';
import { unpackRules } from '@casl/ability/extra';
import { AuthedRequest } from '../interfaces';
import { ClsService } from 'nestjs-cls';

@Injectable()
export class UserIdentityInterceptor implements NestInterceptor {
  public constructor(private readonly cls: ClsService) {}

  public intercept(
    ctx: ExecutionContext,
    next: CallHandler,
  ): Observable<unknown> {
    const request = ctx.switchToHttp().getRequest<AuthedRequest>();

    if (!request.auth) return next.handle();

    const rules = unpackRules(request.auth.rules);
    const ability = createMongoAbility(rules);
    const identity = { id: request.auth.id, ability };
    this.cls.set(USER_IDENTITY_KEY, identity);
    return next.handle();
  }
}
```