Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

Навчально-науковий інститут комп'ютерних наук
та інформаційних технологій
(повне найменування інституту, назва факультету (відділення))

Кафедра комп'ютерних наук
(повна назва кафедри (предметної, циклової комісії))

Магістерська кваліфікаційна робота

другий (магістерський)
(рівень вищої освіти)

на тему: *«Розроблення бекенд частини вебпорталу навчально-наукового інституту»*

Виконав: студент 6 курсу групи КН-63м
спеціальності
122 "Комп'ютерні науки"
(шифр і назва напряму підготовки, спеціальності)

Верцімага Ю. О.
(прізвище та ініціали)

Керівник *Процик Ю. С.*
(прізвище та ініціали)

Рецензент Флуд Л. О.
(прізвище та ініціали)

Львів – 2024

Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

ННІ комп'ютерних наук та інформаційних технологій
Кафедра комп'ютерних наук
Рівень вищої освіти другий (магістерський)
Спеціальність 122 "Комп'ютерні науки"
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Борецька І.Б.
"05" січня 2024 року

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Верцімазі Юрію Олександровичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Розроблення бекенд частини вебпорталу навчально-наукового інституту*

керівник роботи *Процик Юрій Степанович, к.ф.-м.н.*,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від "*13*" *лютого 2023 року* № *С-49*

2. Термін подання студентом роботи *"05" січня 2024 року*

3. Вихідні дані до роботи *Аналіз шляхів вирішення задачі, організаційна структура системи*

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
Вступ. Розділ 1. Стан проблемної області.
Розділ 2. Інформаційне забезпечення.
Розділ 3. Математичне забезпечення.
Розділ 4. Програмне забезпечення.
Розділ 5. Розроблення стартап-проєкту.
Висновки. Список використаної літератури. Додатки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Слайди для доповіді (підготовка матеріалу для доповіді загальним обсягом 10-12 слайдів)

6. Дата видачі завдання *"15" лютого 2023 року*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1.	Огляд літератури згідно досліджуваної теми. Збір необхідних матеріалів	*15.02.2023 р.* *23.03.2023 р.*	Виконав
2.	Постановка задачі і її формалізація	*24.03.2023 р.* *01.05.2023 р.*	Виконав
3.	Виконання досліджень	*02.05.2023 р.* *19.06.2023 р.*	Виконав
4.	Реалізація архітектури проєкту	*20.06.2023 р.* *04.09.2023 р*	Виконав
5.	Реалізація бекенд частини вебпорталу навчально-наукового інституту	*05.09.2023 р.* *27.11.2023 р.*	Виконав
6.	Оформлення пояснювальної записки	*28.11.2023 р.* *05.01.2024 р.*	Виконав

Студент _____ Вецімага Ю. О.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Процик Ю. С.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дипломна робота містить 63 сторінки пояснювальної записки, 15 рисунків, 4 додатків, 13 джерел.

У даній роботі була розроблена бекенд частина вебпорталу навчально-наукового інституту. Реалізовані функціональні можливості включають обробку запитів, валідацію даних, роботу з базою даних та генерацію документації API.

Для розробки були використані наступні технології та бібліотеки: C# та .NET Core 6.0, Visual Studio, AutoMapper, FluentValidation, MediatR, EF Core та Swagger. C# та .NET Core 6.0 були використані як основна мова програмування та фреймворк відповідно.

Ключові слова: *C#, .NET Core 6.0, Visual Studio, AutoMapper, FluentValidation, MediatR, EF Core, Swagger, бекенд, вебпортал, навчально-науковий інститут*.

ABSTRACT

The thesis contains 63 pages of explanatory note, 15 figures, 4 appendices, 13 used literary sources.

In the context of this project, was developed backend part of a web portal for an educational and scientific institute was developed. The implemented functional capabilities include request processing, data validation, database operations, and API documentation generation.

The following technologies and libraries were used for development: C# and .NET Core 6.0, Visual Studio, AutoMapper, FluentValidation, MediatR, EF Core, and Swagger. C# and .NET Core 6.0 were used as the primary programming language and framework, respectively.

Keywords: *C#, .NET Core 6.0, Visual Studio, AutoMapper, FluentValidation, MediatR, EF Core, Swagger, backend, web portal, educational, scientific institute*.

ТЕХНІЧНЕ ЗАВДАННЯ

Необхідно розробити бекенд частину вебпорталу для навчально-наукового інституту, а саме:

1. реалізувати мікросервісну архітектуру для підвищення гнучкості та масштабованості вебпорталу;
2. використати мову програмування C# і фреймворк .NET Core 6.0 для бекенду;
3. інтегруватися з базою даних PostgreSQL за допомогою Entity Framework Core для ефективної роботи з даними;
4. впровадити системи автентифікації та авторизації для забезпечення безпеки;
5. використати додаткові сервіси у Docker, включаючи Seq, Portainer, та PgAdmin для моніторингу, управління контейнерами та базою даних відповідно;
6. розробити механізми обробки запитів, валідації даних та генерації документації API;
7. протестувати функціонал для забезпечення стабільності та відповідності вимогам.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

ПЗ – Програмне забезпечення

ОС – Операційна система

GUI – Graphical User Interface

ВСТУП

В даній роботі розглядається розробка бекенд частини вебпорталу навчально-наукового інституту з використанням сучасних технологій та бібліотек. Основною метою є забезпечення ефективної обробки запитів, валідації даних, роботи з базою даних та генерації документації API. Результати дослідження та розробки можуть мати практичне значення для навчальних закладів, сприяючи поліпшенню процесів навчання та обміну знаннями.

Актуальність теми. В сучасному інформаційному суспільстві розвиток технологій та інформаційних систем набуває все більшого значення в різних сферах діяльності. Особливу увагу приділяється розробці та вдосконаленню вебпорталів, які є важливим інструментом для забезпечення доступу до інформації та послуг. Одним із сфер, де вебпортали мають великий потенціал, є навчально-науковий інститут, де створення ефективної та функціональної веб-платформи може сприяти поліпшенню процесів навчання, обміну знаннями та співпраці.

Об’єктом дослідження є кінцевий продукт, що складається з великої кількості функціоналу і процесів, які забезпечують його роботу. Найчастіше автоматизують:

- Back-end процеси та сервіси
- API
- Запис логів
- Роботу баз даних
- Опції та інструменти сайту, додатку, ресурсу, що часто використовуються: форми реєстрації, системи онлайн-оплат і т.п.
- Автоматизацію заповнення полів, збереження і перевірку
- Валідацію даних, що вводяться
- Зберігання та цілісність даних

Предметом дослідження є інформаційний вебпортал та його функціональність, якою користуються студенти для ознайомлення з програмою курсу, або з особовим складом факультету, або кафедри інституту. Ця інформація є ключовою при виборі вищого навчального закладу студеном при процесі вступу.

Метою є створення функціональної та надійної бекенд частини вебпорталу навчально-наукового інституту, яка відповідатиме сучасним вимогам та стандартам розробки. Це дозволить забезпечити зручний та ефективний доступ до інформації, спростити процеси взаємодії користувачів з порталом, покращити адміністрування та підтримку системи. Використання передових технологій та бібліотек для розробки бекенд частини вебпорталу. Це сприятиме підвищенню продуктивності, швидкодії та масштабованості системи, а також забезпечить високу безпеку та захист від потенційних загроз. Поставлені завдання передбачають створення гнучкого та розширюваного рішення, яке відповідає потребам навчально-наукового інституту та його користувачів.

Основні завдання, що мають бути розроблені у роботі:

- Охарактеризувати об'єкт дослідження, яким є бекенд частина вебпорталу навчально-наукового інституту, і сформулювати задачі, що мають бути розроблені, включаючи обробку запитів, валідацію даних, роботу з базою даних та генерацію документації API.
- Розробка системи обробки запитів: Це включає створення логіки обробки різноманітних запитів, включаючи отримання, створення, оновлення та видалення даних. Розробка ефективного та оптимізованого алгоритму обробки запитів є важливою для забезпечення швидкодії та надійності системи.
- Валідація даних: Для забезпечення вірності та цілісності даних, розробка включає валідацію вхідних даних згідно з встановленими правилами та обмеженнями. Це допомагає уникнути помилок та забезпечити правильну

обробку користувацьких запитів.

- Робота з базою даних: Це включає взаємодію з базою даних для збереження, отримання та оновлення інформації. Розробка механізмів доступу до бази даних, оптимізація запитів та забезпечення безпеки даних є важливими аспектами роботи.
- Генерація документації API: Розробка вебпорталу включає створення документації API, яка дозволяє користувачам легко розуміти та використовувати доступні функції та ендпоінти. Генерація автоматичної документації API спрощує процес розробки та забезпечує зрозумілу комунікацію між розробниками та користувачами.
- Тестування та забезпечення якості: Розробка вимагає проведення тестування для перевірки коректності та надійності розроблених функціональностей. Це включає тестування одиниць коду, інтеграційне тестування та прийомочне тестування для забезпечення якості роботи вебпорталу.

Новизна роботи. Новизною даної роботи є впровадження сучасних технологій та бібліотек для розробки бекенд частини вебпорталу навчально-наукового інституту. Результати дослідження та розробки можуть стати цінним додатком до існуючих знань у галузі веб-розробки та сприяти поліпшенню процесів навчання та обміну знаннями.

Практичне значення одержаних результатів. Одержані результати дослідження та розробки можуть мати практичне значення для навчально-наукових інститутів, які прагнуть розвивати свої вебпортали. Розроблений бекенд може забезпечити ефективну та надійну роботу вебпорталу, поліпшити доступ до інформації та послуг, сприяти автоматизації процесів та покращенню користувацького досвіду.

РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

1.1 Актуальність предметної області

В сучасному освітньому середовищі зростає необхідність у зручному та ефективному доступі до навчальних ресурсів, сприянні організації та обміну інформацією, а також у покращенні ефективності управління навчальним процесом.

Окрім цього, швидкий технологічний прогрес та постійні зміни вимог користувачів ставлять перед навчально-науковими інститутами завдання створення сучасного, функціонального та надійного бекенду для вебпорталу. Цей бекенд повинен забезпечувати широкий функціонал, включаючи обробку запитів, валідацію даних, роботу з базою даних та генерацію документації API.

Таким чином, актуальність даної предметної області полягає в необхідності створення сучасних та ефективних рішень для покращення якості навчання, наукових досліджень та управління в навчально-наукових інститутах. Розробка оптимального бекенду для вебпорталу стає важливим завданням, спрямованим на задоволення потреб користувачів та досягнення максимальної ефективності управління навчальним процесом.

1.2 Аналіз предметної області

Аналіз предметної області, пов'язаної з розробкою бекенд частини вебпорталу для навчально-наукового інституту, дозволяє отримати глибоке розуміння основних вимог та потреб користувачів, а також виявити ключові аспекти, що впливають на успішну реалізацію проекту.

Під час аналізу було проведено дослідження існуючих вебпорталів, що використовуються в навчально-наукових інститутах, а також вивчено актуальні тренди та рекомендації в сфері освіти та наукових досліджень. Основні аспекти,

що були враховані в аналізі, включають функціональність, безпеку, швидкодію, масштабованість та зручність використання.

Результати аналізу предметної області показали, що вебпортали для навчально-наукових інститутів повинні мати широкий спектр функціональних можливостей, таких як керування навчальними курсами, студентські облікові записи, електронний доступ до навчальних матеріалів та бібліотек, спільна робота між студентами та викладачами тощо. Безпека є одним із ключових аспектів, оскільки необхідно забезпечити захист персональних даних студентів та викладачів, а також захист від потенційних кібератак.

Швидкодія та масштабованість є важливими факторами, оскільки вебпортал повинен бути здатний обробляти великий потік запитів та забезпечувати стабільну роботу навіть при великому навантаженні. Зручність використання та інтуїтивний інтерфейс також мають велике значення, оскільки це сприяє залученню користувачів та покращує загальний досвід використання вебпорталу.

Аналіз предметної області дав цінні висновки та настанови для подальшої розробки бекенд частини вебпорталу для навчально-наукового інституту. Засновані на ньому рекомендації та вимоги враховані під час розробки та впровадження системи, що сприятиме досягненню мети проекту та задоволенню потреб користувачів.

1.3 Аналіз різних підходів до побудови архітектури вебпорталів

Аналіз різних підходів до побудови архітектури вебпорталів підтвердив переваги та потенціал мікросервісної архітектури [8] для розробки бекенд частини навчально-наукового вебпорталу.

Мікросервісна архітектура базується на розбитті додатку на невеликі та автономні сервіси, які працюють разом для надання комплексного функціоналу. Цей підхід дозволяє досягти більшої гнучкості та масштабованості системи.

Кожен сервіс може бути розроблений, розгорнутий та масштабований незалежно, що спрощує управління та розвиток системи.

Під час аналізу було виявлено, що мікросервісна архітектура забезпечує модульність та розширюваність системи. Кожен сервіс може виконувати конкретну функцію, таку як обробка запитів, управління користувачами, робота з базою даних тощо. Це сприяє збільшенню швидкодії розробки та підтримки системи, а також полегшує розгортання нових функцій та оновлень.

Крім того, мікросервісна архітектура дозволяє підтримувати високу доступність системи. Завдяки незалежному функціонуванню сервісів, можлива резервна робота окремих компонентів без перебоїв в роботі системи в цілому.

Таким чином, аналіз різних підходів до побудови архітектури веб-порталів підтвердив, що мікросервісна архітектура є оптимальним вибором для розробки бекенд частини навчально-наукового веб-порталу, забезпечуючи гнучкість, масштабованість, модульність та високу доступність системи.

Різниця між монолітом та мікросервісами зображено на рисунку 1.1.

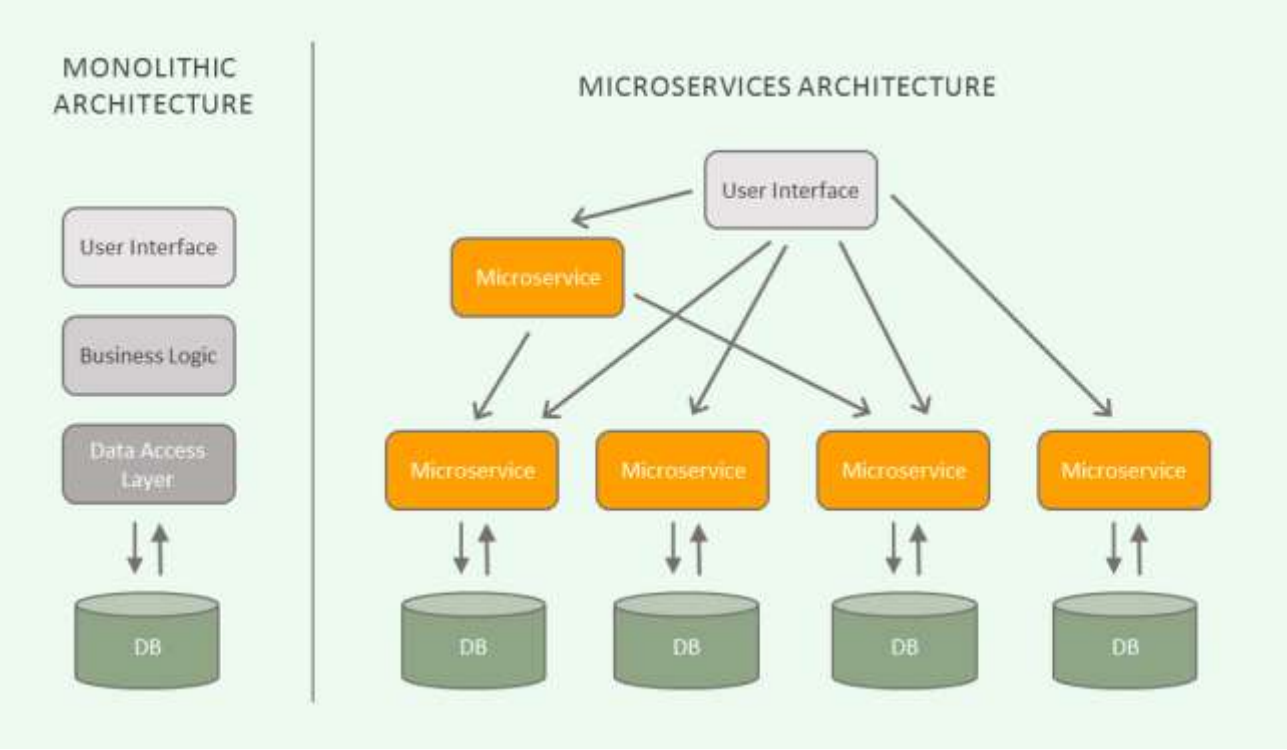

Рисунок 1.1 – Моноліт та мікросервіси

1.4 Постановка завдання

Завданням магістерської роботи є реалізувати систему вебпорталу, основних функцій та інтерфейсу. Основною метою є створення функціональності, яка забезпечить ефективну роботу вебпорталу та задовольнить потреби користувачів. Головні завдання, які потрібно вирішити в рамках постановки завдання, включають наступне:

- Розробка механізму обробки запитів: реалізація системи, яка здатна приймати та обробляти запити від користувачів, забезпечуючи правильну обробку та передачу даних між клієнтом і сервером.
- Валідація даних: реалізація механізму, який забезпечить перевірку та валідацію вхідних даних, щоб запобігти некоректним або неправильним даним, які можуть призвести до проблем в системі.
- Робота з базою даних: розробка моделей даних та взаємодія з базою даних для збереження та отримання інформації, необхідної для роботи вебпорталу.
- Генерація документації API: розробка механізму автоматичної генерації документації API, що дозволить забезпечити чітку та зрозумілу документацію для розробників та користувачів системи.
- Тестування та налагодження: проведення тестування розроблених функцій та виявлення та виправлення можливих помилок та проблем.
- Розробка системи автентифікації і авторизації: реалізація механізму, що забезпечує безпеку та контроль доступу до ресурсів вебпорталу. Це включає ідентифікацію користувачів, перевірку їх прав доступу та захист конфіденційної інформації.
- Розробка API: створення набору програмних інтерфейсів (API) [10] для забезпечення взаємодії між бекендом і фронтендом вебпорталу. Це дозволяє інтегрувати різноманітні функції та можливості, які надає бекенд,

в інтерфейс користувача.

- Оптимізація продуктивності: розробка та впровадження стратегій та методів для покращення продуктивності вебпорталу, таких як кешування, оптимізація запитів до бази даних та шкалювання системи для обробки великого обсягу запитів.
- Забезпечення безпеки: впровадження механізмів захисту даних, виявлення та запобігання атакам, шифрування комунікації та інших заходів для забезпечення безпеки вебпорталу та його користувачів.
- Підтримка розширюваності: розробка архітектурного рішення, яке дозволить легко розширювати функціональність вебпорталу шляхом додавання нових модулів, компонентів або інтеграції з іншими системами.

Постановка завдання є важливим етапом у розробці бекенд частини вебпорталу, оскільки вона визначає основні напрямки та цілі роботи, які потрібно досягти для успішної реалізації проекту.

1.5 Короткий огляд задач для створення програми

Розглянемо роботи поетапно:

- Обробка запитів: розробка логіки обробки запитів вебпорталу, що включає отримання, валідацію та обробку вхідних даних користувачів.
- Робота з базою даних: реалізація взаємодії з базою даних для збереження та отримання необхідної інформації. Це включає створення моделей даних, міграцію бази даних та виконання запитів до бази даних.
- Валідація даних: розробка механізму валідації даних, що дозволяє перевіряти правильність введених користувачем даних перед їх збереженням або обробкою.
- Генерація документації API: інтеграція бібліотеки Swagger для автоматичної генерації документації API вебпорталу. Це дозволяє

забезпечити зрозумілу документацію для розробників, які будуть використовувати API.

