ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
Кафедра Інженерії програмного забезпечення

Пояснювальна записка

до бакалаврської роботи

на ступінь вищої освіти бакалавр

на тему: «**РОЗРОБКА СЕРВЕРНОГО ДОДАТКУ ДЛЯ СИСТЕМИ ДИСТАНЦІЙНОГО НАВЧАННЯ МОВОЮ C#**»

Виконав: студент 4 курсу, групи ПД-44
спеціальності
121 Інженерія програмного забезпечення
(шифр і назва спеціальності)
Мудрик Я.Ю.
(прізвище та ініціали)
Керівник Поперешняк С.В.
(прізвище та ініціали)
Рецензент
(прізвище та ініціали)

Київ – 2023

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ
Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення
Ступінь вищої освіти - «Бакалавр»
Спеціальність - 121 Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувач кафедри
Інженерії програмного
забезпечення
_____О.В. Негоденко

«____»_____2023 року

ЗАВДАННЯ

НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Мудрик Ярослав Юрійович

(прізвище, ім'я, по батькові)

1. Тема роботи: «Розробка серверного додатку для систем дистанційного навчання мовою C#»

Керівник роботи: Поперешняк Світлана Володимирівна к.т.н, доцент

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом вищого навчального закладу від «24» лютого 2023 року №26.

2. Строк подання студентом роботи «1» червня 2023 року
3. Вихідні дані до роботи:

Організація дистанційного навчання, технічна література програмного забезпечення з приводу дистанційного навчання, технічна література .NET, технічні засоби розробки такі як Visual Studio 2022 – середа розробки, методи та бібліотеки C#, MS SQL SERVER, PostgreSQL, браузер для керування і тестування.

4. Зміст розрахунково-пояснювальної записки(перелік питань, які потрібно розробити).

4.1 Аналіз предметної області.

4.2 Вимоги та оцінки якості системи.

4.3 Проектування та реалізація веб-сервісу.

4.4 Тестування системи.

5. Перелік демонстраційного матеріалу (назва основних слайдів)

5.1. Титульний слайд

5.2. Мета, об’єкт та предмет дослідження

5.3. Задачі дипломної роботи

5.4. Аналіз аналогів

5.5. Вимоги до програмного забезпечення

5.6. Засоби реалізації

5.7. Діаграма послідовності

5.8. Архітектура системи

5.9. Діаграми пакетів

5.10. – 5.12. Схема Бази Даних

5.17. Апробація результатів дослідження

5.18. Висновки

5.19. Кінцевий слайд

6. Дата видачі завдання «25» лютого 2023 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	02.03.2023	Виконано
2	Аналіз науково-технічної літератури	05.03.2023	Виконано
3	Проектування системи	06.03.2023	Виконано
4	Створення та тестування програмного рішення	29.04.2023	Виконано
5	Підготовка розділу 1	05.05.2023	Виконано
6	Підготовка розділу 2	08.05.2023	Виконано
7	Підготовка розділу 3	10.05.2023	Виконано
8	Підготовка розділу 4	13.05.2023	Виконано
9	Вступ, висновки, реферат	15.05.2023	Виконано
10	Розробка обов’язкових демонстраційних матеріалів	17.05.2023	Виконано
11	Попередні захист роботи та перевірка на плагіат	25.05.2023	Виконано
12	Здача роботи	15.06.2023	

Студент _____ Мудрик Я.Ю.

(підпис) (прізвище та ініціали)

Керівник роботи _____ Поперешняк С.В.

(підпис) (прізвище та ініціали)

РЕФЕРАТ

Текстова частина бакалаврської роботи 67 с., 44 рис., 8 табл., 2 дод., 20 джерел

ВЕБ ДОДАТОК, МОНОЛІТ, ВЕБ СЕРВІС, URLS, UKRANIAN REMOTE LEARNING SYSTEM, BLEACKBOARD, CLASSROOM, DAPPER, ASP.NET CORE, WEB API, MAILKIT, MOODLE, СЕРВІС ДЛЯ ПІДТРИМКИ НАВЧАЛЬНОГО ПРОЦЕСУ.

- Об’єкт дослідження – процес дистанційного навчання за допомогою серверного додатку.
- Предмет дослідження – серверний додаток для забезпечення доступності функцій дистанційного навчання.
- Мета роботи – покращення процесу взаємодії з клієнтом, за рахунок реалізації API на C#.

Методи дослідження –методи передачі, зберігання та обробки інформації, уніфікований процес створення програмного забезпечення.

Наукова новизна полягає у тому, що дана система немає аналогів та реалізація функціоналу близька за галуззю використання продуктах. URLS дозволяє викладачам та адміністраторам слідкувати за структурою університету, та навчальним процесом, яке дозволяє зберігати інформацію про групи, предмети, пари, журнали, відомості, тощо. Налаштування сповіщень та керування доступом до акаунту є ключовою функціональністю, які відсутні у аналогів.

Веб-сервіс це проект моноліт, розроблено було на мові програмування C# та фреймворків ASP.NET Core, Entity Framework Core. Для баз даних використовувалися ядра SQLServer та PostgreSQL, для зберігання великих файлів було вирішено реалізувати синхронізацію з Google Drive та Azure Blob Storage.

Отже, розроблено веб-сервіс, який підвищує та спрощує навчальний процес за рахунок автоматизації багатьох функцій.

Галузь використання – освітній процес.

Зміст

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

SDK – software development kit

API – application program interface

JSON – JavaScript object notation

ORM – object relation mapping

БД – База Даних

СКБД – Система Керування Базами Даних

HTTPS – Hypertext transfer protocol

MVC – Model View Controller

ASP - Active Server Pages

SQL – Structed query language

LINQ – Language-Integrated Query

IDE – Integrated Development Environment

МБ – Мегабайти

ГБ – Гігабайт

ВСТУП

В сучасному світі, де технології стають все більш невід'ємною частиною нашого життя, дистанційне навчання набуває все більшої популярності. Завдяки Інтернету та комп'ютерним системам, люди отримують можливість отримувати освіту та розвиватися, незалежно від свого місця проживання. Однак, успішна реалізація дистанційного навчання вимагає потужних і надійних інструментів та програмного забезпечення.

Метою роботи є покращення процесу взаємодії з клієнтом, за рахунок реалізації API на C#. Цей додаток буде забезпечувати надійне з'єднання між викладачами та студентами, дозволяючи підтримувати навчальний процес в режимі реального часу.

Одним із головних викликів при розробці даного серверного додатку є забезпечення безпеки та конфіденційності даних, оскільки у системі дистанційного навчання зберігаються особисті та конфіденційні дані користувачів. Для цього будуть використовуватися сучасні методи шифрування та механізми автентифікації, щоб забезпечити безпеку передачі та зберігання інформації.

Окрім того, серверний додаток буде підтримувати широкий набір функціональності, такий як підтримка інформації про структуру університету, про всі його інститути, кафедри, групи, можливості підтримки навчального процесу шляхом надання інструментів для зберігання інформації про предмети, пари, оцінки, звітності, тощо. Налаштування сповіщень та доступу до акаунту.

У цій роботі будуть використовуватися сучасні інструменти розробки, зокрема мова програмування C#. Також будуть використані інші технології та бібліотеки, такі як ASP.NET Core, Entity Framework Core і багато інших, для забезпечення ефективної та швидкої розробки серверного додатку.

Очікується, що результатом цієї роботи буде функціональний та надійний серверний додаток, який дозволить ефективно використовувати систему дистанційного навчання, сприяючи покращенню процесу навчання та забезпеченню доступу до якісної освіти для всіх користувачів системи.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Аналіз предметної області є невід’ємною частиною розробки будь-якого програмного забезпечення. Більшість компанії які планують створити продукт починають аналізувати ринки і конкурентів: що вже є, які є плюси і мінуси і що можна покращити, або створити абсолютно нову концепцію. Інколи через відсутність або поганий аналіз майбутній продукт живе під загрозою зникнення, оскільки через непрофесійність чи недооцінку було втрачено момент випуску продукту.

1.1. Історія розвитку навчального процесу

З давних давен процес навчання відносився до престижних послуг, які могли отримати лише «обрані», оскільки знаючих людей поважали в суспільстві і прислухалися до їх думки. Важливим компонентом в навчальному процесі є викладач, котрий має знати як передати матеріал, а також перевіряти знання людей.

То ж як ми зрозуміли, найважливішим компонентом будь-якого навчання як зараз так і в давні часи є процес комунікації. Так, звичайно розуміння також залежить і від тих людей, що вчяться.

1.2. Розвиток в цифрову епоху

З часом процес комунікації та передачі знань покращувався, а з приходом комп’ютерів та інформаційних технологій відбувся бум. Найпрестижніші і найтехнологічніші університети почали думати про впровадження цього в процес навчання.

Іде 2023 рік, але і досі нема універсальної системи для процесу навчання. Так, існують системи, які максимально уніфіковані в своїй функціоналісті, але для нормальної роботи треба як мінімум декілька систем. Основною проблемою є

різниця в підходах оцінення, проходження навчального плану та рівнях складності навчання.

Також треба згадати, що відносно нещодавно набуло популярності документообіг з успішністю студентів, а це великі папки паперів, документації і архіви. Тому перед цифровими технологіями поставили завдання покращити цей процес. А це в свою чергу зменшує витрати на папір та архіви, також дуже сильно пришвидшує роботу на рівні університету, оскільки більшість операцій можна перенести до цифрового систему.

1.3. Існуюче програмне забезпечення та аналоги

Як було вже сказано існують системи, які можуть допомогти в навчальному процесі, серед них: Google Classroom, Moodle, Blackboard, тощо.

Google Classroom – це система, яка була створена в 2014 році компанією Google для спрощення керування навчальним процесом. Основний рівень навчання це курс, який створює викладач, до якого можуть приєднуватися студенти. На головному сторінці є доступ до ряду курсів (див. Рис. 1.3.1). В кожному курсі можна створювати розділі, розділ включає в себе список завдання (див. Рис. 1.3.2), яке можна налаштувати параметри, починаючи датою здачі закінчуючи обмеженням для розміру завантажених файлів. Також classroom має доступ до сторінки оцінок студентів. Оскільки цей продукт від Google від має інтеграцію з іншими продуктами цієї компанії, такими як: Google Drive, Google Calendar, Google Meet та Google Sheets. Інтеграції подібного роду достатньо сильно спрощують розробку подібних сервісів, а також використання, бо використовує інфраструктуру на повну потужність. В цьому випадку для зберігання даних використовується сервіс Drive, Calendar можна використовувати для розкладу, для онлайн пар Meet і для успішності Sheets. Classroom підтримує не тільки веб-версію а ще і мобільні пристрої (див. Рис. 1.3.3).

Рисунок 1.3.1 – Головна сторінка

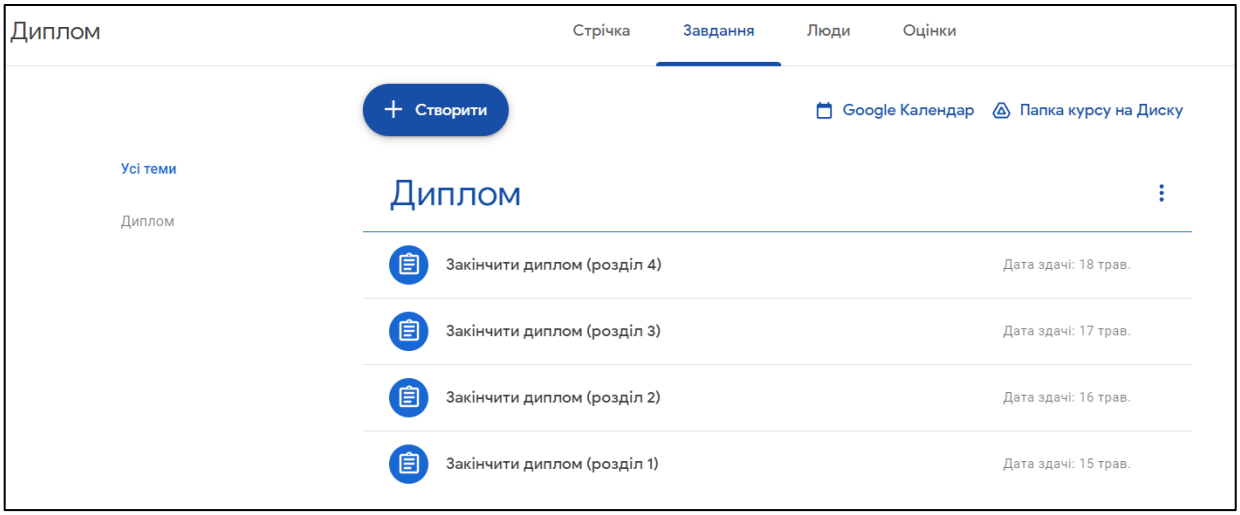


Рисунок 1.3.2 – Список завдань

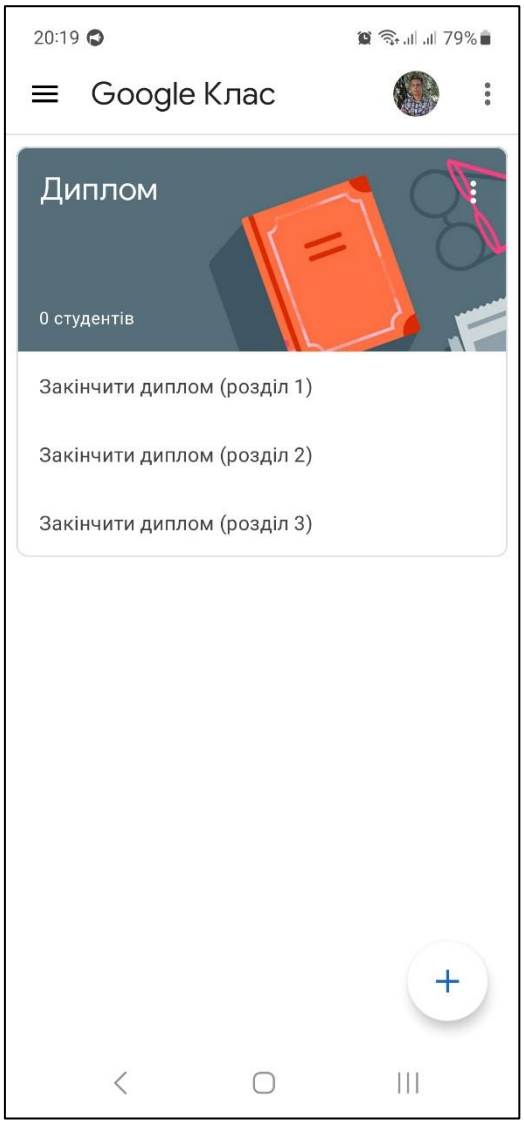


Рисунок 1.3.3 – Мобільний застосунок

До переваг можна віднести:

- Простий інтерфейс.
- Доступний API.
- Наявність мобільного клієнта.
- Google інтеграція.
- Зручна панель керування курсами.
- Коментарі до робіт студентів.
- Перегляд успішності студента.
- Керування ролями для користувачів.
- Інтеграція з навчальними закладами.

- Приєднання по запрошенню;

До недоліків можна віднести:

- Відсутній Todo список для завдань.
- Неповноцінна система сповіщень.
- Вихідний код недоступний.
- Швидкість роботи.
- Приватність даних.
- Відсутня інтеграція з сторонніми сервісами.
- Обмежені можливості адміністрування.
- Відсутність засобів для конкретних предметів.
- Обмежені можливості моніторингу.
- Обмежені можливості відслідковування активності студентів;

Враховуючи вище перераховані можливості, переваги та недоліки можна зробити висновок, що дана система є дуже потужним інструментом по своїм можливостям, але нажаль повноцінною системою дистанційною системою бути не може, оскільки відсутні деякі важливі функції, такі як: оцінки, статистика, сповіщення, вбудований розклад, структура університету, ділення на групи.

Moodle (Modular Object-Oriented Dynamic Learning Environment) – система, яка призначена для навчання, побудована на архітектурі модульності в якій без проблем можна прикрутити дуже багато плагінів, які в свою чергу дають більші можливості у порівняні з базовою версією. Вона є повністю безкоштовною з відкритим вихідним кодом, що дає їй велику перевагу перед іншим подібними системами в цій категорії.

