НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено
Завідувач кафедри
_____ О.В. Коваль
(підпис) (ініціали, прізвище)
“___” _____2020р.

ДИПЛОМНА РОБОТА

на здобуття ступеня бакалавра

з напряму підготовки 121 Інженерія програмного забезпечення

на тему Розробка системи взаємодії індивідуальних сторінок викладачів та студентів. Підсистема: індивідуальні сторінки викладачів

Виконав: студент IV курсу, групи ТІ-61

Гаджалов Олег Володимирович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник Доцент Недашківський Олексій Леонідович _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Консультант _____ _____ _____
(назва розділу) (вчені ступінь та звання, прізвище, ініціали) (підпис)

Рецензент _____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2020 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки: 121 Інженерія програмного забезпечення

Спеціалізація: Програмне забезпечення веб-технологій та мобільних пристроїв

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.В. Коваль
(підпис)
”___”_____2020р.

ЗАВДАННЯ
на дипломну роботу студенту

_____Гаджалову Олегу Володимировичу_____
(прізвище, ім’я, по батькові)

1. Тема роботи Розробка системи взаємодії індивідуальних сторінок викладачів та студентів. Підсистема: індивідуальні сторінки викладачів

керівник роботи Доцент Недашківський Олексій Леонідович
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”25” грудня 2019 р. № __

2. Строк подання студентом роботи ____02.06.2020_____

3. Вихідні дані до роботи сервіси, що надють можливість стоврювати на хмарні сервери, JVM та мова програмування JavaScript, Micronaut

4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) проаналізувати існуючі програмні рішення та можливі засоби реалізації інтеграції, обґрунтувати обрані програмні застосунки та шляхи розробки програмних додатків, розробити програмне забезпечення, розробити користувацький інтерфейс, зробити висновки за результатами роботи_____

5. Перелік ілюстративного матеріалу

1)Діаграма класів, 2)діаграма структури бази даних, 3) діаграма use case_____

_____

_____

_____

_____

6. Дата видачі завдання ”___”_____20____ р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи	25.12.2019	
2.	Вивчення та аналіз задачі	20.02.2020	
3.	Розробка архітектури та загальної структури системи	10.03.2020	
4.	Розробка структур окремих підсистем	05.04.2020	
5.	Програмна реалізація системи	15.05.2020	
6.	Оформлення пояснювальної записки	01.06.2020	
7.	Захист програмного продукту	10.06.2020	
8.	Передзахист	10.06.2020	
9.	Захист	17.06.2020	

Студент _____ _____Гаджалов О. В._____
(підпис) (прізвище та ініціали,)

Керівник роботи _____ _____Недашківський О. Л.__
(підпис) (прізвище та ініціали,)

АНОТАЦІЯ

Мета роботи полягала у створені підсистеми індивідуальних сторінок викладачів з можливістю комунікації викладача зі студентом. Всі обчислення проводяться на стороні сервера повертаючи клієнту отриманий результат. Користувацький додаток представляє собою графічний інтерфейс веб-сторінки. Ключові слова: хмарний сервіс, мікросервіси, Micronaut, REST, React, Message Broker, Object Storage. Записка містить 66 сторінки, 32 рисунки та 20 посилань.

ABSTRACT

The purpose of the work was to create subsystems of individual pages of teachers with the possibility of communication between teacher and student. All calculations are performed on the server side returning the result to the client. The user application is a graphical interface of a web page. Keywords: cloud service, microservices, Micronaut, REST, React, Message Broker, Object Storage. The note contains 66 pages, 32 figures and 20 references.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

JVM – Java Virtual Machine, Віртуальна машина Java.

DI – Dependency injection, Впровадження залежності.

REST – Representational State Transfer, підхід до архітектури мережевих протоколів, які забезпечують доступ до інформаційних ресурсів.

AOP – aspect-oriented programming, Аспектно-орієнтоване програмування.

API – Application Programming Interface прикладний програмний інтерфейс.

HTTP — HyperText Transfer Protocol. Протокол передачі гіпер-текстових документів.

IDE – Integrated development environment. Інтегроване середовище розробки.

ВСТУП

На сьогоднішній день існує проблема комунікації та взаємодії між викладачами та студентами в умовах дистанційного навчання. І на разі, для вирішення цієї проблеми вони вимушені використовувати різне програмне забезпечення проведення навчального процесу: сервіси для спілкування, для поширення матеріалів курсу та для введення поточного контролю.

Сьогодні існує лише декілька систем, які частково вирішують дану проблему. Також більшість з них мають обмежений функціонал, який створений для покриття базових потреб, узагальнених серед всіх навчальних закладів, але при цьому унеможливлюється зміна чи додання нових функцій.

Тому метою даної роботи є вирішення поставленої проблеми шляхом запровадження розподіленої веб-система на базі мікросервісної архітектури з наступним її інтегруванням у сайт кафедри чи факультету.

Така система дистанційної комунікації між студентами та викладачами наразі є актуальною для її використання у період дистанційного навчання та подальшої підтримки учбового процесу. Система буде надавати як базові можливості: централізований доступ до сервісу через веб-браузер, зручний спосіб спілкування, можливість поширювати матеріали лекцій та переглядати студентами поточні оцінки - так і, завдяки мікросервісній архітектурі, легкому впровадженню нового фунціоналу, без обмежень у виборі мов програмування.

Данна підсистема веб-застосунку дозволятиме викладачу обмінюватися повідомленнями у чатах викладач-група, поширювати файли курсу на відведеній сторінці, переглядати розклад та виставляти поточний контроль.

Для реалізації серверної частини веб застосунку було вирішено використовувати фреймворк Micronaut. Його головними перевагами є асинхронність, швидке підняття серверу, мінімальне використовування оперативної пам’яті та велика кількість інструментів для створення швидких і функціональних мікросервісів. Даний фреймворк написаний на мові програмування Java, але також надає можливість використовувати його з такими мовами програмування як Kotlin та

Groovy.

Взаємодія між мікросервісами розроблена з використанням програмного брокера Kafka.

Для надання клієнтській частині сучасного вигляду та швидкої реалізації з можливістю створювати гнучкий інтерфейс було використано бібліотеку React яка базується на мові програмування JavaScript.

1 ЗАГАЛЬНА ЗАДАЧА СТВОРЕННЯ СИСТЕМИ ПІДТРИМКИ НАВЧАННЯ

Оскільки на сьогодні існує проблема комунікації в умовах дистанційного навчання була поставлена загальна мета даної роботи, яка полягає у полегшенні комунікації між викладачами та студентами і підтримки учбового процесу шляхом впровадження системи взаємодії індивідуальних сторінок викладачів та студентів. Також додатковою метою є розробка другорядних підсистем для підтримки чатів, інтеграції з вже існуючою системою розкладу занять НТУУ «КПІ» ім. І. Сікорського, запровадження додаткового функціоналу та підтримки головних сервісів.

Для забезпечення повноти функціоналу та надійності були поставлені такі вимоги до загальної системи програмних засобів розроблених у роботі:

- можливість легкого горизонтального розширення кожної з підсистем;
- можливість легкого інтегрування нових підсистем;
- можливість створювати кластери з однотипних підсистем;
- виведення з ладу одного з екземплярів підсистеми в кластері, не повинно призводити до зупинки всього кластеру підсистеми;
- система повинна бути відмово стійкою.
- система повинна надавати можливість миттєвої передачі повідомлень між користувачами;
- можливість передавати повідомлення як між двома користувачами, так і в середині групи користувачів;
- можливість завантажувати, поширювати, завантажувати та безпечно зберігати файли;

Для досягнення мети та виконання вище перелічених вимог були поставленні та розв’язані наступні завдання:

- вибір фреймворку для реалізації мікросервісної архітектури;
- вибір архітектури мережевих протоколів для комунікації між клієнтом і сервісами та в середині хмари, між сервісами;
- вибір протоколу та його програмної реалізації для миттєвої передачі

повідомлень;

- вибір бази даних для збереження об’єктів бізнес логіки;
- вибір об’єктного сховища для збереження файлів;
- розбиття додаткового функціоналу на логічні підсистеми та їх впровадження;

При виконанні дипломної роботи очікується спроектувати та створити систему, яка дозволяє у зручному режимі спілкуватись викладачам та студентам, та надавати весь можливий функціонал для електронної підтримки процесу навчання.

2 ЗАДАЧА СТВОРЕННЯ ПІДСИСТЕМИ ІНДИВІДУАЛЬНИХ СТОРІНОК ВИКЛАДАЧІВ

Конкретна мета даної роботи полягає у створені підсистеми індивідуальних сторінок викладачів. Підсистема повинна надавати як можна повний функціонал для полегшення введення учбового процесу, комунікації між викладачами та студентами. Також графічний інтерфейс повинен бути зручним та інтуїтивно зрозумілим.

Також були висунуті наступні вимоги до підсистема індивідуальних сторінок викладачів:

- можливість перегляду викладачем його розкладу;
- можливість зберігати електронний матеріал на серверах підсистеми;
- можливість поширювати електронний матеріал між групами, студентами;
- можливість комунікації зі студентами у групових чатах;
- можливість персональної комунікації зі студентом;
- інтуїтивно зрозумілий графічний інтерфейс.

Для досягнення мети та виконання вимог були поставленні та розв’язані наступні завдання:

- створення моделей бізнес логіки та зв’язків між ними;
- створення сервісу авторизації та механізму автентифікації на інших сервісах та підсистемах;
- створення мінімального функціоналу для полегшення супроводу навчального процесу
- розробка дизайну графічного інтерфейсу.

3 АНАЛІЗ ЗАГАЛЬНОЇ ПРЕДМЕТНОЇ ГАЛУЗІ

Системи підтримки учбового процесу та комунікації між її учасниками є достатньо вузькоспеціалізованим програмним забезпеченням, тому не являються розповсюдженими. Серед проаналізованих аналогів загальної системи даної роботи, були відібрані наступні системи: moodle[1] та сайт підтримки навчального процесу НТУУ «КПІ» ім І. Сікорського ecampus.kpi.ua[2].

3.1 Moodle

Moodle(Modular Object-Oriented Dynamic Learning Environment) - це навчальна платформа, створена для того, щоб надати викладачам, адміністраторам та учням єдину надійну, безпечну та інтегровану систему для створення персоналізованих навчальних середовищ. Він надає можливість завантажити програмне забезпечення на власний веб-сервер. На основі Moodle створено десятки тисяч навчальних середовищ у всьому світі, Moodle довіряють великі та малі установи та організації, включаючи Shell, Лондонську школу економіки, Державний університет Нью-Йорка, Microsoft та Open University. Всесвітня кількість аудиторії Moodle - понад 90 мільйонів користувачів на рівні академічного та корпоративного рівня - це найпоширеніша навчальна платформа у світі. Завдяки 10-річному розвитку, керованому педагогікою соціальної конструктивістики, Moodle пропонує потужний набір інструментів, орієнтованих на учнів, та викладачів, що сприяють навчанню та викладанню. Простий інтерфейс, функції перетягування та добре задокументовані ресурси, а також постійні удосконалення роблять Moodle легким у вивченні та використанні.

Moodle надається безкоштовно як програмне забезпечення з відкритим кодом, відповідно до Загальної публічної ліцензії GNU[3]. Будь-хто може адаптувати, розширювати або змінювати Moodle як для комерційних, так і для некомерційних проектів без будь-яких ліцензійних платежів та отримувати вигоду від економічної ефективності, гнучкості та інших переваг використання Moodle. Підхід проекту з відкритим кодом Moodle означає, що Moodle постійно переглядається та

вдосконалюється, щоб відповідати поточним та розвиваючим потребам своїх користувачів.

Рисунок 3.1 – Інтерфейс системи moodle

Багатомовні можливості Moodle забезпечують відсутність лінгвістичних обмежень для навчання в інтернеті. Спільнота Moodle розпочала переклад Moodle більш ніж на 120 мов, щоб користувачі могли легко локалізувати свій сайт Moodle, а також безліч ресурсів, підтримки та обговорень спільноти на різних мовах. Moodle надає найбільш гнучкий набір інструментів для підтримки як змішаного навчання, так і повністю онлайн-курсів. Налаштуйте Moodle, включивши або відключивши основні функції, та легко інтегруйте все необхідне для курсу, використовуючи повний спектр вбудованих функцій, включаючи зовнішні інструменти спільної роботи, такі як форуми, вікі, чати та блоги. Оскільки він є відкритим кодом, Moodle може бути налаштований будь-яким способом та підлаштовуватися під індивідуальні потреби. Його модульне налаштування та сумісна конструкція дозволяє розробникам створювати плагіни та інтегрувати зовнішні програми для досягнення конкретних функціональних можливостей. Розширюйте те, що робить Moodle, використовуючи вільно доступні плагіни та додатки. Від декількох студентів до мільйонів

користувачів Moodle може бути розширений, щоб підтримувати потреби як малих класів, так і великих організацій. Завдяки своїй гнучкості та масштабованості Moodle був пристосований для використання в контексті освіти, бізнесу, некомерційного, урядового та громадського середовища. Прихильний до захисту безпеки даних та конфіденційності користувачів, контроль безпеки постійно оновлюється та впроваджується в процеси розробки та програмного забезпечення Moodle для захисту від несанкціонованого доступу, втрати даних та неправильного використання. Moodle може бути легко розгорнутий на приватному захищеному сервісі-хмарі або сервері для повного контролю.