- Безпека та автентифікація: розробка системи автентифікації та авторизації користувачів, що забезпечує безпеку та контроль доступу до ресурсів вебпорталу.
- Інтеграція з іншими системами: можливість інтеграції з іншими системами або сервісами, які можуть бути необхідні для взаємодії з вебпорталом, наприклад, платіжними шлюзами, системами електронного документообігу тощо.
- Тестування та налагодження: розробка тестів для перевірки коректності та працездатності різних функціональних блоків вебпорталу та вирішення потенційних проблем.

Ці задачі спрямовані на створення функціонального, безпечного та ефективного вебпорталу для навчально-наукового інституту, який забезпечує зручний доступ до інформації та функціональності для користувачів.

1.6 Сфера забезпечення безпеки вебпорталу

Сфера забезпечення безпеки вебпорталу є важливою та необхідною в сучасному цифровому середовищі. Забезпечення безпеки вебпорталу охоплює широкий спектр заходів та стратегій, спрямованих на захист інформації, запобігання несанкціонованому доступу та зловживанням, забезпечення конфіденційності, цілісності та доступності даних.

Одним з ключових аспектів безпеки вебпорталу є захист від зломів та зловживань. Це включає в себе захист від хакерських атак, вторгнень та використання вразливостей системи. Для цього використовуються різні методи та технології, такі як застосування рівня автентифікації та авторизації, шифрування даних, використання захисних механізмів та регулярні оновлення

системи з метою закриття вразливостей.

Також важливим аспектом безпеки вебпорталу є захист від шкідливих програм та вірусів. Це охоплює використання антивірусного ПЗ, регулярне сканування системи на наявність шкідливих програм, контроль вхідних та вихідних даних та використання механізмів виявлення та відновлення в разі виявлення інцидентів безпеки.

Додатково, забезпечення безпеки вебпорталу включає в себе захист від спаму, фішингових атак, крадіжки даних та несанкціонованого доступу. Це може включати в себе встановлення фільтрів та блокування небажаних повідомлень, навчання користувачів про безпеку, регулярні аудити безпеки та застосування строгих політик доступу до даних.

Забезпечення безпеки вебпорталу має велике практичне значення, оскільки воно допомагає захистити конфіденційну інформацію, попередити втрату даних, запобігти фінансовим збиткам та зберегти репутацію організації. Надійна безпека вебпорталу створює довіру серед користувачів та сприяє успішній роботі та розвитку організації.

Висновки до розділу

Провівши аналіз предметної області, виявлено, що розробка бекенд частини вебпорталу для навчально-наукового інституту є актуальною у зв'язку зі зростанням потреб у зручному та ефективному доступі до навчальних ресурсів. Важливими аспектами є також сприяння організації та обміну інформацією, а також покращення ефективності управління навчальним процесом.

Розглядаючи архітектурні підходи, виявлено, що мікросервісна архітектура видається оптимальним вибором для реалізації бекенд частини вебпорталу. Цей підхід дозволяє розділити додаток на невеликі та автономні сервіси, що сприяє гнучкості та масштабованості системи. Незалежний розвиток, розгортання та

масштабування кожного сервісу дозволяють ефективно управляти та розвивати систему.

Отже, розробка бекенд частини вебпорталу за використання мікросервісної архітектури відповідає сучасним вимогам та потребам у сфері освіти, забезпечуючи ефективний та гнучкий інструмент для управління навчальним процесом.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

В цьому розділі ми розглянемо основні технології, що були вибрані для програмної реалізації даної системи. Також визначимо основні переваги та принципи роботи цих технологій, платформ, алгоритмів та підходів.

Основними моментами, що ми повинні вибрати та дослідити являються:

- Платформа, що буде використовуватися для написання основного додатку
- Мова програмування
- Технологія створення фреймворку

2.1 Мова програмування C#

Мова програмування C# [1] є однією з найпопулярніших мов програмування у сфері розробки ПЗ. Вона є частиною .NET-екосистеми і надає розробникам потужні інструменти для створення різноманітних програмних рішень.

C# була розроблена з метою поєднати силу традиційних мов програмування, таких як C++ і Java, з простотою використання та ефективністю розробки. Вона має синтаксис, схожий на мови C і C++, що полегшує розуміння для розробників з досвідом у цих мовах.

Переваги мови програмування C# включають:

- Об'єктно-орієнтований підхід: C# підтримує об'єктно-орієнтоване програмування, що дозволяє створювати класи, об'єкти і наслідування. Це полегшує організацію коду, підвищує повторне використання і поліпшує модульність програми.
- Багатофункціональність: C# надає широкий набір вбудованих бібліотек і фреймворків для реалізації різноманітних функціональних можливостей. Це включає роботу з базами даних, мережеве програмування, графічний інтерфейс, безпеку та багато іншого.

- Кросплатформенність: Завдяки платформі .NET, програми, написані на C#, можуть працювати на різних операційних системах, включаючи Windows, macOS і Linux. Це робить C# ідеальним вибором для розробки кросплатформенних додатків.
- Легкість використання: C# має синтаксис, схожий на мови C і C++, що полегшує розуміння для розробників з досвідом у цих мовах. Вона також надає чистий і читабельний код, що сприяє зручній розробці і підтримці програмного продукту.
- Інтеграція з .NET-екосистемою: C# є основною мовою програмування для платформи .NET, яка забезпечує багато інструментів і сервісів для розробки ПЗ. Це включає розробку веб-додатків, мобільних додатків, хмарних рішень та багато іншого.
- Підтримка великих проектів: C# надає можливості для розробки масштабних проектів з великою командою розробників. Вона підтримує модульність, паралельне програмування, тестування і інші розширені функціональні можливості, що сприяють розробці складних програмних систем.
- Безпека: C# має вбудовану підтримку для безпеки ПЗ, включаючи механізми контролю доступу, шифрування даних і перевірку цілісності. Це допомагає забезпечити захист від різноманітних загроз і зловживань.
- Велике співтовариство розробників: C# має велике та активне співтовариство розробників, що надає доступ до багато ресурсів, документації, форумів, бібліотек і фреймворків. Це сприяє підтримці, обміну досвідом і розвитку вмінь у програмуванні на C#.
- Інструменти розробки: Для розробки на C# доступні потужні інтегровані середовища розробки (IDE), такі як Visual Studio, які надають багато інструментів і можливостей для зручного написання, відлагодження та керування кодом.

Окрім того, C# надає широкий набір вбудованих бібліотек і фреймворків, що дозволяють розробникам швидко реалізувати різноманітні функціональність, таку як робота з базами даних, мережеве програмування, обробка даних, графічний інтерфейс та багато іншого. Це зменшує час і зусилля, необхідні для розробки складних програмних продуктів.

Завдяки платформі .NET, програми, написані на C#, можуть працювати на різних операційних системах, включаючи Windows, macOS і Linux. Це робить мову C# універсальним інструментом для розробки кросплатформенних додатків.

Враховуючи всі ці переваги, мова програмування C# є ідеальним вибором для розробників, які прагнуть створювати потужні, надійні та масштабовані програмні продукти. Вона надає гнучкість, продуктивність і широкі можливості для розвитку різних видів додатків у різних сферах, включаючи веб-розробку, мобільні додатки, ігри, штучний інтелект та багато іншого.

2.1.1 Легкість у навчанні

Легкість у навчанні є однією з важливих переваг мови програмування C#. Деякі особливості, які роблять її легкою для вивчення, включають:

- Синтаксис: C# має синтаксис, схожий на мови C і C++, що робить її зрозумілою для розробників, які мають попередній досвід у цих мовах. Багато структур і концепцій, таких як змінні, функції і умовні оператори, є спільними з цими мовами.
- Читабельність: C# надає можливість писати чистий і легко читабельний код. Це допомагає розробникам швидко розуміти функціональність програми і легко виявляти помилки.
- Об'єктно-орієнтований підхід: C# підтримує об'єктно-орієнтоване програмування, що є одним з найпоширеніших підходів до розробки програмного забезпечення. Цей підхід дозволяє організовувати код у

вигляді класів і об'єктів, що полегшує розуміння програми.

- Багатофункціональність: C# має велику кількість вбудованих бібліотек і фреймворків, які надають готові рішення для різних завдань. Це дозволяє розробникам ефективно використовувати готові компоненти і скорочувати час розробки.
- Ресурси для навчання: Існує велика кількість документації, онлайн-курсів, підручників і відеоуроків, які допомагають вивчити мову програмування C#. Багато з цих ресурсів доступні безкоштовно і надають зрозумілі пояснення та приклади для вивчення.

Загалом, легкість у навчанні робить мову програмування C# привабливою для початківців і дозволяє швидко освоїти основи програмування та розробку програмних продуктів.

2.1.2 Гнучкість та розширюваність

- Гнучкість: C# пропонує різноманітні можливості програмування і дозволяє розробникам вибирати найбільш підходящий підхід до вирішення конкретних завдань. Мова підтримує різні парадигми програмування, такі як процедурне, об'єктно-орієнтоване, функціональне та асинхронне програмування. Це дозволяє розробникам підлаштовувати стиль програмування під вимоги проекту.
- Розширюваність: C# має розширюваність, що означає можливість додавати нові функціональні можливості до мови за допомогою розширень (extensions) і бібліотек. Розробники можуть створювати свої власні розширення, які додають нові функції та можливості до мови, що спрощує розробку складних програмних продуктів.
- Підтримка .NET-екосистеми: C# є основною мовою програмування для розробки на платформі .NET. Це означає, що мова має доступ до широкого спектру бібліотек і фреймворків, які допомагають розробникам ефективно

виконувати різні завдання. Ця велика екосистема сприяє гнучкості і розширюваності програм, оскільки розробники можуть використовувати готові компоненти для реалізації різних функціональних можливостей.

- Розповсюдженість і підтримка: C# є однією з найпоширеніших мов програмування в індустрії. Вона має велику спільноту розробників, що означає наявність обширної бази знань, документації та онлайн-ресурсів для допомоги розробникам. Крім того, мова має активну підтримку з боку Microsoft, що забезпечує оновлення, виправлення помилок та розвиток мови.

2.1.3 Підтримка спільноти

Спільнота розробників, яка складається з програмістів, експертів та ентузіастів, активно співпрацює та спільно вирішує проблеми, пов'язані з розробкою на C#.

Основні переваги підтримки спільноти включають:

- Форуми та дискусійні групи: Існує безліч форумів та дискусійних груп, присвячених мові програмування C#. Тут розробники можуть задавати питання, ділитися досвідом та отримувати відповіді від інших учасників спільноти.
- Блоги та статті: Багато розробників та експертів по C# пишуть блоги та публікують статті, в яких поділяються корисними порадами, техніками та рішеннями. Це дозволяє розробникам вивчати нові можливості мови та вдосконалювати свої навички.
- Конференції та зустрічі: Спільнота C# організовує різноманітні конференції, семінари та зустрічі, на яких розробники можуть обмінюватись досвідом, відкривати нові напрями розробки та навчатися від експертів.
- Відкриті джерела та проекти: Багато проектів на C# є відкритими

джерелами, що означає, що їх вихідний код доступний для розробників. Це дає можливість вивчати інші реалізації, співпрацювати з іншими розробниками та вносити свій внесок у розвиток спільноти.

Завдяки підтримці спільноти, розробники на C# мають доступ до широкого кола ресурсів та експертів, що сприяє їх професійному росту та розвитку.

2.1.4 Тестування

C# має багато вбудованих функціональностей та інструментів, які полегшують розробку тестів і забезпечують надійність та якість програмного продукту.

Основні переваги тестування на C# включають:

- Широкий вибір фреймворків: У мові C# доступно багато різноманітних фреймворків для автоматизованого тестування, таких як NUnit, MSTest, xUnit і FluentAssertions. Ці фреймворки дозволяють розробляти та виконувати тести з різних аспектів програми, забезпечуючи повноту тестового покриття.
- Інструменти для юніт-тестування: C# має вбудовані засоби для написання і виконання юніт-тестів, таких як атрибути [TestMethod], [TestClass] та [TestInitialize], які спрощують розробку тестових наборів та перевірку результатів.
- Інтеграція з іншими інструментами: C# може легко інтегруватися з іншими інструментами тестування, такими як фреймворки для UI-тестування (наприклад, Selenium), фреймворки для вимірювання покриття коду (наприклад, OpenCover) та системи для автоматизованого розгортання (наприклад, Jenkins). Це дає можливість створювати повноцінні тестові середовища та забезпечувати автоматичне виконання тестів у процесі розробки.

- Можливість розширення: C# підтримує розширення мови за допомогою власних бібліотек та фреймворків, що дозволяє розробникам створювати спеціалізовані інструменти для тестування своїх програм. Завдяки цьому, можна створювати власні розширення, які допомагатимуть у забезпеченні потреб тестування конкретного проекту.

Тестування на C# дозволяє ефективно перевіряти функціональність програми, забезпечуючи високу якість і надійність. Завдяки широким можливостям та інструментарію мови, розробники можуть швидко та ефективно писати тести, автоматизувати їх виконання та забезпечувати безперебійну роботу своїх програмних продуктів.

2.2 Visual Studio IDE та існуючи аналоги

Visual Studio IDE (Integrated Development Environment) є одним з найпопулярніших та потужних інтегрованих середовищ розробки програмного забезпечення для мови програмування C# та інших мов. Воно надає розширений набір інструментів та функціональностей, що допомагають розробникам створювати, тестувати та налагоджувати програми швидко та ефективно.

Основні переваги Visual Studio IDE включають:

- Повний набір інструментів розробки: Visual Studio має вбудовані засоби для розробки, налагодження та тестування програм. Вона надає доступ до різноманітних інструментів, таких як редактор коду з підсвічуванням синтаксису, автодоповненням та перевіркою помилок, вбудована система контролю версій, візуальний дизайнер форм, відлагоджувач, аналізатори продуктивності та багато іншого. Це дозволяє розробникам працювати ефективно та швидко.
- Інтеграція з платформою .NET: Visual Studio повністю підтримує платформу .NET, що дозволяє розробникам створювати різноманітні програми та сервіси, використовуючи багатофункціональні бібліотеки та

фреймворки. Завдяки інтеграції з платформою .NET, Visual Studio надає доступ до розширеного екосистеми, що допомагає в розробці та розгортанні програмного забезпечення.

- Розширюваність та плагіни: Visual Studio підтримує розширення та плагіни, які дозволяють розробникам налаштувати робоче середовище під свої потреби. Є багато сторонніх розширень, що додають додаткові функції та можливості до Visual Studio, такі як інструменти для роботи зі специфічними фреймворками, плагіни для підвищення продуктивності, інструменти для тестування та багато іншого. Це дає розробникам більшу свободу та гнучкість у використанні IDE.

На рисунку 2.1 зображено середовище розробки програмного забезпечення «Visual Studio».

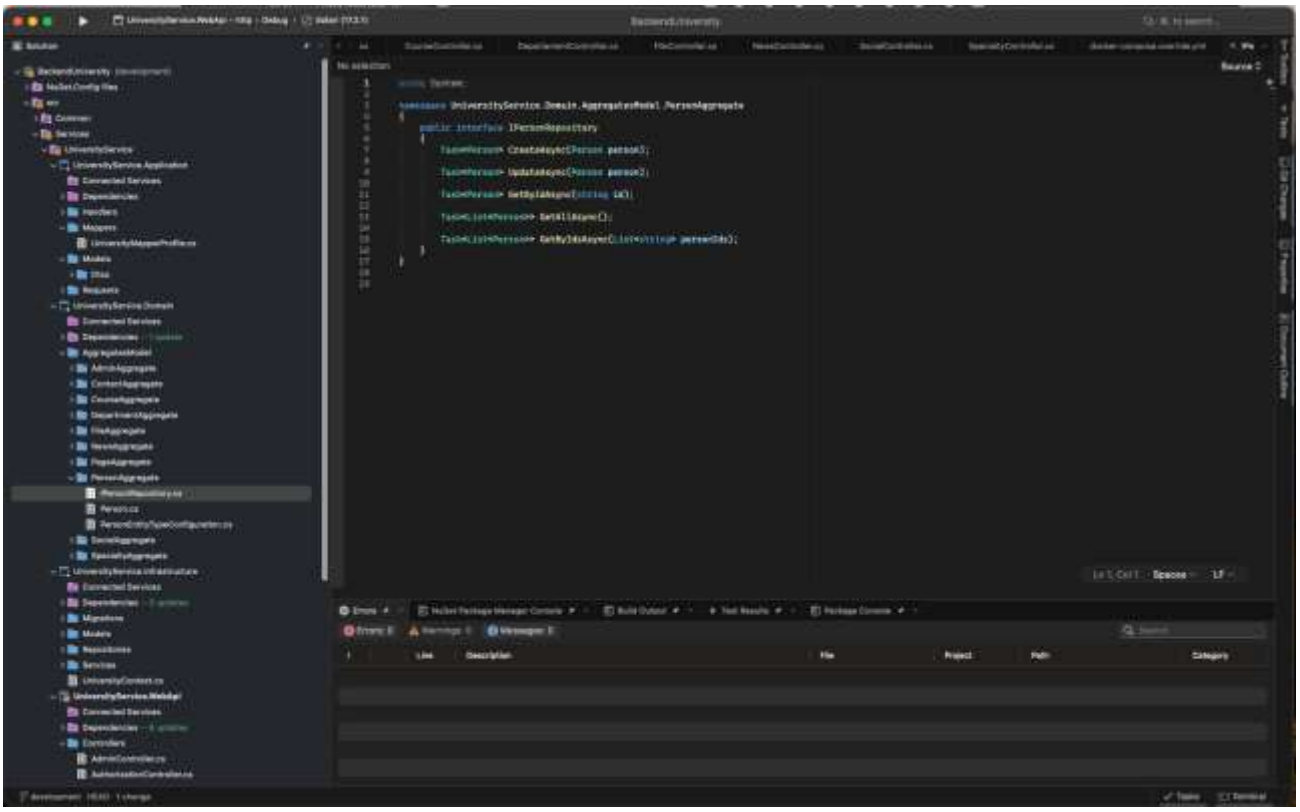

Рисунок 2.1 – Середовище розробки програмного забезпечення «Visual Studio»

Існують також інші аналоги Visual Studio IDE, які також популярні серед розробників C# та інших мов програмування. Деякі з них включають:

- JetBrains Rider: Це інтегроване середовище розробки, яке підтримує мову C# та інші мови програмування. Воно володіє потужними функціями для розробки та налагодження програм, а також підтримує різні фреймворки та технології.
- Visual Studio Code: Це легке та розширюване редактор-коду, який підтримує різні мови програмування, включаючи C#. Він має багато розширень та можливостей для розробки програмного забезпечення.
- Xamarin Studio: Це середовище розробки, спеціально призначене для розробки мобільних додатків на базі платформи Xamarin, що підтримує мову C# та .NET Framework.

2.3 Фреймворк .net core 6

.NET Core 6 є одним з найновіших версій фреймворку .NET Core [2, 3], який розробляється компанією Microsoft. Це крос-платформенний відкритий фреймворк, який дозволяє розробникам створювати різноманітні додатки для різних платформ, таких як Windows, macOS та Linux.

Особливості та переваги .NET Core 6 включають:

- Крос-платформенність: .NET Core 6 дозволяє розробляти додатки, які можуть запускатися на різних операційних системах, незалежно від того, чи це Windows, macOS чи Linux. Це робить фреймворк відмінним вибором для розробників, які хочуть мати можливість переносити свої додатки на різні платформи.
- Висока продуктивність: .NET Core 6 має вбудовану оптимізацію та підтримку асинхронного програмування, що дозволяє створювати швидкі та ефективні додатки. Він має оптимізовану систему зборки сміття, що забезпечує ефективне управління пам'яттю та покращує продуктивність

додатків.