Базовий функціонал передбачає багато можливостей, як для студента так і для викладача. Після успішного входу йде перенаправлення на головну сторінку, яка має меню і поточні курси студента (див. Рис. 1.3.4), натиснувши на будь-який курс завантажується інформація і його опис, студенти, успішність і ToDo список нагадування студенту про найближчі завдання, у яких підходить термін здачі (див. Рис. 1.3.5). Є можливість передивитися розділи і завдання до кожного з них,

перевірити чи виконали завдання (див. Рис. 1.3.6). Також великі можливості для викладача, який може створювати курси (див. Рис. 1.3.7), контролювати оцінки, виставляти відсутність студентів на парах та навіть створювати розклад в вбудованому календарі (див. Рис. 1.3.8).

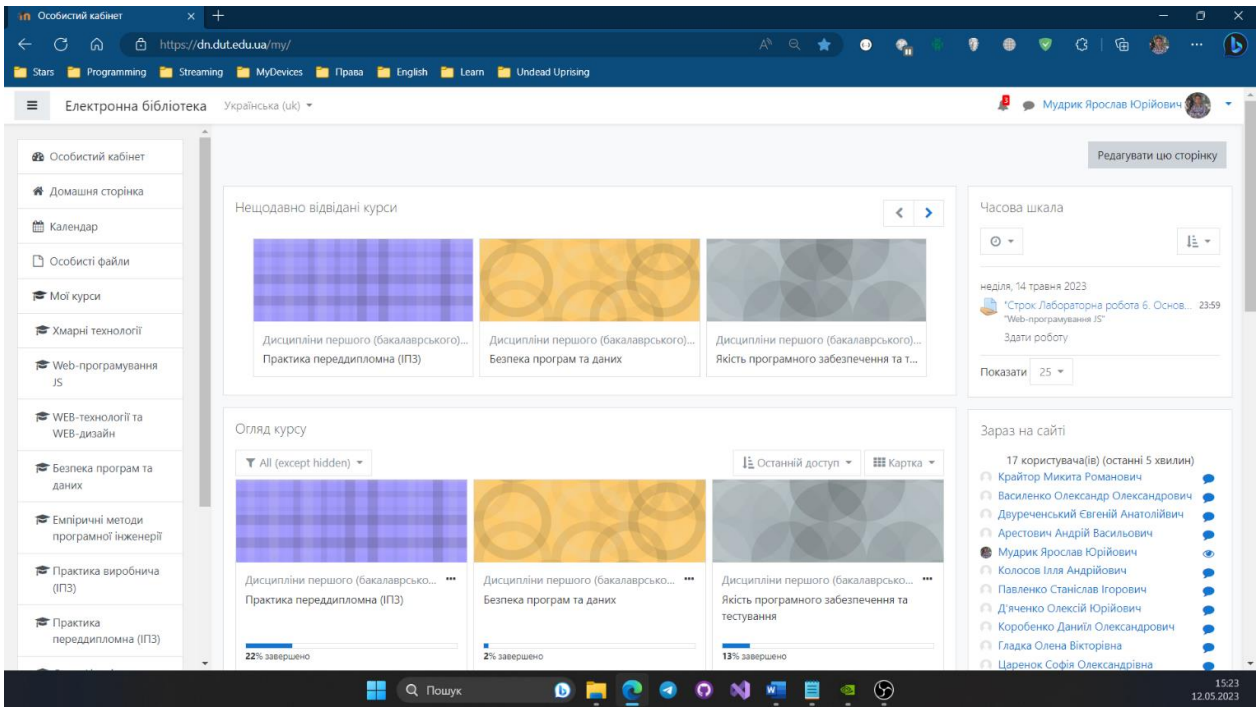


Рисунок 1.3.4 – Головна сторінка

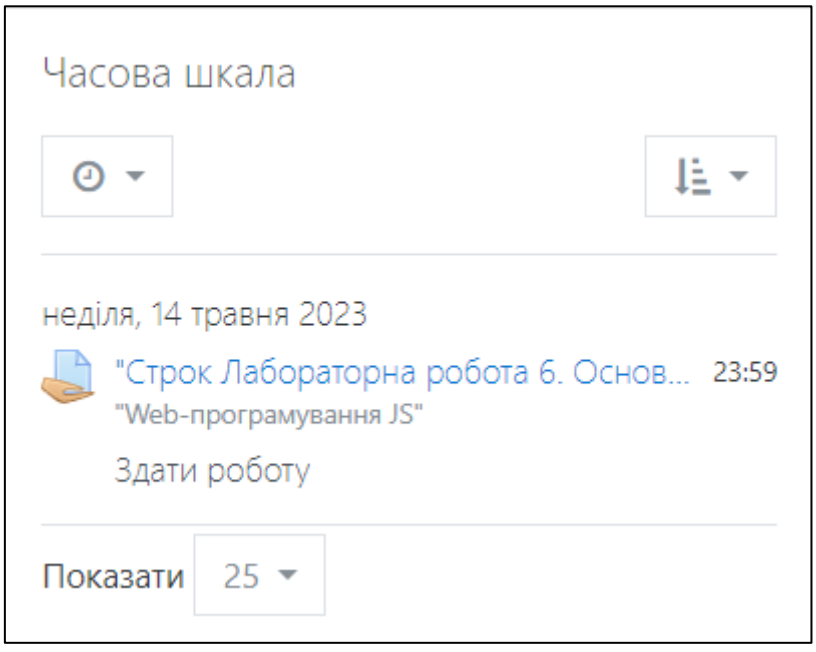


Рисунок 1.3.5 – Нагадування про завдання

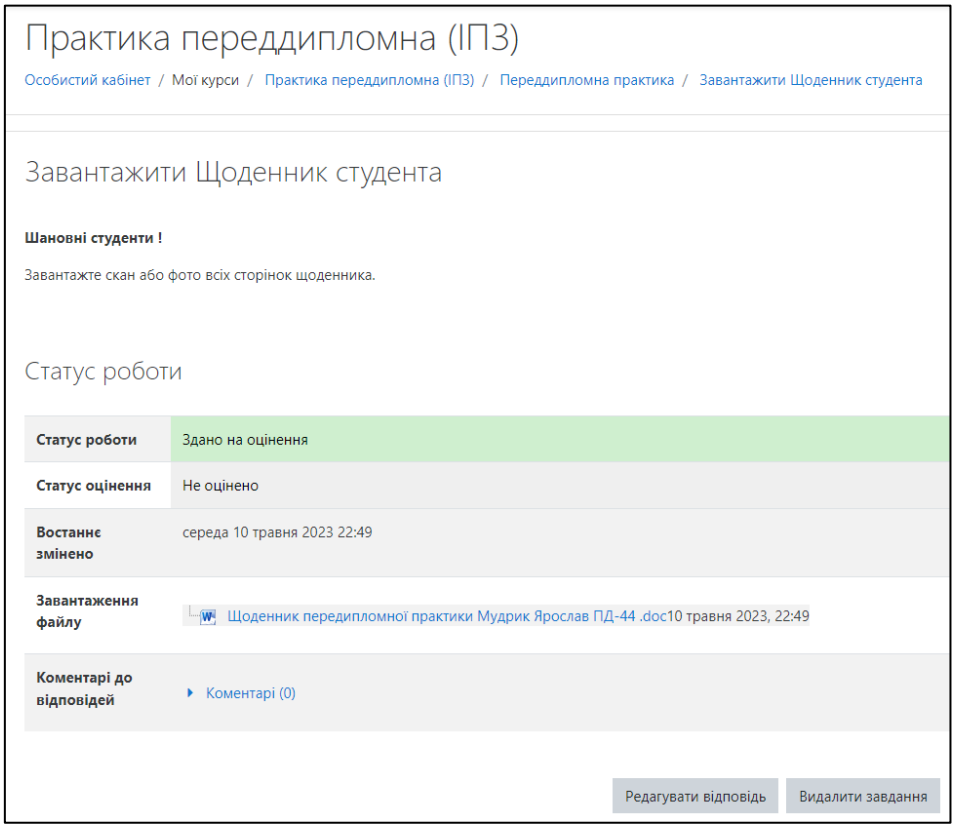


Рисунок 1.3.6 – Сторінка перевірки завдання

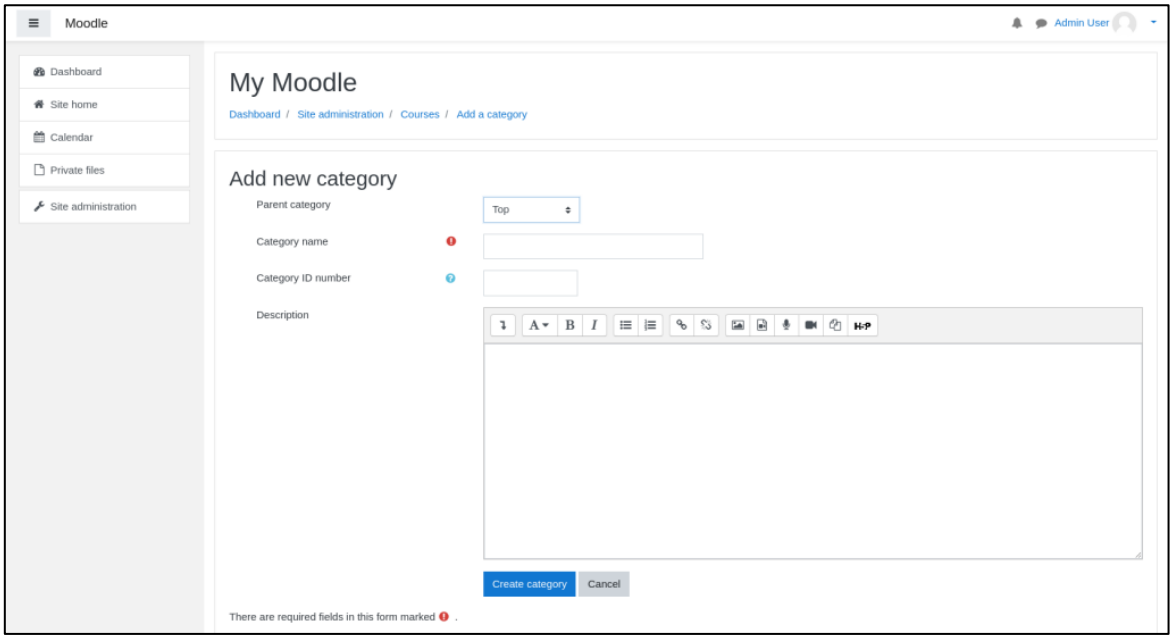


Рисунок 1.3.7 – Сторінка створення курсу

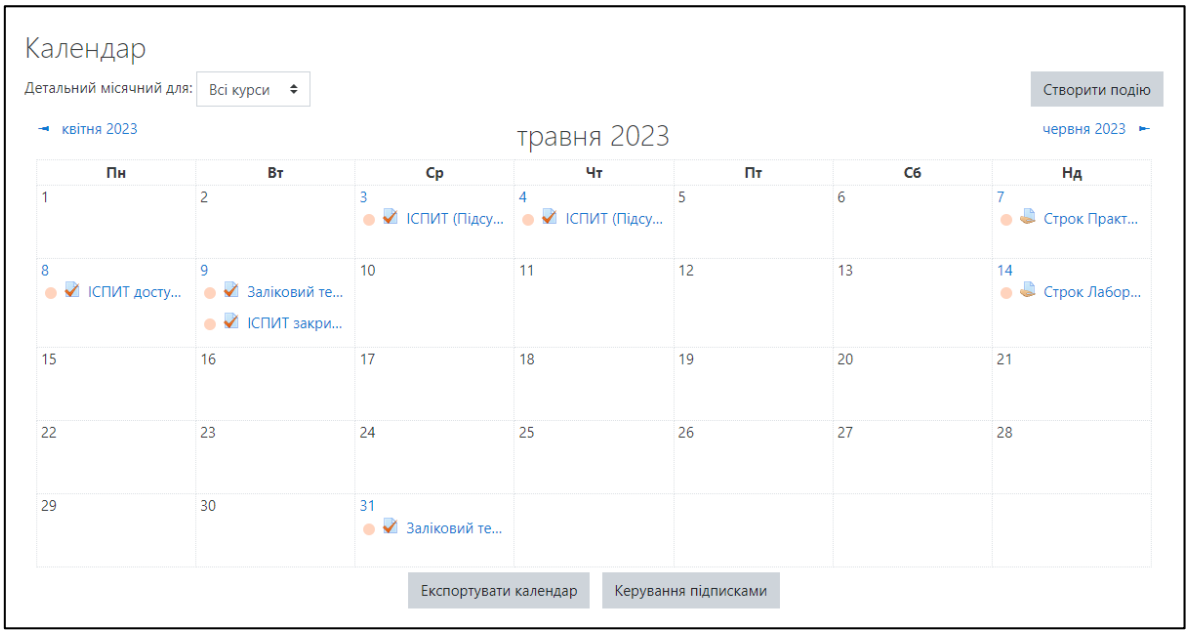


Рисунок 1.3.8 – Сторінка календаря

Переваги системи Moodle:

- Відкритий вихідний код.
- Безкоштовний.
- Гнучкий до розширювання.
- Багато базового функціоналу.
- Оптимізація вебу під всі відомі платформи включаючи Android, iOS.
- Сучасний UI.
- Проводження тестів.
- Детальне відслідковування прогресу.
- Розподіл доступності контенту між студентами та викладачами.
- Проста інтеграція.
- Підлатування курсу під кожного студента.
- Відкритий API;

Недоліки системи Moodle:

- Складність входження.
- Високі вимоги до ресурсів.
- Оновлення на нову версія.

- Відсутність нормальної комунікації між студентом та викладачем.
- Підключення існуючої системи до сторонніх сервісів.
- Слабка кастомізації інтерфейсу.
- Високе споживання трафіку.
- Відсутність відеоконференцій.
- Відсутність мобільного клієнта.
- Складність з перевантаженням серверу;

Тож можна зробити висновок, що Moodle є дуже потужною системою, але через складність розуміння функціональності для початківців можуть виникнути проблеми. Хоча підтримка її інфраструктури є достатньо складною та може бути непередбачуваною, все одно більшість навчальних закладів бере її за основу, оскільки кращих альтернатив з такою гнучкістю налаштування не знайти.

Blackboard – це платформа призначена для навчального процесу, яка є достатньо популярною по всьому світу. Важливо зазначити, шо це комерційний програмний продукт, і немає навіть безкоштовного версії на 30 днів. Його ціна починається від 10 000$ на рік. Це достатньо відчутна сума як для навчального закладу в Україні. Ця система має власну бібліотеку інструментів розширень, яку можуть поповнювати звичайні програмісти, створюючи доповнення, наприклад для курсів по математиці чи англійській мові.

Серед основного функціоналу можна виділити створення курсів (див. Рис. 1.3.9) та керування матеріалами в них, проходження тестів та виконання іспитів. Комунікація між студентами та викладачами засобами електронної пошти, чатів а відео-конференцій. Створення та перегляд розкладу всередині застосунку. Можливість формування оцінок, звіту навчання та групове виконання робіт.

Також ця платформа має інтеграцію з сервісами авторизації. Різні мобільні додатки (див. Рис. 1.3.10) для студентів і викладачів дозволяють ефективно слідкувати за змінами в навчальному процесі і моментально реагувати на різні ситуації. Присутня ще інтеграція з сторони навчальними, бібліотечними та відео сервісами, які прискорюють роботу, оскільки використовують всім відомі сервіси.

Статистика з аналітикою створені для покращення сервісу та виявлення багів чи надання інформації журналістам за запитом.