Moodle базується на веб-серверній архітектурі, тому до нього можна отримати доступ з будь-якої точки світу. Завдяки інтерфейсу для мобільних пристроїв за замовчуванням та сумісності між браузерами вміст на платформі Moodle легко доступний та сумісний у різних веб-браузерах та пристроях. Отримайте доступ до великої документації щодо Moodle та форумів користувачів на різних мовах, безкоштовного контенту та курсів, якими користуються користувачі Moodle у всьому світі, а також сотень плагінів, внесених великою світовою спільнотою.

Проект Moodle підтримується активною міжнародною спільнотою, командою відданих розробників та штатних партнерів Moodle. Керуючись відкритою співпрацею та великою підтримкою громади, проект продовжує досягати швидких виправлень помилок та вдосконалень, з новими основними випусками кожні півроку.

Як переваги даної системи можна перелічити наступні пункти:

- Сучасний, простий у користуванні інтерфейс;
- Персоналізована інформаційна панель;
- Інструменти та заходи для спільної роботи;
- Календар;
- Зручне управління файлами;
- Простий та інтуїтивно зрозумілий текстовий редактор;
- Сповіщення;
- Відстеження прогресу.

Серед недоліків можна виділити наступне:

- Немає зручної системи для миттєвої комунікації між студентами та студентами і викладачами;
- Цілодобова підтримку користувачів, яка через велику кількість запитів не завжди має можливість швидко відповісти.

3.2 Система ЕК НТУУ «КПІ»

Електронний кампус НТУУ «КПІ» ecampus.kpi.ua є складовою Корпоративного порталу НТУУ «КПІ», що є підсистемою Єдиного Інформаційного Середовища НТУУ «КПІ». Система була розроблена для інформаційної підтримки навчального процесу університету з метою підвищення якості навчання студентів, забезпечення навчального процесу сучасними інформаційними технологіями. Серед функціоналу система надає наступні рішення. Віртуальні кабінети за профілями користувачів: студент, що розширює функціонал, якщо студент є старостою чи профоргом; викладач-науковець, розширюється якщо профіль є куратором групи; методист кафедри; завідувач кафедрою. Роботу з ЕК НТУУ «КПІ» підтримують всі сучасні браузери, такі як Internet Explorer, Mozilla Firefox, Opera і Google Chrome. Головна сторінка ПВК системи ЕК НТУУ «КПІ» містить кнопки переходу до наступних розділів: мій профіль, щоденник, контакти, довідка, форум, дошка оголошення, повідомлення та розкладу (рис. 3.2).

Рисунок 3.2 – Головна сторінка ПВК системи ЕК НТУУ «КПІ»

Серед переваг системи можна виділити наступне:

- Простий для розуміння інтерфей;
- Доступ до усієї потрібної інформації такої як поточного контролю, контактів, довідки, форуму, дошки оголошень та розкладу;
- Анонімна система оцінювання викладачів.

Серед недоліків ЕК НТУУ «КПІ» є такі як:

- Не повний функціонал усіх вкладок;
- Недороблена основна версія кампусу (поточна робота з електронним кампусом ведеться у тимчасовій версії);
- Технічні помилки в інтерфейсі.

Кожна з систем має свої недоліки, через що, повнота функціоналу не задовольняє сучасним потребам до систем інформаційної підтримки навчального процесу університету.

Також, спільним недоліком є відсутність або погано реалізована підсистема комунікації між студентами в групах та студентами і викладачами. З огляду на це, для вдосконалення функціоналу об’єктів даної предметної галузі, було вирішено впровадити сучасну реалізацію миттєвого обміну між користувачами та збереження корисних можливостей існуючих систем.

4 РОЗПОДІЛ ОБОВ'ЯЗКІВ МІЖ ВИКОНАВЦЯМИ

Так-як реалізуєма система складається з двох головних підсистем та декількох допоміжних, було поставлено завдання розподілу обов’язків між її виконавцями. Головними задачами до розподілу обов’язків є мінімізування загального часу реалізації всієї системи та розробка кожним виконавцем сервісів, в реалізуванні яких він має більше досвіду.

Виходячи з зазначеного вище було прийнято рішення по розподілу обов’язків наступним чином. Перший виконавець відповідальний за розробку таких сервісів та підсистем:

- Підсистема індивідуальних сторінок викладачів;
- Сервіс автентифікації;
- Сервер об’єктного сховища.

Другий виконавець має реалізувати наступні елементи системи:

- Підсистема індивідуальних сторінок студентів;
- Сервіс розкладу;
- Сервер брокеру повідомлень.

5 ЗАСОБИ РОЗРОБКИ ЗАГАЛЬНОЇ ПРОГРАМНОЇ СИСТЕМИ

Для реалізації серверної частини загальної програмної системи було обрано мову програмування Java[4]. Серед всіх фреймворків даної мови програмування було проаналізовано та обрано два, які розраховані на створення серверних додатків: Spring[5] та Micronaut[6]. Spring являється одним з основних фреймворків для створення серверних додатків, так-як він має широкий спектр для вирішення базових задач та існує використовується з 2004 року. Micronaut є новим фреймворком, так як він існує з 2019 року, але попри це, як і Spring, має достатній функціонал та інтеграцію з іншими серверними додатками. Порівнюючи ці фреймворки, було вирішено використовувати Micronaut, так-як як він має наступуні переваги, які дозволяють легше прототипувати мікросервісну архітектуру:

- інструменти для автоматичного визначення середовища та подальшого налаштування додатку;
- інструменти для легкої конфігурації додатка, в залежності від середовища розгортання;
- базова підтримка як асинхронної так і синхронної обробки запитів;
- інтеграція з бібліотеками та додатками брокерів повідомлень;

Для тестування було обрано бібліотеку Spock[7]. Ця бібліотека дозволяє використовувати підхід behaviour-driven development(з англ. - керована поведінкою розробка). Цей підхід дозволяє тестувати не конкретний об’єкт чи метод, а деяку поведінку чи специфікацію.

Клієнтську частину системи було вирішено реалізовувати на мові програмування JavaScript[8] з використанням бібліотеки React[9], яка є бібліотекою програмних засобів з відкритим вихідним кодом, розробленою компанією Facebook. Дана бібліотека дозволяє створювати графічний інтерфейс, що складається з програмних одиниць - «компонентів». Вони можуть бути впроваджені розробником або іншими бібліотеками. Дана бібліотека дозволяє використовувати один компонент у багатьох інших компонентах або сторінках. Такий підхід дозволяє пришвидшити

прототипування дизайну клієнтської частини і відділити функціональну логіку обробки графічного інтерфейсу, від логіки комунікації з серверною частиною.

Так-як серверна частина побудована за принципом мікросервісної архітектури, важливим елементом стає брокер повідомлень, який виконує функцію забезпечення передачі повідомлень між мікросервісами. В ситуації коли відправлене повідомлення одного сервісу може бути не прийнятим іншим, перевантаженим, сервісом, воно втрачається, що призводить до втрати важливої інформації. Тому для комунікацію між сервісами використовують брокер повідомлень, який гарантує передачу повідомлень. В даній системі було вирішено використовувати брокер повідомлень Apache Kafka[10] розроблений компанією LinkedIn. Його головними перевагами є легке горизонтальне розширення та тимчасового зберігання даних, що були передані.

5.1 Фреймворк Micronaut

Micronaut – фреймворк на базі JVM для побудови модульних, з легким тестуванням мікросервісних та безсерверних додатків. При його проектуванні, розробники мали на меті забезпечити всі інструменти, необхідні для створення повнофункціональних мікросервісних додатків, включаючи:

- Продумані значення за замовчуванням та автоматична конфігурація
- Можливість конфігурації та поширення конфігурацій
- Знаходження сервісів
- Маршрутизація HTTP
- HTTP-клієнт з балансуванням навантаження на стороні клієнта

У той же час Micronaut уникнає недоліків інших фреймворків, забезпечуючи:

- Швидкий час запуску
- Зниження відбитку використовування пам'яті
- Мінімальне використання рефлексії
- Мінімальне використання проксі
- Легке блочне тестування

Також такі фреймворки, як Spring та Grails[11], не розроблялися для використання у таких сценаріях, як безсерверних функції, додатки для Android або мікросервіси з низьким використанням пам'яті. На відміну від них, Micronaut розроблений таким чином, щоб він підходить для всіх цих сценаріїв.

Ця мета досягається завдяки використанню процесорів анотацій Java, які можна використовувати на будь-якій мові JVM, яка їх підтримує та використанню HTTP-серверу та клієнту, побудованих на Netty[12]. Для того, щоб надати подібну модель програмування як у Spring чи Grails, ці процесори анотацій попередньо компілюють необхідні метадані для виконання DI, визначення AOP проксі та налаштування додатка для роботи в середовищі мікросервісів.

5.1.1 Інверсія контролю(Inversion of Control)

Інверсія контролю також відома як ін'єкція залежностей (DI), яка піклується про розподіляння залежностей між різними шарами або компонентами шляхом спільних абстракцій. У об'єктно-орієнтованому програмуванні DI - це комбінація фабричного шаблону, шаблону знаходження сервісів, шаблону стратегії та шаблону «одинак». При впроваджені даного шаблону створюється IoC контейнер. Він створює об'єкт зазначеного класу, а також вводить усі об'єкти залежності через конструктор, поле або метод під час виконання та позбувається його у відповідний час. Це робиться для того, щоб розробник не створював та керував об'єктами вручну.

Micronaut використовує даний шаблон програмування за деякими оптимізаціями та покращеннями. Так як рефлексія в Java є використовує багато ресурсів, то на відміну від Spring, який покладається виключно на рефлексію під час виконання програми та впроваджень проксі, Micronaut використовує дані що створюються під час компіляції для ін'єкції залежностей. Так-як Micronaut розроблений для побудови мікросервісів на стороні сервера він надає багато тих же інструментів та утиліт, що і Spring, але без використання рефлексії чи кешування надмірної кількості метаданих відображень. IoC контейнер Micronaut використовує наступні оптимізації:

- Використовує рефлексію в крайньому випадку
- Уникає проксі

- Оптимізує час запуску
- Зменшує відбиток використаної пам’яті
- Забезпечує чітку, зрозумілу обробку помилок

Також модуль IoC Micronaut може використовуватися повністю незалежно від самого Micronaut для будь-якого типу програм.

При використані шаблону інверсії контролю підконтрольні об’єкти IoC контейнера називають бінами(від англ. – bean). Весь життєвий цикл біна контролюється контейнером. Для того, щоб фреймворк почав розпізнавати клас як бін, треба додати одну із анотацій: @Singleton, @Context, @Prototype, @Infrastructure, @ThreadLocal, @Refreshable або @RequestScope. Приклад створення біну та його введені в інший клас наведено на рис. 5.1.

```
public interface Engine { 1
    int getCylinders();
    String start();
}

@Singleton 2
public class V8Engine implements Engine {
    public String start() {
        return "Starting V8";
    }

    public int getCylinders() {
        return cylinders;
    }

    public void setCylinders(int cylinders)
        this.cylinders = cylinders;
    }

    private int cylinders = 8;
}

@Singleton
public class Vehicle {
    private final Engine engine;

    public Vehicle(Engine engine) { 3
        this.engine = engine;
    }
```

Рисунок 5.1 - Створення біну та його введені в інший клас. Цифрами позначено: (1) опис стандартного інтерфейсу; (2) опис класу та позначення його анотацією

@Singleton; (3) введення залежності.

5.1.2 HTTP сервер

Micronaut включає як неблокуючі HTTP-сервер, так і клієнтські API на основі фреймворку Netty. Netty - це клієнт-серверний фреймворк на основі концепції NIO, який дозволяє швидко та просто розроблювати мережеві додатки. Це значно спрощує та впорядковує мережеве програмування, такі як TCP та UDP сокет-сервери. Netty був ретельно розроблений з досвідом, зробленим від впровадження безлічі протоколів, таких як FTP, SMTP, HTTP та різних бінарних та текстових застарілих протоколів. У результаті Netty вдалося знайти спосіб досягти простоти розвитку, продуктивності, стабільності та гнучкості.

Назва концепції NIO пішла від Java пакету java.nio (NIO розшифровується як неблокуючі введення / виведення). java.nio це сукупність API, які пропонують функції для інтенсивних операцій вводу / виводу. API NIO були розроблені, щоб забезпечити доступ до низькорівневих операцій вводу / виводу сучасних операційних систем. Хоча API самі по собі відносно високого рівня, метою є сприяння реалізації, яка може безпосередньо використовувати найефективніші операції базової платформи.