- Широкий вибір мов програмування: .NET Core 6 підтримує різні мови програмування, включаючи C#, F#, Visual Basic та інші. Це дозволяє розробникам вибирати мову, з якою вони найбільш комфортно працюють, і розробляти додатки на базі .NET Core.
- Модульність та розширюваність: .NET Core 6 побудований на основі модульної архітектури, що дозволяє розробникам використовувати лише необхідні компоненти фреймворку. Він також підтримує розширення та плагіни, що дає можливість розширювати функціональність фреймворку залежно від потреб проекту.
- Велика спільнота та підтримка: .NET Core має активну та велику спільноту розробників, яка забезпечує великий обсяг ресурсів, документації, підручників та підтримку. Це означає, що розробники можуть швидко знайти відповіді на свої питання та рішення для своїх проблем.
- Нові можливості: .NET Core 6 пропонує ряд нових можливостей, таких як підтримка гарячого перезапуску додатків, покращення роботи з пам'яттю та продуктивність, підтримка графічного інтерфейсу з використанням Blazor та багато іншого. Ці нові можливості роблять фреймворк більш потужним та зручним для розробки додатків.

Загалом, .NET Core 6 є потужним фреймворком для розробки крос-платформенних додатків з високою продуктивністю, гнучкістю та широкою підтримкою спільноти. Він надає розробникам багато можливостей та інструментів для створення сучасних та інноваційних додатків.

2.4 Документація API

Swagger є набором інструментів, які дозволяють автоматично створювати, відображати та документувати API [5]. Основна мета Swagger - спростити процес розробки та спілкування між розробниками, які працюють над різними

частинами проекту.

Документація API з використанням Swagger здійснюється наступним чином:

- Опис специфікації API: Swagger використовує специфікацію OpenAPI для опису структури та функціональності API. Специфікація включає інформацію про шляхи (endpoints), параметри, типи даних, відповіді та інші деталі API.
- Анотування коду: Розробники використовують анотації або атрибути в своєму коді для вказівки Swagger, які методи і класи повинні бути документовані. Ці анотації містять інформацію про URL, параметри, типи даних та інші важливі деталі API.
- Автоматичне генерування документації: За допомогою Swagger, документація API генерується автоматично на основі анотацій, що використовуються в коді. Swagger сканує код та витягує всю необхідну інформацію для створення документації, включаючи URL, параметри, типи даних, описи методів та інші деталі.
- Візуальне відображення документації: Swagger надає веб-інтерфейс, який дозволяє розробникам переглядати та взаємодіяти з документацією API. Цей інтерфейс має зручну навігацію, яка дозволяє переглядати доступні шляхи, виконувати запити та переглядати відповіді.
- Додаткові можливості: Swagger надає багато додаткових можливостей для документації API, таких як автоматична генерація коду клієнта для різних мов програмування, тестування API безпосередньо з інтерфейсу Swagger та інтеграція з іншими інструментами розробки.

На рисунку 2.2 представлено Swagger GUI бекенд частини вебпорталу для навчально-наукового інституту.

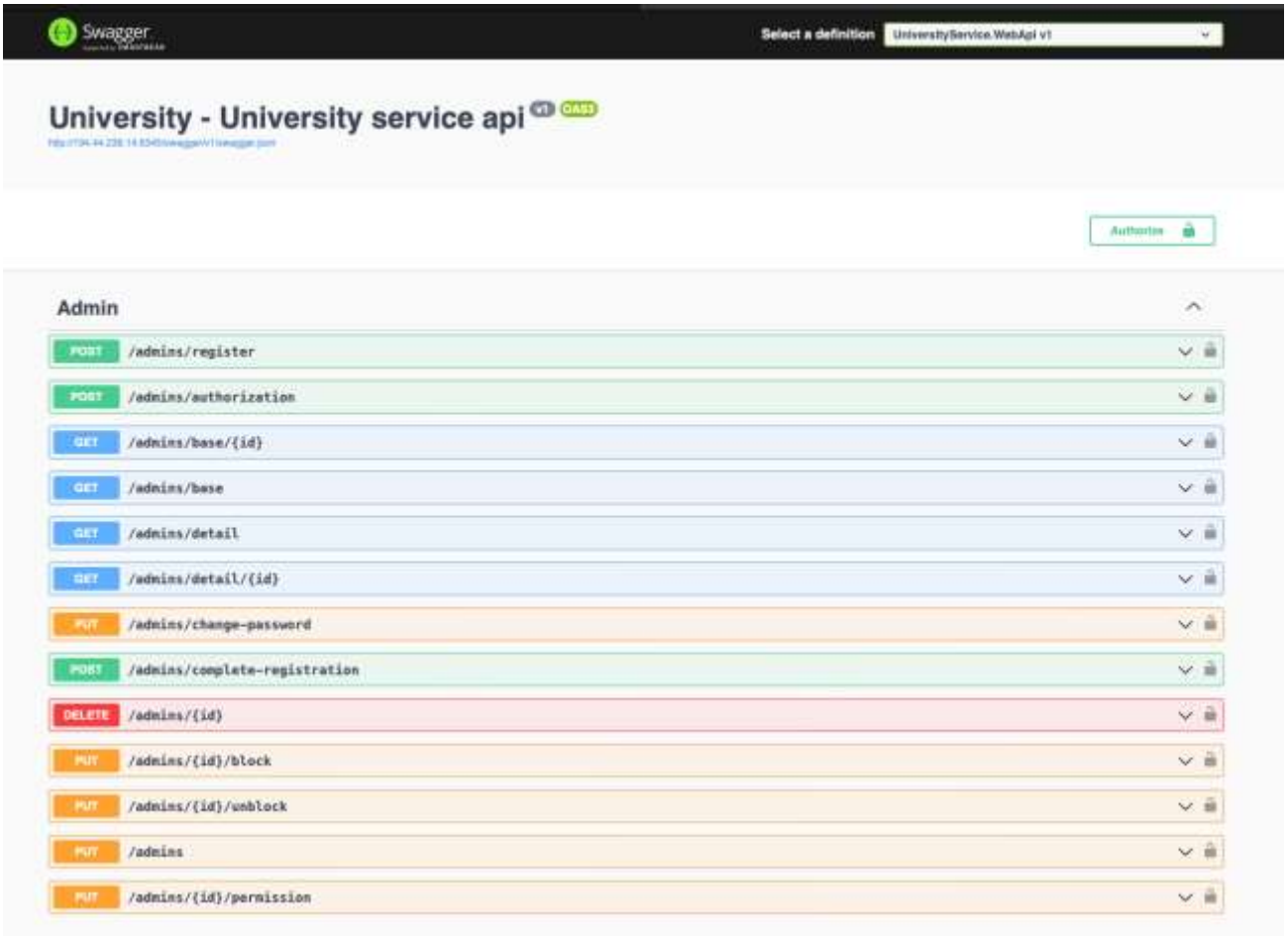

Рисунок 2.2 – Swagger GUI

Загалом, Swagger дозволяє розробникам автоматично створювати документацію API з використанням анотацій у коді. Це збільшує продуктивність та полегшує спілкування між розробниками, а також надає веб-інтерфейс для зручного перегляду та тестування API.

2.5 Взаємодія з базою даних

Entity Framework (EF) Core [4] є сучасним ORM (Object-Relational Mapping) фреймворком, який забезпечує зручну та ефективну взаємодію з базою даних у дотриманні принципів об'єктно-орієнтованого програмування. Використання EF Core дозволяє розробникам працювати з базою даних, не звертаючи уваги на деталі SQL і складні операції з базою даних.

Основні кроки взаємодії з базою даних за допомогою EF Core включають:

- Визначення моделі даних: Розробники визначають класи, які представляють сутності бази даних (таблиці), а також взаємозв'язки між ними. Ці класи називаються моделями. EF Core використовує ці моделі для створення відображення бази даних на об'єкти.
- Конфігурація контексту бази даних: Розробники визначають клас, який наслідуєся від DbContext, що представляє контекст бази даних. У цьому класі визначаються налаштування зв'язку з базою даних, таблиці, відношення між ними та інші параметри. Контекст бази даних виконує функцію посередника між програмою та базою даних.
- Використання CRUD операцій: EF Core надає зручний API для виконання операцій створення (Create), читання (Read), оновлення (Update) та видалення (Delete) даних в базі даних. Розробники можуть використовувати LINQ (Language-Integrated Query) для формулювання складних запитів до бази даних, які автоматично перетворюються на SQL запити.
- Міграції бази даних: EF Core надає можливість автоматичного керування схемою бази даних. Розробники можуть визначити зміни у моделі даних і застосувати їх до бази даних за допомогою міграцій. Міграції дозволяють автоматично створювати таблиці, змінювати структуру бази даних та вносити інші зміни безпечним способом.
- Оптимізація запитів: EF Core надає можливість оптимізувати запити до бази даних. Розробники можуть використовувати методи та атрибути EF Core, щоб вказати, як EF Core повинен виконувати запити та які дані повинні бути завантажені з бази даних, що дозволяє зменшити кількість запитів до бази даних та покращити продуктивність.

Взаємодія з базою даних за допомогою EF Core дозволяє розробникам ефективно працювати з базою даних, зосереджуючись на розробці

функціональності своєї програми, не звертаючи багато уваги на низькорівневі деталі взаємодії з базою даних.

Висновки до розділу

Проаналізувавши технічні аспекти розробки програмного забезпечення на мові програмування C# з використанням Visual Studio IDE та фреймворку .NET Core 6, можна зробити висновок про потужність та ефективність цих інструментів у сфері програмування.

Мова програмування C# виокремлюється своєю популярністю та внутрішньою логікою, а також інтеграцією з .NET-екосистемою, що робить її привабливим вибором для розробників. Visual Studio IDE, як основне середовище для роботи з C# та іншими мовами, забезпечує зручний та розширений інтерфейс для розробки, тестування та налагодження програм.

Фреймворк .NET Core 6 виокремлюється крос-платформенністю та можливістю створення додатків для різних операційних систем. Його оновлені можливості роблять його привабливим для сучасної розробки програмного забезпечення.

Щодо документації API, використання Swagger дозволяє автоматизувати та спростити процес розробки, а також поліпшує комунікацію між розробниками. Це важливий інструмент для створення зрозумілих та добре задокументованих API.

Обрані технічні засоби (мова програмування C#, Visual Studio IDE, .NET Core 6 та Swagger) є потужними та сучасними інструментами для розробки програмного забезпечення, які відповідають вимогам сучасної індустрії та забезпечують зручність та ефективність у процесі розробки.

РОЗДІЛ 3. МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Роль авторизації та прав доступу

Авторизація та управління правами доступу (рівнем доступу) — це критичні аспекти в розробці веб-додатків, оскільки вони визначають, хто і як може використовувати різні ресурси системи. У даній роботі використовується JWT-авторизації в .NET для авторизації та управління доступом користувачів, також вона пов'язана з рядом математичних операцій та концепцій.

Структура JWT: Токен складається з трьох частин, розділених крапками: Header (заголовок), Payload (вантаж), та Signature (підпис). Header зазвичай містить тип токена та використовуваний алгоритм шифрування. Payload містить заяви (claims), які включають інформацію про користувача та інші метадані. Signature генерується за допомогою Header, Payload та секретного ключа, що гарантує незмінність та автентичність токена. Підпис гарантує незмінність та автентичність токена.

Безпека JWT: Зазвичай JWT підписуються за допомогою одного з трьох алгоритмів: HMAC, RSA або ECDSA. Кожен з цих алгоритмів має свої особливості та сфери застосування. Наприклад, HMAC використовує спільний ключ для створення та перевірки JWT, що робить його простим у використанні, але менш безпечним у деяких сценаріях. RSA та ECDSA - асиметричні алгоритми, що дозволяють верифікувати JWT за допомогою публічного ключа без можливості створення нового. Це підвищує безпеку, оскільки тільки власник приватного ключа може створювати валідні JWT. Ці алгоритми, HMAC (Hash-based Message Authentication Code) або RSA (Rivest–Shamir–Adleman), забезпечують, що тільки ті, хто володіє секретним ключем, можуть генерувати або перевіряти підпис. Математичні основи цих алгоритмів, як наприклад складність обернення хеш-функцій у випадку HMAC або складність факторизації великих чисел у випадку RSA, забезпечують безпеку даних.

Використання HMAC означає, що без доступу до секретного ключа неможливо згенерувати валідний підпис, що робить JWT надійним для перевірки автентичності. RSA додає додатковий рівень безпеки завдяки складності факторизації великих чисел, що робить його важким для зламу. У моєму проекті ці принципи гарантували безпечне управління правами доступу, запобігаючи несанкціонованому доступу до ресурсів.

JWT-токен захищається шляхом використання цифрового підпису або шляхом генерування Коду Автентифікації Повідомлень (MAC). Для підписання токену використовується асиметричний алгоритм (наприклад, RS256), який застосовує приватний ключ суб'єкта, що генерує токен. З іншого боку, симетричний алгоритм (наприклад, HS256) використовує спільний секретний ключ, відомий обом сторонам, тобто генератору та валідатору токенів.

На рисунку 3.1 показано, як симетричний та асиметричний алгоритми можуть використовуватися разом для аутентифікації споживача API до провайдера API через Сервер Аутентифікації. Споживач API (мікросервіс A) генерує HS256 JWT-токен, використовуючи власний спільний секретний ключ, та відправляє його на сервер аутентифікації. Сервер аутентифікації валідує HS256 JWT-токен, використовуючи спільний секретний ключ мікросервісу A. Якщо перевірка токена проходить успішно, сервер аутентифікації генерує RS256 JWT-токен, використовуючи власний приватний ключ, та відправляє його назад мікросервісу A. Мікросервіс A відправляє запит мікросервісу B з RS256 JWT-токеном, який він отримав від сервера аутентифікації. Мікросервіс B може аутентифікувати запит від мікросервісу A, перевіривши RS256 JWT-токен, використовуючи публічний ключ сервера аутентифікації. Оскільки мікросервіс A вже встановив довіру з сервером аутентифікації, мікросервіс B може довіряти запиту від мікросервісу A, якщо RS256 токен успішно валідований.

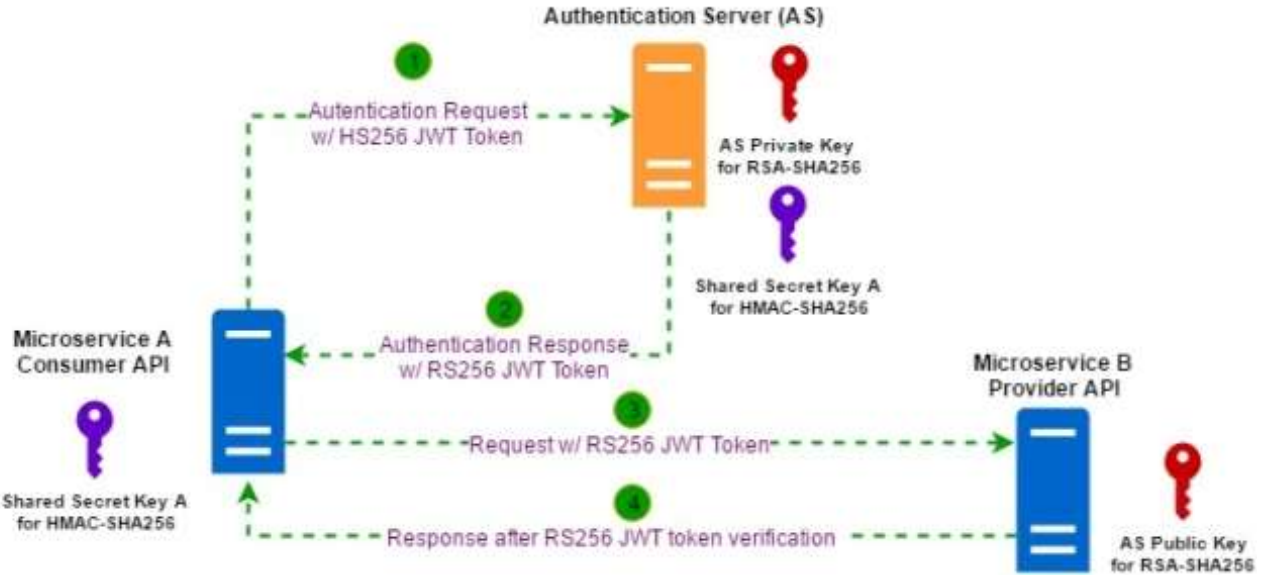

Рисунок 3.1 – Використання симетричного та асиметричного алгоритмів для автентифікації API

3.2 Оптимізація Бази Даних

У своїй роботі з EF Core та PostgreSQL я зіткнувся з необхідністю оптимізувати запити та структуру бази даних для підвищення ефективності взаємодії з даними. Оптимізація бази даних - це комплексний процес, що включає аналіз та вдосконалення різних аспектів зберігання та обробки даних.

Індексація: Використання індексів є критичним для покращення швидкості витягування даних. Математичний принцип під індексацією полягає у створенні оптимізованої структури даних, такої як B-дерева або хеш-таблиці, які дозволяють швидко знаходити рядки в таблиці без необхідності сканування всієї таблиці. Вибір відповідного типу індексу залежить від конкретного використання та властивостей даних.

Нормалізація: Нормалізація даних полягає у структуруванні бази даних таким чином, щоб зменшити дублювання та забезпечити логічну цілісність. Це включає розділення даних на таблиці з визначенням відносин між ними.

Використання нормалізації вимагає розуміння математичних концепцій залежностей та ключів у базах даних.

Оптимізація запитів: Оптимізація SQL-запитів є важливою для забезпечення швидкого та ефективного доступу до даних. Математичний аналіз таких аспектів, як вибірка (SELECT), з'єднання (JOIN) та сортування (ORDER BY), дозволяє розробляти запити, що мінімізують обсяг оброблюваних даних та час відгуку.

PostgreSQL, як і багато інших систем управління базами даних, використовує низку математичних алгоритмів для оптимізації зберігання, витягування, управління та обробки даних. Ось деякі з ключових алгоритмів та математичних концепцій, які використовуються в PostgreSQL:

- B-дерева (B-Trees): Найбільш поширений тип індексації у базах даних. Вони дозволяють швидко читати та писати дані, забезпечуючи логарифмічний час пошуку, вставки та видалення.
- Хешування (Hash Indexing): Використовується для створення індексів, де вхідні дані перетворюються хеш-функцією у хеш-значення, яке потім може бути використано для швидкого пошуку відповідних записів.
- GiST (Generalized Search Tree): Це балансовані дерева, які дозволяють створювати ефективні індекси для складних типів даних та умов пошуку, таких як геометричні дані чи текстовий пошук.
- GIN (Generalized Inverted Indexes): Оптимізовані для індексації складних структур даних, як-от масиви чи JSON-документи, де один ключ може асоціюватися з багатьма значеннями.
- Алгоритми сортування (Sorting Algorithms): Використовуються для сортування даних перед їх зберіганням чи виводом. Наприклад, алгоритми QuickSort, MergeSort, або варіації HeapSort можуть бути використані для ефективного сортування великих наборів даних.

- Алгоритми сканування (Scan Algorithms): Використовуються для просканування таблиць чи індексів для знаходження записів, які відповідають певним критеріям.
- Оптимізатор запитів (Query Optimizer): Використовує математичні моделі для визначення найефективнішого шляху виконання запиту, аналізуючи різні можливі стратегії і вибираючи найоптимальніший план запиту.

Ці алгоритми та структури даних дозволяють PostgreSQL ефективно управляти широким спектром операцій з даними, забезпечуючи високу продуктивність та надійність для різноманітних застосувань.

3.3 Алгоритми управління персональними даними

У контексті моєї роботи з .NET 6, захист персональних даних, особливо паролів, є критично важливим. Щоб забезпечити цю безпеку, я використовую хеш-функції разом із солінгом, які є стандартною практикою в індустрії безпеки. Суть цих методів полягає у перетворенні первинної інформації – в даному випадку пароля – у новий, фіксований набір символів, що забезпечує односторонність та унікальність. Для цього я використовую бібліотеку Bcrypt.Net [13]. Ця бібліотека імплементує алгоритм BCrypt, який є одним із найнадійніших методів для шифрування паролів, і ось як вона працює:

- Генерація Солі: Коли створюється новий пароль, Bcrypt.Net автоматично генерує випадкову "сіль" – унікальну послідовність байтів, що додається до пароля. Ця додаткова інформація забезпечує, що навіть ідентичні паролі будуть мати унікальні хеші, роблячи марними спроби атак з використанням вже відомих хешів.
- Шифрування Паролю: Після того, як сіль згенеровано, вона комбінується з паролем, і ця комбінація піддається процесу шифрування алгоритмом BCrypt. Процес включає в себе кілька ітерацій шифрування, кількість яких залежить від встановленого робочого фактору. Результатом є хеш, який

потім зберігається у базі даних.