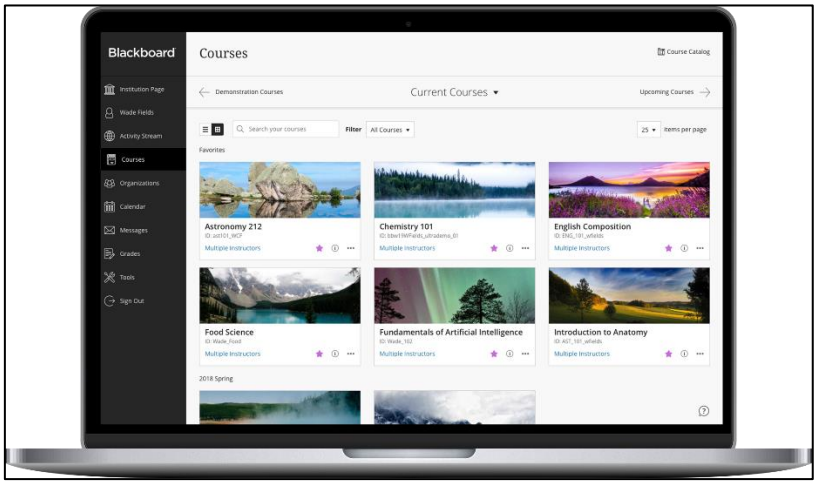


Рисунок 1.3.9 – Сторінка курсів

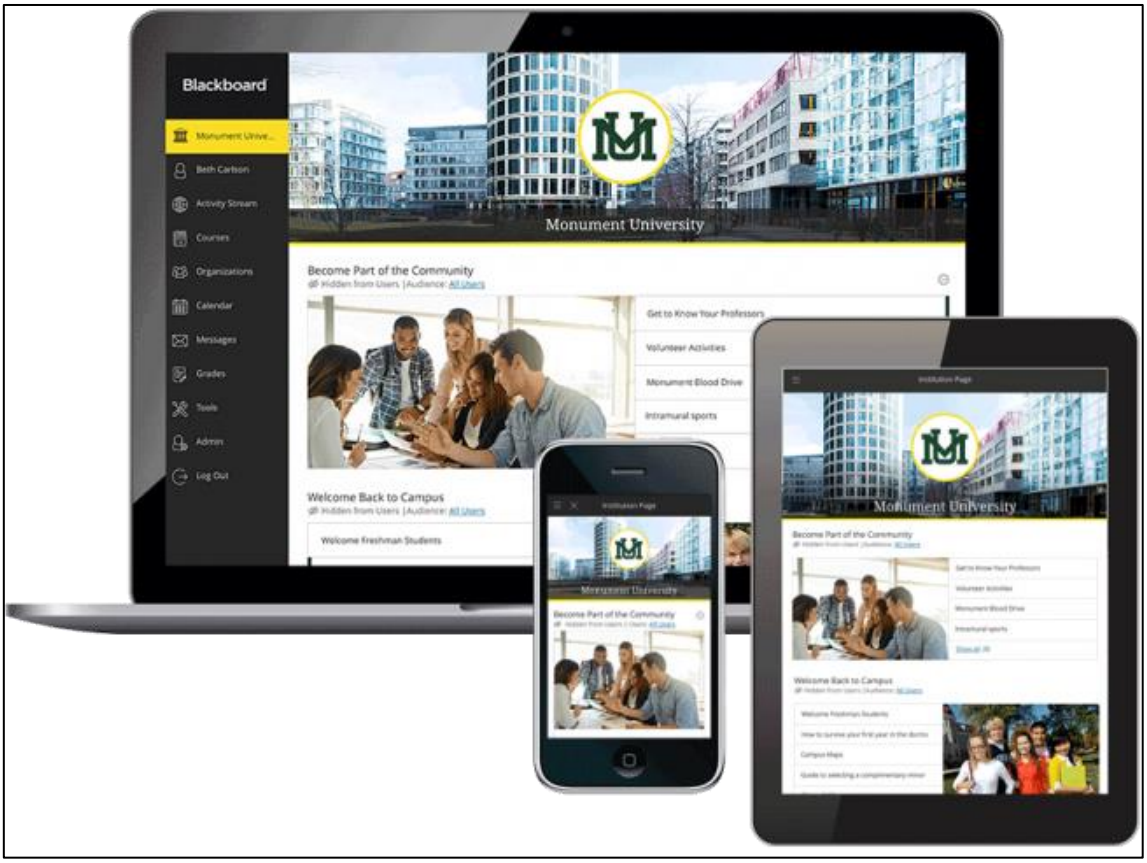


Рисунок 1.3.10 – Мобільні додатки

Рисунок 1.3.11 – Сторінка статистики

Переваги Blackboard:

- Централізація.
- Доступність.
- Гнучкість.
- Співпраця.
- Звітність.
- Оцінювання.
- Безпека.
- Система підтримки.
- Інтеграції.
- Різні формати та типи навчання.
- Автоматизація.
- Прогрес студентів.
- Підтримка багатьох мов;

Недоліки Blackboard:

- Складна у використані.

- Відсутня можливість дизайнінгу курсів.
- Низька продуктивність.
- Обмеження функціоналу на мобільних пристроях.
- Проста аналітика.
- Обмеження відеоконференцій та відео-контенту.
- Висока вартість.
- Обмеження в розгортанні standalone застосунку [1];

Отже можна зробити висновки, що Blackboard надає централізовану платформу, яка дозволяє забезпечити організацію навчального процесу, взаємодію між викладачами та студентами, і доступ до необхідних матеріалів та ресурсів з будь-якого місця. Це програмне забезпечення допомагає зменшити географічні обмеження та забезпечує гнучкість у розкладі та навчанні, особливо для студентів, які не мають можливості фізично присутніх у навчальному закладі. Blackboard також сприяє активній взаємодії і спільноті шляхом функцій комунікації та спільної роботи, таких як форуми, чати та спільні проекти.

Загалом, програмне забезпечення Blackboard є корисним інструментом для дистанційного навчання, яке допомагає забезпечити доступ до навчальних матеріалів та комунікацію в онлайн-середовищі. Враховуючи його переваги та обмеження, Blackboard може бути ефективним рішенням для підтримки освітнього процесу на відстані.

2. ВИМОГИ ТА ОЦІНКА ЯКОСТІ СИСТЕМИ

При розробці будь-якої системи має бути виділено функціональні та нефункціональні вимоги, які окреслюють рамки роботи застосунку.

Вони поділяються на:

- Функціональні: це вимоги, які повинні описувати роботу системи (маніпулювання даними, розрахунки чи обчислення).
- Нефункціональні: це вимоги, які вказують якою система має бути (які навантаження повинна витримувати, безпека, масштабування);

2.1. Функціональні вимоги

2.1.1. Аутентифікація та авторизація

Система підтримує 3 ролі: Адміністратор, Викладач, Студент. Для кожної з цих ролей існує різні шляхи реєстрації. Адміном можна зробити будь-якого користувача. Реєстрація викладача та студента мають спільні кроки та включають наступні дані: прізвище, ім’я, по-батькові, логін (пошта), пароль та код-запрошення. Важливим пунктом реєстрації студента є згода чи відмова викладача групи, до якої намагається приєднатися студент. Вхід в систему здійснюється за допомогою логіна та пароля (див. рис. 2.1.1). Також підтримується прив’язка сторонніх сервісів авторизації для спрощення входу, серед них: Google, Microsoft, Facebook. Присутня можливість відновлення пароля. Дво-факторна авторизація вмикається для способу входу по логіну і паролю. При потребі кожен користувач має можливість змінити як свій логін так і пароль. Кожен користувач має інколи потребу увійти з декількох пристроїв, тому була розроблена можливість сесійного доступу, це означає, що кожен вхід в систему створює сесію і будь-коли користувач може переглянути її деталі (для прикладу: де був вхід, з якого пристрою і в якому додатку) або припинити сесію на іншому девайсі. При спробі увійти ввівши 5 разів невірний пароль – акаунт блокується автоматично на годину. Розблокувати його може тільки адміністратор.

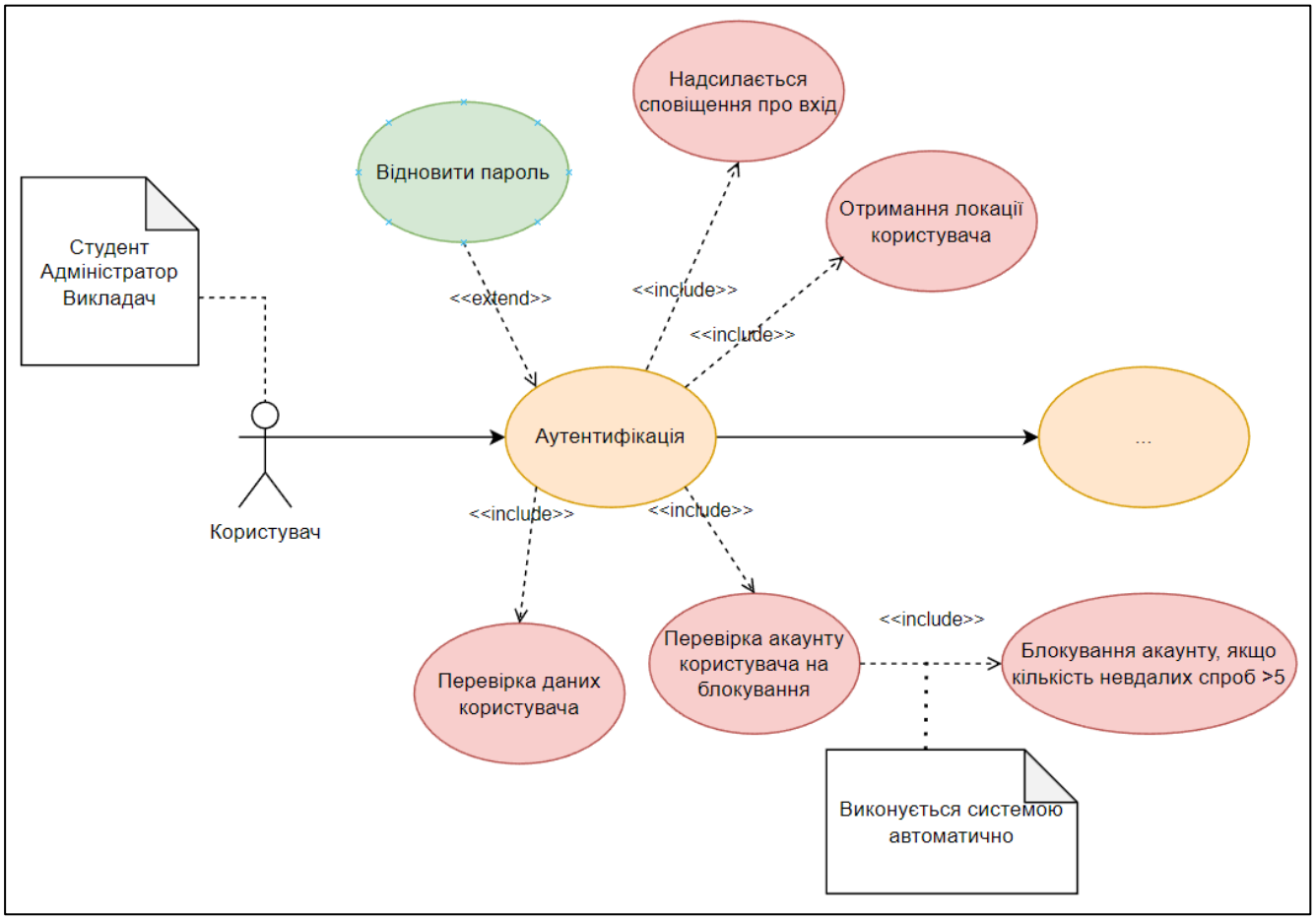


Рисунок 2.1.1 – Діаграма використання входу

2.1.2. Керування ролями і доступом

Кожен шлях в URLS має перевірку на дозвіл виклику його, для прикладу: студенти можуть тільки переглядати розклад або структуру університету, а вже викладачі можуть частково редагувати розклад пар, які з ними пов’язані.

Крім трьох базових ролей є можливість створити окремі ролі, які будуть відповідати наприклад за створення розкладу чи редагування інформації про університет тощо.

2.1.3. Профіль користувача

Викладач, що зареєструвався в системі має можливість змінити прізвище, ім’я, по-батькові, нікнейм та які сповіщення мають надходити. Студент може

змінити лише нікнейм та сповіщення. Адміністратор може змінювати всі дані у всіх користувачів.

2.1.4. Структура університету

URLS має повноцінну підтримку структур українських університетів, тому є можливість створення, редагування та видалення наступних рівнів структур:

- Університет.
- Інститут (факультет).
- Спеціальність (кафедра).
- Група;

2.1.5. Перегляд та керування групами

Для навчання студент має бути в конкретній групі. Для приєднання до групи створюється так званий «код-приєднання», він має певну назву, час життя, та активність. Найголовніше правило системи, у одного студента може бути тільки одна група. Створенням та редагуванням назви групи, зображенням та її функціоналом займається адміністратор або викладач групи. Після закінчення навчання на рівні бакалаврат та магістр йде Керування групою відбувається великою кількість функціоналу, яка починається від звичайної зміни назви закінчуючи оголошеннями, а саме:

- Створення, редагування та видалення кодів приєднання.
- Згода чи відмова на вступ студента.
- Згода на приєднання усіх нових студентів групи.
- Редагування чи видалення студентів групи.
- Перегляд інформації про студента.
- Переведення групи на наступний курс.
- Зміна або призначення керівника групи.
- Керування ролями групи (куратор, керівник, староста, студент).

- Створення нових та редагування існуючих ролей групи з певними доступом.
- Видалення ролей групи.
- Створення, редагування, видалення оголошень в групі.
- Створення коментарів до оголошень.
- Можливість видалення коментарів чи їх редагування.
- Додавання реакцій на оголошення;

2.1.6. Створення та перегляд розкладу

Система забезпечує доступ до створення, перегляду та видаленню розкладу. Усі дані аудиторій, викладачів, часу та типу пар ведуться в актуальній БД. Студентам доступне лише перегляд розкладу будь-яких груп, в той час як адміністраторам доступні усі дії. А викладачі можуть редагувати, створювати чи видаляти пари тільки власних предметів, для прикладу викладач Іноземних мов може відредагувати тільки предмет «Англійська мова [ІМ]» і тільки в тій групі, в якій веде цей предмет. Хоча у викладачів достатньо обмежений функціонал, це не забороняє зареєструватися ще одній людині + створити ще одну роль, яка буде безпосередньо керувати розкладом. Присутній також білдер, який допомагає створити розклад за правилами, для прикладу: може допомогти створити розклад з 2 парами в один тиждень і однією і другий тиждень.

2.1.7. Керування предметами

Система повинна забезпечувати створення, редагування, видалення та перегляд предметів групи (див. рис. 2.1.2). Існує 2 типи предмету, звичайний і шаблонний:

- Звичайний – це предмет, який пов'язаний з конкретною групою та конкретним викладачем.
- Шаблонний – це предмет, який є шаблоном, він не прив’язаний ні до ніякої групи і немає викладача, але може бути взятий за основу звичайного

предмету. Рішенням про його додавання став висновок, що майже кожен рік буде йти перетворення предметів і щоб не витрачати багато часу на заповнення даних було створено шаблоні предмети.

Предмет включає в себе таку інформацію:

- Назва.
- Опис.
- Дата початку навчання.
- Дата кінець навчання.
- Кількість годин.
- Максимальна оцінка.
- Чи є екзамен.
- Максимальні оцінки до екзамену і під час екзамену.
- Рекомендовано для курсу.
- Викладач.
- Список пар;

Протягом певного часу є можливість створювати відомості, по яким можна робити висновки успішності.