Дизайн сервера HTTP в Micronaut оптимізований для обміну повідомленнями між мікросервісами, як правило, в форматі JSON, і не є повноцінною структурою для MVC-сервера. Наприклад, наразі немає підтримки для перегляду на стороні сервера або функцій, типових для традиційного MVC-сервера. Мета сервера HTTP - зробити його максимально простим, для легкого відкриття кінцевих точок API, які можуть приймати запити HTTP-клієнтів, якою б мовою вони не були написані.

Для відкриття однієї кінцевої точки API потрібно створити клас «контролер». За та допомогою анотацій вказати шлях, по якому буде відкритий доступ до нього, та які методи повинні викликатись в залежності від HTTP-метода запита. Анотація @Controller відповідає за створення біну кінцевої точки. Анотація @Get над методом index() вказує на те, що даний метод буде викликано, коли прийде запит з HTTP-методом GET. Приклад наведений на рис. 5.2.

```
@Controller("/hello") (1)
public class HelloController {

    @Get(produces = MediaType.TEXT_PLAIN) (2)
    public String index() {
        return "Hello World"; (3)
    }
}
```

Рисунок 5.2 – Створення контролеру. Цифрами позначено: (1) клас визначається як контролер з анотацією @Controller, відображеною на шлях /hello; (2) Метод відповість на запит GET; (3) Відповідь з тілом “Hello world”

5.1.3 Безпека

Мікронавт поставляється з можливостями впровадження безпеки на базі Json Web Token (JWT). JWT - стандарт IETF, який визначає безпечний спосіб інкапсуляції довільних даних, які можуть надсилатися через незахищені URL-адреси. Також JWT може бути зашифрований, щоб забезпечити цілісність між сторонами. Підписані жетони можуть перевірити цілісність сигнатур, що містяться в ньому, а зашифровані маркери приховують ці сигнатуру від інших сторін. Коли токени підписуються за допомогою пар відкритих / приватних ключів, підпис також засвідчує, що лише сторона, що тримає приватний ключ, є тією, яка його підписала.

Схема процесу аутентифікації наведена на рис. 5.3.

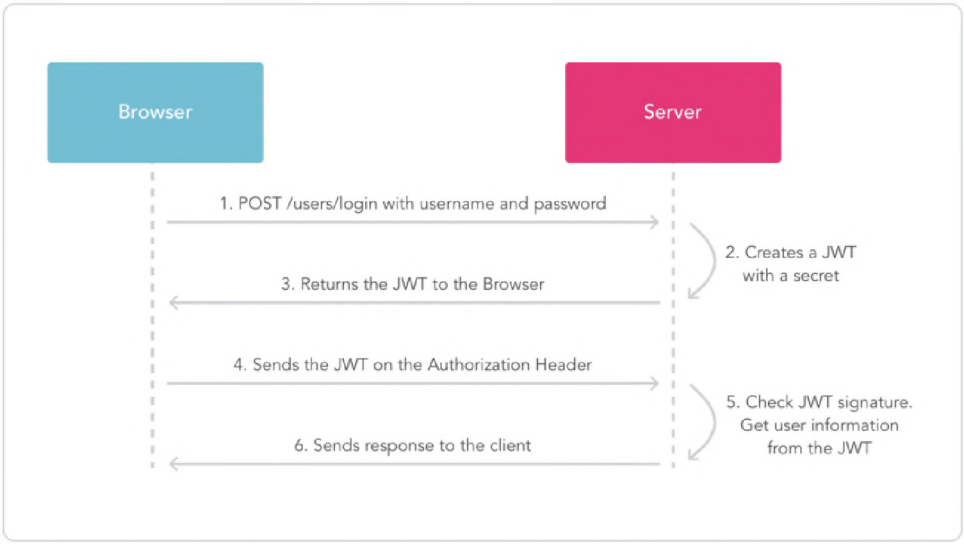

Рисунок 5.3 – схема авторизації через JWT токен

5.1.4 Інтеграція з Hibernate ORM

Micronaut Data - це інструментарій доступу до бази даних, який використовує компіляцію Ahead of Time (AoT) для попереднього обчислення запитів інтерфейсів репозиторіїв, які потім виконуються під час роботи програми. Micronaut Data були створені по принципу GORM та Spring Data, однак покращують рішення цих біліотек наступним чином:

- Не створює моделі під час виконання - GORM і Spring Data підтримують мета-моделі, що створюються під час виконання програми, які використовують рефлексію для моделювання взаємозв'язків між класами об'єктів бізнес логіки. Ця модель споживає значні об'єми пам'яті, а зі збільшенням розміру вашої програми зростають потреби в пам'яті. Проблема стає ще гіршою у поєднанні з Hibernate, який підтримує власну мета-модель, но призводить до дублікату мета-моделей.
- Немає перекладу запитів - І GORM, і Spring Data використовують регулярні вирази та відповідність шаблонів у поєднанні з проксі-серверами, що генеруються під час виконання програми, щоб перетворити метод в інтерфейсі Java в SQL запит. У Micronaut Data немає такого перетворення під час роботи програми, так-як ця робота проводиться під час компіляції.
- Немає рефлексії чи проксі під час виконання програми – Micronaut Data не використовують проксі чи рефлексію при виконанні програми, що призводить до кращої продуктивності, менших відбитків в стеку та зменшення споживання пам'яті через повну відсутність кеш-рефлексій.
- Безпечні типи – Micronaut Data активно перевіряє під час компіляції, чи може бути реалізовано метод репозиторію, і якщо ні, компіляція не вдасться.

Micronaut Data надає загальний API для перекладу моделі запиту під час компіляції в запит та забезпечує підтримку для наступних бібліотек:

- JPA (Hibernate)
- SQL (JDBC)

В даній роботі було використано бібліотеку Hibernate. Hibernate ORM дозволяє розробникам легше писати програми, дані яких потрібно також зберігати поза виконанням програми. Як фреймворк об’єкту/реляційного відлбраження(ORM), Hibernate піклується про стійкістю даних, оскільки це стосується реляційних баз даних (через JDBC). Hibernate побудовано за специфікацією JPA(Java Persistence API).

JPA піклується про постійне зберігання, що може означати будь-який механізм, за допомогою якого об'єкти Java можуть існувати поза виконанням програми, яка їх створила. Не всі об’єкти Java потрібно зберігати, але більшість програм зберігають ключові бізнес-об’єкти. Специфікація JPA дозволяє визначити, які об’єкти слід зберігати, а як ні.

JPA сама по собі не є інструментом або фреймворком; скоріше, він визначає набір концепцій, які можуть бути реалізовані будь-яким інструментом або фреймворком. Хоча модель об'єктно-реляційного відображення (ORM) JPA спочатку базувалася на Hibernate, на даний час вона є самостійною бібліотекою. Так само, як спочатку JPA призначався для використання з реляційними / SQL базами даних, деякі реалізації JPA були розширені для використання зі сховищами даних NoSQL. Популярною основою, яка підтримує JPA з NoSQL, є EclipseLink, реалізація стандарту JPA 2.2.

5.2 Бібліотека React.js

Для розробки клієнтської частини було вирішено створити веб-застосунок по принципу односторінкового застосунку SPA (single-page application). Такі веб-застосунки чи веб-сайти, які вміщується на одній сторінці з метою забезпечити користувачу досвід близький до користування настільною програмою. В односторінковому застосунку весь необхідний код - HTML, JavaScript, та CSS - завантажується разом зі сторінкою, або динамічно довантажується за потребою, зазвичай у відповідь на дії користувача. Сторінка не оновлюється і не перенаправляє користувача до іншої сторінки у процесі роботи з нею. Взаємодія з односторінковим застосунком часто включає в себе динамічний зв'язок з веб-сервером.

Для реалізації даної парадигми було вирішено використовувти бібліотек React.js. React.js це відкрита JavaScript бібліотека для створення інтерфейсів користувача, яка покликана вирішувати проблеми часткового оновлення вмісту веб-сторінки, з якими стикаються в розробці односторінкових застосунків. Розробляється Facebook, Instagram і спільнотою індивідуальних розробників.

React дозволяє розробникам створювати великі веб-застосунки, які використовують дані, котрі змінюються з часом, без перезавантаження сторінки. Його мета полягає в тому, щоб бути швидким, простим, масштабованим. React обробляє тільки користувацький інтерфейс у застосунках. Це відповідає видові у шаблоні модель-вид-контролер (MVC), і може бути використане у поєднанні з іншими JavaScript бібліотеками або в великих фреймворках MVC, таких як AngularJS. Він також може бути використаний з React на основі надбудов, щоб піклуватися про частини без користувацького інтерфейсу побудови веб-застосунків. Як бібліотеку інтерфейсу користувача React найчастіше використовують разом з іншими бібліотеками, такими як Redux.

React підтримує віртуальний DOM, а не покладається виключно на DOM браузера. Це дозволяє бібліотеці визначити, які частини DOM змінилися, порівняно (diff) зі збереженою версією віртуального DOM, і таким чином визначити, як найефективніше оновити DOM браузера. Таким чином програміст працює зі сторінкою, вважаючи що вона оновлюється вся, але бібліотека самостійно вирішує які компоненти сторінки треба оновити.

Компоненти React зазвичай написані на JSX. Код написаний на JSX компілюється у виклики методів бібліотеки React. Розробники можуть так само писати на чистому JavaScript. JSX нагадує іншу мову, яку створили у компанії Фейсбук для розширення PHP, XHP.

6 МЕТОДИ РОЗРОБКИ ПІДСИСТЕМИ

Обрані засоби розробки програмного забезпечення дозволяють досягти мети даної роботи, розробки підсистеми для підтримки навчального процесу та комунікації з викладача зі студентом, декількома шляхами. Так-як до даної підсистеми були висунуті специфічні вимоги, було вирішено використовувати наступні методи розробки програмного забезпечення.

6.1 Архітектура REST API

Для забезпечення можливості перегляду викладачем його розкладу, було розроблено програмний елемент для комунікації з існуючим сервісом НТУУ «КПІ» ім. І. Сікорського rozklad.kpi.ua через шаблон мережевої архітектуру REST API.

Мережева архітектура REST API використовує протокол HTTP, його команди GET, POST, PUT, DELETE та URI для однозначної передачі, отримування чи модифікації інформаційних об’єктів, різних тематичних доменів сервісу. Дані передаються в форматі структурованого тексту, який визначається серверною частиною. Найбільш популярні формати: JSON та XML.