- Перевірка Паролю: Коли користувач намагається увійти, Bcrypt.Net знову генерує хеш із введеного пароля та солі, що була використана при створенні оригінального хешу. Якщо згенерований хеш збігається з хешем, збереженим у базі даних, доступ дозволяється.
- Адаптивність: Однією з ключових особливостей BCrypt є його адаптивність. Оскільки обчислювальні можливості сучасних комп'ютерів зростають, можна просто збільшити робочий фактор, щоб зробити процес створення хешу більш складним і таким чином підвищити безпеку.

Вибір алгоритму SHA-384 в бібліотеці Bcrypt.Net мотивований кількома ключовими факторами, які забезпечують баланс між продуктивністю, безпекою та захистом від зіткнень, а також відсутністю вразливостей до атак з розширенням довжини. Ось докладніший аналіз:

- Баланс Між Продуктивністю та Безпекою: SHA-384 є частиною сімейства SHA-2 і забезпечує високий рівень безпеки, не вимагаючи при цьому надмірних обчислювальних ресурсів. Це робить його підходящим для більшості додатків, де потрібно забезпечити безпеку без значного впливу на продуктивність системи.
- Захист від Зіткнень: SHA-384 має довжину хешу 384 біти, що забезпечує величезний простір можливих хешів і, відповідно, високий рівень захисту від зіткнень (ситуації, коли два різні вхідні набори даних дають однаковий хеш).
- Відсутність Вразливостей до Атак з Розширенням Довжини: SHA-384 не схильний до атак з розширенням довжини, які є потенційною проблемою для деяких інших хеш-функцій. При атаці з розширенням довжини, атакуючий може використовувати відомий хеш для генерування нових хешів з додатковими даними без необхідності знати початкові дані. SHA-384 захищений від такого типу атак, що підвищує загальну безпеку хеш-

функції.

- Захист від Інших Відомих Вразливостей: На відміну від деяких інших версій SHA (наприклад, SHA-1), SHA-384 не має відомих вразливостей або слабких місць, що робить його надійним вибором для криптографічного застосування.

Використання SHA-384 в Bcrypt.Net є виразом прагнення розробників забезпечити безпеку на високому рівні, водночас зберігаючи гарний рівень продуктивності і уникнення потенційних вразливостей, які можуть бути присутні в інших хеш-функціях.

Ця бібліотека використовує математичні принципи криптографії, зокрема шифрування, для забезпечення безпеки паролів. Основою цього процесу є використання хеш-функції SHA-384, яка є частиною сімейства алгоритмів SHA-2. SHA-384 застосовує математичні операції для перетворення вхідних даних (у цьому випадку, пароля) у вихідний хеш фіксованої довжини. Цей процес включає в себе багато ітерацій складних математичних перетворень, що забезпечують високий рівень криптографічної безпеки.

Використання SHA-384 у контексті Bcrypt.Net надає додаткову безпеку, оскільки SHA-384 не схильний до атак з розширенням довжини, на відміну від деяких інших хеш-функцій. Це означає, що атакуючі не можуть легко додати додаткові дані до оригінального повідомлення і отримати валідний хеш без знання оригінального повідомлення. Таким чином, використання SHA-384 забезпечує більшу безпеку від різних видів криптографічних атак, зберігаючи при цьому ефективність обчислень.

На рисунку 3.2 зображено використання шифрування паролю в поточній роботі.

Рисунок 3.2 – Використання шифрування паролю

Загалом, Bcrypt.Net з використанням SHA-384 демонструє, як можна ефективно застосовувати математичні принципи криптографії для захисту персональних даних у сучасних програмних продуктах.

Висновки до розділу

У ході розгляду різних аспектів програмної розробки, зокрема взаємодії з базою даних, оптимізації та алгоритмів управління персональними даними в .NET 6, визначено ключовий вплив математичних принципів на створення надійних та ефективних програм.

Взаємодія з базою даних вимагає оптимізації, яка базується на принципах індексації, нормалізації та оптимізації запитів. Цей підхід, з користю використовуючи математичні концепції, спрямований на підвищення ефективності взаємодії з даними.

Оптимізація бази даних використовує різноманітні математичні алгоритми, що дозволяє забезпечити ефективне зберігання та обробку даних. Це віддзеркалює важливість математичних концепцій у досягненні високої продуктивності баз даних.

У розділі про алгоритми управління персональними даними в .NET 6, відзначено критичне значення безпеки, яке досягається за допомогою математичних методів, таких як хеш-функції та солінг. Обрання алгоритму SHA-384 у бібліотеці Bcrypt.Net вказує на відповідальний підхід до захисту

персональних даних.

Математика виступає ключовим інструментом в усіх аспектах програмної розробки, сприяючи створенню безпечних, ефективних та надійних програмних продуктів.

РОЗДІЛ 4. ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

4.1 Загальна структура програмного коду

Проект, заснований на принципах DDD (Domain-Driven Design) [6] та "Чистої Архітектури" [7, 9], включає кілька ключових компонентів, що допомагають зберегти високу розширюваність, легкість тестування та збереження бізнес-логіки в центрі розробки. Основні компоненти структури проекту включають:

- “Application” проект: Відповідає за реалізацію додаткового шару, який взаємодіє з доменним шаром та інфраструктурними сервісами для надання функціональності користувачам. Він може містити сервіси додатку, DTO (Data Transfer Objects), фасади та інші компоненти, які реалізовують виконання конкретних вимог та використання доменного шару.
- “Domain” проект : Відображає бізнес-логіку вашого додатку та включає об'єкти, сутності, агрегати, сервіси та інші компоненти, що представляють основу вашого домену. Він визначає правила та обмеження бізнес-процесів, забезпечує становлення доменної моделі та її взаємодію з іншими компонентами системи.
- “Infrastructure” проект :Включає компоненти, які забезпечують інфраструктуру та підтримують функціонування вашого додатку. Сюди можуть входити бази даних, зовнішні сервіси, репозиторії, інструменти логування, кешування, комунікація з зовнішніми системами та інші технічні аспекти. Інфраструктурний шар може бути розділений на декілька підпроектів або модулів в залежності від потреб вашого додатку.
- “WebApi” проект: Представляє веб-інтерфейс вашого додатку, який надає зовнішній доступ до функціональності через API. Він може містити контролери, моделі представлення, фільтри, middleware та інші компоненти, необхідні для обробки запитів, аутентифікації, авторизації та

інших аспектів веб-взаємодії.

На рисунку 4.1 представлено структуру проекту з використанням DDD підходу та “Чистої архітектури”.

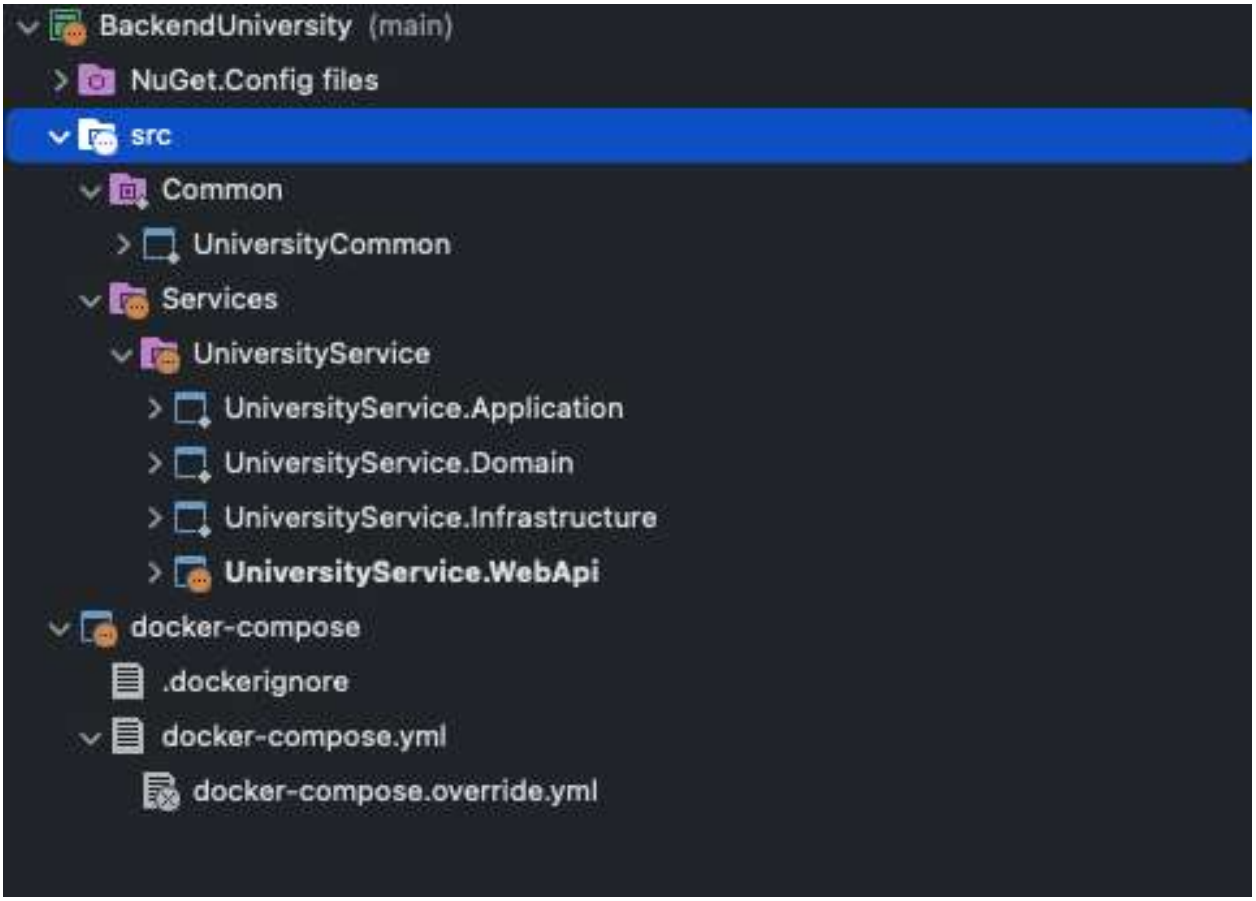

Рисунок 4.1 – Структура проекту з використанням DDD

4.2 Опис використаних сторонніх бібліотек та модулів

Entity Framework Core: Це фреймворк для доступу до даних, який дозволяє працювати з базами даних за допомогою об'єктно-орієнтованого підходу. Entity Framework Core використовується для взаємодії з базою даних в проекті. Він надає зручний API для створення, зчитування, оновлення та видалення даних з бази даних, а також підтримку механізмів міграцій та кешування.

AutoMapper: Це бібліотека, яка допомагає автоматично відображати дані між різними типами об'єктів. AutoMapper використовується для зручного та автоматизованого мапінгу даних між DTO (Data Transfer Objects) та моделями домену або моделями представлення. Це дозволило ефективно передавати дані між компонентами програми та зменшити зусилля, пов'язані з ручним мапінгом. На рисунку 4.2 представлено налаштування моделей з використанням бібліотеки AutoMapper.

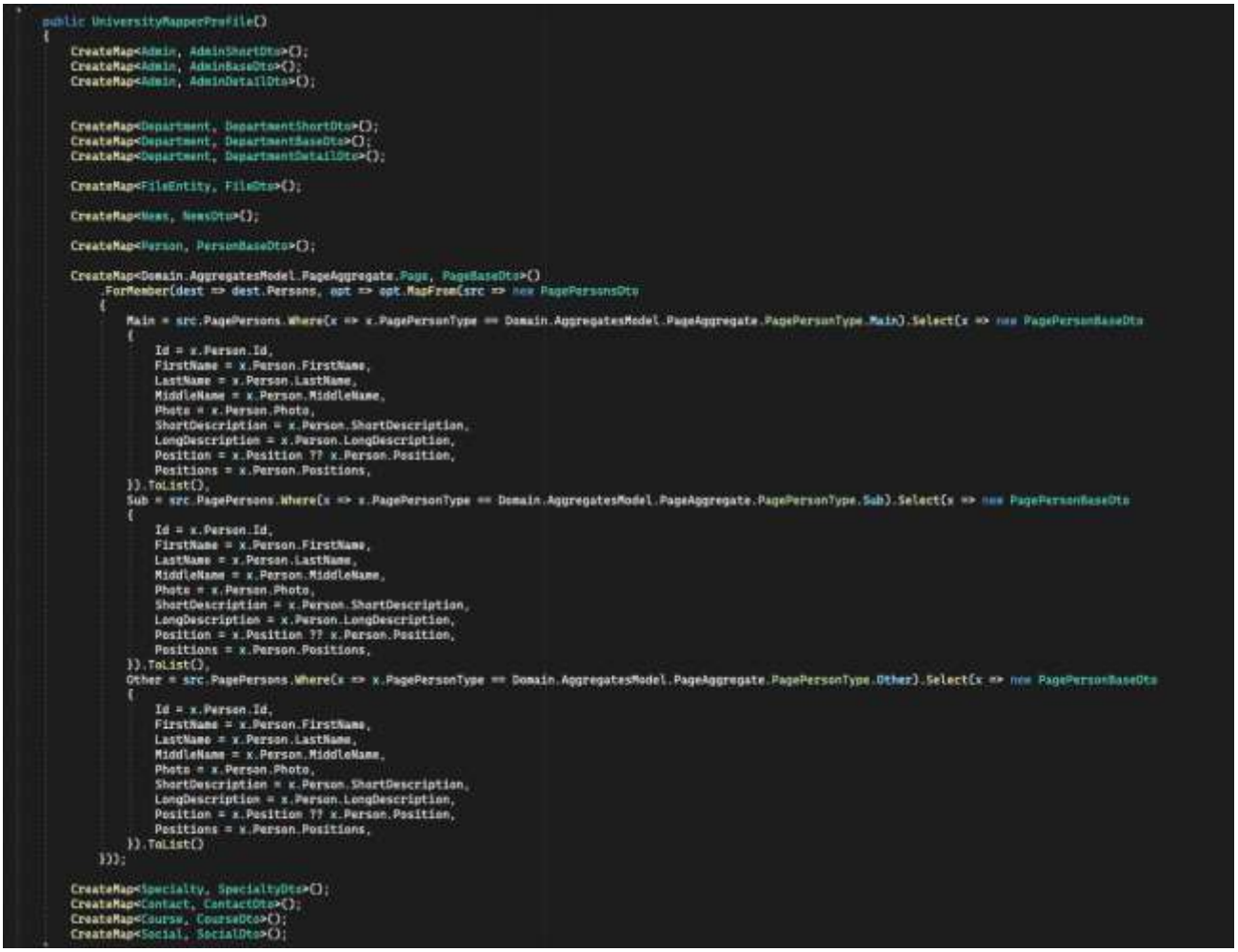

Рисунок 4.2 – Налаштування моделей з використанням бібліотеки AutoMapper

Swagger: Це інструмент для створення документації API та виконання різних операцій, пов'язаних з розробкою API. Використовували Swagger для автоматичної генерації документації API на основі анотацій та конфігурацій, що

додаються до коду. Це надає зручний спосіб документування та спілкування з користувачами API.

MediatR: Це бібліотека, яка надає шаблон Mediator для реалізації медіатора (посередника) для обробки запитів та команд у програмі. Використання MediatR для розподілу логіки додатку на окремі запити та команди, дозволило зберігати код чистим та зменшувати залежності між різними компонентами.

FluentValidation: Це бібліотека для валідації даних. Використовується для визначення правил валідації для моделей даних та автоматичної валідації даних, що передаються в програму. Це дозволяє забезпечити правильність та цілісність даних, які використовуються у програмі.

4.3 Додаткові сервіси та Docker

В цьому розділі ми розглянемо кілька ключових сервісів, які вирішують різноманітні завдання у сфері розробки, контейнеризації та керування застосунками. Серед цих інструментів визначаються Docker, Seq, Portainer та PostgreSQL, які відіграють важливу роль у спрощенні процесів створення, розгортання та моніторингу програмного забезпечення.

4.3.1 Docker

Docker - це платформа, яка дозволяє розробляти, доставляти та запускати застосунки у вигляді контейнерів. Контейнер - це автономний виконуваний пакет програмного забезпечення, який включає в себе все необхідне для роботи програми, таке як код, середовище виконання, бібліотеки та конфігураційні файли.

Одна з головних переваг Docker полягає в тому, що він дозволяє ізолювати застосунки та їх залежності, полегшуючи розгортання та запуск на різних платформах. Використовуючи легкий та ефективний механізм віртуалізації на

рівні операційної системи, Docker дозволяє контейнерам швидко запускатися та спільно використовувати ресурси хост-системи.

Докер надає інструменти для створення, керування та розгортання контейнерів, а також автоматизацію процесів розгортання та масштабування застосунків.

4.3.2 Seq

Seq - це потужний інструмент для збору, аналізу та візуалізації журналів (логів) у реальному часі. Цей інструмент дозволяє розробникам та адміністраторам систем отримувати інсайти щодо роботи застосунків та швидко виявляти можливі проблеми.

Основні особливості Seq включають:

- Реальний час: Seq дозволяє вам переглядати журнали в реальному часі, що є важливим для оперативного виявлення проблем та моніторингу діяльності застосунків.
- Структурований підхід до логів: Seq підтримує структуровані дані у форматі JSON, що робить аналіз журналів більш ефективним і дозволяє швидко фільтрувати та розуміти дані.
- Потужна пошукова система: Інструмент має розширену систему пошуку, яка дозволяє вам ефективно взаємодіяти з великим обсягом журналів і швидко знаходити необхідну інформацію.
- Візуалізація даних: Seq надає можливості візуалізації даних у вигляді графіків та діаграм, що полегшує розуміння тенденцій та аналіз роботи застосунків.
- Розширення можливостей за допомогою плагінів: Seq підтримує розширення за допомогою плагінів, що дозволяє користувачам налаштовувати функціонал під свої потреби.

Seq використовується для виявлення помилок, відстеження діяльності користувачів, аналізу продуктивності та вирішення інших завдань, пов'язаних із збором та аналізом журналів у реальному часі. Особливо важливим є використання Seq у розподілених системах та мікросервісах, де велика кількість компонентів може генерувати значний обсяг журналів.

На рисунку 4.3 представлено Seq сервіс логування для бекенд частини вебпорталу для навчально-наукового інституту.