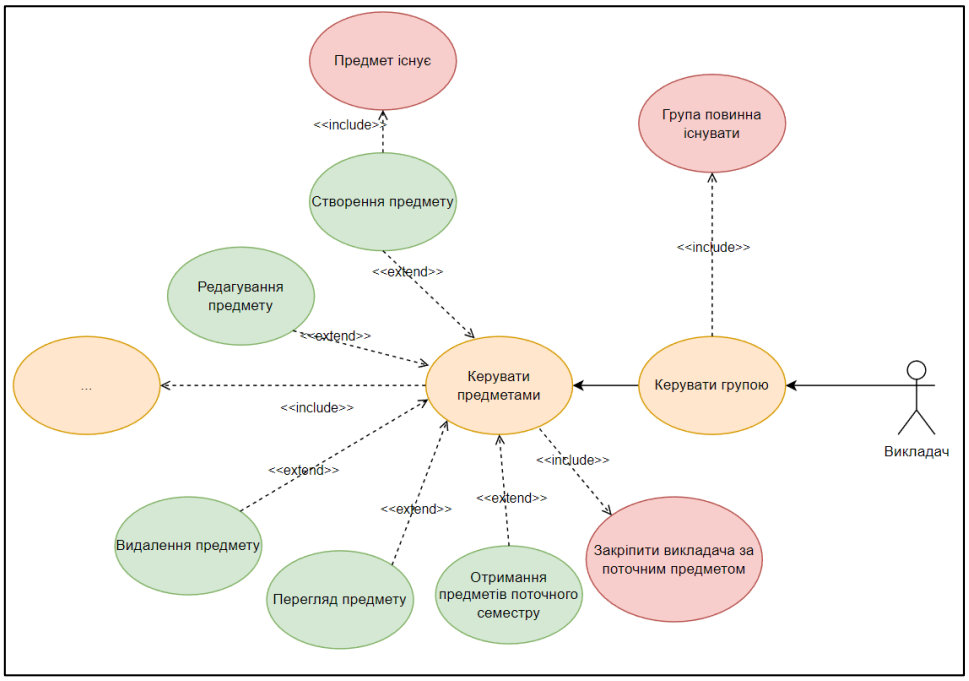


Рисунок 2.1.2 – Діаграма використання предмети

2.1.8. Керування парами

Кожен предмет має власний список пар, який є точним аналогом реальної пари в тому сенсі в якому ми звикли розуміти. Створенням, редагуванням та видаленням пар займається викладач цього предмету або адміністратор (див рис. 2.1.3). Кожна пара має наступні дані:

- Тема.
- Опис.
- Журнал.
- Дата і час (коли відбувається пара).
- Тип пари (лекція, практика, екзамен).
- Домашня завдання.
- Вкладення.
- Минула пара.
- Наступна пара.
- Викладач, що заміняє;

Пара має журнал, а це потужний інструмент, оскільки представляє повний список групи студентів в якому можна відмічати присутність студентів на парі або виставлення оцінок. У журналі присутня маленька статистика, в статистика включені такі поля:

- Кількість студентів.
- Кількість присутніх студентів.
- Кількість студентів з оцінками і без;

Кожна пара яка створена і співпадає по даті в розкладі, то вона автоматично буде прив’язана перехресним посиланням з розкладом, але буде доступна лише для учасників групи, викладача і адміністратора.

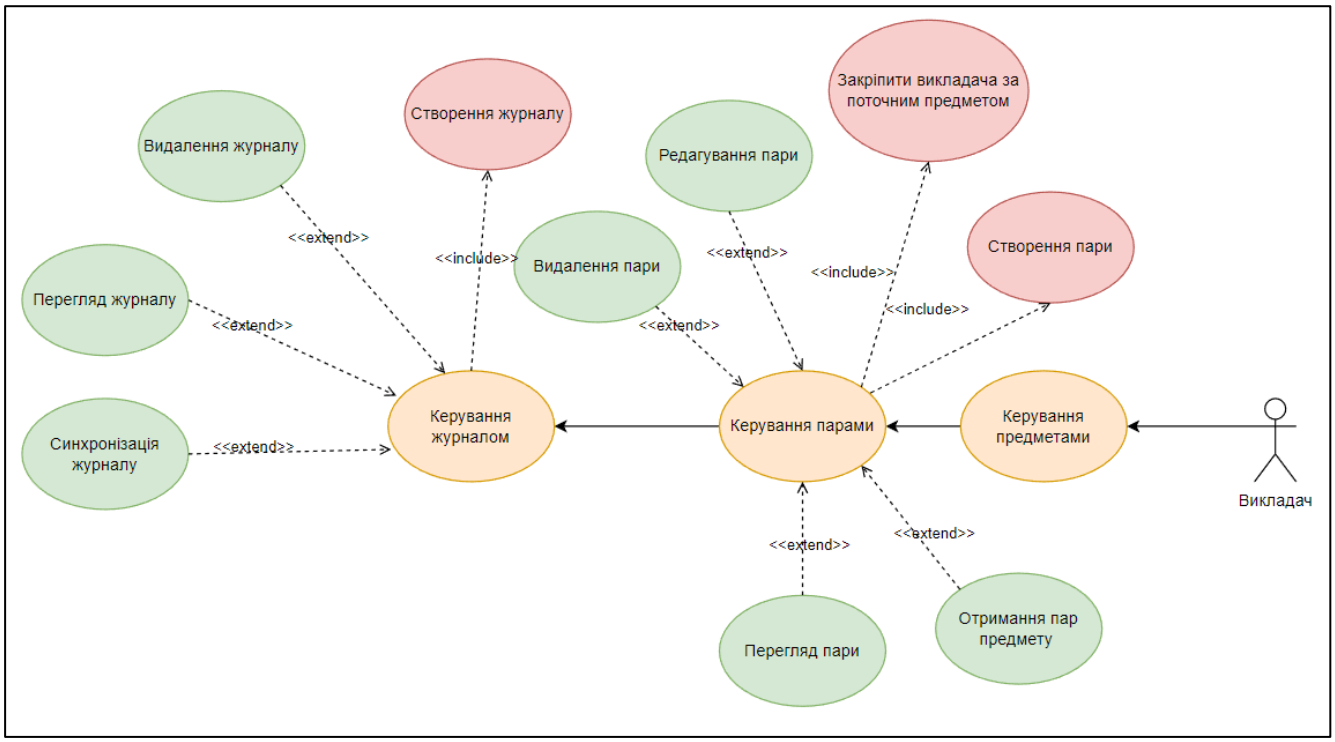


Рисунок 2.1.3 – Діаграма використання пар

2.1.9. Проходження тестів

Перевірка знань студентів є невід'ємною частиною навчального процесу, тому розробка тестування була необхідною. Функціональність для створення тестів достатньо велика, вона покриває майже 90% можливого функціоналу для розробки тестів (див. рис. 2.1.4). Тест може бути прив'язаний до якогось конкретного предмету або у доступним всім бажаючим.

Тест включає в себе наступні дані:

- Назва.
- Опис.
- Конфігурація.
- Автор.
- Термін з якого доступний тест.
- Термін до якого доступний тест.
- Чи доступний тест.
- Чи це шаблон.
- Питання.

- Результати;

Конфігурація кожного тесту відбувається окремо і має наступні властивості до налаштування:

- Максимальна кількість спроб.
- Оцінка за тест.
- Час для проходження.
- Випадкове розташовування питання та відповіді.
- Чи доступні результати учаснику.
- Чи показуються правильні відповіді;

Автор тесту може бути як викладач поточного університету так і випадкова людина, тому при необхідності можна заповнити наступні дані:

- Повне ПІБ людини, яка створювала тест.
- Її вчене звання (якщо таке є).
- Шлях до зображення;

Для кожного питання можна виставити його порядковий номер в тесті при умові, що випадкове розташування вимкнуте. Також можна вказати, чи поточне питання підтримує декілька відповідей. Відповідь включає в себе текст і галочку, чи правильна відповідь.

Результати тестів зберігаються з наступними даними:

- Оцінка.
- Спроба.
- Коли почалося проходження тестів.
- Коли закінчилося проходження тестів.
- Статистика.
- Чи здав тест після дедлайну.
- Відповіді;

В статистиці зберігається інформація:

- Кількість питань.
- Кількість правильних відповідей.

- Кількість пропущених питань;

Кожна відповідь має декілька варіантів і є заповнені поля з коректністю відповіді і чи вибрав студент саме цю відповідь. На базі цього формується звіт про проходження тестів студентами.

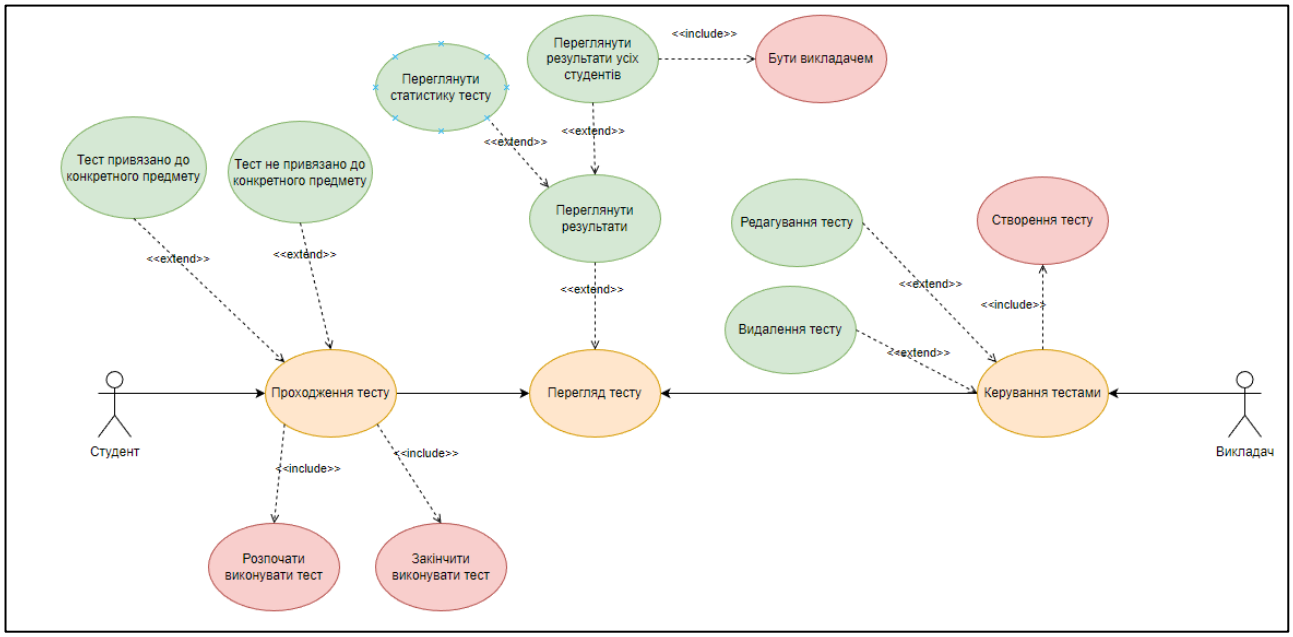


Рисунок 2.1.4 – Діаграма використання тестування

2.1.10. Експорт та імпорт

Для спрощення роботи системи на деяких функціях є підтримка імпорту та експорту excel. Наприклад список нових студентів можна імпортувати через excel і на базі цих даних буде створено повноцінну групу.

Експорт доступний для наступних функцій:

- Експорт групи.
- Експорт оцінок конкретної пари.
- Експорт оцінок по предмету;

Кількість функцій обмежена, але їх збільшення не є великою проблемою.

2.1.11. Система відгуків

Відгуки є невід’ємною частиною будь-якого застосунку, оскільки на бази них формується список бажаного функціоналу користувачами.

Також, якщо будуть присутні баги, користувачі зможуть надсилати і в швидкому порядку будуть виправлятися. Присутній механізм автоматичного надсилання багів в разі поломки застосунку.

2.1.12. Система сповіщень

Сповіщення сьогодні відіграють важливу роль в житті будь-якої системи, оскільки надають можливість автоматично нагадувати користувачу про якісь події, які є важливими в подібних системах. Не менш важливим є можливість їх налаштувань.

На даний момент підтримується наступні сповіщення:

- Приєднання студента до групи.
- Новий вхід.
- Спроба входу.
- Зміна пароля.
- Вихід з облікового запису.
- Нове оголошення.
- Змінене оголошення.
- Блокування.
- Зміна доступив у ролі в якій знаходиться користувач.
- Зміна ролі користувача.
- Додавання нової пари.
- Зміну розкладу.
- Встановлення оцінки;

2.2. Нефункціональні вимоги

2.2.1. Безпека

URLS повинна виконувати наступні вимоги відносно безпеки:

- Хешування паролів користувачів.

- Захист від sql-ін’єкції [2].
- Усі запити мають йти по https протоколу, які є надійно захищеним.
- Фільтри для перевірки запитів.
- Підтримка механізмів аутентифікації і авторизації.
- Захищене з’єднання з БД.
- Доступ студентів тільки до власних ресурсів.
- Доступ до пар тільки студентам групи та викладачу.
- Заборона редагування розкладу користувачам.
- Перевірка даних на предмет образливої чи ненормативної лексики.
- Перевірка даних на образу інших расових відмінностей;

2.2.2. Продуктивність

Продуктивність визначається деякими правилами, які враховують швидкість певних запитів та загальної здатності сервера пропускати потоки інформації. Використання RateLimit [3] бібліотеки спросить навантаження на систему, налаштувавши певні обмеження на виконання запитів окремого клієнта. Система повинна витримувати ріст користувачів без втрати швидкості обробки їх запитів.

2.2.3. Сумісність

Система URLS має забезпечувати деякі вимоги:

- Робота на ОС Windows, Linux, MacOS.
- Підтримка роботи з https по сертифікатам.
- API включає версіонування.
- API повинен підтримувати формат запиту\відповіді у XML чи JSON.
- Сумісність з сторонніми клієнтами через API SDK та бібліотеки моделей;

2.2.4. Масштабованість

Масштабування по горизонталі та по вертикалі має проходити максимально

просто та безболісно як для користувачів так і для розробників.

2.2.5. Підтримка

Для більш ефективної підтримки такої складності застосунку треба відповідні умови:

- Повинен проводитися аналіз систем кожні 5-10 хв на робото здатність, щоб швидко виявляти коли система впала.
- Створені pipelines [4] для швидкого розгортання фіксів та оновлень.
- Логування системи має проводитися в усіх місцях проекту і має включати в себе час, формат, повідомлення, дані, стектрейс та RequestId для швидкого визначення проблеми та її подальшого усунення;

2.2.6. Зручність використання API

Система має задовольняти наступні вимоги для використання API:

- API включає в себе документацію по взаємодії з собою через схему OpenAPI 3.
- Підтримка декількох типів контенту, а саме XML та JSON.
- Мають слідкувати принципам Restful [5].
- Підтримка пагінації та сортування.
- Повертати правильні статус коди:
    - 200 – успішне виконання.
    - 400 – клієнтська помилка.
    - 401 – несанкціонований доступ.
    - 403 – користувачу не вистачає прав.
    - 404 – ресурс не знайдено.
    - 500 – серверна помилка.
    - 503 – сервіс тимчасово недоступний;

2.2.7. Сповіщення

Вимоги які потребують сповіщення наступні:

- Система повинна забезпечувати швидку доставку.
- Система повинна надійно відправляти сповіщення.
- Система повинна підтримувати налаштування.
- Система повинна справлятися з ростом навантаження без втрати продуктивності;

2.3. Системні вимоги та розгортання

Для коректної роботи платформи бажано наступні виділені ресурси:

- CPU: Intel Xeon E5 3.30 GHz.
- RAM: 32Gb.
- SDD: 10TB.
- OS: Windows, Linux, MacOS.
- LAN: 10Gig;

Також треба ще врахувати ресурси для UI, який буде працювати окремо.

При відсутності можливостей розгорнути на власному сервері це можна зробити на серверах Azure від Microsoft, AWS від Amazon і GCloud від Google. Всі ці хмарні сервіси надають послуг розгортання сторонніх систем у довільному порядку. У кожного провайдера є безкоштовний термін на 30 днів, після чого доведеться платити за послуги. Ці плани включають як місце на сервері під застосунок, виділений хостинг, зареєстровану адресу сайту так і робоче ядро бази даних.

Кожен з цих провайдерів має свої переваги.

Переваги Azure:

- Проста інтеграція з продуктами Microsoft.
- Гнучкість і масштабованість.
- Велика кількість доступних сервісів.

- Велика кількість дата-центрів по всьому світу.
- Підтримка усіх реляційних баз даних.
- SDK створені майже для всіх мов на платформ.
- Можливість навчити штучний інтелект на серверах.
- Можливість зберігати бекапи з різних платформ.
- Єдиний центр аутентифікації та авторизації.
- Безпека даних;

Переваги AWS:

- Велика можливість конфігурацій сервісів.
- Доступність пов’язаних сервісів.
- Гнучкий підхід до виділення ресурсів під час роботи.
- Зручна екосистема.
- Налаштований Load balancer.
- Окремий сервіс логування.
- Детальна документація.
- Розділення акаунтів по доступам та регіонам.
- Підтримка більшості реляційних баз даних та документ-орієнтованих;

3. ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ ВЕБ-СЕРВІСУ

Проектування є критично важливим етапом, оскільки він буде визначати успіх та ефективність сервісу, тому дуже важливо вибрати інструменти, програмні модулі та правильно спроектувати сервіс.