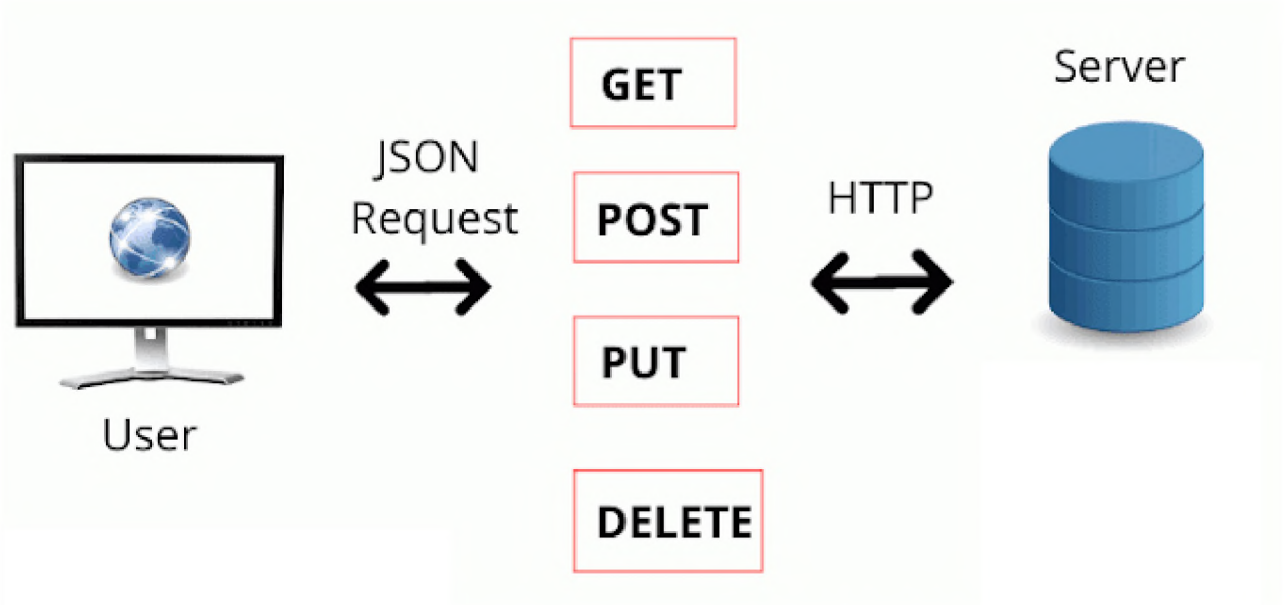

Рисунок 6.1 – Схема роботи REST API

HTTP(Hyper text transport protocol) - це протокол, який використовується для

передачі даних через Інтернет та інші мережі. Він є частиною набору Інтернет-протоколів і визначає команди та сервіси, що використовуються для передачі даних веб-сторінок. HTTP використовує модель сервер-клієнт. Клієнтом, наприклад, може бути домашній комп'ютер, ноутбук або мобільний пристрій. Сервер HTTP, як правило, являє собою веб-хост, на якому працює програмне забезпечення веб-сервера. При відкритті веб-сайту, браузер надсилає запит на відповідний веб-сервер, і він відповідає кодом статусу HTTP. Якщо URL-адреса дійсна і з'єднання надано, сервер надішле веб браузеру веб-сторінку та пов'язані з ними файли. HTTP має декілька класів кодів статусу. Коди статусу – це тризначні коди, що можуть бути класифіковані в різні класи, де перша цифра означає відповідний клас статусу. Коди першого класу представляють інформацію для обробки та надсилаються під час запиту, вони включають код статусу 100 та код статусу 102. Коди другого класу представляють успішну операцію. Один з найпоширеніших кодів статусу HTTP, який починається з 2, - код статусу 200 ОК. Код статусу третього класу являє собою переспрямування і повертається, якщо запитуваний документ тепер доступний за іншою адресою. Тому обробка ще не завершена і вимагає подальших дій клієнта. Деякі з найважливіших кодів статусу цього класу щодо оптимізації пошукових систем - це код 301 та код статусу 302. Коди четвертого класу представляють помилки клієнта, тобто помилки, що виникають внаслідок несправного запиту клієнта. Хороший приклад цього класу - код статусу 404 Не знайдено. П'ятий клас містить помилки сервера. Це помилки, які приписуються серверу. Код статусу 500 Внутрішня помилка сервера та код статусу 503 Служба недоступна - хороші приклади цього класу. Дев'ятий клас статусу охоплює як стандартизовані коди статусу, так і фірмові коди, які можуть виникати за певних обставин. Тут помилка приписується мережі, і клієнт зобов’язаний повторно надіслати запит. Найбільш поширені в цьому класі - код статусу 906 та код статусу 950.

Також для інтеграції даної мережевої архітектури потрібно, що функціонал клієнта та сервера відповідали наступним принципам:

- Клієнт-сервер - відокремивши проблеми користувальницького інтерфейсу від проблем зберігання даних, покращується переносимість призначеного для

користувача інтерфейсу на кілька платформ і підвищується масштабованість за рахунок спрощення серверних компонентів.

- Тимчасовість стану - кожен запит від клієнта до сервера повинен містити всю інформацію, необхідну для розуміння запиту, і не може зберігатись у серверному контексті. Тому стан сеансу повністю зберігається на клієнті.
- Кеш - обмеження кешування вимагають, щоб дані у відповіді на запит були неявно або явно позначені як кешувальні або не кешувальні. Якщо відповідь є кешувальною, то для клієнтського кешу надається право повторно використовувати дані відповіді для наступних, еквівалентних запитів.
- Уніфікований інтерфейс - застосовуючи принцип програмної інженерії загального характеру до інтерфейсу, спрощується загальна архітектура системи і поліпшується видимість взаємодій. Для отримання уніфікованого інтерфейсу необхідна наявність безлічі архітектурних обмежень, що визначають поведінку компонентів. REST визначається чотирма обмеженнями інтерфейсу: ідентифікація ресурсів; маніпулювання ресурсами через команди; самоопис повідомлень; і, гіпермедіа як двигун для стану програми.
- Багаторівнева система - Стиль багаторівневої системи дозволяє побудувати архітектуру, що складається з ієрархічних шарів, обмежуючи поведінку компонентів таким чином, що кожен компонент не може "бачити" за межами безпосереднього шару, з яким вони взаємодіють.
- Код за запитом (опціонально) - REST дозволяє розширити функціональність клієнта, завантаживши та виконавши код у вигляді аплетів або скриптів. Це спрощує роботу з клієнтами, зменшуючи кількість функцій, які повинні бути попередньо реалізовані.

6.2 Управління файлами

Для надання можливості завантажувати електронний матеріал у вигляді файлів серверна частина повинна правильно та розумно управляти їх зберіганням та транспортуванням. Для виконання цієї вимоги було вирішено дві проблеми. Перша – так-як принцип REST API не розрахований на передачу файлів, потрібно було обрати

інший метод для їх передачі. Друга – реляційні бази не пристосовані для ефективного зберігання та забезпечення доступу до неструктурованих типів даних, таких як файли, то потрібно було використати інший тип баз даних.

Для вирішення першої проблеми було використано метод з використанням multipart/form-data. Визначення даних, що складаються з декількох multipart/form-data. Вибирається boundary(межа) – послідовність символів, яка не зустрічається в жодному з наборів даних. Іноді цей вибір робиться виподково. Кожне поле форми відправляється в тому порядку, в якому воно зустрічається в формі, як частина потоку даних, що складається з декількох частин. Кожна частина ідентифікує ім'я вводу. Кожна частина повинна мати відповідне маркування відповідним content-type, якщо відомий тип або як application/octet-stream. Медіа-тип даних, що складаються з декількох multipart/form-data, дотримується правил всіх потоків даних, що складаються з декількох частин MIME, як зазначено в RFC 1521[13]. Він призначений для використання при поверненні даних, одержуваних при заповненні форми. У формі існує ряд полів, які повинен надати користувач, що заповнює форму. Кожне поле має ім'я. Усередині даної форми імена унікальні. multipart/form-data містять ряд частин. Кожна частина повинна містити заголовок "content-disposition ", де значення - " form-data", а атрибут імені визначає ім'я поля всередині форми, наприклад, content-disposition:form-data, name=xxxxx, де xxxx - це ім'я поля , відповідне цьому полю. Імена полів, спочатку не в наборах символів ASCII, можна закодувати за допомогою методу, описаного в RFC 1522[14]. Як і у всіх типах багатокомпонентних MIME, кожна частина має додатковий Content-Type, який за замовчуванням має значення text / plain. якщо вміст файлу повертається шляхом заповнення форми, то введення файлу ідентифікується як application/octet-stream або як відповідний тип носія, якщо він відомий. Якщо в результаті заповнення однієї форми повинно бути повернуто кілька файлів, то вони можуть бути повернуті у вигляді multipart/mixed, вбудованих в multipart/form-data. Кожна частина може бути закодована і заголовок "Контент-передача-кодування". поставляється, якщо значення цієї частини не відповідає значенню за замовчуванням кодування. Файлові входи можуть також ідентифікувати ім'я файлу. Файл може бути описана за допомогою параметра filename або заголовку

content-disposition . Це не обов'язково, але настійно рекомендується в будь-якому випадку, де відомо оригінальне ім'я.

```
POST https://someLMShost.edu/d21/api/eP/{version}/import/new HTTP/1.1
Content-Type: multipart/form-data; boundary=xxBOUNDARYxx
Content-Length: {POST body length in bytes}

--xxBOUNDARYxx
Content-Type: text/plain
Content-Disposition: form-data; name="targetUsers"

12345
--xxBOUNDARYxx
Content-Type: text/plain
Content-Disposition: form-data; name="targetUsers"

67890
--xxBOUNDARYxx
Content-Type: applicaiton/zip
Content-Disposition: form-data; name="file"; filename="import-package.zip"

{file data bytes}
--xxBOUNDARYxx
```

Рисунок 6.2 – Приклад передачі файлу через multipart/form-data

Для вирішення другої проблеми було використано бази даних типу об’єктного сховища. Об’єктне сховище - це стратегія зберігання даних, яка розділяє дані на окремі одиниці або об'єкти, які зберігаються в ізольованому сховищі разом з усіма відповідними метаданими і унікальним ідентифікатором. Об'єктне сховище являє собою масивне, масштабуєме рішення для зберігання статичних даних. На відміну від файлових або блокових сховищ, в об'єктному сховище не використовується ієрархія або дерево каталогів. Замість цього кожна окрема одиниця даних існує на одному рівні в сховищі. Така унікальна конструкція робить об'єктне сховище більш масштабуємим, ефективним і надійним, ніж всі інші рішення для зберігання статичних даних. Багато організацій настільки зросли, що їх статичне сховище складається з петабайт даних. Традиційні системи зберігання, такі як SAN і NAS, не можуть масштабуватися до потрібної величини в хмарному сховищі. Об'єктне сховище, з іншого боку, може легко масштабуватися до такої величини, так-як не має меж. Сховище об'єктів не призначене для прямого доступу операційної системи. Замість цього взаємодія відбувається через API на рівні додатків через угоди RESTful.

Як правило, така взаємодія відбувається на одній кінцевій точці, що виключає зіставлення LUN і топологію мережевого сховища з процесу проектування додатків. Об'єктне сховище має різні варіанти використання, які вимагають, щоб зберігання даних масштабується до петабайт і вище, і все це при збереженні високого рівня доступності та продуктивності.

До даних, які найбільше виграють від об'єктного зберігання, відносяться:

- неструктуровані дані, такі як музика, зображення та відео;
- створення резервних копій та лог-файли;
- великі набори історичних даних;
- архівні файли.

З цього списку легко побачити, де об'єктне сховище має велике значення для корпоративних потреб. Резервні копії та архіви, наприклад, критично важливі для аварійного відновлення; набори історичних даних можна використовувати для виконання аналізу великих обсягів даних на таких джерелах, як веб-сайти електронної комерції, соціальні мережі, датчики IoT і багато іншого. Але сховище об'єктів . У розглянутих вище випадках використання воно значно краще ніж методи, такі як блочне зберігання. Однак існують сценарії, в яких об'єктне сховище не є найкращим рішенням.

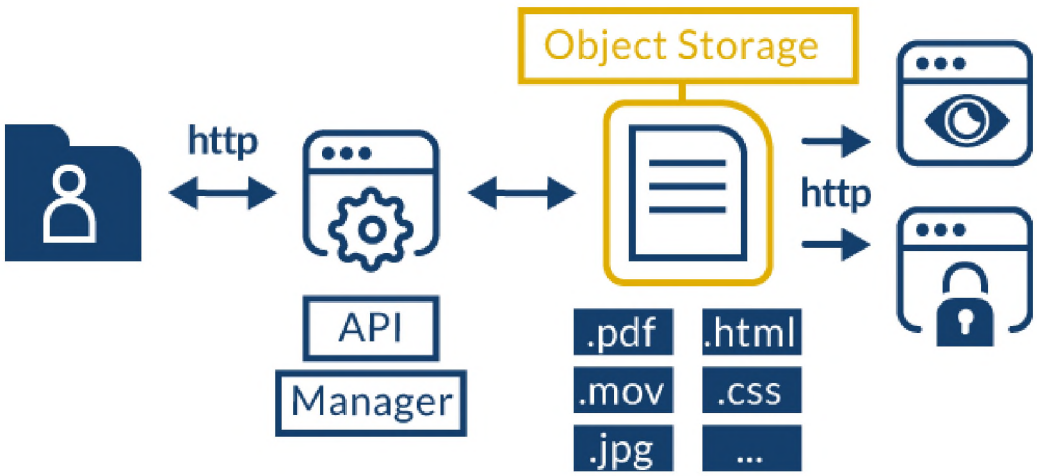

Рисунок 6.3 – Схематичне зображення роботи об’єктного сховища

6.3 Протокол WebSocket

Так-як підсистема потребує одночасної підтримки миттєвого обміну

повідомленнями між декількома користувачам, архітектура REST API та протокол HTTP не надають такого функціоналу, тому для рішення такої задачі було вирішено використовувати протокол WebSocket[15]. На початку 2000 веб був побудований на ідеї, що робота клієнта полягає в тому, щоб запитувати дані з сервера, а робота сервера полягає в тому, щоб виконувати ці запити. Ця парадигма залишалася безперечною протягом декількох років, але з введенням AJAX почали вивчати можливості встановлення двонапрямлених з'єднань між клієнтом і сервером. Сьогодні веб-додатки збільшились та споживають більше даних, ніж будь-коли раніше. Найбільше їх стримувала традиційна HTTP-модель клієнтських транзакцій. Для подолання цього було розроблено кілька різних стратегій, що дозволяють серверам передавати дані клієнта. Однією з найбільш популярних з цих стратегій була long-pooling. Це реалізовувалось підтримку HTTP-з'єднання відкритим до тих пір, поки у сервера не з'являться дані для передачі клієнту. Проблема всіх цих рішень полягає в тому, що вони несуть накладні витрати HTTP. Кожен раз, коли ви робите HTTP запит, на сервер передається купа заголовків і куки-даних. Це може додати до досить великій кількості даних, які необхідно передати, що, в свою чергу, збільшує затримку.