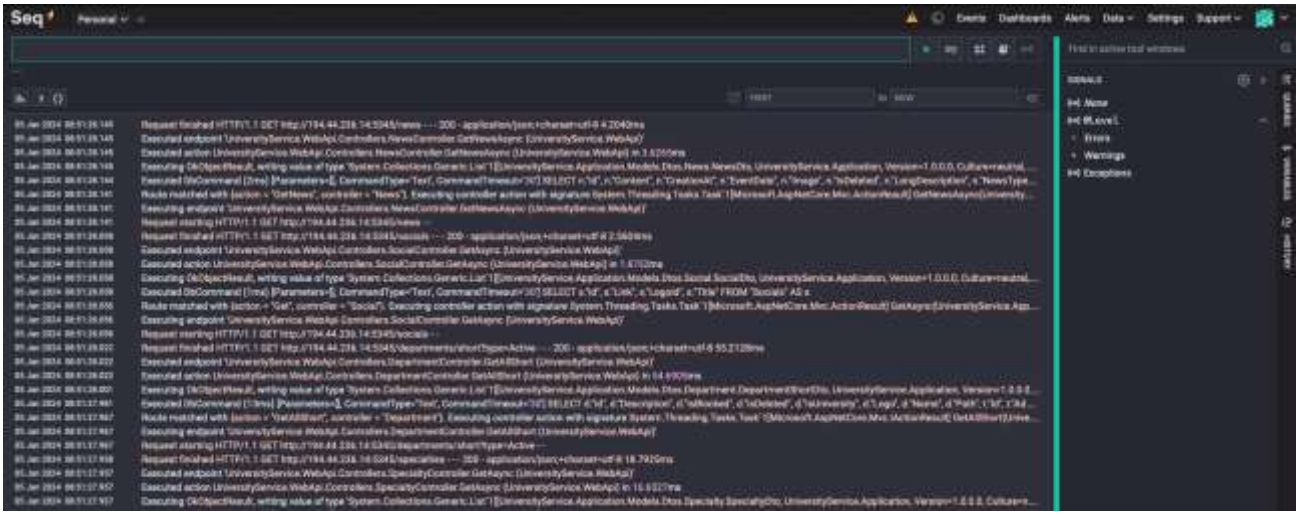

Рисунок 4.3 – Seq сервіс

4.3.3 Portainer

Portainer - це відкрите програмне забезпечення для керування та моніторингу контейнерів у середовищі Docker. Його графічний інтерфейс надає користувачам простий спосіб взаємодії з Docker і використання його можливостей без необхідності введення складних команд через командний рядок. На рисунку 4.4 представлено основу сторінку керування в Portainer.

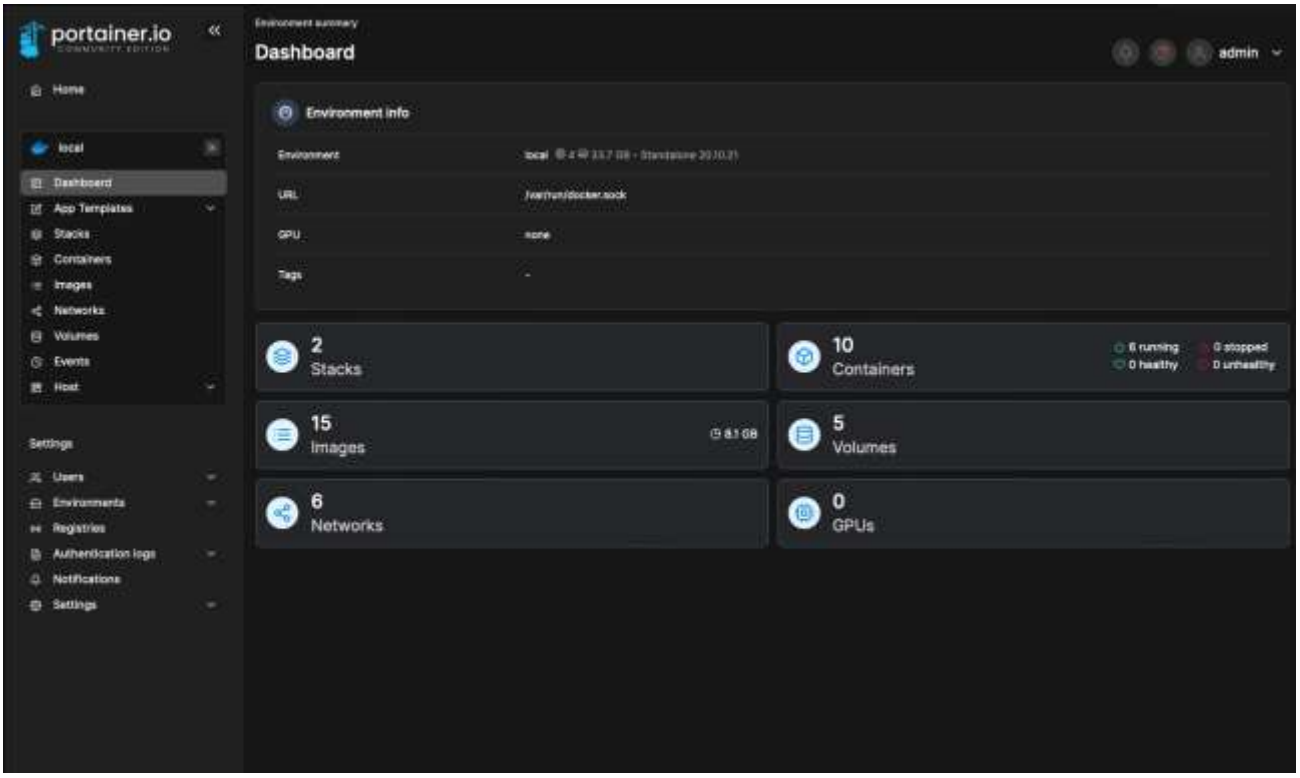

Рисунок 4.4 – Основна сторінка керування в Portainer

Основні функціональні можливості Portainer включають:

- Графічний інтерфейс: Portainer надає інтуїтивно зрозумілий графічний інтерфейс, який дозволяє легко керувати контейнерами, образами, мережами та іншими компонентами Docker.
- Множинні середовища: Ви можете використовувати Portainer для керування контейнерами в різних середовищах, включаючи локальні і віддалені інстанції Docker.
- Моніторинг та статистика: Portainer надає інформацію щодо ресурсів, використовуваних контейнерами, що дозволяє вам відстежувати продуктивність і вчасно виявляти можливі проблеми.
- Управління образами та стеками: Ви можете легко керувати Docker-образами, стеками та іншими ресурсами, використовуючи Portainer.

– Ролі та автентифікація: Portainer підтримує систему ролей і автентифікацію, дозволяючи обмежувати доступ користувачів до конкретних ресурсів.

Portainer робить процеси розгортання та управління контейнерами Docker більш доступними для широкого кола користувачів, зокрема для тих, хто може не мати глибоких знань командного рядка. Це інструмент особливо корисний для адміністраторів систем, розробників та команд, які використовують контейнеризацію для своїх проектів. На рисунку 4.5 зображено список контейнерів в Portainer.

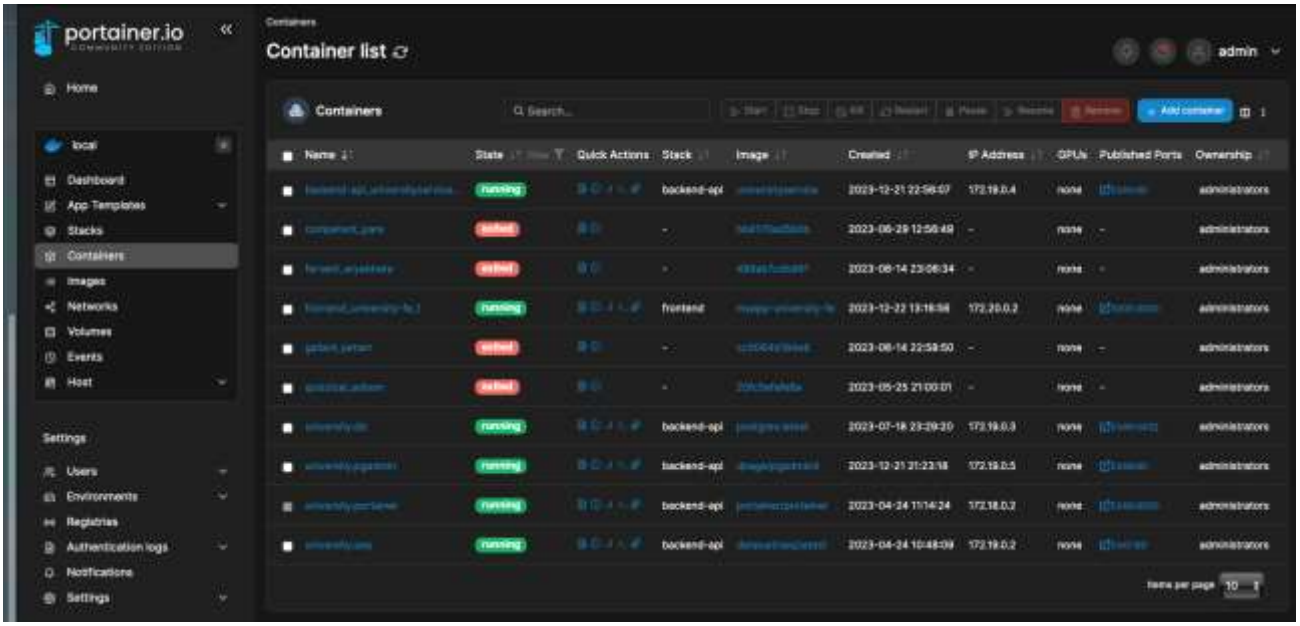

Рисунок 4.5 – Сторінка з контейнерами в Portainer

4.3.4 PgAdmin

PgAdmin - це програмне забезпечення для керування та адміністрування баз даних у системі управління реляційними базами даних PostgreSQL. Забезпечуючи графічний інтерфейс, PgAdmin дозволяє легко керувати серверами та взаємодіяти з базами даних без необхідності використання складних команд SQL.

Основні можливості PgAdmin включають:

- Інтуїтивний інтерфейс: PgAdmin пропонує зрозумілий графічний інтерфейс, що спрощує адміністрування PostgreSQL-серверів та управління базами даних.
- Створення та управління об'єктами бази даних: Користувачі можуть легко створювати нові бази даних, таблиці, індекси та інші об'єкти, а також здійснювати їхнє керування через графічний інтерфейс.
- Виконання SQL-запитів: PgAdmin дозволяє зручно виконувати SQL-запити безпосередньо через інтерфейс для тестування та роботи з даними.
- Моніторинг сервера та аналіз логів: Підтримка моніторингу активності сервера та аналізу логів сприяє ефективному виявленню та вирішенню можливих проблем.
- Управління користувачами та доступом: PgAdmin надає інструменти для керування користувачами, надання ролей та прав доступу до різних об'єктів бази даних.
- Підтримка віддаленого доступу та забезпечення безпеки: PgAdmin дозволяє працювати з віддаленими серверами PostgreSQL та надає засоби для забезпечення безпеки, такі як використання SSL-з'єднань.

PgAdmin спрощує управління базами даних PostgreSQL і використовується адміністраторами баз даних, розробниками та іншими фахівцями для зручної роботи з цією системою управління базами даних.

Використовуючи "PgAdmin", можна здійснити з'єднання з базою даних та дослідити, як саме вони зберігаються. На рисунку 4.6 представлено інтерфейс "PgAdmin".

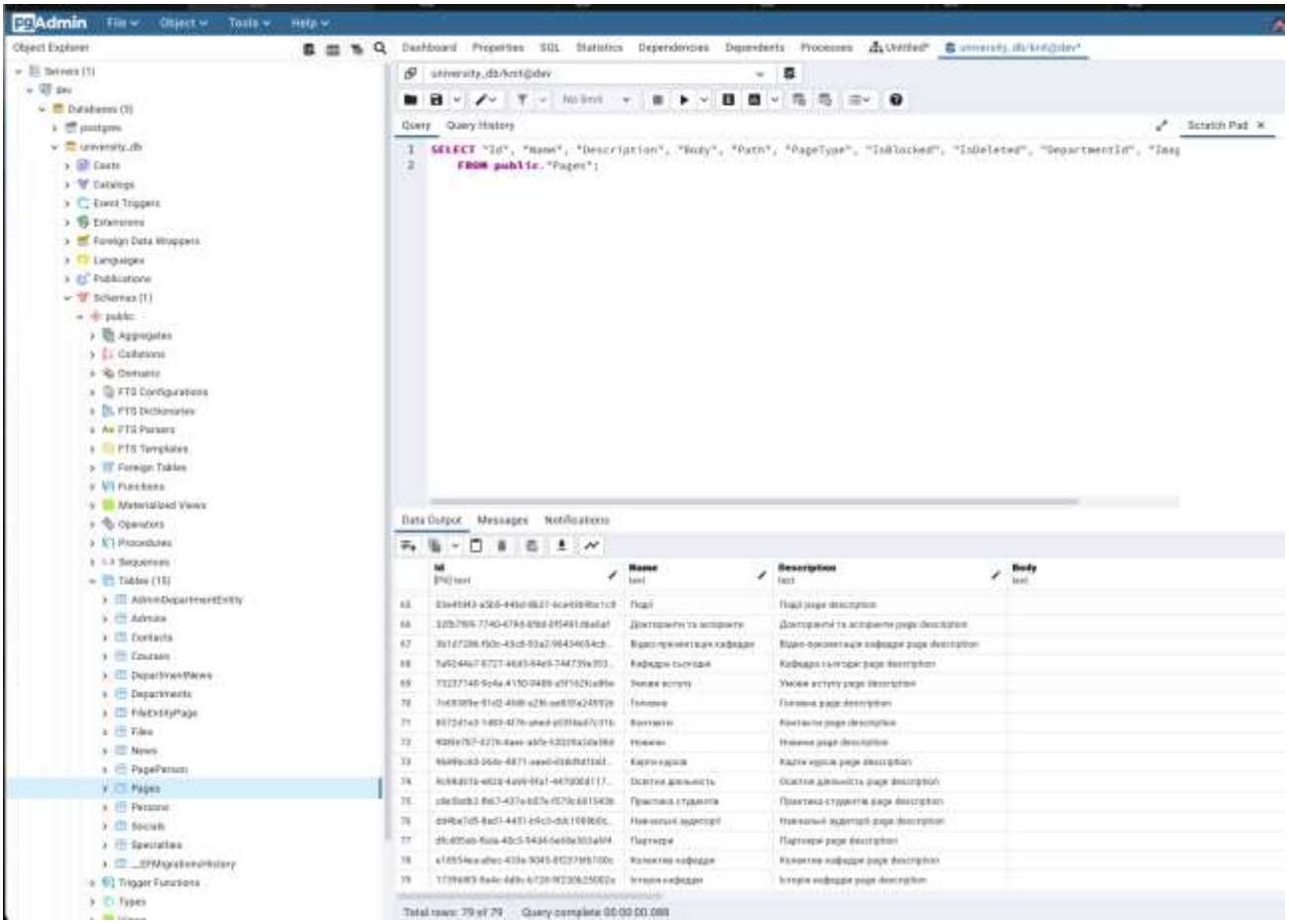

Рисунок 4.6 – PgAdmin сервіс

4.4 Програмна реалізація бази даних

З метою досягнення поставленого завдання було обрано базу даних - "PostgreSQL". PostgreSQL представляє собою високофункціональну систему управління реляційними базами даних (СУБД), яка вирізняється багатством функцій і розширюваністю. Зосереджуючись на стандартизації та дотриманні принципів ACID, PostgreSQL є важливим інструментом в областях з великим обсягом даних, вимогами до високої доступності і потребою у роботі з різноманітними видами інформації.

Основні характеристики PostgreSQL включають:

- Розширені типи даних: Поміж стандартних типів даних PostgreSQL підтримує багато розширених форматів, таких як геометричні, зворотні, та

масиви.

- Множинні індекси та оптимізації: PostgreSQL надає різноманітні типи індексів і оптимізацій для ефективного використання даних в різних випадках.
- Географічні та геоспеціальні функції: Вбудована підтримка географічних об'єктів і операцій, що робить PostgreSQL популярним у геоінформаційних системах.
- Підтримка JSON і інших неструктурованих форматів: PostgreSQL може ефективно зберігати та опрацьовувати неструктуровані дані, такі як JSON.
- Розширені можливості розгортання: Здатний працювати в різних умовах, включаючи великі віддалені системи, кластери та конфігурації високої доступності.
- Розширення та плагіни: Підтримка розширень, що дозволяє розширювати функціональність через власні модулі.
- Ліцензія з відкритим кодом: PostgreSQL поширюється під вільною ліцензією, що надає велику вільність у використанні, модифікації та поширенні коду.

PostgreSQL є популярним вибором для розробників та організацій, оскільки він надає високий рівень надійності та гнучкість для вирішення різноманітних викликів у сфері управління та обробки даних. Це потужна система управління реляційними об'єктами з відкритим вихідним кодом, яка активно розвивається протягом понад 30 років і зарекомендувала себе як надійна, функціональна та продуктивна.

На рисунку 4.7 представлено Entity-Relationship Diagram (ERD) бази даних для бекенд частини вебпорталу для навчально-наукового інституту.

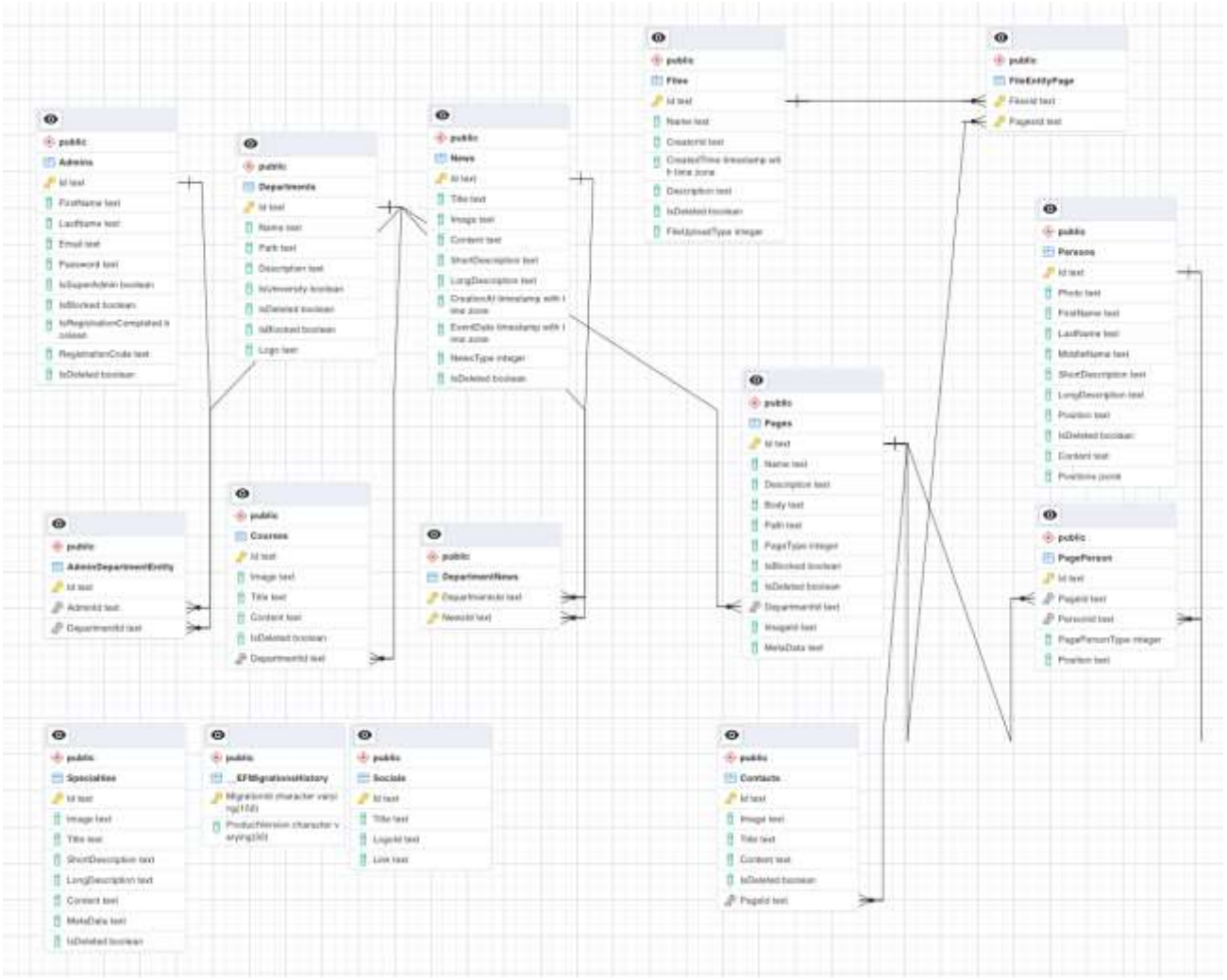

Рисунок 4.7 – Структура бази даних бекенд частини вебпорталу для навчально-наукового інституту

4.5 Безперервна інтеграція

Команди з великим досвідом інтенсивно застосовують автоматизацію інфраструктури при розробці додатків.