3.1. Інструменти для розробки

Найголовнішим компонентом будь-якого продукту є інструменти розробки, оскільки безпосередньо впливають на швидкість створення, також зручність розробки є важливою складовою, оскільки всі звикли до сучасних серед розробки, які роблять підказки, а деякі можуть навіть покращувати код, чи підказувати контекст.

3.1.1. Visual Studio 2022

Visual Studio 2022 Enterprise (VHF9H-NXBBB-638P6-6JHCY-88JWH) [6] від Microsoft, була використана як інтегрована середа розробки. Підтримує аналіз, рефакторинг, дебагінг коду. Є можливість переглядати так званий MSIL код [7] та код від декомпільованих бібліотек. Присутня можливість вкладок, закріплень, розширень, брекпоентів та оглядач бази даних (див. рис. 3.1.1).

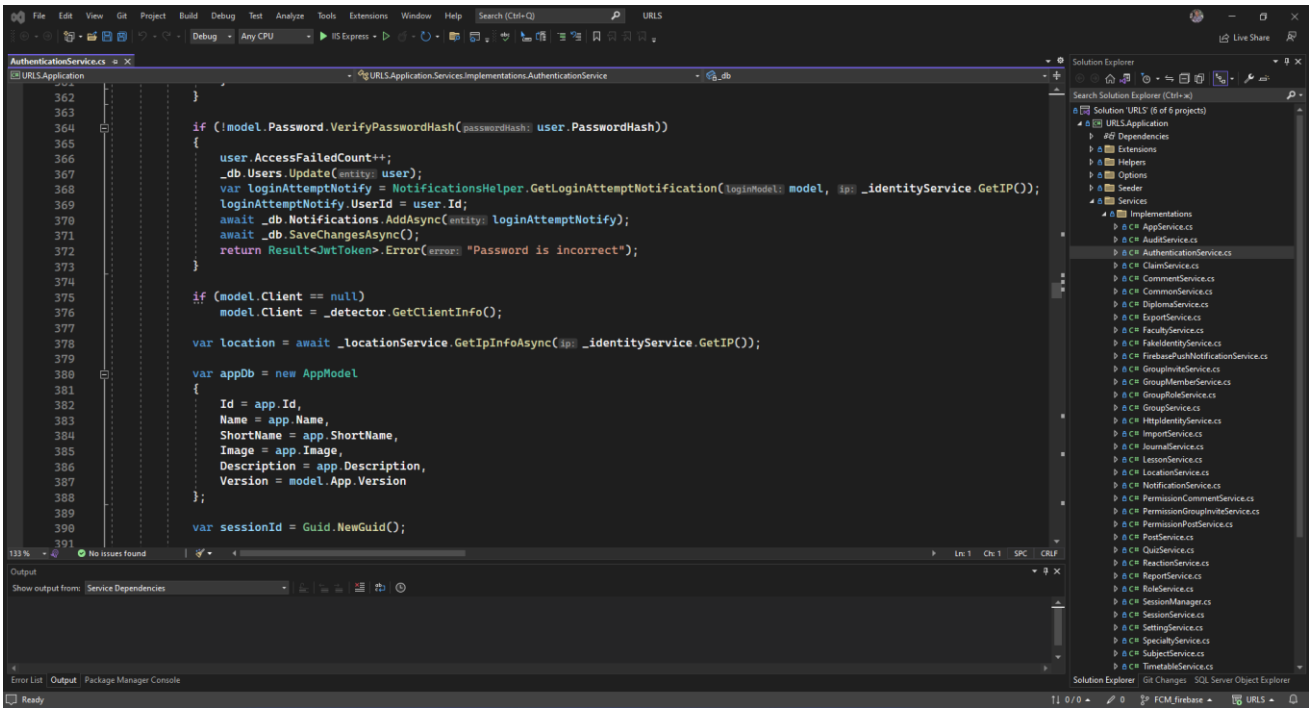


Рисунок 3.1.1 – Visual Studio 2022

3.1.2. Git

Git є однією з найбільш поширених та потужних серед подібних систем. Вона дозволяє відслідковувати кожну зміну у файлах проекту, додавати нові файли, замінювати старі, створювати гілки, ревертатися до попередніх. Ці всі функції достатньо сильно спрощують розробку в командах. Для взаємодії з системою існують 3 типи клієнтів: консольний, десктопний так вбудований. Майже всі сучасні IDE мають підтримку системи git. Під час розробки даної система використовувалися десктопна (див. рис. 3.1.2) та вбудована git (див. рис. 3.1.3).

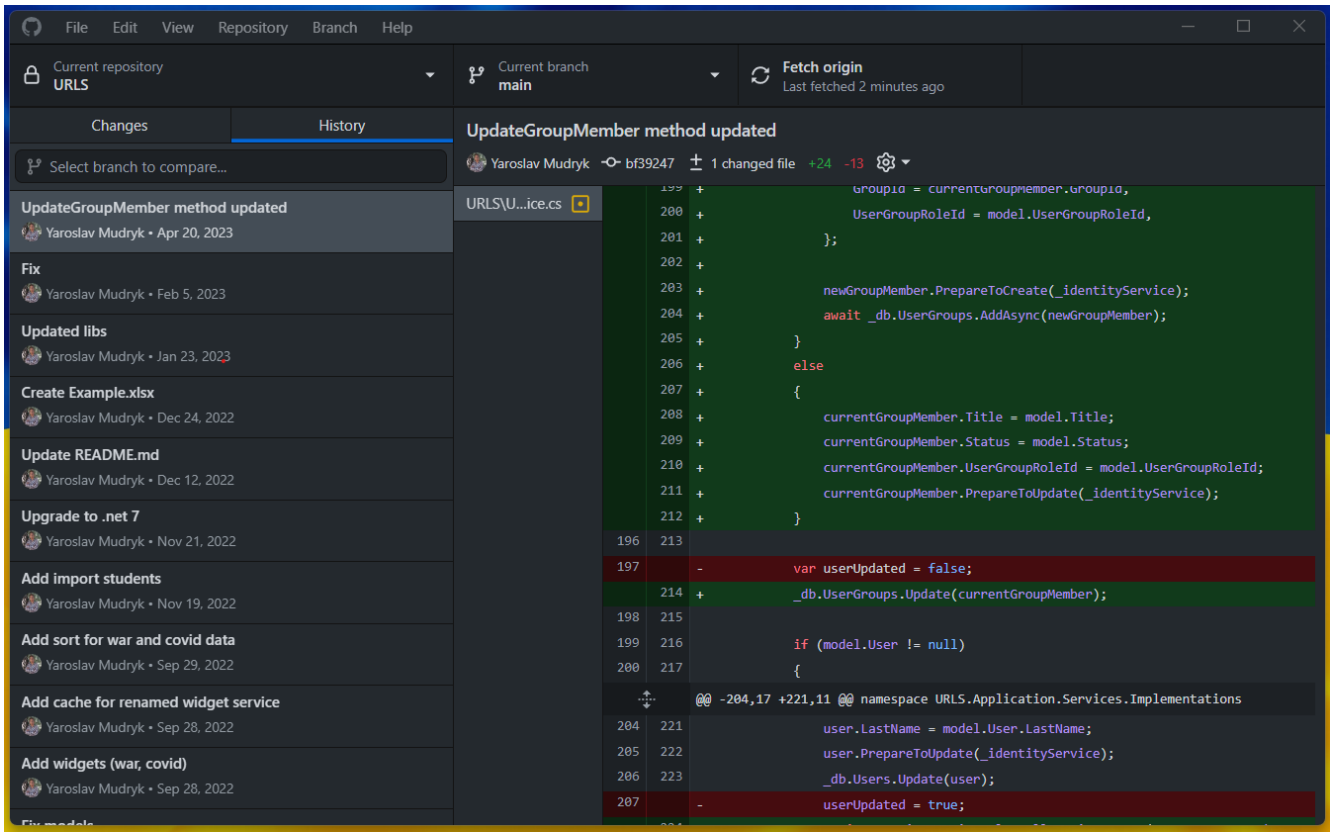


Рисунок 3.1.2 – GitHub Desktop

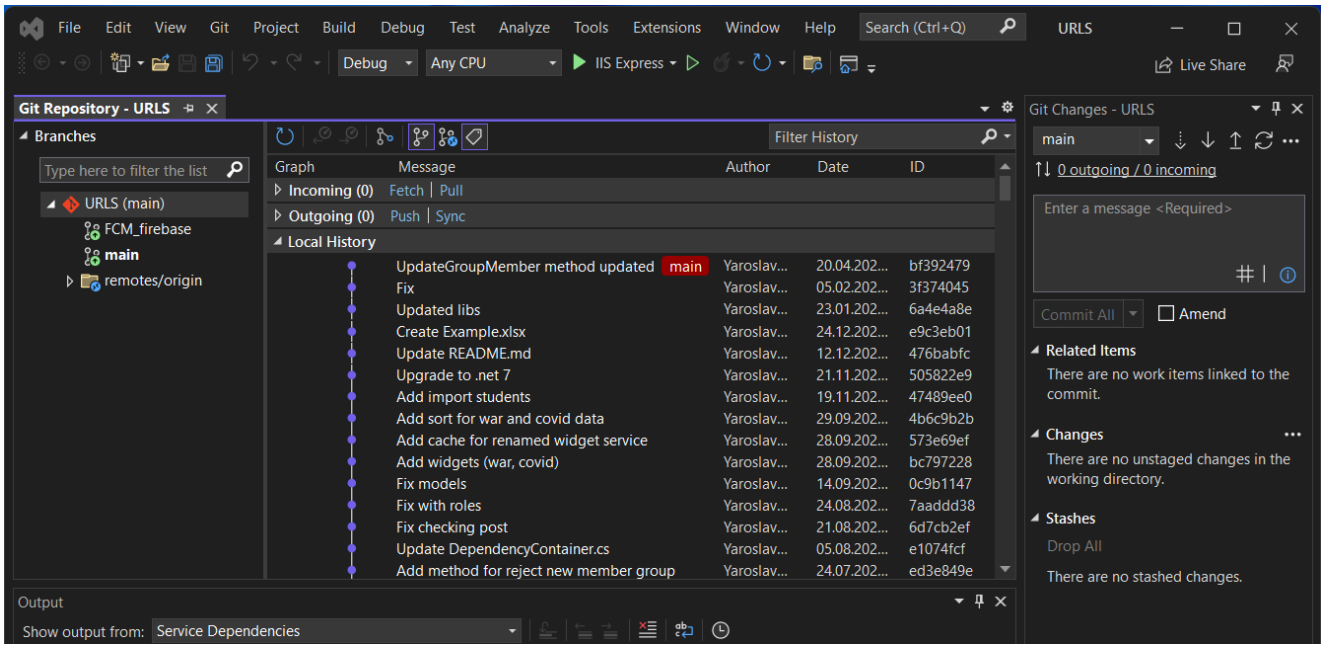


Рисунок 3.1.3 – Вбудований git

3.1.3. Azure DevOps

Azure DevOps [8] (див. рис. 3.1.4) є комплексним продуктом, який включає в

себе всі основні функції для розробки будь-якого продукту, починаючи від вікі сторінок закінчуючи git репозиторіями, які підтримують великі об’єми даних у порівняні з GitHub, який має більш обмежений функціонал. Подібні сервіси можна як орендувати так купити інстанс і розмістити на власному сервері. Для розробки даного проекту використовувалися усі можливості цього сервісу, а саме:

- Wiki-сторінки: це певна структурна одиниця, яка описує якусь поведінку системи чи контракти API.
- Dashboard-сторінки: це динамічна сторінка, яка дозволяє показувати інформацію про будь-що, це може бути як прогрес контурної тіми в спринті так і загальний прогрес по імплементації продукту або його версії.
- Дошки завдань: в кожній тімі є певна дошка завдань, яка показує поточний стан відносно завдань.
- Керування списком завдань в беклозі: кожне завдання, над яким не ведеться робота в даний момент знаходиться в беклозі, по факту в певному резерві.
- Слідкування за спринтами: якщо роботи йде в рамках Agile [9], то є певне розділення по короткотривалим термінам (спринтам).
- Аналіз ефективності та доцільності: звіт аналітики відбувається автоматично щоночі чи по запиту адміністратора.
- Репозиторії: це певне сховище даних, яке дозволяє відслідковувати зміни, створювати гілки, відновлювати зміни та ефективно працювати в командах.
- Pipelines розгортання: під час використання хмар, розробники потребують автоматизації дії (розгортання, перезапис, тощо). Цією автоматизацією займаються так звані Jobs або Pipeline (список Job).
- Збірка проекту: найважливіший компонент Job, який має зробити збірку автоматично.
- Розгортання проекту: одна з Jobs, яка робить деплой на сервер.
- Створення та керування тестами: окремий компонент системи, який відповідає за перевірку нових тестів, їх виконання, надання звіту та формування висновку по зміні роботоспроможності системи після змін;

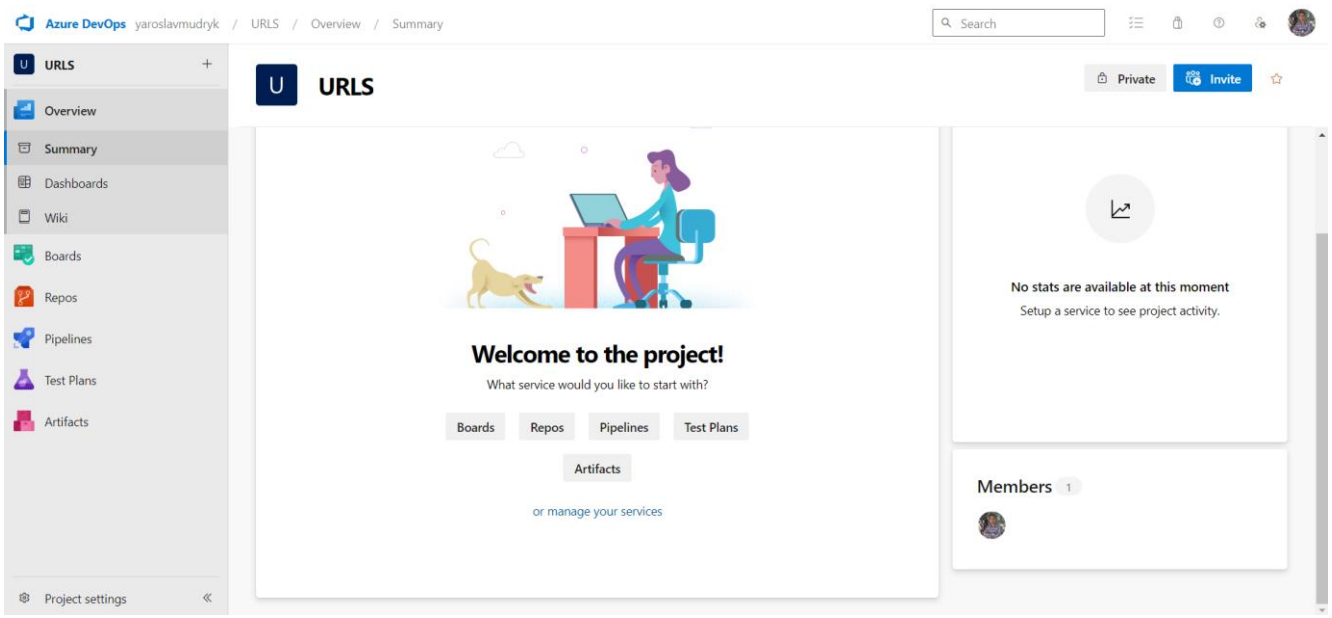


Рисунок 3.1.4 – Azure DevOps

3.1.4. DBeaver

Будь-який проект потребує БД, дана система не є виключенням, тому найважливішим інструментом є оглядач БД. Кожна СУБД зазвичай має власне ПЗ в своєму “арсеналі”, проте в цьому проекті використовується дві БД від різних розробників, а качати багато ПЗ не кожен захоче. В даному випадку було обрано саме DBeaver (див. рис. 3.1.5), оскільки він забезпечує стабільну роботу з найбільш популярними провайдерами баз даних, серед них: SqlServer, MySql, Sqlite, PostgreSql, Oracle, MariaDB, тощо. Присутня можливість роботи з NoSql та кеш сервісами формату Redis [10]. Кількість функціоналу є достатньо великою. Можна переглядати дані (див. рис. 3.1.5), переглядати структуру таблиці (див. рис. 3.1.6) чи навіть модифікувати окремі поля (див. рис. 3.1.7).