WebSockets забезпечує постійне з'єднання між клієнтом і сервером, яку обидві сторони можуть використовувати для початку відправки даних в будь-який час. Клієнт встановлює з'єднання WebSocket за допомогою процесу рукостискання. Цей процес починається з відправки клієнтом звичайного HTTP-запиту на сервер. У цьому запиті міститься заголовок Upgrade, який інформує сервер про те, що клієнт бажає встановити WebSocket-з'єднання.

Нижче наведено спрощений приклад заголовків початкового запиту(рис 6.4).

```
GET ws://websocket.example.com/ HTTP/1.1
Origin: http://example.com
Connection: Upgrade
Host: websocket.example.com
Upgrade: websocket
```

Рисунок 6.4 – Заголовки з’єднання WebSocket

Якщо сервер підтримує протокол WebSocket, він погоджується на оновлення і повідомляє про це через заголовок Upgrade у відповіді(рис. 6.5). Тепер, коли рукостискання завершено, початкове HTTP-з'єднання замінено на WebSocket-з'єднання, що використовує те ж саме, що і базове TCP / IP-з'єднання. На цьому етапі будь-яка зі сторін може почати відправку даних.

```
HTTP/1.1 101 WebSocket Protocol Handshake
Date: Wed, 16 Oct 2013 10:07:34 GMT
Connection: Upgrade
Upgrade: WebSocket
```

Рисунок 6.5 – Заголовки з’єднання WebSocket

WebSocket - це фреймовий протокол, що означає, що фрагмент даних (повідомлення) ділиться на кілька дискретних фрагментів, розмір яких закодований у фреймі. Фрейм включає в себе тип фрейму, довжину корисного навантаження і порцію даних. Вигляд кадру наведено в RFC 6455 (рис. 6.6).

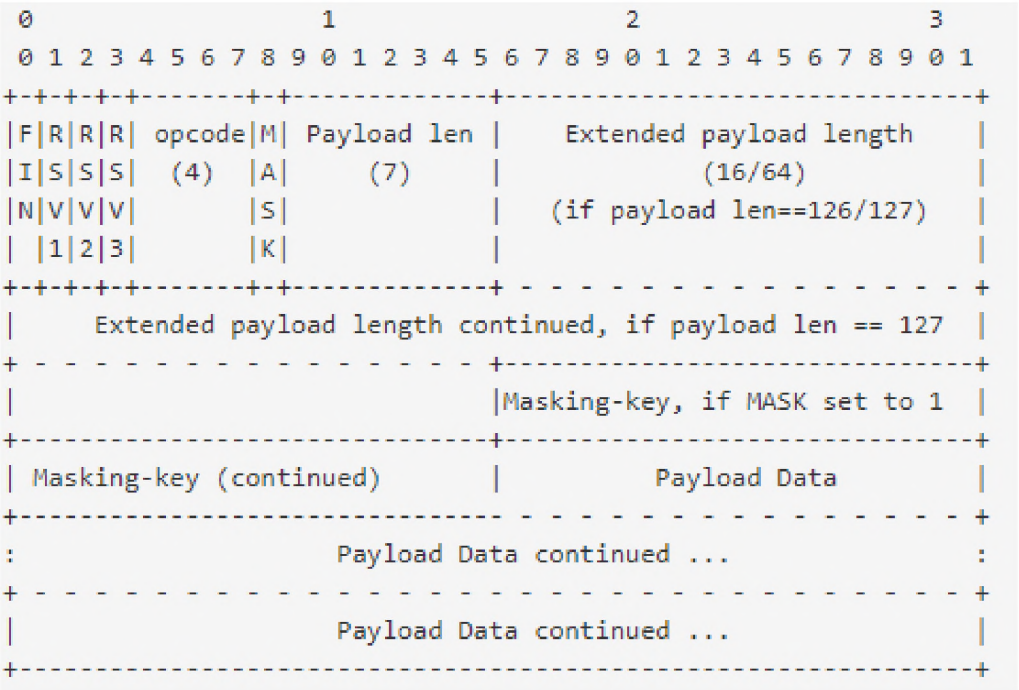

Рисунок 6.6 – Фрейм протоколу WebSocket

Перший біт заголовка WebSocket - це біт Fin. Цей біт встановлюється, якщо даний кадр є останнім даними для заповнення даного повідомлення. Біти RSV1, RSV2,

RSV3 зарезервовані для подальшого використання. Біт маски - установка цього біта в 1 включає маскування. WebSockets вимагає, щоб все корисне навантаження було скрите за допомогою випадкового ключа (маски), обраного клієнтом. Маскуючий ключ комбінується з даними корисного навантаження за допомогою операції XOR перед відправкою даних корисного навантаження. Це маскування запобігає неправильній інтерпретації WebSocket-фреймів як кешованих даних. В ході розробки протоколу WebSocket було показано, що якщо з'являється скомпрометований сервер, а клієнти підключаються до цього сервера, то можлива наявність проміжних проксі-серверів або інфраструктури, кешуючих відповіді скомпрометованого сервера, щоб майбутні клієнти, запитують ці дані, отримували неправильну відповідь. Ця атака називається отруєнням кешу, і є результатом того, що ми не можемо контролювати, як невірно поводяться проксі-сервери в дикій природі. Це особливо проблематично при поданні нового протоколу, такого як WebSocket, який повинен взаємодіяти з існуючою інфраструктурою інтернету.

7 АРХІТЕКТУРА СИСТЕМИ

Виходячи з огляду засобів програмної реалізації та методів розробки була створена модель архітектури для підсистеми індивідуальних сторінок викладачів, що реалізує функціонал поставлений в задачах роботи та відповідає висунутим вимогам. Загальна система складається з головних сервісів та допоміжних. До головних сервісів відносяться:

- Сервіс викладачів;
- Сервіс студентів.

До допоміжних:

- Сервіс розкладу;
- Сервіс аутентифікації;
- Сервіс розподілу навантаження(шлюз);
- Сервіс брокера повідомлень;
- Сервер управління базою даних;
- Сервер управління об’єктним сховищем.

Структурна схема системи показана на рисунку 7.1.

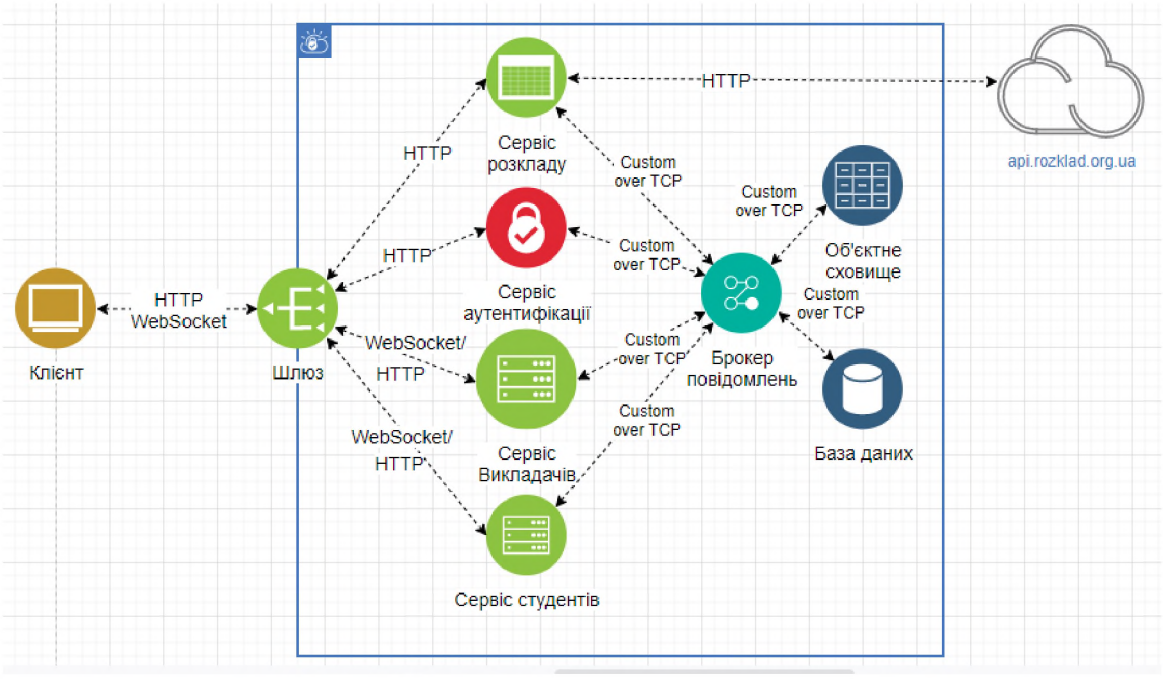

Рисунок 7.1 - Структурна схема системи

7.1 Сервіс викладачів

Даний сервіс є одним із головних і являє собою підсистему, яка є об'єктом даної роботи. Він спілкується з іншими сервісами та обробляє запити, які пов'язані зі персональною сторінкою викладача, такі як:

- Надання інформації про викладача, його розклад, матеріали, предмети, які він веде, групи які з ним пов'язані та список чатів груп;
- Обробка завантажених файлів та перенаправлення їх до об'єктного сховища;
- Синхронізація даних з сервісом розкладу.

Підсистема тісно пов'язана як з підсистемою студентів так і з сервісом аутентифікації та об'єктним сховищем.

7.2 Сервіс автентифікації

Сервіс автентифікації відповідає за реєстрування нових користувачів, створення та оновлення JWT токенів та впровадження ролей.

Даний сервіс надає можливість реєстрації декількох користувачів різних ролей, в залежності від того, яка роль у реєструючого користувача. В системі усніє 4 ролі: «Адміністратор», «Викладач», «Староста» та «Студент». Адміністратор може зареєструвати викладача або старосту. Староста може зареєструвати студентів своїє групи.

Процес реєстрації проходить у два етапи. Перший, коли Адміністратор або Староста заповнює таблицю нових користувачів. Головним полем кожного запису є електронна скринька, так-як на неї прийде запрошення на реєстрацію. Другий етап – студент або викладач отримують електронний лист з посиланням на встановлення паролю. Кожне посилання має в собі унікальний токен, тому кожне посилання однозначно відповідає раніше введеним даним. Після вводу паролю на сторінці встановлення паролю та його підтвердження, в системі створюється користувач з раніше визначеною роллю. Нижче приведена діаграма Use Case реєстрації ново користувача Адміністратором (рис.7.2)

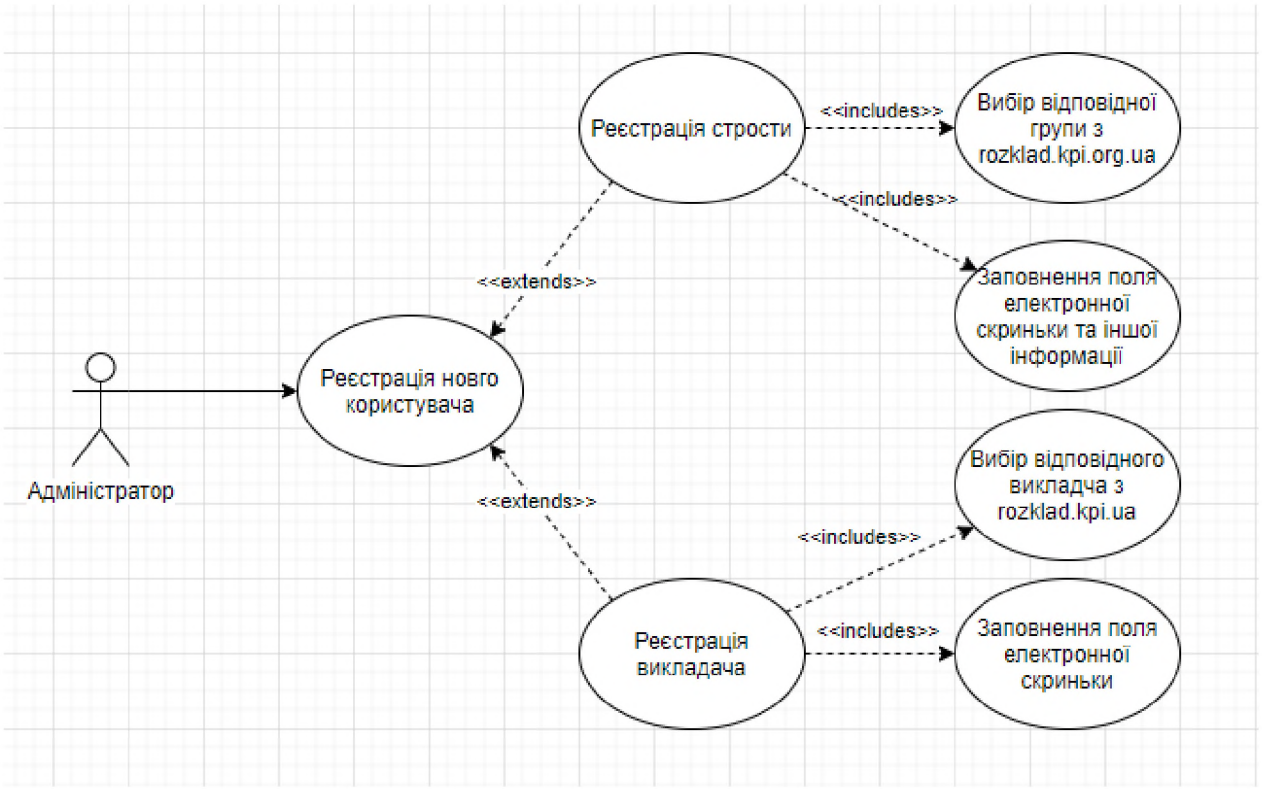

Рисунок 7.2 – Use Case діаграма реєстрації нового користувача Адміністратором

Для розсилки запрошень сервіс підтримує дві платформи надсилання електронних листів: GMAIL та SendGrid.