Платформи Continuous Integration та Continuous Delivery [11, 12], які впроваджують цю концепцію, підтримують регулярне автоматизоване збирання проекту для оперативного виявлення дефектів та вирішення інтеграційних проблем. У стандартному підході, де розробники працюють незалежно над різними частинами системи, стадія інтеграції є заключною, і виявлення помилок може несподівано затримати завершення проекту. Перехід до безперервної

інтеграції дозволяє знизити складність роботи та зробити її більш передбачуваною, завдяки ранньому та безперервному виявленню та виправленню помилок і протиріч.

Концепція Continuous Integration та Continuous Delivery втілює ідеологію злиття розробки та експлуатації ПЗ (DevOps) та відповідає основним принципам Agile, що стосуються використання автоматичного тестування для швидкого налаштування робочої версії програмного забезпечення.

Основна проблема при переході до Continuous Integration та Continuous Delivery полягає в адаптації підходу, де процеси визначаються на першому плані, а технології відводяться на другий план. Необхідно створювати нові процеси, визначати нові ролі людей і знаходити точки інтеграції між існуючими та новими процесами. Зміщення акценту з програмного забезпечення та апаратного забезпечення на людей і організаційні аспекти стає серйозним викликом для багатьох.

Інструменти Continuous Integration допомагають швидко знаходити причини дефектів для кожного коміту, забезпечуючи раннє виявлення та усунення помилок. Платформа Continuous Integration та Continuous Delivery не лише допомагає ефективно тестувати, але й впроваджувати новий функціонал для кінцевого користувача так, щоб у разі виявлення помилок завжди була можливість або швидко їх усунути, або "відкотити" ітерацію рішення на крок назад.

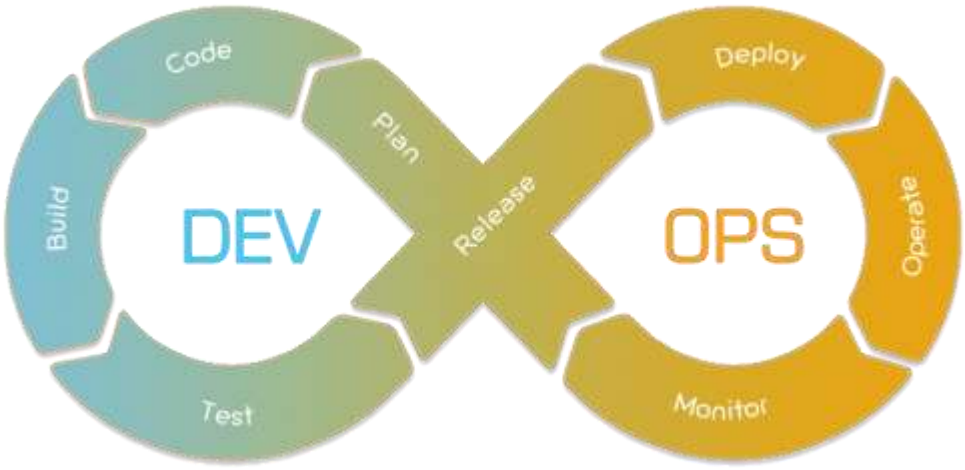

Рисунок 4.8 – Постійна інтеграція / Постійне впровадження

Цей процес подібний до життєвого циклу розробки програмного забезпечення, що зображений на рисунку 4.9.

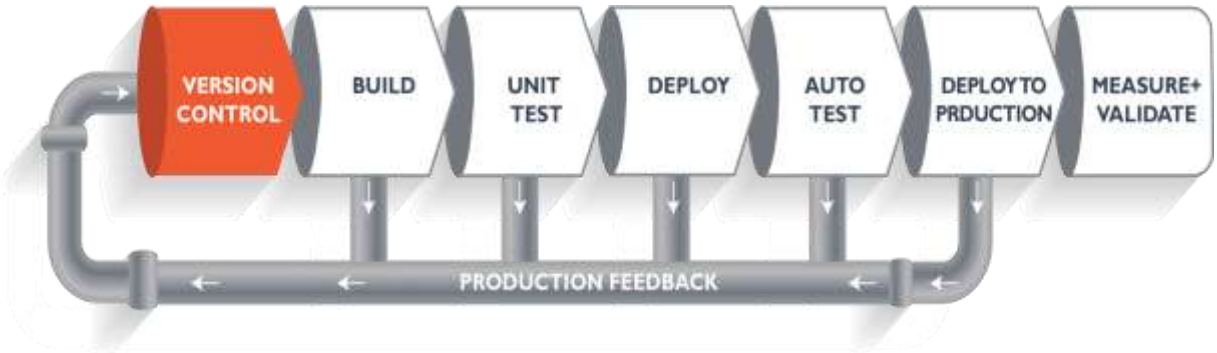

Рисунок 4.9 – Життєвий цикл розробки програмного забезпечення

Завдання полягає у автоматизації всього процесу, з моменту, коли команда розробників передає код і здійснює його до того моменту, коли його отримаємо у виробництво. Конвеєр автоматизовується, щоб зробити весь життєвий цикл розробки програмного забезпечення в DevOps / автоматизованому режимі. Для цього нам знадобляться засоби автоматизації. На даний момент є багато засобів

автоматизації, найпопулярніші з них це Jenkins та GitLab piplines. GitLab piplines зображено на рисунку 4.10.

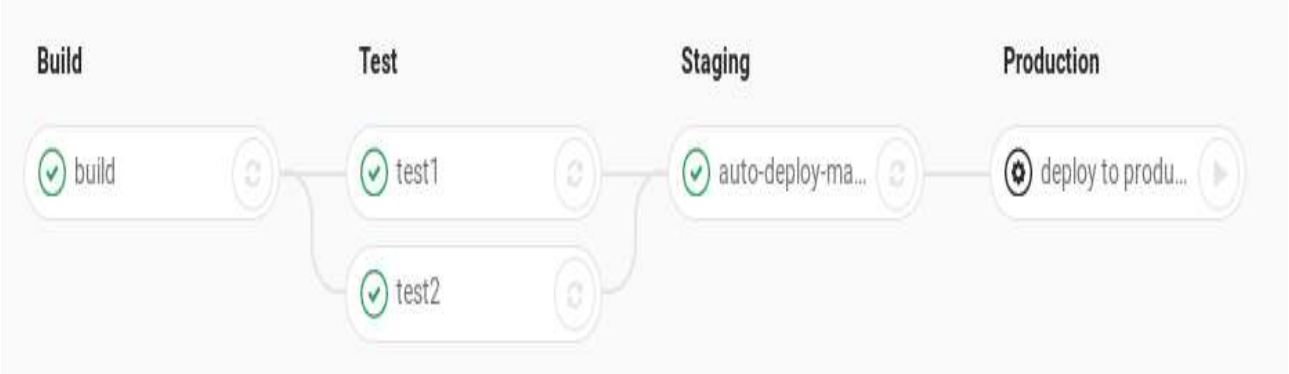

Рисунок 4.10 – GitLab piplines

Висновки до розділу

Аналізуючи загальну структуру програмного коду та використані компоненти, можна визначити, що проект ґрунтується на сучасних та ефективних підходах до розробки програмного забезпечення. Використання принципів DDD та "Чистої Архітектури" свідчить про спрямованість на збереження бізнес-логіки в центрі розробки, що сприяє високій розширюваності та легкості тестування.

Структура, побудована на основі DDD, призначена для відображення домену додатку у коді, що дозволяє зберігати логіку бізнес-процесів чітко та систематично. Принцип "Чистої Архітектури" визначає чітку ієрархію шарів та їх взаємодію, забезпечуючи незалежність різних компонентів та можливість зміни одного шару без впливу на інші.

Щодо використаних сторонніх бібліотек та модулів, вони вибрані з урахуванням потреб проекту. Entity Framework Core використовується для роботи з базою даних, AutoMapper допомагає в автоматизованому мапінгу об'єктів, Swagger надає інструменти для автоматичної генерації та

документування API, MediatR служить для впровадження паттерну Mediator для відокремлення обробки запитів та команд, а FluentValidation використовується для зручної валідації даних.

Розглянуті сервіси, такі як Docker, Seq, Portainer та PostgreSQL, представляють сучасні інструменти, спрямовані на полегшення розробки, моніторингу та управління різноманітними аспектами інформаційних систем. Docker дозволяє упаковувати та ізолювати застосунки в контейнери для консистентного та ефективного розгортання. Seq надає потужний інструмент для аналізу та візуалізації логів, сприяючи вчасному виявленню проблем. Portainer, зі своїм графічним інтерфейсом, спрощує керування контейнерами Docker, забезпечуючи зручність у взаємодії. PostgreSQL, зі своєю реляційною базою даних, вирізняється розширеними можливостями для зберігання та обробки даних. Ці сервіси, взаємодіючи, формують комплексний інструментарій для створення, оптимізації та підтримки сучасних інформаційних рішень.

Проект має добре структуровану архітектуру, готову до розширення та легкої тестованості, а використані сторонні бібліотеки відповідають сучасним стандартам та сприяють ефективній розробці.

РОЗДІЛ 5. РОЗРОБЛЕННЯ СТАРТАП-ПРОЄКТУ

5.1 Опис ідеї проєкту

Проєкт полягає у розробці бекенд-частини вебпорталу для освітньо-наукового інституту. Він складається з адміністративної панелі для управління вебсайтом університету або школи. Ця платформа дозволяє добавляти нові сторінки, публікувати новини, керувати інформацією про кафедри та персонал. Основною перевагою є гнучкість та можливість швидко адаптувати контент, що робить її корисною для динамічного освітнього середовища. Цей проект сприяє покращенню комунікації та представлення освітнього закладу в інтернеті, забезпечуючи актуальність та релевантність інформації на сайті. Його мета - поліпшити доступність та ефективність використання освітніх та наукових ресурсів через сучасний веб-інтерфейс.

Проект важливий, оскільки він сприяє підвищенню якості освітнього процесу та наукових досліджень, забезпечуючи студентам та викладачам швидкий і безпечний доступ до необхідних матеріалів та інструментів. Також він сприяє оптимізації управління вебсайтом освітніх установ, надаючи зручні інструменти для швидкого оновлення інформації та забезпечення її актуальності.

Загальні напрямки використання проєкту включають:

- Управління контентом вебсайту: Можливість додавати та оновлювати сторінки, публікувати новини, забезпечуючи своєчасне та релевантне інформування студентів та викладачів.
- Керування інформацією про кафедри: Актуалізація даних про науково-педагогічний склад, кафедри та навчальні програми, підтримуючи їхню точність та актуальність.
- Комунікація зі студентами та спільнотою: Підтримка ефективного діалогу зі спільнотою через регулярні оновлення та анонси подій, що сприяють залученню аудиторії.

- Поліпшення доступу до освітніх ресурсів: Забезпечення легкого та зручного доступу до навчальних матеріалів, лекцій, та наукових публікацій.
- Адаптація до змін у навчальному процесі: Гнучке внесення змін на сайт відповідно до еволюції освітніх програм та методик.

5.2 Розроблення ринкової стратегії

Ринкова стратегія буде базуватися на маркетинговій програмі. Але є ще деякі основні інструменти, які будуть у ринковій стратегії.

Надсилання електронних листів

Email маркетинг дозволяє ефективно звертатися до потенційних клієнтів з персоналізованим зверненням. Використання цієї стратегії може включати регулярні інформаційні бюлетені, спеціальні пропозиції, оновлення продуктів, а також запрошення до участі в опитуваннях або заходах. Ефективність цього підходу залежить від збору відповідних контактних даних, розробки цікавого контенту та здатності адаптувати повідомлення під потреби кожного отримувача. Також важливим є відстеження відгуку клієнтів для покращення майбутніх кампаній.

Створення блогу

Створення блогу як частини маркетингової програми включає вибір тематики, яка відповідає інтересам цільової аудиторії, регулярне публікування змістовного контенту, SEO-оптимізацію для підвищення видимості в пошукових системах, сприяння інтерактивності та залучення аудиторії через коментарі. Важливо також інтегрувати блог з іншими маркетинговими каналами та аналізувати ефективність для покращення майбутніх публікацій.

Публікації в соціальних мережах

Маркетинг у соціальних мережах вимагає вибору платформ, які найкраще підходять для вашої цільової аудиторії. Створення привабливого контенту, який

резонує з інтересами аудиторії, є ключовим. Важливо дотримуватися регулярного графіка публікацій, щоб підтримувати активну присутність у мережі. Ефективна взаємодія з аудиторією, включаючи відповіді на коментарі та запитання, допомагає підтримувати залученість. Моніторинг та аналіз допомагають визначити ефективність публікацій та вдосконалити стратегію.

5.3 Розроблення маркетингової програми

Розроблення маркетингової програми включає визначення цілей, аналіз цільової аудиторії, вибір маркетингових каналів (як онлайн, так і офлайн), розробку контенту, встановлення бюджету та графіка кампанії, реалізацію стратегії, моніторинг результатів і постійне вдосконалення програми. Важливо інтегрувати різні методи, такі як емейл-маркетинг, соціальні медіа, блоги, SEO, та партнерські взаємодії, для створення комплексного підходу.

Концепція товару

Концепція товару проекту полягає в створенні інноваційної адміністративної панелі для управління вебсайтами освітніх закладів. Цей продукт забезпечує гнучкість та легкість у додаванні та оновленні контенту, новин, інформації про кафедри та персонал. Він спрощує процес управління веб-ресурсами, роблячи їх більш актуальними та відповідними до сучасних освітніх потреб.

Збут та продажі

Збут та продажі проекту можуть бути організовані через прямі звернення до освітніх закладів, участь у освітніх виставках та конференціях, а також через онлайн-маркетинг. Продажі можуть включати демонстрацію продукту потенційним клієнтам, надання індивідуальних пропозицій та підтримку клієнтів після продажу. Важливо також використовувати стратегії залучення через партнерства з освітніми організаціями та впливовими особами в галузі освіти.

Просування

Просування проекту може включати використання онлайн-маркетингу, такого як соціальні мережі, емейл-маркетинг, блоги та SEO. Також важливим є участь у професійних освітніх виставках та конференціях. Стратегія може включати створення партнерських відносин з освітніми організаціями та лідерами думок. Використання кейс-стаді та відгуків задоволених клієнтів також може підвищити довіру до продукту.

Висновки до розділу

Важливо розуміти, що успіх стартап-проєкту, який фокусується на розробці вебпорталу в освітній сфері, залежить не лише від інноваційності ідеї, а й від чіткості ринкової стратегії та ефективності маркетингу. Цей проєкт має потенціал вирішити сучасні виклики, пропонуючи високоефективне рішення для ефективного управління освітніми ресурсами. Ринкова стратегія та маркетингова програма повинні бути спрямовані на залучення цільової аудиторії, забезпечуючи швидкий вихід на ринок та стабільний розвиток проєкту. Крім того, необхідно враховувати технічні та безпекові аспекти, оскільки стабільність, гнучкість та безпека системи є важливими для її успішного функціонування та масштабування в майбутньому.

ВИСНОВКИ

Під час розробки бекенд частини вебпорталу було проведено широкий спектр робіт з розробки та інтеграції різних компонентів і функціональностей. Основним завданням було створення потужного та масштабованого сервісу, який забезпечує надійну роботу та ефективну обробку запитів вебпорталу.

У процесі розробки було використано принципи та концепції "Чистої архітектури" та "DDD" (Domain-Driven Design). Це дозволило розбити систему на окремі модулі та компоненти, які мають чітко визначені обов'язки та взаємодіють один з одним через доменні моделі. Використання цих підходів сприяло зменшенню залежностей та забезпечило більшу гнучкість, розширюваність та тестируваність коду.

Для реалізації бекенд частини вебпорталу було використано різноманітні бібліотеки та фреймворки. Наприклад, Entity Framework Core був використаний для забезпечення доступу до бази даних та управління моделями даних. Це дозволило зручно працювати з базою даних, виконувати запити, забезпечувати міграції та інші операції.

Крім того, для реалізації API та обробки запитів було використано фреймворк ASP.NET Core. Він надав зручний механізм для реалізації роутингу, контролерів, фільтрів та інших компонентів, необхідних для побудови потужного та функціонального API.

Завдяки використанню цих технологій та інструментів, бекенд частина вебпорталу була успішно розроблена, забезпечуючи високу продуктивність, масштабованість та надійність системи.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. C# Programming Guide [Електронний ресурс]. – Режим доступу: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/.
2. ASP.NET Core documentation [Електронний ресурс]. – Режим доступу: https://docs.microsoft.com/en-us/aspnet/core/.
3. .NET Core documentation [Електронний ресурс]. – Режим доступу: https://docs.microsoft.com/en-us/dotnet/core/.
4. Entity Framework Core documentation [Електронний ресурс]. – Режим доступу: https://docs.microsoft.com/en-us/ef/core/.
5. Swagger documentation [Електронний ресурс]. – Режим доступу: https://swagger.io/.
6. Domain-Driven Design [Електронний ресурс]. – Режим доступу: https://martinfowler.com/tags/domain-driven-design.html.
7. Clean Architecture [Електронний ресурс]. – Режим доступу: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html.
8. Microservices [Електронний ресурс]. – Режим доступу: https://microservices.io/.
9. SOLID principles [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/SOLID.
10. RESTful API design [Електронний ресурс]. – Режим доступу: https://restfulapi.net/.
11. Continuous Integration (CI) [Електронний ресурс]. – Режим доступу: https://www.atlassian.com/continuous-delivery/continuous-integration.
12. Continuous Deployment (CD) [Електронний ресурс]. – Режим доступу: https://www.atlassian.com/continuous-delivery/continuous-deployment.
13. Bcrypt.Net [Електронний ресурс]. – Режим доступу: https://github.com/BcryptNet/bcrypt.net.