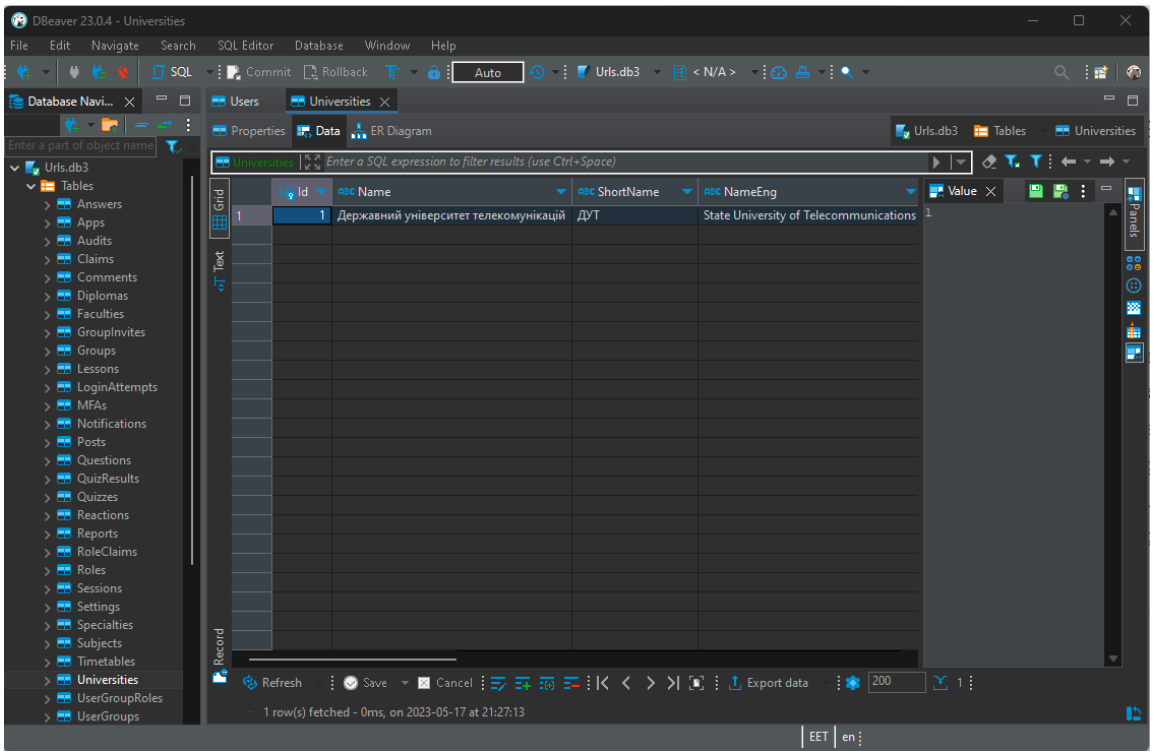


Рисунок 3.1.5 – Дані DBeaver

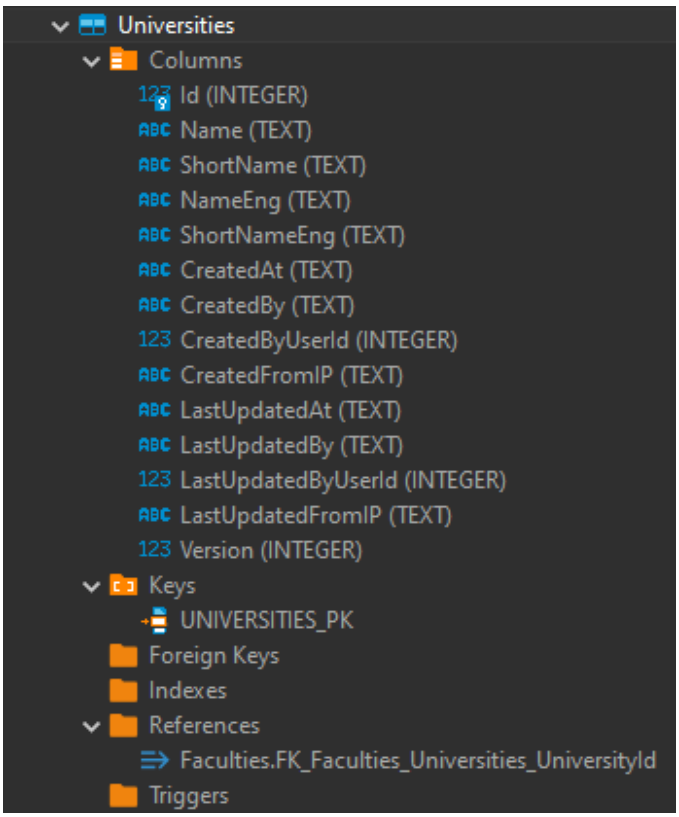


Рисунок 3.1.6 – Структура таблиці

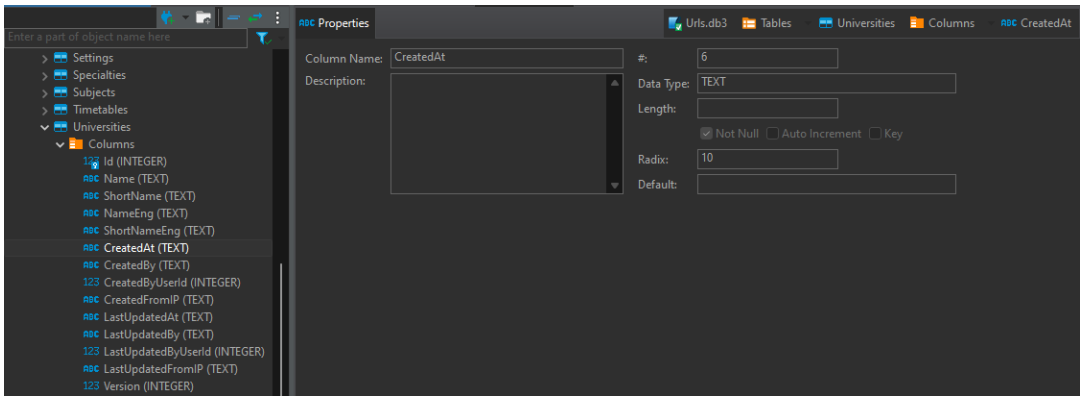


Рисунок 3.1.7 – Модифікація поля

3.2. Архітектура системи

Даний веб-сервіс використовує архітектуру типу моноліт [11], ця структурна архітектура набагато простіше у розробці, підтримці та має більше продуктивність у випадках, якщо це не дуже великий проект.

Моноліт дозволяє:

- Займатися розробкою у спрощеному підході.
- Розгортати більш швидко.
- Економити на ресурсах.
- Економити гроші витрачені на проект.
- Пришвидшити роботу за рахунок відсутності транспортування пакетів;

Рішення складається з семи проектів, які поділені наступним чином:

- Аутентифікація та авторизація: проект який включає в себе усі функції по реєстрації, входу, сесіям, перевірки ролей, доступів та зміни користувачів.
- Університетська структура: проект, що включає в себе усі обробки та зберігання структури університету, інституту, кафедр та деканатів.
- Груповий: проект, який має відповідальність за групи, а саме інформацію про групу, учасників та запрошення.
- Навчальний процес: проект в якому йде обробка навчального процесу, а саме робота з предметами, парами, журналом, оцінками та відомістю.
- Тестування: проект, який забезпечує можливість повноцінної системи

тестування.

- Сповіщення: проект, який забезпечує безперебійну роботу сповіщень усім користувачам.
- Відгуки: проект відповідальний за прийом і обробку відгуків користувачів системи для її покращення або надсилання звітності по багам.
- Web: проект (API), який отримує запити від клієнтів та відповідає за комунікацію;

Загальна архітектура виглядає подібним чином (див. рис. 3.2.1).

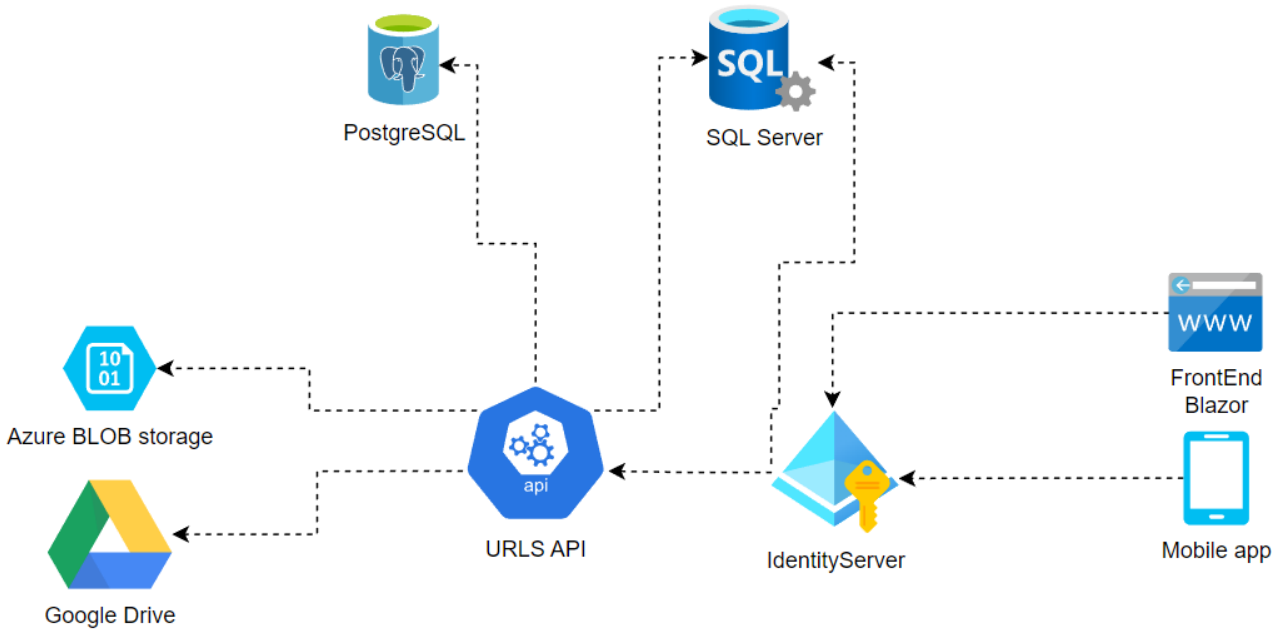


Рисунок 3.2.1 – Архітектура системи

3.2.1. Реалізація моноліту на ASP.NET 7.0

Фреймворк ASP.NET 7.0 (див. рис. 3.2.2) та мова програмування C# 11 (.NET 7.0) є фундаментом цього проекту, а безпосередньо розробка велася у IDE Visual Studio 2022 Enterprise (VHF9H-NXBBB-638P6-6JHCY-88JWH).

ASP.NET – це фреймворк який дозволяє розробляти веб-застосунки по сучасним підходам, підтримується роботи у хмарах. Працює як на операційних системах Windows так і Linux з MacOS. Включає величезну кількість бібліотек, які спрощують розробку у всіх місцях, від створення контрактів до роботи з базою.

Фреймворк має такі переваги:

- Кросплатформеність: дозволяє працювати на багатьох операційних системах.
- Продуктивність: за рахунок того, що фреймворк був повністю переписаний в 2015 році вдалося впровадити усі сучасні підходи та переваги .NET платформи. Швидкість виконання може перевищувати 7 000 000 запитів на секунду.
- Модульність: будь-який застосунок складається з маленьких частинок, тому має перевагу у гнучкості налаштування, можна підключити тільки потрібні бібліотеки.
- Web-services: побудова веб-сервісів підтримується з “коробки”, робота по HTTPS, GRPC [12] та SOAP [13] протоколам.
- Безпека: фреймворк включає велику кількість базових бібліотек для налаштувань безпеки та доступу.
- Гнучкість використання: цей фреймворк можна запускати як на власно розробленому сервері Kestrel [14] так і на сторонніх, типу IIS, HTTP.sys, Nginx, Apache, тощо.
- Технологій реально часу: підтримується можливість роботи в реальному часі по протоколам: Web-Socket, SSE через бібліотеку SignalR.
- IoC контейнер: також присутній вбудований контейнер залежностей, який спрощує роботу з абстракцією.
- Простота міграції: до даного фреймворку існував ASP.NET Framework, який працював тільки на ОС Windows і не мав такої модульності, тому присутні інструменти по міграції з старого asp.net на новий.
- Розгортання: даний фреймворк має вбудовані інструменти розгортання для будь-якої платформи, від Docker [15] до хмарних середовищ.
- Blazor: присутня бібліотека, яка дозволяє писати UI на C# та Razor Pages;

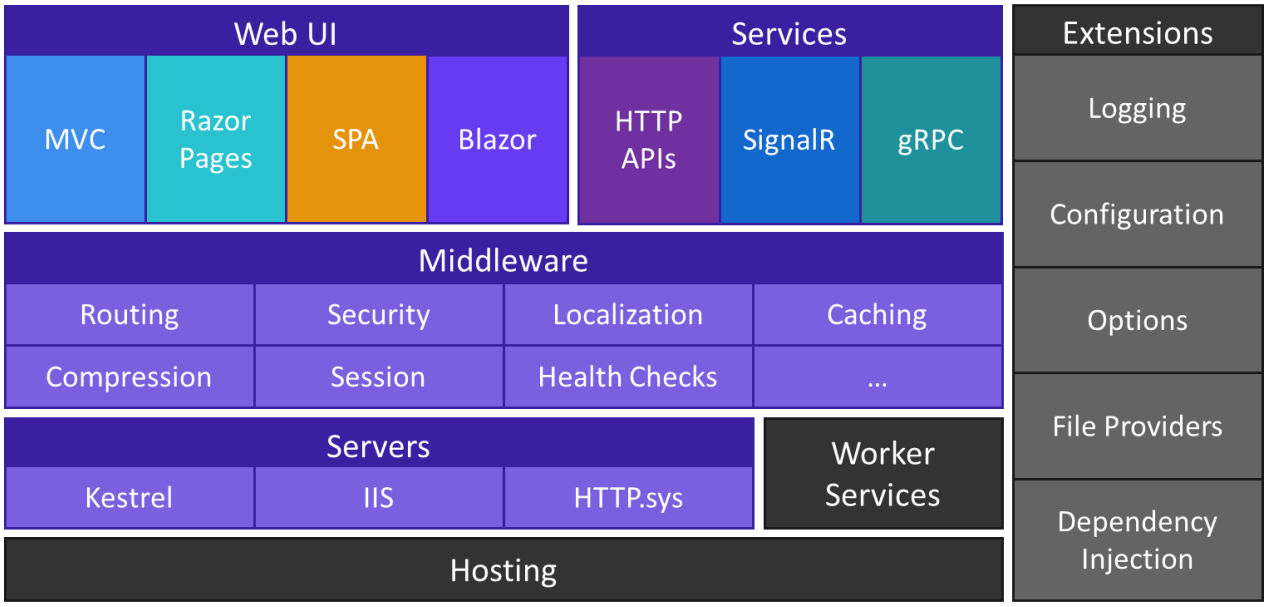


Рисунок 3.2.2 – ASP.NET

3.2.2. SQLServer та PostgreSQL як джерела даних

Реляційні (SQL) бази є найпоширенішим типом баз даних у світі. Їх основою є реляційна модель, створена у 1970-х роках. При цьому підході усі дані зберігаються у таблицях, а кожна таблиця має стовпці а рядки. Рядки по факту є окремими записами, а стовпці описують тип даних, в якому форматі можна зберігати. Ефективність цих типів баз даних довели за великий проміжок часу. Існують багато інструментів, реалізацій та документацій по покращенню взаємодії з БД.