7.3 Сервер об’єктного сховища

Для роботи з файлами, їх збереження та надання до них швидкого доступу було використано програмне забезпечення MinIO[16]. MinIO – це високопродуктивний програмний пакет MinIO для зберігання об'єктів, який дозволяє клієнтам створювати хмарну інфраструктуру даних для збереження великої кількості файлів, даних для машинного навчання, аналізів та архівів. Дане програмне забезпечення розробляється та поширюється під ліцензією Apache License v2.0[17], що дає можливість використовувати його у даній системі. Головними перевагами є:

- Можливість легкого запуску через образ Docker;
- Інтеграція з Kubernetes;
- Легке налаштування доступу до кінцевих точок API;
- Можливість створення кластеру під управлінням MinIO Operator.

Структурна схема кластеру зображена на рисунку 7.3

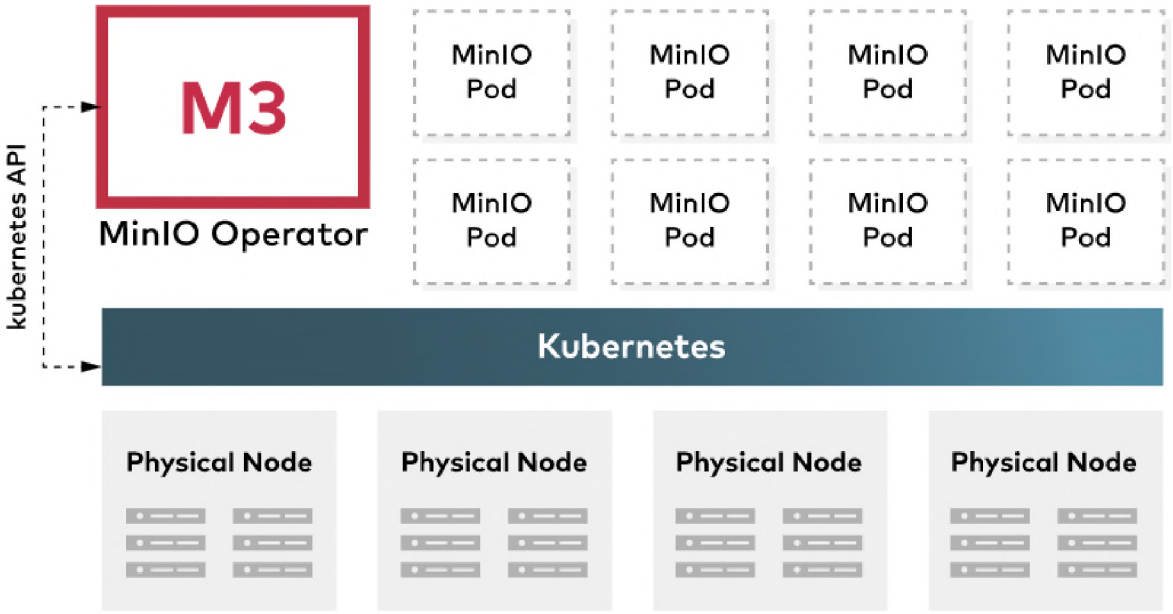

Рисунок 7.3 – Кластер MinIO

8 ПРОГРАМНА РЕАЛІЗАЦІЯ ПІДСИСТЕМИ

Програмна реалізація підсистеми складається з двох компонентів. Бібліотеки класів моделей бізнес-логіки та самого коду підсистеми. Таку структуру було впроваджено через те, що комунікація між мікро-сервісами здійснюється за допомогою сеарелізації моделей об'єктів в JSON формат. Тому для уникнення дуплікації коду проміжних моделей бізнес логіки, було вирішено винести ці моделі в окрему бібліотеку, якою будуть користуватися всі підсистеми.

8.1 Структура проекту

Структура проекту (рис. 8.1) в загалом поділяється на шість основних пакунків:

- Контролери – відповідають за первинну обробку HTTP запитів REST API;
- Репозиторії – відповідають за роботу з реляційною базою даних;
- Сервіси – є проміжним шаром між контролерами та репозиторіями. Реалізують управління об'єктами бізнес-логіки;
- Веб-сокети – пакунок містить класи, що відповідають за повну обробку та управління інформацією передану через протокол WebSocket.
- Повідомлення – пакунок містить клієнти та споживачі повідомлень брокера повідомлень Apache Kafka;
- Утіліти – класи шо мають специфічний функціонал, та використовуються у різних шарах чи рівня підсистеми.

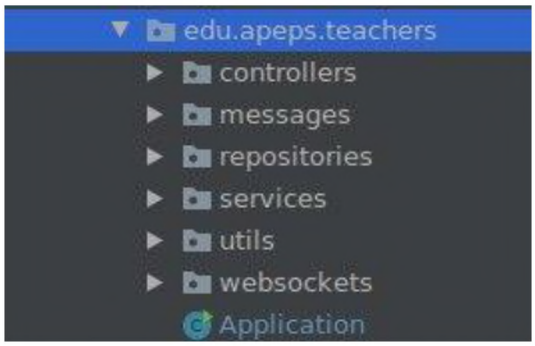

Рисунок 8.1 – Структура проекту

8.2 Бізнес-модель

Бібліотека бізнес-моделей edu.apeps.models містить в собі чотири пакунки на кожну підсистему та сервіс: auth, schedule, teachers та students. Пакунок students

описує бізнес-моделі даної підсистеми.

Зв’язки між класами об’єктів зображені на рисунку 8.2. Також даний пакунок має зв’язок з пакунком auth, так-як в ньому описані об’єкти авторизації користувачів системи, які пов’язані з моделлю студента.

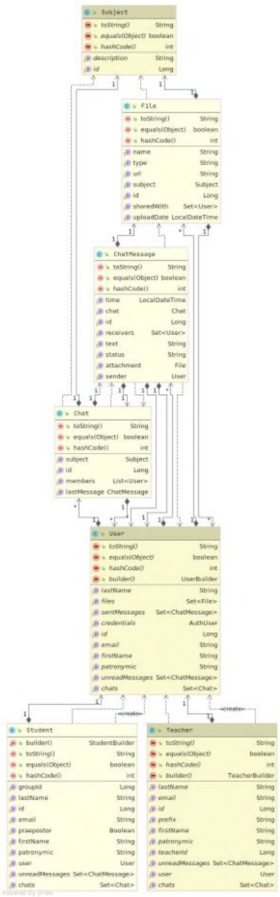

Рисунок 8.2 – UML діаграма класів моделі бізнес логіки

Головні класи пакунку:

- User – містить загальну інформацію про користувача;
- Student – містить інформацію про групу, та чи поле позначення статусу старости. Також має зв’язок один-до-одного з класом User;
- Subject – містить опис об’єкту предмета;
- Chat – містить опис об’єкту чата. Має зв’язки з один-до-одного з Subject, один-до-багатьох з ChatMessage та багато-до-багатьох з User;
- File – містить опис об’єкту файлу;
- ChatMessage – опис об’єкту повідомлення в чаті.

8.3 Структура бази даних

База даних містить в собі 11 таблиць, 8 з яких містять безпосередньо дані об’єктів, та 3 для реалізації зв’язків багато-до-багатьох між ними. Схему бази даних зображено на рисунку 8.3

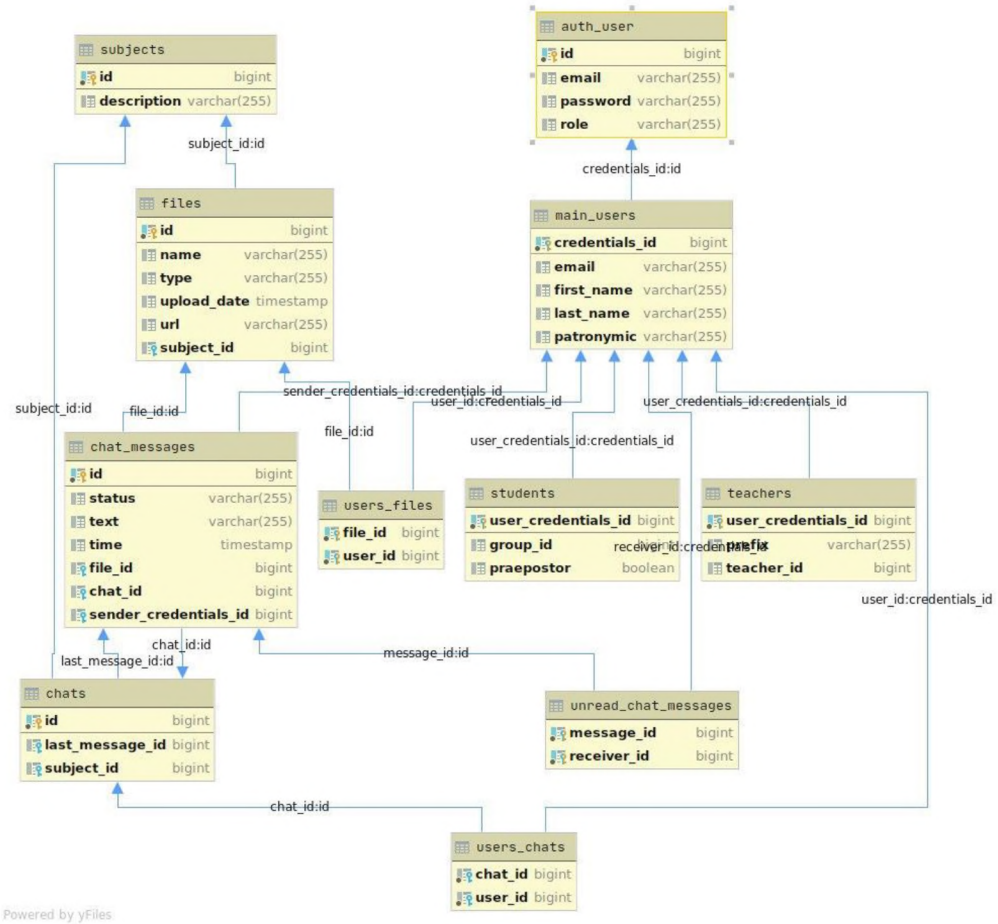

Рисунок 8.3 – Схема public бази даних підсистеми

9 МЕТОДИКА РОБОТИ КОРИСТУВАЧА З ПІДСИСТЕМОЮ

Підсистема представляє собою веб-сайт. Для користувачів, які не автентифіковані в системі можливий доступ лише до сторінки логіну та сторінки встановлення паролю. При будь-який спробах перейти на сторінки, які потребують автентифікації, користувач буде переправлений на сторінку логіну.

Після успішною автентифікації викладач потрапляє на головну сторінку сервісу (рис. 9.1).