ДОДАТКИ

ДОДАТОК А

Додавання сервісів

```
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers()
    .AddJsonOptions(options =>
    {
        options.JsonSerializerOptions.Converters.Add(new JsonStringEnumConverter());
    });

builder.Services.AddAutoMapper(typeof(AuthorizeAdminRequest).Assembly);
builder.Services.AddFluentValidationAutoValidation();
builder.Services.AddValidatorsFromAssemblyContaining<AuthorizeAdminRequest>();

builder.Services.AddMediatR(typeof(AuthorizeAdminRequest));

builder.Services.Configure<JwtInfo>(builder.Configuration.GetSection("Jwt"));
builder.Services.Configure<EmailSettingsConfiguration>(builder.Configuration.GetSection("EmailSettings"));
builder.Services.Configure<FileStorageInfo>(builder.Configuration.GetSection("FileStorage"));

builder.Services.AddDbContext<UniversityContext>(options =>
            options.UseNpgsql(builder.Configuration.GetConnectionString("University")));

builder.Services.AddSingleton<IEnailManager, EmailManager>();

builder.Services.AddScoped<IAdminRepository, AdminRepository>();
builder.Services.AddScoped<IDepartmentRepository, DepartmentRepository>();
builder.Services.AddScoped<IFileRepository, FileRepository>();
builder.Services.AddScoped<ITagRepository, TagRepository>();
builder.Services.AddScoped<IPageRepository, PageRepository>();
builder.Services.AddScoped<IPersonRepository, PersonRepository>();

builder.Services.AddScoped<IJwtTokenService, JwtTokenService>();
builder.Services.AddScoped<IClaimsService, ClaimsService>();
builder.Services.AddScoped<IStorageService, StorageService>();

builder.Services.ConfigureAuthentication(builder.Configuration["Jwt:Secret"]);
builder.Services.AddAuthorization();

builder.Services.AddScoped<UniversitySession>();
builder.Services.AddHttpContextAccessor();

builder.Services.AddLogger(builder.Configuration);
builder.Services.Configure<ApiBehaviorOptions>(options => options.SuppressInferBindingSourcesForParameters = true);


// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
```

```
var app = builder.Build();

await app.AutoMigrateAsync<UniversityContext>();

app.UseCors(x => x.AllowAnyHeader().AllowAnyMethod().AllowAnyOrigin());

app.UseRouting();

app.UseAuthentication();
app.UseAuthorization();

app.UseMiddleware<SessionMiddleware>();
app.UseCors(x => x.AllowAnyHeader().AllowAnyMethod().AllowAnyOrigin());

app.UseSwagger();
app.UseSwaggerUI();

app.UseHttpsRedirection();

app.MapControllers();

app.Run();
```

ДОДАТОК Б

Скрипт (міграція) створення бази даних

```
migrationBuilder.CreateTable(
                name: "Admins",
                columns: table => new
                {
                    Id = table.Column<string>(type: "text", nullable: false),
                    FirstName = table.Column<string>(type: "text", nullable:
true),
                    LastName = table.Column<string>(type: "text", nullable:
true),
                    Email = table.Column<string>(type: "text", nullable:
false),
                    Password = table.Column<string>(type: "text", nullable:
true),
                    RegistrationCode = table.Column<string>(type: "text",
nullable: false),
                    IsRegistrationCompleted = table.Column<bool>(type:
"boolean", nullable: false),
                    IsSuperAdmin = table.Column<bool>(type: "boolean",
nullable: false),
                    IsBlocked = table.Column<bool>(type: "boolean", nullable:
false),
                    IsDeleted = table.Column<bool>(type: "boolean", nullable:
false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Admins", x => x.Id);
                });

            migrationBuilder.CreateTable(
                name: "Departments",
                columns: table => new
                {
                    Id = table.Column<string>(type: "text", nullable: false),
                    Logo = table.Column<string>(type: "text", nullable: true),
                    Name = table.Column<string>(type: "text", nullable: false),
                    Path = table.Column<string>(type: "text", nullable: false),
                    Description = table.Column<string>(type: "text", nullable:
true),
                    IsUniversity = table.Column<bool>(type: "boolean",
nullable: false),
                    IsBlocked = table.Column<bool>(type: "boolean", nullable:
false),
                    IsDeleted = table.Column<bool>(type: "boolean", nullable:
false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Departments", x => x.Id);
                });
```

```
            migrationBuilder.CreateTable(
                name: "Files",
                columns: table => new
                {
                    Id = table.Column<string>(type: "text", nullable: false),
                    Name = table.Column<string>(type: "text", nullable: false),
                    Description = table.Column<string>(type: "text", nullable:
true),
                    FileUploadType = table.Column<int>(type: "integer",
nullable: false),
                    CreatorId = table.Column<string>(type: "text", nullable:
false),
                    CreatedTime = table.Column<DateTime>(type: "timestamp with
time zone", nullable: false),
                    IsDeleted = table.Column<bool>(type: "boolean", nullable:
false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Files", x => x.Id);
                });
            migrationBuilder.CreateTable(
                name: "News",
                columns: table => new
                {
                    Id = table.Column<string>(type: "text", nullable: false),
                    Title = table.Column<string>(type: "text", nullable:
false),
                    Image = table.Column<string>(type: "text", nullable: true),
                    Content = table.Column<string>(type: "text", nullable:
false),
                    ShortDescription = table.Column<string>(type: "text",
nullable: true),
                    LongDescription = table.Column<string>(type: "text",
nullable: true),
                    CreationAt = table.Column<DateTime>(type: "timestamp with
time zone", nullable: false),
                    EventDate = table.Column<DateTime>(type: "timestamp with
time zone", nullable: true),
                    NewsType = table.Column<int>(type: "integer", nullable:
false),
                    IsDeleted = table.Column<bool>(type: "boolean", nullable:
false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_News", x => x.Id);
                });
            migrationBuilder.CreateTable(
                name: "Persons",
                columns: table => new
                {
                    Id = table.Column<string>(type: "text", nullable: false),
                    Photo = table.Column<string>(type: "text", nullable: true),
                    FirstName = table.Column<string>(type: "text", nullable:
```

```
false),
                    LastName = table.Column<string>(type: "text", nullable:
false),
                    MiddleName = table.Column<string>(type: "text", nullable:
false),
                    ShortDescription = table.Column<string>(type: "text",
nullable: false),
                    LongDescription = table.Column<string>(type: "text",
nullable: false),
                    Content = table.Column<string>(type: "text", nullable:
false),
                    Position = table.Column<string>(type: "text", nullable:
false),
                    Positions = table.Column<List<string>>(type: "jsonb",
nullable: true),
                    IsDeleted = table.Column<bool>(type: "boolean", nullable:
false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Persons", x => x.Id);
                });

            migrationBuilder.CreateTable(
                name: "Socials",
                columns: table => new
                {
                    Id = table.Column<string>(type: "text", nullable: false),
                    Title = table.Column<string>(type: "text", nullable:
false),
                    LogoId = table.Column<string>(type: "text", nullable:
false),
                    Link = table.Column<string>(type: "text", nullable: false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Socials", x => x.Id);
                });

            migrationBuilder.CreateTable(
                name: "Specialties",
                columns: table => new
                {
                    Id = table.Column<string>(type: "text", nullable: false),
                    Image = table.Column<string>(type: "text", nullable: true),
                    Title = table.Column<string>(type: "text", nullable:
false),
                    ShortDescription = table.Column<string>(type: "text",
nullable: true),
                    LongDescription = table.Column<string>(type: "text",
nullable: true),
                    Content = table.Column<string>(type: "text", nullable:
true),
                    MetaData = table.Column<string>(type: "text", nullable:
true),
```

```
                    IsDeleted = table.Column<bool>(type: "boolean", nullable:
false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Specialties", x => x.Id);
                });

            migrationBuilder.CreateTable(
                name: "AdminDepartmentEntity",
                columns: table => new
                {
                    Id = table.Column<string>(type: "text", nullable: false),
                    AdminId = table.Column<string>(type: "text", nullable:
false),
                    DepartmentId = table.Column<string>(type: "text", nullable:
false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_AdminDepartmentEntity", x => x.Id);
                    table.ForeignKey(
                        name: "FK_AdminDepartmentEntity_Admins_AdminId",
                        column: x => x.AdminId,
                        principalTable: "Admins",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
                    table.ForeignKey(
                        name:
"FK_AdminDepartmentEntity_Departments_DepartmentId",
                        column: x => x.DepartmentId,
                        principalTable: "Departments",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
                });

            migrationBuilder.CreateTable(
                name: "Courses",
                columns: table => new
                {
                    Id = table.Column<string>(type: "text", nullable: false),
                    Image = table.Column<string>(type: "text", nullable: true),
                    Title = table.Column<string>(type: "text", nullable:
false),
                    Content = table.Column<string>(type: "text", nullable:
false),
                    IsDeleted = table.Column<bool>(type: "boolean", nullable:
false),
                    DepartmentId = table.Column<string>(type: "text", nullable:
true)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Courses", x => x.Id);
```

```
                    table.ForeignKey(
                        name: "FK_Courses_Departments_DepartmentId",
                        column: x => x.DepartmentId,
                        principalTable: "Departments",
                        principalColumn: "Id");
                });
            migrationBuilder.CreateTable(
                name: "Pages",
                columns: table => new
                {
                    Id = table.Column<string>(type: "text", nullable: false),
                    Name = table.Column<string>(type: "text", nullable: false),
                    Description = table.Column<string>(type: "text", nullable:
true),
                    ImageId = table.Column<string>(type: "text", nullable:
true),
                    Body = table.Column<string>(type: "text", nullable: false),
                    Path = table.Column<string>(type: "text", nullable: false),
                    MetaData = table.Column<string>(type: "text", nullable:
true),
                    PageType = table.Column<int>(type: "integer", nullable:
false),
                    IsBlocked = table.Column<bool>(type: "boolean", nullable:
false),
                    IsDeleted = table.Column<bool>(type: "boolean", nullable:
false),
                    DepartmentId = table.Column<string>(type: "text", nullable:
false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Pages", x => x.Id);
                    table.ForeignKey(
                        name: "FK_Pages_Departments_DepartmentId",
                        column: x => x.DepartmentId,
                        principalTable: "Departments",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
                });

            migrationBuilder.CreateTable(
                name: "DepartmentNews",
                columns: table => new
                {
                    DepartmentsId = table.Column<string>(type: "text",
nullable: false),
                    NewsId = table.Column<string>(type: "text", nullable:
false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_DepartmentNews", x => new {
x.DepartmentsId, x.NewsId });
                    table.ForeignKey(
```

```
                        name: "FK_DepartmentNews_Departments_DepartmentsId",
                        column: x => x.DepartmentsId,
                        principalTable: "Departments",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
                    table.ForeignKey(
                        name: "FK_DepartmentNews_News_NewsId",
                        column: x => x.NewsId,
                        principalTable: "News",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
                });
            migrationBuilder.CreateTable(
                name: "Contacts",
                columns: table => new
                {
                    Id = table.Column<string>(type: "text", nullable: false),
                    Image = table.Column<string>(type: "text", nullable: true),
                    Title = table.Column<string>(type: "text", nullable:
false),
                    Content = table.Column<string>(type: "text", nullable:
false),
                    IsDeleted = table.Column<bool>(type: "boolean", nullable:
false),
                    PageId = table.Column<string>(type: "text", nullable: true)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Contacts", x => x.Id);
                    table.ForeignKey(
                        name: "FK_Contacts_Pages_PageId",
                        column: x => x.PageId,
                        principalTable: "Pages",
                        principalColumn: "Id");
                });
            migrationBuilder.CreateTable(
                name: "FileEntityPage",
                columns: table => new
                {
                    FilesId = table.Column<string>(type: "text", nullable:
false),
                    PagesId = table.Column<string>(type: "text", nullable:
false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_FileEntityPage", x => new { x.FilesId,
x.PagesId });
                    table.ForeignKey(
                        name: "FK_FileEntityPage_Files_FilesId",
                        column: x => x.FilesId,
                        principalTable: "Files",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
```

```
                    table.ForeignKey(
                        name: "FK_FileEntityPage_Pages_PagesId",
                        column: x => x.PagesId,
                        principalTable: "Pages",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
                });
            migrationBuilder.CreateTable(
                name: "PagePerson",
                columns: table => new
                {
                    Id = table.Column<string>(type: "text", nullable: false),
                    PageId = table.Column<string>(type: "text", nullable:
false),
                    PersonId = table.Column<string>(type: "text", nullable:
false),
                    PagePersonType = table.Column<int>(type: "integer",
nullable: false),
                    Position = table.Column<string>(type: "text", nullable:
true)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_PagePerson", x => x.Id);
                    table.ForeignKey(
                        name: "FK_PagePerson_Pages_PageId",
                        column: x => x.PageId,
                        principalTable: "Pages",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
                    table.ForeignKey(
                        name: "FK_PagePerson_Persons_PersonId",
                        column: x => x.PersonId,
                        principalTable: "Persons",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
                });
migrationBuilder.InsertData(
                table: "Departments",
                columns: new[] { "Id", "Description", "IsBlocked", "IsDeleted",
"IsUniversity", "Logo", "Name", "Path" },
                values: new object[] { "088e46e2-1111-0000-1111-865313c9a21d",
"Description of University", false, false, true, null, "Name of University",
"university" });
            migrationBuilder.InsertData(
                table: "AdminDepartmentEntity",
                columns: new[] { "Id", "AdminId", "DepartmentId" },
                values: new object[] { "088e46e2-0000-3333-1111-865313c9a21d",
"088e46e2-1111-1111-1111-865313c9a21d", "088e46e2-1111-0000-1111-865313c9a21d"
});
            migrationBuilder.InsertData(
                table: "Pages",
                columns: new[] { "Id", "Body", "DepartmentId", "Description",
"ImageId", "IsBlocked", "IsDeleted", "MetaData", "Name", "PageType", "Path" },
```

```
                values: new object[,]
                {
                    { "088e46e2-1001-1111-1111-865313c9a21d", "", "088e46e2-
1111-0000-1111-865313c9a21d", "Description of page Головна", null, false,
false, null, "Головна", 0, "main" },
                    { "088e46e2-1002-1111-1111-865313c9a21d", "", "088e46e2-
1111-0000-1111-865313c9a21d", "Description of page Новини", null, false, false,
null, "Новини", 0, "news" },
                    { "088e46e2-1003-1111-1111-865313c9a21d", "", "088e46e2-
1111-0000-1111-865313c9a21d", "Description of page Події", null, false, false,
null, "Події", 0, "events" },
                    { "088e46e2-1004-1111-1111-865313c9a21d", "", "088e46e2-
1111-0000-1111-865313c9a21d", "Description of page Умови вступу", null, false,
false, null, "Умови вступу", 0, "conditions-of-entry" },
                    { "088e46e2-1005-1111-1111-865313c9a21d", "", "088e46e2-
1111-0000-1111-865313c9a21d", "Description of page Варстіть навчання", null,
false, false, null, "Варстіть навчання", 0, "cost-of-education" },
                    { "088e46e2-1006-1111-1111-865313c9a21d", "", "088e46e2-
1111-0000-1111-865313c9a21d", "Description of page Історія інституту", null,
false, false, null, "Історія інституту", 0, "history" },
                    { "088e46e2-1007-1111-1111-865313c9a21d", "", "088e46e2-
1111-0000-1111-865313c9a21d", "Description of page Дирекція", null, false,
false, null, "Дирекція", 0, "directorate" },
                    { "088e46e2-1008-1111-1111-865313c9a21d", "", "088e46e2-
1111-0000-1111-865313c9a21d", "Description of page Наукова діяльність", null,
false, false, null, "Наукова діяльність", 0, "research-activities" },
                    { "088e46e2-1009-1111-1111-865313c9a21d", "", "088e46e2-
1111-0000-1111-865313c9a21d", "Description of page Міжнародна діяльність",
null, false, false, null, "Міжнародна діяльність", 0, "international-activity"
},
                    { "088e46e2-1010-1111-1111-865313c9a21d", "", "088e46e2-
1111-0000-1111-865313c9a21d", "Description of page Вчена рада інституту", null,
false, false, null, "Вчена рада інституту", 0, "academic-council" },
                    { "088e46e2-1011-1111-1111-865313c9a21d", "", "088e46e2-
1111-0000-1111-865313c9a21d", "Description of page Науково методична рада",
null, false, false, null, "Науково методична рада", 0, "scientific-methodical-
council" },
                    { "088e46e2-1012-1111-1111-865313c9a21d", "", "088e46e2-
1111-0000-1111-865313c9a21d", "Description of page Відомі випускники", null,
false, false, null, "Відомі випускники", 0, "famous-graduates" },
                    { "088e46e2-1013-1111-1111-865313c9a21d", "", "088e46e2-
1111-0000-1111-865313c9a21d", "Description of page Розклад", null, false,
false, null, "Розклад", 0, "study-schedule" },
                    { "088e46e2-1014-1111-1111-865313c9a21d", "", "088e46e2-
1111-0000-1111-865313c9a21d", "Description of page Студентська рада", null,
false, false, null, "Студентська рада", 0, "student-council" },
                    { "088e46e2-1015-1111-1111-865313c9a21d", "", "088e46e2-
1111-0000-1111-865313c9a21d", "Description of page Списки груп і наставники",
null, false, false, null, "Списки груп і наставники", 0, "list-of-groups" },
                    { "088e46e2-1016-1111-1111-865313c9a21d", "", "088e46e2-
1111-0000-1111-865313c9a21d", "Description of page Рейтингові списки,
стипендія", null, false, false, null, "Рейтингові списки, стипендія", 0,
"rating-lists" },
                    { "088e46e2-1017-1111-1111-865313c9a21d", "", "088e46e2-
1111-0000-1111-865313c9a21d", "Description of page Гуртожиток ІКНІТ", null,
false, false, null, "Гуртожиток ІКНІТ", 0, "dormitory" },
                    { "088e46e2-1018-1111-1111-865313c9a21d", "", "088e46e2-
```

```
1111-0000-1111-865313c9a21d", "Description of page Довідки, заяви, обхідні",
null, false, false, null, "Довідки, заяви, обхідні", 0, "references" },
                    { "088e46e2-1019-1111-1111-865313c9a21d", "", "088e46e2-
1111-0000-1111-865313c9a21d", "Description of page Вибіркові дисципліни", null,
false, false, null, "Вибіркові дисципліни", 0, "disciplines" }
                });
            migrationBuilder.CreateIndex(
                name: "IX_AdminDepartmentEntity_AdminId",
                table: "AdminDepartmentEntity",
                column: "AdminId");

            migrationBuilder.CreateIndex(
                name: "IX_AdminDepartmentEntity_DepartmentId",
                table: "AdminDepartmentEntity",
                column: "DepartmentId");

            migrationBuilder.CreateIndex(
                name: "IX_Contacts_PageId",
                table: "Contacts",
                column: "PageId");

            migrationBuilder.CreateIndex(
                name: "IX_Courses_DepartmentId",
                table: "Courses",
                column: "DepartmentId");

            migrationBuilder.CreateIndex(
                name: "IX_DepartmentNews_NewsId",
                table: "DepartmentNews",
                column: "NewsId");

            migrationBuilder.CreateIndex(
                name: "IX_FileEntityPage_PagesId",
                table: "FileEntityPage",
                column: "PagesId");

            migrationBuilder.CreateIndex(
                name: "IX_PagePerson_PageId",
                table: "PagePerson",
                column: "PageId");

            migrationBuilder.CreateIndex(
                name: "IX_PagePerson_PersonId",
                table: "PagePerson",
                column: "PersonId");

            migrationBuilder.CreateIndex(
                name: "IX_Pages_DepartmentId",
                table: "Pages",
                column: "DepartmentId");
```