Основою взаємодії з БД є SQL (structured query language), це спеціальна мова запитів до джерел даних, за допомогою неї можна писати звичайні запити, збережені функції та викликати зарезервовані слова.

SQL в свою чергу складається з:

- DDL (data definition language): мова для структуризації бази.
- DML (data manipulation language): мова для маніпуляції даними.
- DCL (data control language): мова для роботи з правами доступу.
- TCL (transaction control language): мова для роботи з транзакціями;

З часом сформувалися вимоги до систем, які можуть забезпечувати стабільну і передбачену роботу системи – ACID [16], це акронім від слів Atomicity, Consistency, Isolation, Durability.

Atomicity (атомарність): це стан системи коли під час виконання транзакції стається помилка і всі минулі операції відкочуються до вихідного стану.

Consistency (послідовність): успішне виконання транзакції говорить про те що, в базі фіксовані тільки допустимі результати.

Isolation (ізоляція): умова при якій під час виконанні однієї транзакції паралельні транзакції не повинні впливати на неї.

Durability (довговічність): після успішного виконання транзакції гарантується, що зроблені зміни не будуть відмінені через збій системи.

SQLServer та PostgreSQL є прямими конкурентами, їхній функціонал багато в чому схожий, але є також і різниця між ними. Наприклад в тому, що SQLServer є кооперативним продуктом від Microsoft, а PostgreSQL повністю відкритий продукт.

Основною перевагою PostgreSQL є її оптимізація під Unix системи, в свою чергу продукт від Microsoft почав підтримувати Unix системи відносно нещодавно, тому ще не всі функції стабільно швидко працюють як на системі Windows. SQLServer має декілька планів підписок продукту, для навчання, для розробки, для малого бізнесу і для ентерпрайз систем. В залежності від планів швидкодія може змінюватися.

Порівняння цих двох БД краще подивитися нижче (див. табл. 3.2.1). Тестування проводилось в максимально схожих умовах і на одному девайсі, який включає в себе наступні характеристики:

- CPU: AMD Ryzen 7 5800H
- RAM: 32GB DDR4 (3200 Ghz)
- SSD: Samsung 970 Evo Plus 2TB (3500/3300 Mb/s)

При цьому було встановлено SQL Server 2019.22 Developer Edition та PostgreSQL 15.03.

Таблиця 3.2.1 – Порівняння продуктивності

Дія	SQL Server	PostgreSQL
INSERT (1000 count)	504ms	721ms
INSERT (10000 count)	4912ms	8372ms
SELECT (1000 count)	16ms	23ms
SELECT (10000 count)	50ms	49ms
SELECT (500 count with WHERE)	33ms	38ms
UPDATE (1000 count)	519ms	653ms
UPDATE (10000 count)	5598ms	4927ms
REMOVE (1000 count)	301ms	300ms
REMOVE (10000 count)	708ms	458ms

Як ми бачимо з результатів продуктивність SQLServer трохи краща, проте в деяких сценаріях PostgreSQL випереджає.

Отже, можна зробити деякий висновок, що вибираючи SQLServer ви отримуєте максимальну продуктивність, проте треба платити. Вибравши PostgreSQL ви отримуєте якісний продукт в рамках безоплатної основи.

3.2.3. Azure BLOB storage і Google Drive

Кожний проект має потребу зберігати файли, зображення, документи, відео чи архіви. Всі об’єкти мають різний розмір, аватарки можуть займати по 1Мб в той час, як відеоматеріал може займати десятки гігабайт. Тому використання сховища є дуже важливим компонентом. В даному проекті було вирішено зберігати дані в двох різних сховищах в Azure BLOB storage та Google Drive, є можливість налаштувати роботу як в паралельному режимі так і односторонньому. Якщо ми кажемо про надійність, то звичайно краще користуватися у паралельному режимі, проте це може зайняти багато місця і коштувати відносно дорого.

Google Drive – це хмарний сервіс, який пропонує місце на сервері за плату. Є можливість синхронізації через Windows, Android та iOS. Безкоштовно надається 15 ГБ. При потребі є можливість збільшити це місце до 100 ГБ за 45 грн/м та на

200 ГБ за 80 грн/м (див. рис. 3.2.3). Для навчальних закладів існує можливість купити або отримати доступ до 100TB (див. рис. 3.2.4).

Azure BLOB storage – це сервіс, який допомагає з зберіганням даних у хмарі. Сховище цього сервісу оптимізоване під зберігання даних, які не є структоризованими, це означає, що формат зберігання може бути будь-який від текстового до файлів формату .uasset. Ціни в даному сервісі є більшими, проте і рахуються по іншій формулі, тут вказується ціна за 1 ГБ і вона дорівнює 0.15$ (див. рис. 3.2.5).

Так, ці сервіси мають різну цінову політику, різний підхід для зберігання, але мають і подібний функціонал. Наприклад: можливість створення папок, відслідковування змін, редагування файлів прямо у хмарі, перегляд в онлайн режимі, створення посилання на завантаження та роботу з бібліотеками в коді C#.

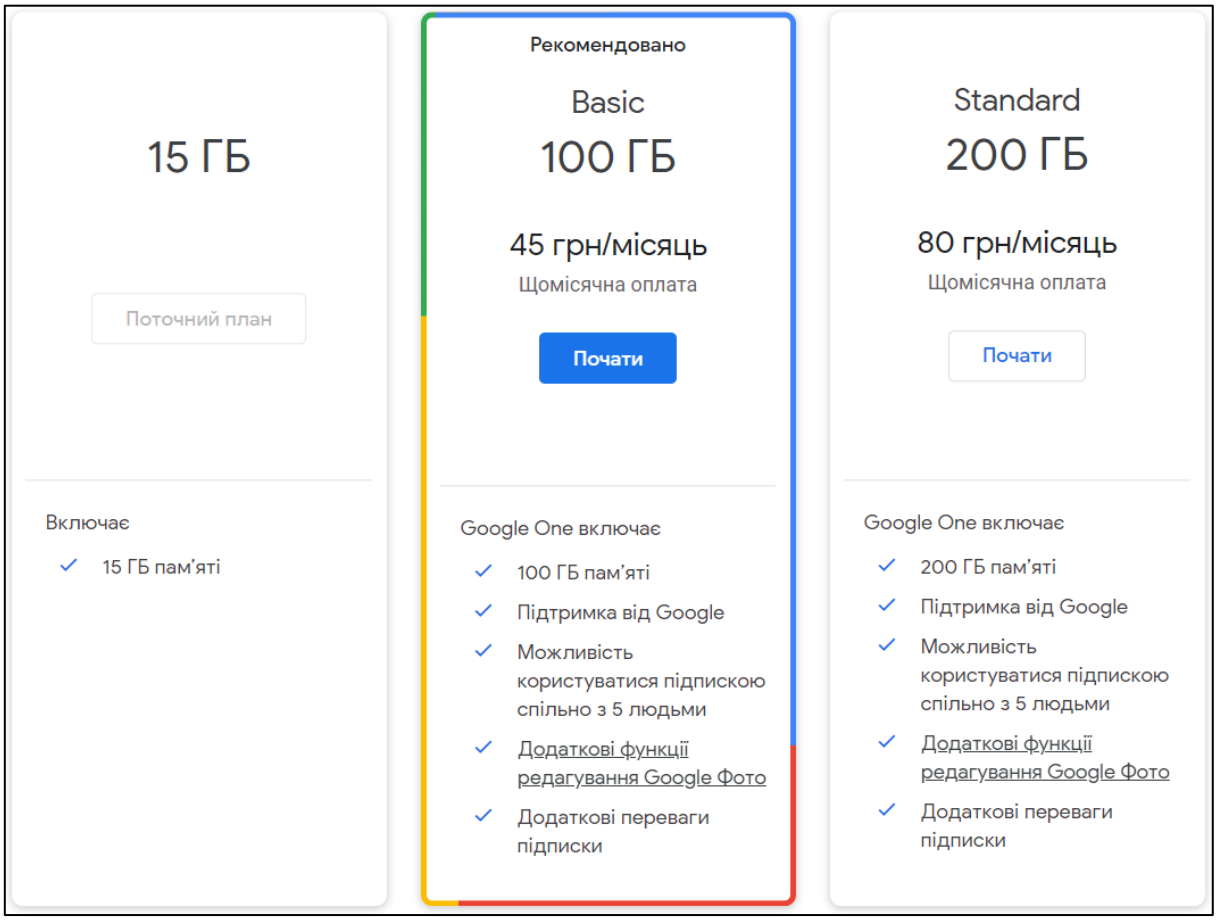


Рисунок 3.2.3 – Google Drive плани

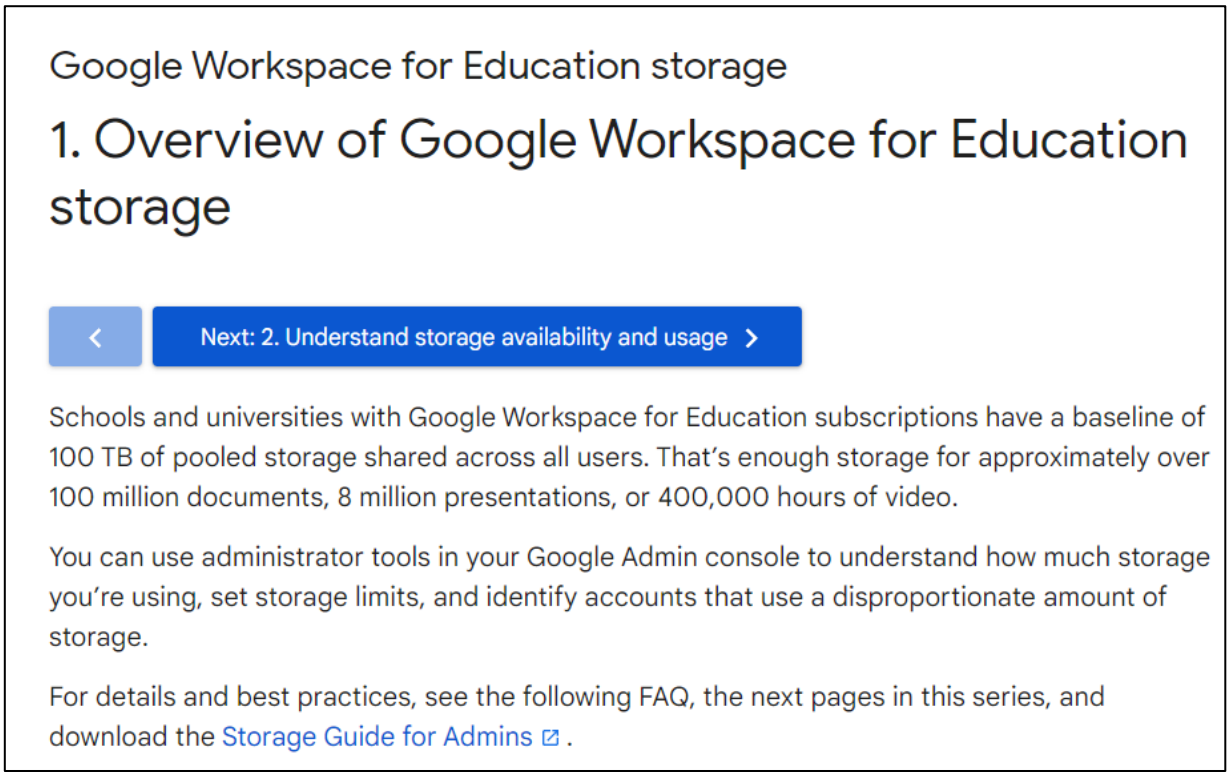


Рисунок 3.2.4 – Google Drive Education план

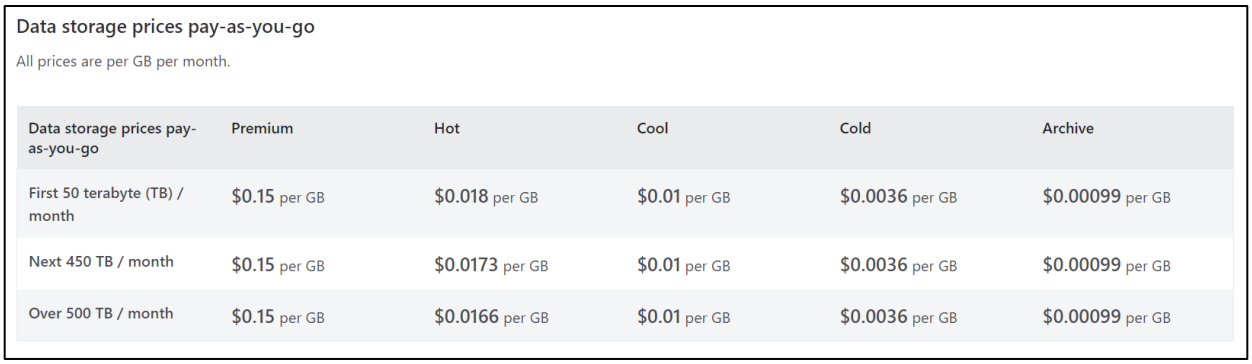


Рисунок 3.2.5 – Azure Storage ціни

3.2.4. Структура рішення

Даний проект є монолітом, найвищий рівень системи це рішення, яке включає в себе проекти (див. рис. 3.2.6). Для початку було вирішено розробити моноліт, проте розділити їх на 6 проектів, які в майбутньому при необхідності можна було б безболісно винести і зробити мікро-сервісну архітектуру.

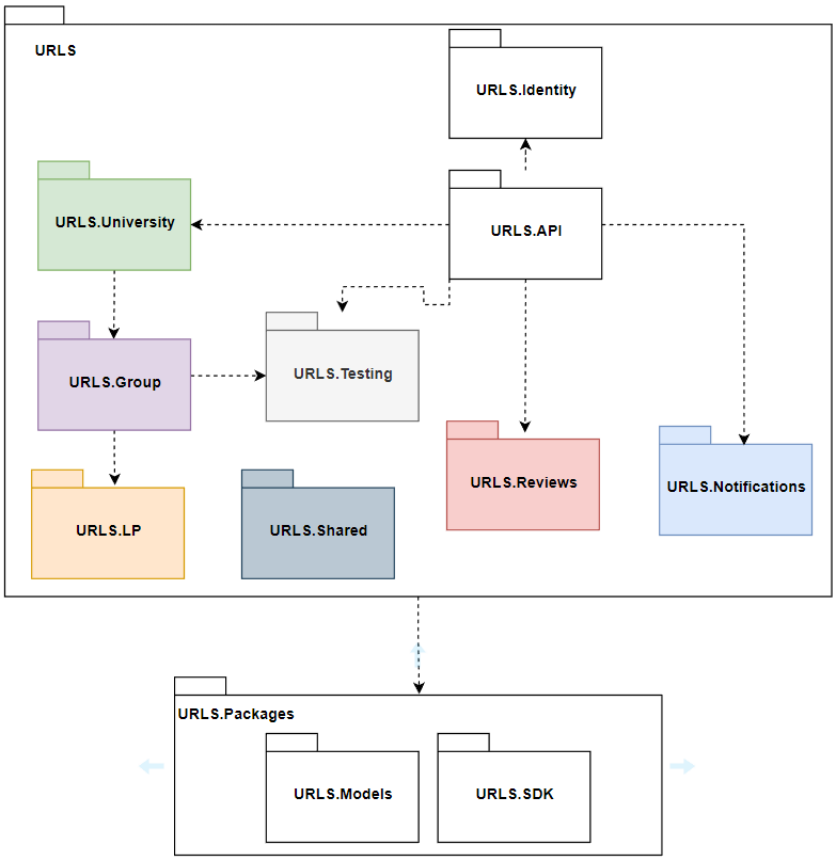


Рисунок 3.2.6 – Діаграма пакетів

Увесь доступний функціонал системи розділений на бібліотеки та пакети, які можуть зберігається в Azure DevOps, також частина з них представлена в nuget.org, для більш детальної інформації про пакети (див. табл. 3.2.2).