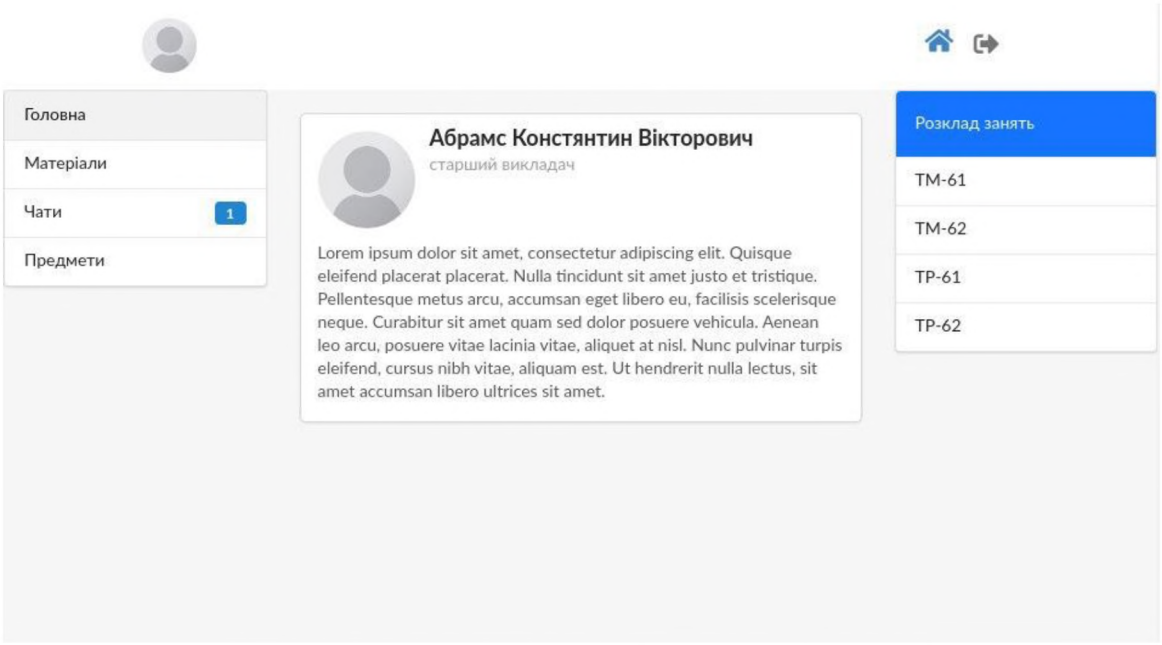

Рисунок 9.1 – Головна сторінка викладача

На головній сторінці викладача, користувачу доступні два меню. Меню з права містить переходи до розкладу занять, список груп, у яких викладач проводить заняття. У меню зліва містить кнопки переходу до модулів Матеріали, Чати та Предмети. На центральній частині модулю Головна розташована інформація про викладача.

При переході на модуль Матеріли, в центральній частині екрану з’являється список матеріалів, які викладач завантажував, згруповані по предметам(рис. 9.2). При натискані на файл, починається процес завантаження його на персональний комп’ютер користувача. При натисканні на іконку контекстного меню, зв’являється можливість переслати файл у чат. Для завантаження файлу достатньо перетягнути

файл до списку матеріалів, після чого з’явиться модульне вікно, в якому потрібно буде вказати, до якого предмету цей файл належить.

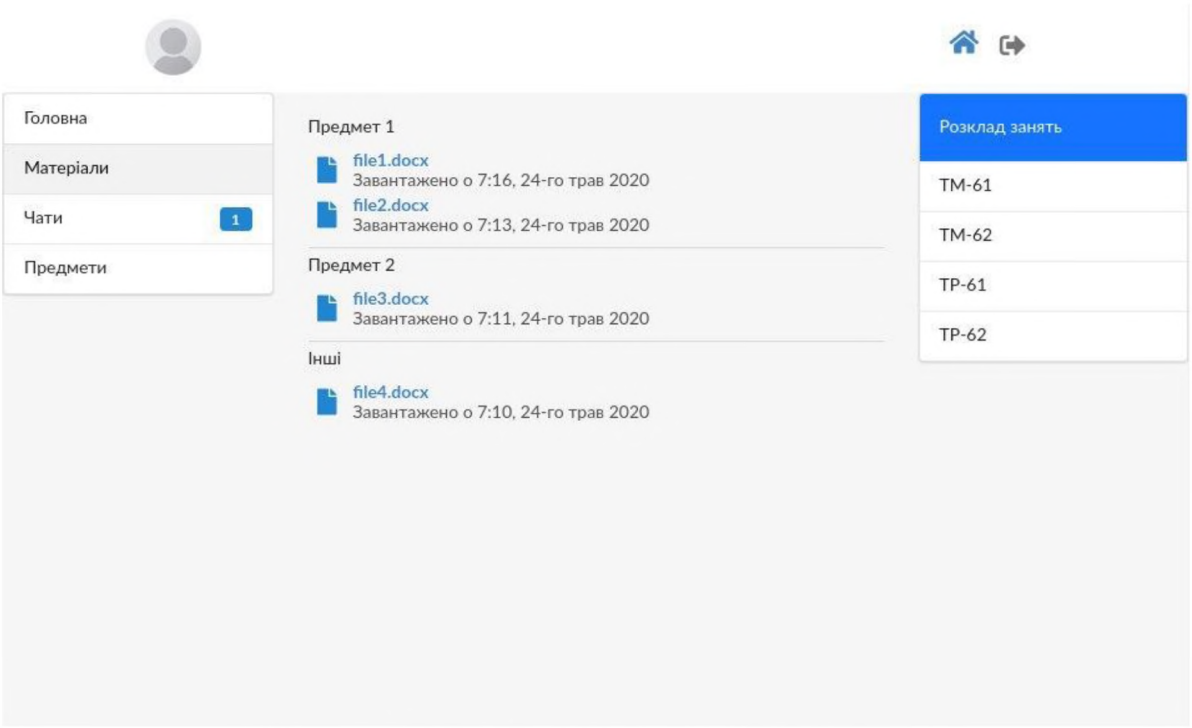

Рисунок 9.2 – Вигляд модулю Матеріали

При переході на модуль Чати (рис. 9.3), в центральній частині екрану з’являється список чатів. Кожна з карток чату містить його назву або назву предмету, до якого відноситься даний час, час останнього повідомлення, відправника цього повідомлення та частину тексту повідомлення. Також на кожній з карток міститься кількість не прочитаних повідомлень користувача.

Модуль Предмети (рис. 9.4)містить в собі список предметів викладача, які він викладає. Кожний елемент списку містить назву предмету, перелік груп та короткий опис предмету. При натисканні на кожен з предметів, користувач переходить до сторінки повного опису предмету.

Також, кожна з підсистем містить шапку, в якій відображається фото користувача, та кнопки додому та виходу. При натисканні на мініатюру фото користувача, користувач переходить до сторінки редактування профілю.

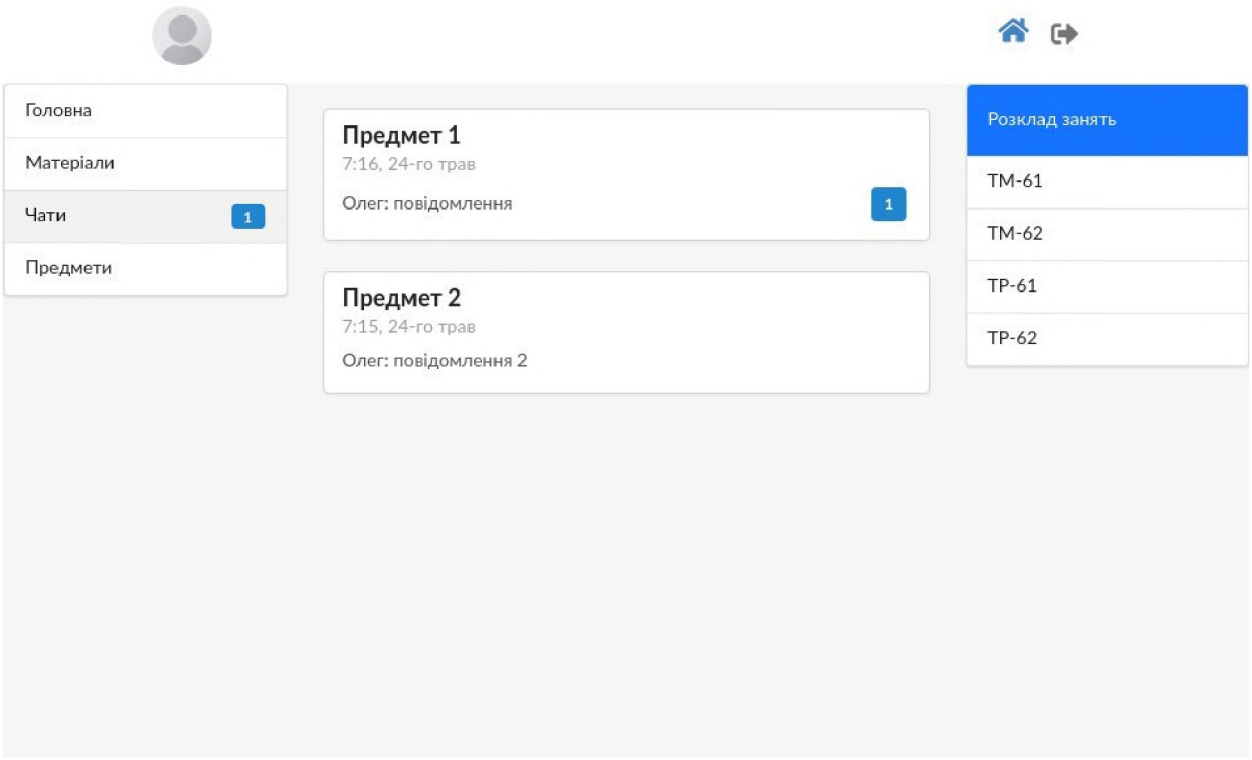

Рисунок 9.3 – Вигляд модулю Чати

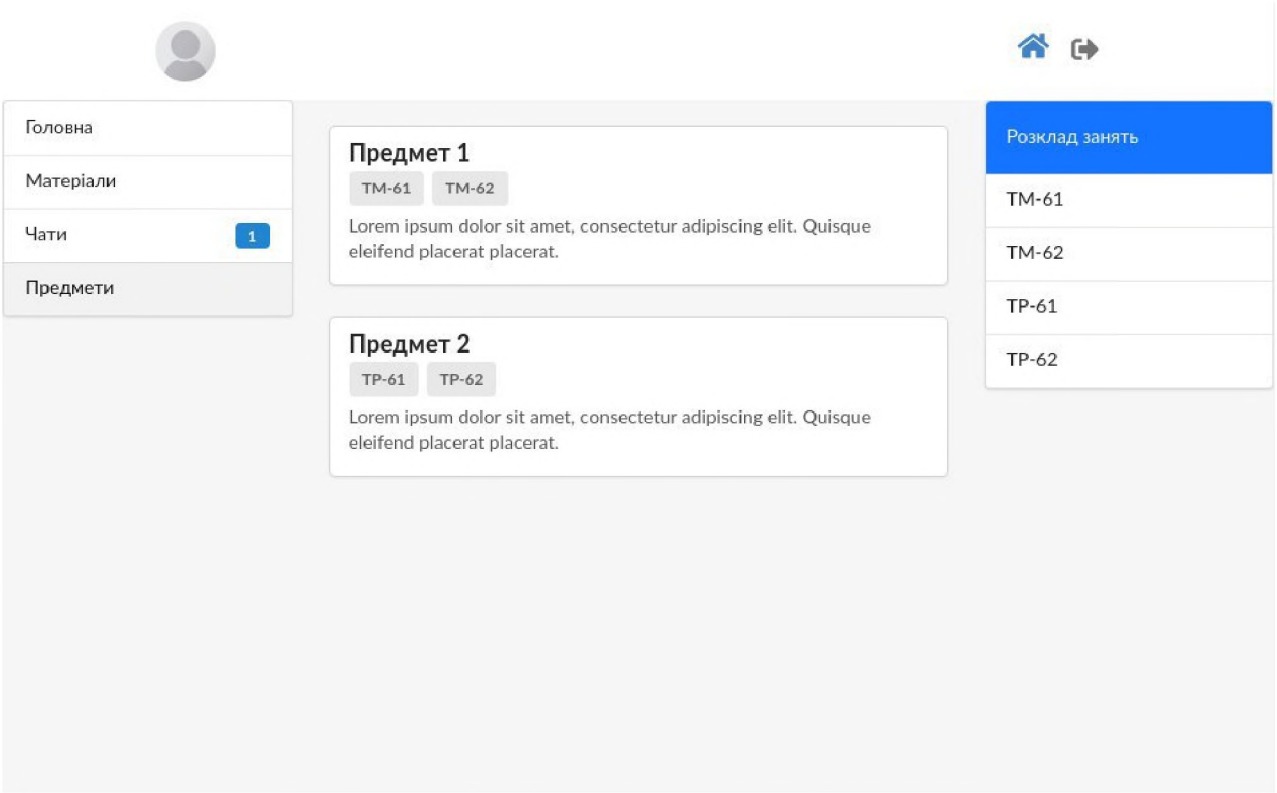

Рисунок 9.4 – Вигляд модулю Предмети

ВИСНОВКИ

Дипломна робота присвячена створенню підсистеми індивідуальних сторінок в загальній системі веб-сайту для підтримки навчального процесу та комунікації між студентами та викладачами. На сьогоднішній день проблема комунікації та взаємодії між викладачами та студентами в умовах дистанційного навчання є як ніколи актуальною. На разі, для вирішення цієї проблеми університет використовує стару систему підтримки навчального процесу, яка не відповідає вимогам які ставить сьогочасна ситуація.

Результатом виконання роботи є опис і створення підсистеми індивідуальних сторінок викладачів. При виконанні роботи були виконані всі завдання та більшість вимог, поставлених на початку роботи над системою.