ДОДАТОК В

Вибіркові файли програмного коду бекенду

```
using System;
using MediatR;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using UniversityService.Application.Requests.Admin;
using UniversityService.Application.Requests.Department;

namespace UniversityService.WebApi.Controllers
{
    [ApiController]
    [Route("admins")]
    public class AdminController : ControllerBase
    {
        private readonly IMediator _mediator;

        public AdminController(IMediator mediator)
        {
            _mediator = mediator;
        }

        [HttpPost("register")]
        [Authorize(Roles = "SuperAdmin,Admin")]
        public async Task<IActionResult> Registration(RegisterAdminRequest
request)
        {
            var token = await _mediator.Send(request);

            return Ok(token);
        }

        [HttpPost("authorization")]
        [Authorize(Roles = "SuperAdmin,Admin")]
        [AllowAnonymous]
        public async Task<IActionResult> Authorization(AuthorizeAdminRequest
request)
        {
            var token = await _mediator.Send(request);

            return Ok(token);
        }

        [HttpGet("base/{id}")]
        [Authorize(Roles = "SuperAdmin,Admin")]
        public async Task<IActionResult> GetBaseDataById(GetAdminBaseRequest
request)
        {
            var admin = await _mediator.Send(request);

            return Ok(admin);
        }

        [HttpGet("base")]
        [Authorize(Roles = "SuperAdmin,Admin")]
        public async Task<IActionResult> GetBaseData(GetAdminsBaseRequest
request)
```

```
        {
            var admins = await _mediator.Send(request);

            return Ok(admins);
        }

        [HttpGet("detail")]
        [Authorize(Roles = "SuperAdmin,Admin")]
        public async Task<IActionResult> GetDetailData(GetAdminsDetailRequest
request)
        {
            var admins = await _mediator.Send(request);

            return Ok(admins);
        }

        [HttpGet("detail/{id}")]
        [Authorize(Roles = "SuperAdmin,Admin")]
        public async Task<IActionResult> GetDetailDataById(GetAdminDetailRequest
request)
        {
            var admin = await _mediator.Send(request);

            return Ok(admin);
        }

        [HttpPut("change-password")]
        [Authorize(Roles = "SuperAdmin,Admin")]
        public async Task<IActionResult>
ChangePassword(AdminChangePasswordRequest request)
        {
            try
            {
                await _mediator.Send(request);
            }
            catch (Exception ex)
            {
                return Conflict();
            }

            return Ok();
        }

        [HttpPost("complete-registration")]
        [AllowAnonymous]
        public async Task<IActionResult>
CompleteRegistration(CompleteRegistrationRequest request)
        {
            var adminTokenInfo = await _mediator.Send(request);

            return Ok(adminTokenInfo);
        }

        [HttpDelete("{id}")]
        [Authorize(Roles = "SuperAdmin")]
        public async Task<IActionResult> Delete(DeleteAdminRequest request)
        {
            await _mediator.Send(request);
```

```
            return Ok();
        }

        [HttpPut("{id}/block")]
        [Authorize(Roles = "SuperAdmin,Admin")]
        public async Task<IActionResult> Block(BlockAdminRequest request)
        {
            await _mediator.Send(request);

            return Ok();
        }

        [HttpPut("{id}/unblock")]
        [Authorize(Roles = "SuperAdmin,Admin")]
        public async Task<IActionResult> Unblock(UnblockAdminRequest request)
        {
            await _mediator.Send(request);

            return Ok();
        }

        [HttpPut]
        [Authorize(Roles = "SuperAdmin,Admin")]
        public async Task<IActionResult> Update(UpdateAdminRequest request)
        {
            var admin = await _mediator.Send(request);

            return Ok(admin);
        }

        [HttpPut("{id}/permission")]
        [Authorize(Roles = "SuperAdmin,Admin")]
        public async Task<IActionResult>
UpdatePermission(UpdateAdminPermissionRequest request)
        {
            var admin = await _mediator.Send(request);

            return Ok(admin);
        }
    }
}

using MediatR;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using UniversityService.Application.Requests.ContactRequests;

namespace UniversityService.WebApi.Controllers;

[Authorize]
[Route("contacts")]
[ApiController]
public class ContactController : ControllerBase
{
    private readonly IMediator _mediator;

    public ContactController(IMediator mediator)
    {
        _mediator = mediator;
```

```
    }

    [HttpPost]
    [Authorize(Roles = "SuperAdmin,Admin")]
    public async Task<ActionResult> CreateAsync(CreateContactRequest request)
    {
        var contact = await _mediator.Send(request);

        return Ok(contact);
    }

    [HttpPut("{id}")]
    [Authorize(Roles = "SuperAdmin,Admin")]
    public async Task<ActionResult> UpdateAsync(UpdateContactRequest request)
    {
        var contact = await _mediator.Send(request);

        return Ok(contact);
    }

    [HttpDelete("{id}")]
    [Authorize(Roles = "SuperAdmin,Admin")]
    public async Task<ActionResult> DeleteAsync(DeleteContactRequest request)
    {
        await _mediator.Send(request);

        return Ok();
    }

    [HttpGet("{id}")]
    [AllowAnonymous]
    public async Task<ActionResult> GetByIdAsync(GetContactRequest request)
    {
        var contact = await _mediator.Send(request);

        return Ok(contact);
    }

    [HttpGet]
    [AllowAnonymous]
    public async Task<ActionResult> GetAsync(GetContactsRequest request)
    {
        var contacts = await _mediator.Send(request);

        return Ok(contacts);
    }
}

using MediatR;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using UniversityService.Application.Requests.CourseRequests;

namespace UniversityService.WebApi.Controllers;

[Authorize]
[Route("courses")]
[ApiController]
public class CourseController : ControllerBase
```

```
{
    private readonly IMediator _mediator;

    public CourseController(IMediator mediator)
    {
        _mediator = mediator;
    }

    [HttpPost]
    [Authorize(Roles = "SuperAdmin,Admin")]
    public async Task<ActionResult> CreateAsync(CreateCourseRequest request)
    {
        var course = await _mediator.Send(request);

        return Ok(course);
    }

    [HttpPut("{id}")]
    [Authorize(Roles = "SuperAdmin,Admin")]
    public async Task<ActionResult> UpdateAsync(UpdateCourseRequest request)
    {
        var course = await _mediator.Send(request);

        return Ok(course);
    }

    [HttpDelete("{id}")]
    [Authorize(Roles = "SuperAdmin,Admin")]
    public async Task<ActionResult> DeleteAsync(DeleteCourseRequest request)
    {
        await _mediator.Send(request);

        return Ok();
    }

    [HttpGet("{id}")]
    [AllowAnonymous]
    public async Task<ActionResult> GetByIdAsync(GetCourseRequest request)
    {
        var course = await _mediator.Send(request);

        return Ok(course);
    }

    [HttpGet]
    [AllowAnonymous]
    public async Task<ActionResult> GetAsync(GetCoursesRequest request)
    {
        var courses = await _mediator.Send(request);

        return Ok(courses);
    }
}

using MediatR;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using UniversityService.Application.Requests.Admin;
using UniversityService.Application.Requests.Department;
```

```
using UniversityService.Domain.AggregatesModel.DepartmentAggregate;

namespace UniversityService.WebApi.Controllers
{
    [ApiController]
    [Route("departments")]
    public class DepartmentController : ControllerBase
    {
        private readonly IMediator _mediator;

        public DepartmentController(IMediator mediator)
        {
            _mediator = mediator;
        }

        [HttpGet("/university")]
        [Authorize(Roles = "SuperAdmin,Admin")]
        public async Task<IActionResult> GetUniversityData(GetUniversityRequest
request)
        {
            var university = await _mediator.Send(request);

            return Ok(university);
        }

        [HttpGet("/university/short")]
        [AllowAnonymous]
        public async Task<IActionResult>
GetUniversityBaseData(GetUniversityShortRequest request)
        {
            var university = await _mediator.Send(request);

            return Ok(university);
        }

        [HttpGet("base")]
        [Authorize(Roles = "SuperAdmin,Admin")]
        public async Task<IActionResult> GetBaseData(GetDepartmentsBaseRequest
request)
        {
            var departments = await _mediator.Send(request);

            return Ok(departments);
        }

        [HttpGet("{path}/short")]
        [AllowAnonymous]
        public async Task<IActionResult>
GetShortByPath(GetDepartmentByPathShortRequest request)
        {
            var department = await _mediator.Send(request);

            return Ok(department);
        }

        [HttpGet("short")]
        [AllowAnonymous]
        public async Task<IActionResult> GetAllShort(GetDepartmentsShortRequest
request)
```

```
        {
            var departments = await _mediator.Send(request);

            return Ok(departments);
        }

        [HttpGet("{id}/detail")]
        [Authorize(Roles = "SuperAdmin,Admin")]
        public async Task<IActionResult>
GetDetailDataById(GetDepartmentDetailRequest request)
        {
            var department = await _mediator.Send(request);

            return Ok(department);
        }

        [HttpPost]
        [Authorize(Roles = "SuperAdmin,Admin")]
        public async Task<IActionResult> Create(CreateDepartmentRequest request)
        {
            var departament = await _mediator.Send(request);

            return Ok(departament);
        }

        [HttpPut("{id}")]
        [Authorize(Roles = "SuperAdmin,Admin")]
        public async Task<IActionResult> Update(UpdateDepartmentRequest request)
        {
            var departament = await _mediator.Send(request);

            return Ok(departament);
        }

        [HttpDelete("{id}")]
        [Authorize(Roles = "SuperAdmin,Admin")]
        public async Task<IActionResult> Delete(DeleteDepartmentRequest request)
        {
            await _mediator.Send(request);

            return Ok();
        }

        [HttpPut("{id}/block")]
        [Authorize(Roles = "SuperAdmin,Admin")]
        public async Task<IActionResult> Block(BlockDepartmentRequest request)
        {
            await _mediator.Send(request);

            return Ok();
        }

        [HttpPut("{id}/unblock")]
        [Authorize(Roles = "SuperAdmin,Admin")]
        public async Task<IActionResult> Unblock(UnblockDepartmentRequest
request)
        {
            await _mediator.Send(request);
```

```
            return Ok();
        }
    }
}

using System;
using MediatR;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using System.ComponentModel.DataAnnotations;
using UniversityService.Application.Requests.File;
using UniversityService.Application.Models.Dtos.File;

namespace UniversityService.WebApi.Controllers;

[Authorize]
[Route("files")]
[ApiController]
public class FileController : ControllerBase
{
    private const string OctetStreamContentType = "image/jpeg";

    private readonly IMediator _mediator;

    public FileController(IMediator mediator)
    {
        _mediator = mediator;
    }

    /// <summary>
    ///     Asynchronously upload file.
    /// </summary>
    /// <param name="file">The upload file model.</param>
    /// <returns>A task that represents asynchronous upload operation.</returns>
    [HttpPost]
    [Authorize(Roles = "SuperAdmin,Admin")]
    public async Task<ActionResult> PostFileAsync(UploadFileRequest request)
    {
        var result = await _mediator.Send(request);

        return Ok(result);
    }

    /// <summary>
    ///     Asynchronously download file by id.
    /// </summary>
    /// <param name="id">The file id.</param>
    /// <returns>A task that represents asynchronous upload operation.</returns>
    [HttpGet("{id}")]
    [AllowAnonymous]
    public async Task<IActionResult> DownloadAsync([FromRoute, Required] string
id)
    {
        var result = await _mediator.Send<DownloadFileDto>(new
DowndloadFileRequest { Id = id });

        return File(result.Stream, OctetStreamContentType, result.FileName);
    }
```

```
    [HttpPost("get")]
    [Authorize(Roles = "SuperAdmin,Admin")]
    public async Task<IActionResult> GetFiles(GetFilesRequest request)
    {
        var files = await _mediator.Send(request);

        return Ok(files);
    }

    [HttpPut("{id}")]
    [Authorize(Roles = "SuperAdmin,Admin")]
    public async Task<IActionResult> UpdateFile(UpdateFileRequest request)
    {
        var file = await _mediator.Send(request);

        return Ok(file);
    }

    [HttpDelete("{id}")]
    [Authorize(Roles = "SuperAdmin,Admin")]
    public async Task<IActionResult> DeleteFile(DeleteFileRequest request)
    {
        await _mediator.Send(request);

        return Ok();
    }
}

using MediatR;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using UniversityService.Application.Requests.NewsRequests;
using UniversityService.Domain.AggregatesModel.SpecialtyAggregate;

namespace UniversityService.WebApi.Controllers;

[Authorize]
[Route("news")]
[ApiController]
public class NewsController : ControllerBase
{
    private readonly IMediator _mediator;

    public NewsController(IMediator mediator)
    {
        _mediator = mediator;
    }

    [HttpPost]
    [Authorize(Roles = "SuperAdmin,Admin")]
    public async Task<ActionResult> CreateNewsAsync(CreateNewsRequest request)
    {
        var news = await _mediator.Send(request);

        return Ok(news);
    }

    [HttpPut("{id}")]
    [Authorize(Roles = "SuperAdmin,Admin")]
```

```
    public async Task<ActionResult> UpdateNewsAsync(UpdateNewsRequest request)
    {
        var news = await _mediator.Send(request);

        return Ok(news);
    }

    [HttpDelete("{id}")]
    [Authorize(Roles = "SuperAdmin,Admin")]
    public async Task<ActionResult> DeleteNewsAsync(DeleteNewsRequest request)
    {
        await _mediator.Send(request);

        return Ok();
    }

    [HttpGet("{id}")]
    [AllowAnonymous]
    public async Task<ActionResult> GetNewsByIdAsync(GetNewsByIdRequest request)
    {
        var news = await _mediator.Send(request);

        return Ok(news);
    }

    [HttpGet("departments/{departmentId}")]
    [AllowAnonymous]
    public async Task<ActionResult>
GetNewsByDepartnemtIdAsync(GetNewsByDepartmentIdRequest request)
    {
        var news = await _mediator.Send(request);

        return Ok(news);
    }

    [HttpGet("/departments/by-path/{path}/news")]
    [AllowAnonymous]
    public async Task<ActionResult>
GetNewsByDepartnemtPathAsync(GetNewsByDepartmentPathRequest request)
    {
        var news = await _mediator.Send(request);

        return Ok(news);
    }

    [HttpGet]
    [AllowAnonymous]
    public async Task<ActionResult> GetNewsAsync(GetNewsRequest request)
    {
        var news = await _mediator.Send(request);

        return Ok(news);
    }
}

using MediatR;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using UniversityService.Application.Requests.Department;
```

```
using UniversityService.Application.Requests.Page;

namespace UniversityService.WebApi.Controllers
{
    [ApiController]
    [Route("pages")]
    public class PageController : ControllerBase
    {
        private readonly IMediator _mediator;

        public PageController(IMediator mediator)
        {
            _mediator = mediator;
        }

        [HttpGet("/department/{departmentId}/pages/base")]
        [Authorize(Roles = "SuperAdmin,Admin")]
        public async Task<IActionResult> GetBase(GetPagesBaseRequest request)
        {
            var pages = await _mediator.Send(request);

            return Ok(pages);
        }

        [HttpGet("/department/{departmentId}/pages/by-path/{path}")]
        [AllowAnonymous]
        public async Task<IActionResult> GetByPath(GetPageByPathRequest request)
        {
            var page = await _mediator.Send(request);

            return Ok(page);
        }

        [HttpPost("/department/{departmentId}/pages")]
        [Authorize(Roles = "SuperAdmin,Admin")]
        public async Task<IActionResult> Create(CreatePageRequest request)
        {
            var page = await _mediator.Send(request);

            return Ok(page);
        }

        [HttpPost("/department/by-path/{path}/pages/by-type/{type}")]
        [Authorize(Roles = "SuperAdmin,Admin")]
        public async Task<IActionResult>
Create(GetPageByTypeDepartmentByPathRequest request)
        {
            var page = await _mediator.Send(request);

            return Ok(page);
        }

        [HttpGet("{id}")]
        [Authorize(Roles = "SuperAdmin,Admin")]
        public async Task<IActionResult> Get(GetPageRequest request)
        {
            var page = await _mediator.Send(request);

            return Ok(page);
```

```
        }

        [HttpPut("{id}")]
        [Authorize(Roles = "SuperAdmin,Admin")]
        public async Task<IActionResult> Update(UpdatePageRequest request)
        {
            var page = await _mediator.Send(request);

            return Ok(page);
        }

        [HttpDelete("{id}")]
        [Authorize(Roles = "SuperAdmin,Admin")]
        public async Task<IActionResult> Delete(DeletePageRequest request)
        {
            await _mediator.Send(request);

            return Ok();
        }

        [HttpPut("{id}/block")]
        [Authorize(Roles = "SuperAdmin,Admin")]
        public async Task<IActionResult> Block(BlockPageRequest request)
        {
            await _mediator.Send(request);

            return Ok();
        }

        [HttpPut("{id}/unblock")]
        [Authorize(Roles = "SuperAdmin,Admin")]
        public async Task<IActionResult> Unblock(UnblockPageRequest request)
        {
            await _mediator.Send(request);

            return Ok();
        }

        [HttpPut("{id}/persons")]
        [Authorize(Roles = "SuperAdmin,Admin")]
        public async Task<IActionResult>
UpdatePagePersons(UpdatePagePersonsRequest request)
        {
            await _mediator.Send(request);

            return Ok();
        }
    }
}

using MediatR;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using UniversityService.Application.Requests.File;
using UniversityService.Application.Requests.Person;

namespace UniversityService.WebApi.Controllers;

[Authorize]
```

```
[Route("persons")]
[ApiController]
public class PersonController : ControllerBase
{
    private readonly IMediator _mediator;

    public PersonController(IMediator mediator)
    {
        _mediator = mediator;
    }

    [HttpPost]
    [Authorize(Roles = "SuperAdmin,Admin")]
    public async Task<ActionResult> CreatePersonAsync(CreatePersonRequest
request)
    {
        var person = await _mediator.Send(request);

        return Ok(person);
    }

    [HttpPut("{id}")]
    [Authorize(Roles = "SuperAdmin,Admin")]
    public async Task<ActionResult> UpdatePersonAsync(UpdatePersonRequest
request)
    {
        var person = await _mediator.Send(request);

        return Ok(person);
    }

    [HttpDelete("{id}")]
    [Authorize(Roles = "SuperAdmin,Admin")]
    public async Task<ActionResult> DeletePersonAsync(DeletePersonRequest
request)
    {
        await _mediator.Send(request);

        return Ok();
    }

    [HttpGet("{id}/public")]
    [AllowAnonymous]
    public async Task<ActionResult> GetPersonAsync(GetPersonRequest request)
    {
        var person = await _mediator.Send(request);

        return Ok(person);
    }

    [HttpPost("get")]
    [AllowAnonymous]
    public async Task<ActionResult> GetPersonsAsync(GetPersonsRequest request)
    {
        var persons = await _mediator.Send(request);

        return Ok(persons);
    }
}
```

```
using MediatR;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using UniversityService.Application.Requests.SocialRequests;

namespace UniversityService.WebApi.Controllers;

[Authorize]
[Route("socials")]
[ApiController]
public class SocialController : ControllerBase
{
    private readonly IMediator _mediator;
    public SocialController(IMediator mediator)
    {
        _mediator = mediator;
    }
    [HttpPost]
    [Authorize(Roles = "SuperAdmin,Admin")]
    public async Task<ActionResult> CreateAsync(CreateSocialRequest request)
    {
        var social = await _mediator.Send(request);
        return Ok(social);
    }
    [HttpPut("{id}")]
    [Authorize(Roles = "SuperAdmin,Admin")]
    public async Task<ActionResult> UpdateAsync(UpdateSocialRequest request)
    {
        var social = await _mediator.Send(request);
        return Ok(social);
    }

    [HttpDelete("{id}")]
    [Authorize(Roles = "SuperAdmin,Admin")]
    public async Task<ActionResult> DeleteAsync(DeleteSocialRequest request)
    {
        await _mediator.Send(request);

        return Ok();
    }

    [HttpGet("{id}")]
    [AllowAnonymous]
    public async Task<ActionResult> GetByIdAsync(GetSocialRequest request)
    {
        var social = await _mediator.Send(request);

        return Ok(social);
    }

    [HttpGet]
    [AllowAnonymous]
    public async Task<ActionResult> GetAsync(GetSocialsRequest request)
    {
        var socials = await _mediator.Send(request);

        return Ok(socials);
    }
}
```

```
using MediatR;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using UniversityService.Application.Requests.SpecialtyRequests;

namespace UniversityService.WebApi.Controllers;

[Authorize]
[Route("specialties")]
[ApiController]
public class SpecialtyController : ControllerBase
{
    private readonly IMediator _mediator;
    public SpecialtyController(IMediator mediator)
    {
        _mediator = mediator;
    }
    [HttpPost]
    [Authorize(Roles = "SuperAdmin,Admin")]
    public async Task<ActionResult> CreateAsync(CreateSpecialtyRequest request)
    {
        var specialty = await _mediator.Send(request);
        return Ok(specialty);
    }
    [HttpPut("{id}")]
    [Authorize(Roles = "SuperAdmin,Admin")]
    public async Task<ActionResult> UpdateAsync(UpdateSpecialtyRequest request)
    {
        var specialty = await _mediator.Send(request);

        return Ok(specialty);
    }

    [HttpDelete("{id}")]
    [Authorize(Roles = "SuperAdmin,Admin")]
    public async Task<ActionResult> DeleteAsync(DeleteSpecialtyRequest request)
    {
        await _mediator.Send(request);

        return Ok();
    }

    [HttpGet("{id}")]
    [AllowAnonymous]
    public async Task<ActionResult> GetByIdAsync(GetSpecialtyRequest request)
    {
        var specialty = await _mediator.Send(request);

        return Ok(specialty);
    }

    [HttpGet]
    [AllowAnonymous]
    public async Task<ActionResult> GetAsync(GetSpecialtiesRequest request)
    {
        var specialties = await _mediator.Send(request);
        return Ok(specialties);
    }
```

ДОДАТОК Г

Підключенні бібліотеки