Були створені моделі бізнес логіки та зв'язків між ними, які відповідають області роботи підсистеми, описаної в даній роботі. Головними моделями є класи викладача, предмету, чату, повідомлень та файлових матеріалів.

Були створені однозначні відображення моделей підсистеми з моделями системи розкладу занять НТУУ «КПІ» ім. І. Сікорського, такі як урок, викладач та предмет. Також була розроблений сервіс синхронізації даних даної підсистеми з електронним розкладом занять НТУУ «КПІ» ім. І. Сікорського.

Був створений функціонал, який задовольняє сучасні потреби до підтримки навчального процесу: відображення розкладу занять, можливість завантажувати матеріал та можливість комунікації зі студентами.

Був розроблений зручний графічний інтерфейс користувача, з акцентом на максимально інтуїтивне розташування компонентів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Електронна система Moodle [Електронний ресурс] – Режим доступу: https://moodle.org/.
2. Електронний кампус КПІ ім. І. Сікорського [Електронний ресурс] – Режим доступу: https://ecampus.kpi.ua.
3. The GNU Operating System [Електронний ресурс] – Режим доступу: https://www.gnu.org.ua/.
4. Java [Електронний ресурс] – Режим доступу: https://www.java.com/.
5. Spring Framework [Електронний ресурс] – Режим доступу: https://spring.io//.
6. Micronaut [Електронний ресурс] – Режим доступу: https://micronaut.io/.
7. Spock Framework [Електронний ресурс] – Режим доступу: http://spockframework.org/.
8. JavaScript language [Електронний ресурс] – Режим доступу: https://www.javascript.com/.
9. React.js [Електронний ресурс] – Режим доступу: https://reactjs.org/.
10. Брокер повідомлень Apache Kafka [Електронний ресурс] – Режим доступу: https://kafka.apache.org/.
11. Grails [Електронний ресурс] – Режим доступу: https://grails.org/.
12. Server Netty [Електронний ресурс] – Режим доступу: https://netty.io/.
13. MIME Specification RFC 1521 [Електронний ресурс] – Режим доступу: https://tools.ietf.org/html/rfc1521/.
14. MIME Specification Part 2 RFC 1522 [Електронний ресурс] – Режим доступу: https://tools.ietf.org/html/rfc1522/.
15. The WebSocket Protocol [Електронний ресурс] – Режим доступу: https://tools.ietf.org/html/rfc6455.
16. MinIO [Електронний ресурс] – Режим доступу: https://min.io/.
17. APACHE LICENSE, VERSION 2.0 [Електронний ресурс] – Режим доступу: https://www.apache.org/licenses/LICENSE-2.0.
18. Електронна система e-APEPS [Електронний ресурс] – Режим доступу: https://e-

apeps.ml/.

19. Erik Wilde, Cesare Pautasso. REST: From Research to Practice. — Springer Science & Business Media, 2011. — 528 p

20. The WebSocket Protocol [Електронний ресурс] – Режим доступу: https://tools.ietf.org/html/rfc6455.

ДОДАТОК А

Розробка системи взаємодії індивідуальних сторінок викладачів та студентів.
Підсистема: індивідуальні сторінки викладачів

Специфікація

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТІ6159_20Б

Аркушів 1

Київ 2020

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТІ6159_20 Б ПЗ	Записка.docx	Текстова частина дипломної роботи
Комплекс		
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТІ6159_20 Б	teachers	Підсистема індивідуальних сторінок викладачів. Розроблена у даній роботі
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТІ6159_20 Б	students	Підсистема індивідуальних сторінок студентів.
Компоненти		
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТІ6159_20 Б К1	client	Основна папка, яка містить файли для клієнта.
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТІ6159_20 Б К2	students	Основна папка, яка містить файли для сервісу студента.
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТІ6159_20 Б К4	teachers	Основна папка, яка містить файли для сервісу викладача.
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТІ6159_20 Б К5	schedule	Основна папка, яка містить файли для сервісу розкладу.
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТІ6159_20 Б К6	auth	Основна папка, яка містить файли для сервісу розкладу.

ДОДАТОК Б

Розробка системи взаємодії індивідуальних сторінок викладачів та студентів.
Підсистема: індивідуальні сторінки викладачів

Текст програми

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТІ6159_20Б

Аркушів 4

Київ 2020

```
package edu.apeps.main.controllers;

import edu.apeps.main.services.UserService;
import edu.apeps.models.general.Student;
import edu.apeps.models.general.Teacher;
import edu.apeps.main.repositories.*;
import edu.apeps.models.general.User;
import io.micronaut.http.MediaType;
import io.micronaut.http.annotation.Controller;
import io.micronaut.http.annotation.Get;
import io.micronaut.http.annotation.Put;
import io.micronaut.security.annotation.Secured;
import io.reactivex.Maybe;

import javax.inject.Inject;
import javax.inject.Singleton;
import java.security.Principal;

@Controller("/user")
@Secured("isAuthenticated()")
public class UserController {
    @Inject
    UserService userService;


    @Get("/")
    public Maybe<User> index() {
        return userService.getUser();
    }

    @Get("/student")
    @Secured({"STUDENT","PRAEPOSTOR"})
    public Maybe<Student> student() {
        return userService.getStudent();
    }

    @Get("/teacher")
    @Secured({"TEACHER"})
    public Maybe<Teacher> teacher() {
        return userService.getTeacher();
    }

    @Put(uri = "/student", consumes = {MediaType.APPLICATION_JSON})
    @Secured({"STUDENT","PRAEPOSTOR"})
    public Maybe<Student> editStudent(Student student) {
        return userService.editStudent(student);
    }

    @Put(uri = "/teacher", consumes = {MediaType.APPLICATION_JSON})
    @Secured({"TEACHER"})
    public Maybe<Teacher> editTeacher(Teacher teacher) {
        return userService.editTeacher(teacher);
```

```
    }
}

package edu.apeps.main.controllers;


import edu.apeps.main.services.ChatService;
import edu.apeps.main.services.SubjectService;
import edu.apeps.models.general.Chat;
import edu.apeps.models.general.Subject;
import io.micronaut.http.annotation.*;
import io.micronaut.security.annotation.Secured;
import io.reactivex.Flowable;
import io.reactivex.Maybe;
import org.reactivestreams.Publisher;

import javax.annotation.Nullable;
import javax.inject.Inject;
import java.util.List;

@Controller("/chats")
@Secured("isAuthenticated()")
public class ChatController {
    @Inject
    ChatService chatService;

    @Get("/{?ids}")
    @Secured({"STUDENT","PRAEPOSTOR","TEACHER"})
    public Publisher<Chat> get(@Nullable @QueryValue("ids") List<Long>
ids) {
        assert ids != null;
        return Flowable.fromIterable(ids).flatMapMaybe(id ->
chatService.get(id));
    }

    @Put("/")
    @Secured({"TEACHER"})
    public Maybe<Chat> update(Chat chat) {
        return chatService.update(chat);
    }

    @Post("/")
    @Secured({"TEACHER"})
    public Maybe<Chat> create(Chat chat) {
        return  chatService.update(chat);
    }
}

package edu.apeps.main.websockets;

import edu.apeps.main.caches.ChannelCache;
import edu.apeps.main.services.MessageService;
```

```
import edu.apeps.models.general.ChatMessage;
import io.micronaut.websocket.WebSocketSession;
import io.reactivex.Maybe;

import javax.inject.Inject;
import javax.inject.Singleton;
import java.security.Principal;
import java.util.HashSet;
import java.util.stream.Collectors;

@Singleton
public class WebSocketService {


    @Inject
    private ChannelCache channelCache;

    public Maybe<Channel> getChannel(Long id, ChatMessage message,
WebSocketSession session){
        return channelCache.getFromCache(message.getChat())
                .defaultIfEmpty(
                        Channel.builder()
                                .id(message.getChat())
                                .sessionId( new HashSet<>(){
                                                {

add(session.getId());
                                                }
                                            }
                                )
                                .users( new HashSet<>(){
                                                {

add(message.getSender().getId());
                                                }
                                            }
                                )
                                .build()
                )
                .filter(any -> id.equals(message.getSender().getId()))
                .flatMap( channel -> {
                    var openSessionIds =
session.getOpenSessions().stream()
                            .map(WebSocketSession::getId)
                            .collect(Collectors.toSet());
                    channel.getSessionId().add(session.getId());

                    channel.setSessionId(
                            channel.getSessionId().stream()
                            .filter(openSessionIds::contains)
                            .collect(Collectors.toSet())
                    );
```

```
channel.getUsers().add(message.getSender().getId());
                    return
channelCache.putIntoCache(channel.getId(),channel);
                });
    }
}
```

```
package edu.apeps.models.general;

import com.fasterxml.jackson.annotation.JsonIgnore;
import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import io.micronaut.core.annotation.Introspected;
import lombok.*;
import lombok.experimental.Delegate;

import javax.persistence.*;

@Getter @Setter @RequiredArgsConstructor
@ToString(doNotUseGetters = true)
@EqualsAndHashCode(doNotUseGetters = true)
@AllArgsConstructor
@Builder
@JsonIgnoreProperties(value = { "user",
"files","sentMessages","credentials"})
public class Teacher{
    private Long id;
    private Long teacherId;
    private String prefix;

    @Delegate(excludes = User.UserExclude.class)
    @Builder.Default
    private User user = new User();
}
```

ДОДАТОК В

Розробка системи взаємодії індивідуальних сторінок викладачів та студентів.
Підсистема: індивідуальні сторінки викладачів

Опис програмного модулю

УКР.НТУУ"КПІ"_ТЕФ_АПЕПС_ТІ6159_20Б

Аркушів 7

Київ 2020

АННОТАЦІЯ

Даний додаток містить опис мікросервіс системи взаємодії індивідуальних сторінок викладачів та студентів, підсистеми індивідуальних сторінкок викладачів. Розроблено на базі фреймворку Micronaut для надання можливості комунікації викладачів зі студентами та підтримки навчального процесу. Створено веб-сервіс мікросервісного типу. Сервіс виконує наступні завдання:

- Надання розкладу;
- Редагування опису предметів;
- Можливість спілкування у чаті групи;
- Надсилання результату обрахунків на клієнт.

При розробці веб-сервісу використовувався мова програмування Java, фреймворк Micronaut та бібліотека React.js (на мові програмування Javascript).

ЗМІСТ

ЗАГАЛЬНІ ВІДОМОСТІ

У цьому додатку міститься опис веб-сервісу з підсистемою індивідуальних сторінок викладачів системи e-APEPS. У додатку Б міститься програмний код головних модулів розробленого серверу. Веб-сервіс виконано для подальшої інтеграції на серверну віддалену машину. Для коректної роботи серверу необхідно встановити всі залежні модулі на сервер.

ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Розроблений мікросервіс виконує завдання надання розкладу, редагування опису предметів, можливість спілкування у чаті групи, надсилання результату обрахунків на клієнт. Розроблений сервіс може використовуватись в якості системи підтримки навчання, надання учбових матеріалів та комунікації між студентами та викладачами. Функціональних обмеження на використання веб-сервісу полягає лише в його апаратному забезпеченні.

ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Вся система поділяється на 8 компонентів: сервіс викладачів, сервіс студентів, сервіс розкладу, сервіс аутентифікації, сервер брокера повідомлень, сервер об’єктного сховища, сервер управління базою даних та розподільник навантаження. Діпсистема що розроблялася у даному дипломі відповідає за обробку інформації пов’язаною з функціоналом сторінки викладачів.

ВИКОРИСТОВУВАНІ ТЕХНІЧНІ ЗАСОБИ

Для запуску та тестуванні системи у наближених до реальних умовах було використано сервіси від Google Cloud Platform. Завдяки можливостям фреймворку Micronaut, було розгорнуто і впроваджено систему повністю з мінімальними налаштуваннями.

Розроблений веб-сервіс працює на будь-якій платформі у будь-якому браузері. У подальшому необхідно підняти веб-сервіс на віддаленому сервері для його стабільної роботи та доступу у мережі інтернет.

ВИКЛИК І ЗАВАНТАЖЕННЯ

Для запуску всіх сервісів потрібно інсталювати наступне програмне забезпечення: docker, nginx та npm. Для запуск серверу клієнта та сервісів автентифікації, розкладу, викладачів та студентів необхідно виконати команду "./docker-build.sh" у кожному з каталогів з вихідним кодом сервісів.

Також з головної директорії проекту необхідно перенести конфігураційний файл для розподільника навантаження nginx.

ВХІДНІ І ВИХІДНІ ДАНІ

Вхідними даними підсистеми є інформації про викладачів, матеріали та файли, повідомлення та інша інформація. Так-як система займається обробкою та збереженням даних, вихідні дані в більшості збігаються з вхідними і невелику кількість додаткової інформації.