Таблиця 3.2.2 – Опис пакетів

Назва пакету	Опис
URLS	Головний пакет, який містить весь функціонал системи
URLS.Identity	Пакет, який відповідає за функціонал авторизації, аутентифікації та зберігання даних користувачів
URLS.API	Пакет, який відповідає за комунікацію з зовнішніми клієнтами
URLS.University	Пакет, що включає в себе структуру університету, його кафедр та зберігання

URLS.Group	Пакет, який відповідає за роботу з групами, кодами-приєднання та учасниками
URLS.LP	Пакет, який відповідає за функціонал навчального процесу. Робота з предметами, парами, домашнім завданням, електронним журналом та оцінками
URLS.Testing	Пакет, відповідальний за функціонал створення, проходження чи зміну тестів
URLS.Reviews	Пакет, який виступає у ролі отримувача і отримує відгуки про роботу системи чи баги, які є критичними
URLS.Notifications	Пакет, який відповідальний за надсилання сповіщень користувачам
URLS.Shared	Пакет, який включає в себе загальні константи, розширення, функціонал, тощо
URLS.Models	Пакет з моделями, який використовується для більш простої інтеграції
URLS.SDK	Пакет, який має функціонал використання API для сторонніх сервісів

Більшість з цих пакетів є окремими проектами, тому функціонал так різниться, через деякі обмеження в характеристиках девайсу рішення, проекту та бібліотеку було вимушено зробити окремими частинами коду.

3.3. Моделювання та налаштування баз даних

3.3.1. ORM

При побудові будь-якого сервісу буде обов’язково робота з базою даних, тому бібліотеки є важливою складовою. Існує декілька підходів роботи з базою даних. Перший це звичайно створення запитів переважно мовою SQL, отримання результату та його обробку в зрозумілу відповідь для серверу. Другий спосіб ORM,

вважається набагато простішим у використанні, проте деякі бібліотеки мають поганий перекладач у створені запитів. ORM або object-relational mapping – це технологія, яка допомагає в обробці запитів бази даних. Основний принцип полягає в тому, що кожна окрема таблиця є певним класом чи об’єктом в коді. Зазвичай ORM підтримують увесь спектр CRUD операцій. Перевага цієї технології в тому, що зазвичай під кожну конкретну базу даних використовується певний провайдер, який знає синтаксис певної бази і використовує його для формування запитів. Швидкість виконання та правильність формування запитів залежить вже від конкретної мови та бібліотеки.

Бібліотека Entity Framework Core 7.0 може працювати з наступними провайдерами:

- SqlServer.
- Sqlite.
- InMemory.
- CosmosDb.
- PostgreSQL.
- MySql.
- Oracle.
- Firebird.
- Ibm.
- Google.Cloud;

Оскільки вона вміє працювати одночасно як з SqlServer та і з PostgreSQL, тому було вирішено обрати її для роботи як основний.

3.3.2. Таблиці SQL Server

Схема бази даних після закінчення проектування можна побачити на ER-Diagram (див. рис. 3.3.1, 3.3.2, 3.2.3)

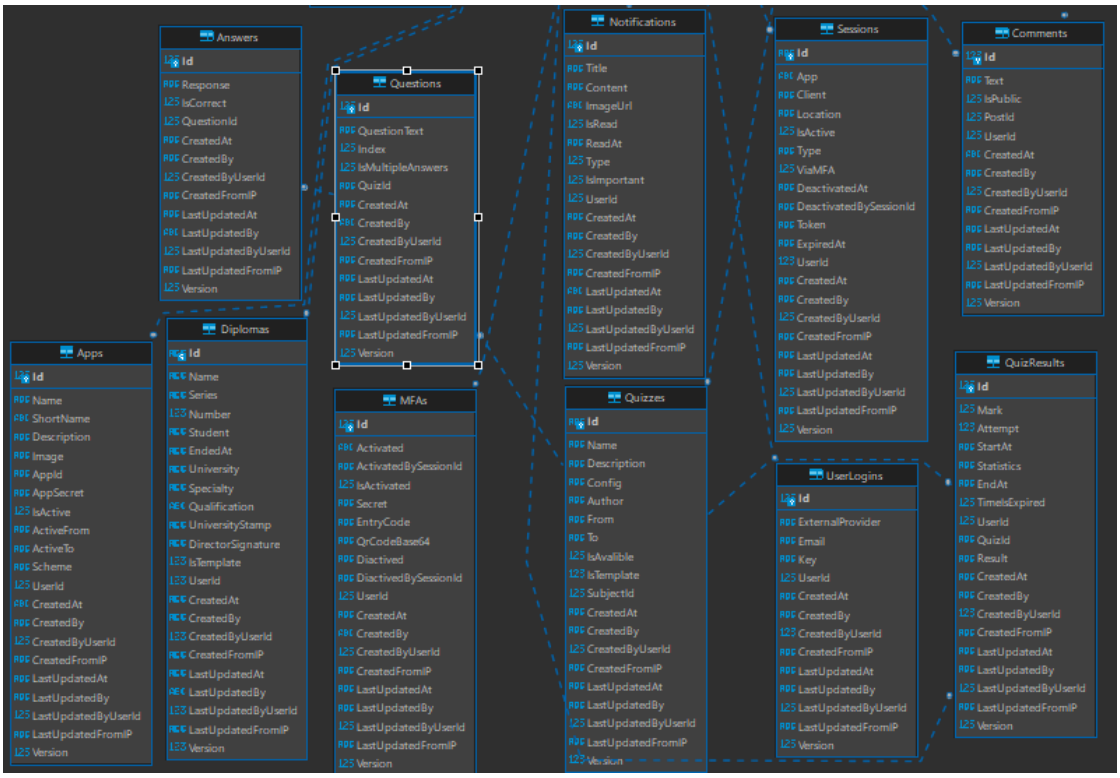


Рисунок 3.3.1 – Схема БД частина 1

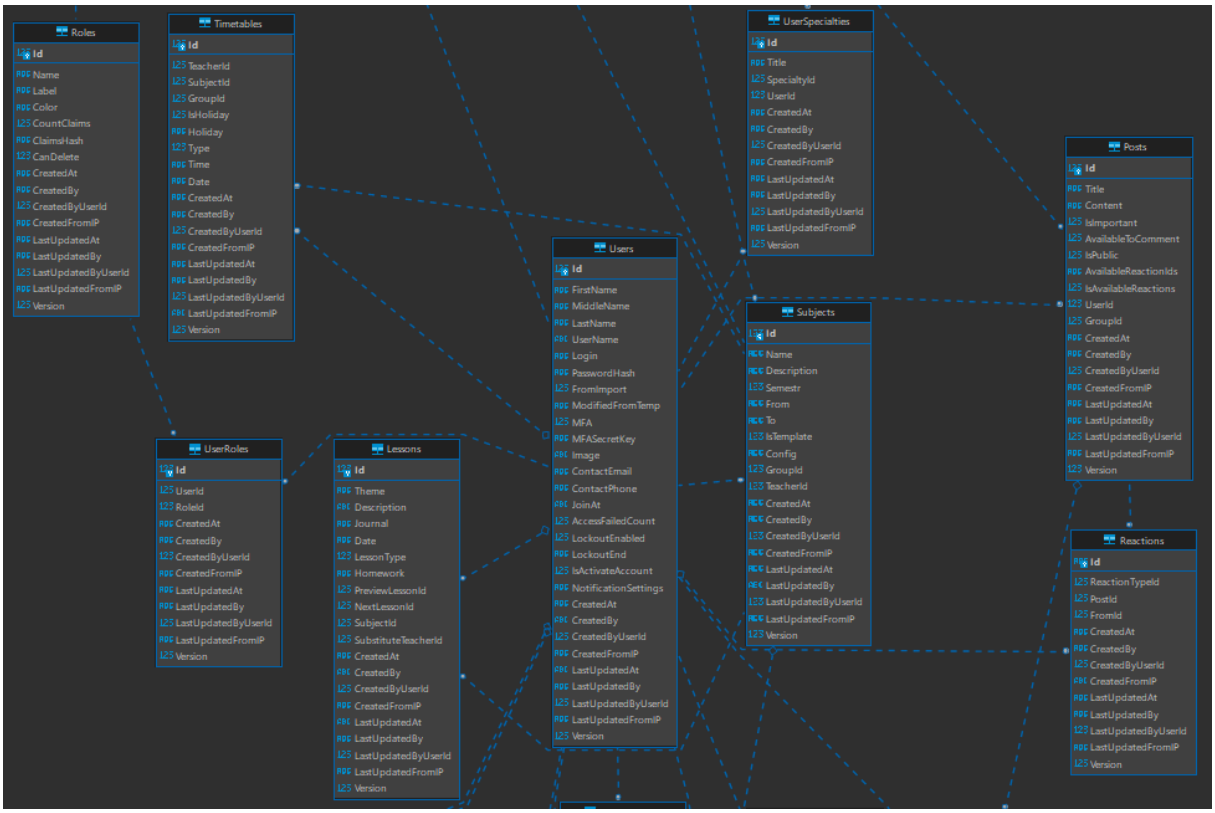


Рисунок 3.3.2 – Схема БД частина 2

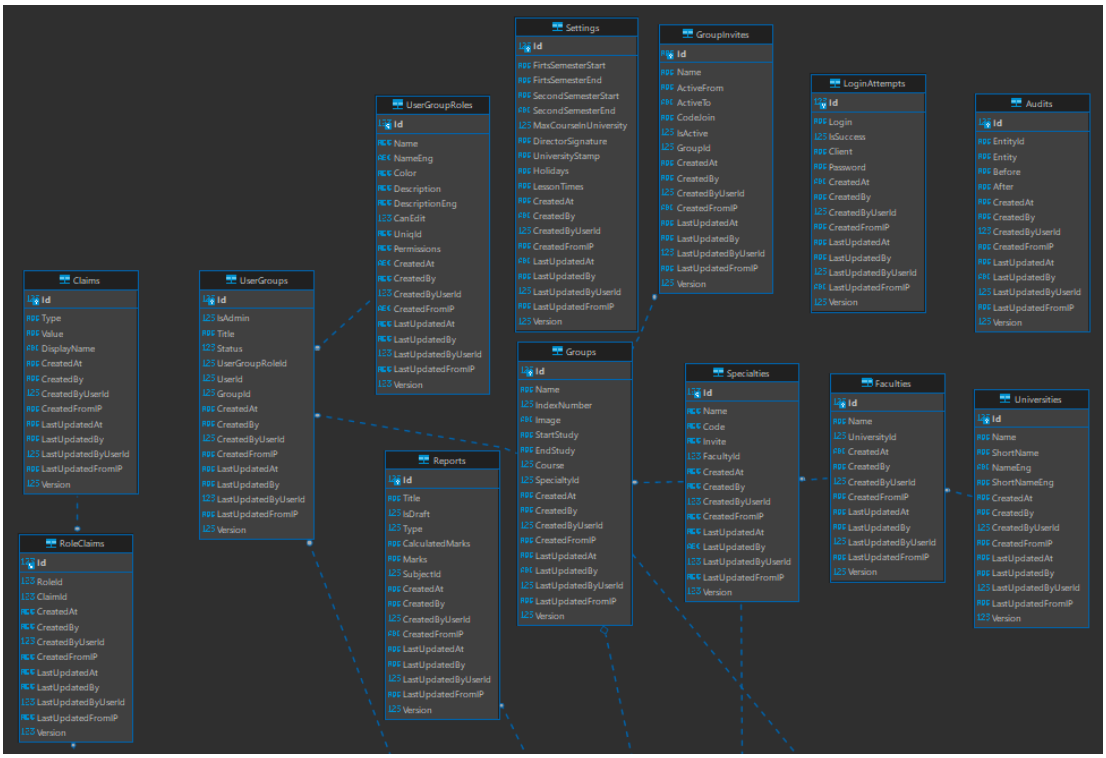


Рисунок 3.3.3 – Схема БД частина 3

3.3.3. Таблиці PostgreSQL

Схема бази даних представлена на ER-Діаграмі, яка демонструє сутності та зв’язки (див. рис. 3.3.4, 3.3.5).

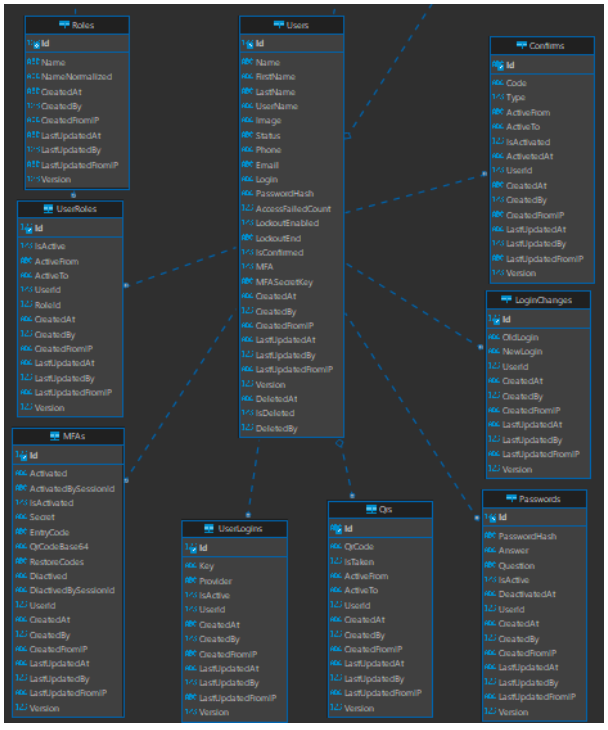


Рисунок 3.3.4 – Схема БД частина 1

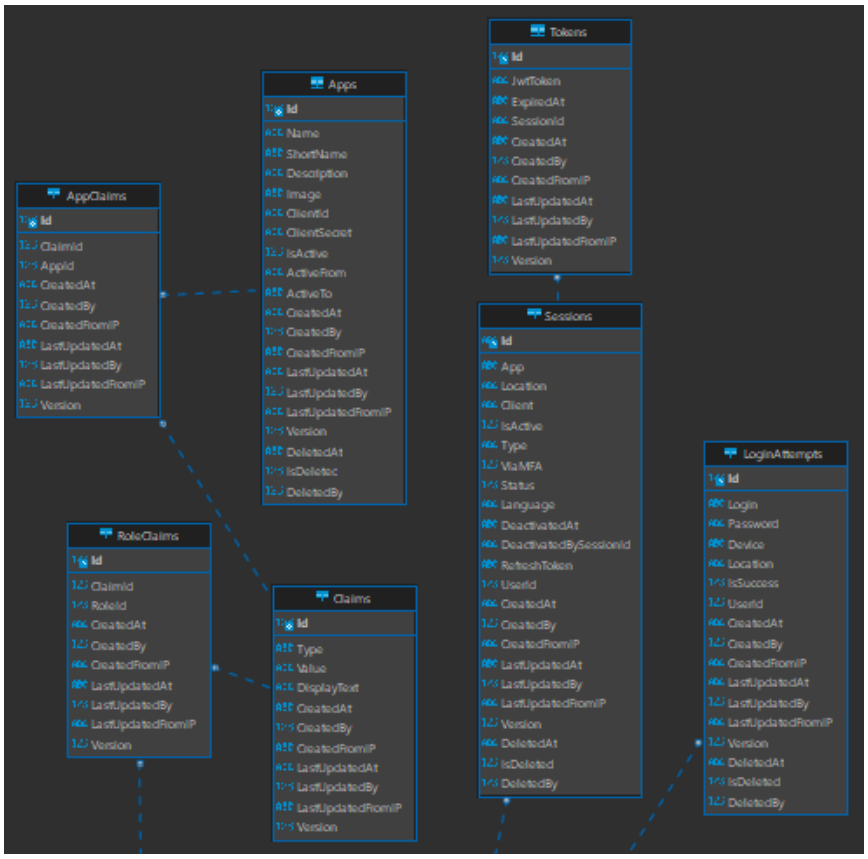


Рисунок 3.3.5 – Схема БД частина 2

3.3.4. Міграції

Міграція – це процес який змінює структуру бази даних в автоматичному режимі. Міграції є важливою компонентом при роботи з ORM, оскільки таблиці в БД та класи в коді мають бути однаковими для коректної роботи.

Уявимо ситуація, з’явилась потреба додати декілька полів в якусь таблицю. Для цього необхідно спочатку додати поля в клас, після чого виконати деякі команди, щоб створити міграцію. Для застосування змін в БД необхідно виконати ще одну команду і таким чином в результаті структура БД і коду буде синхронізована.

Особливості ORM міграцій:

- Автоматичне налаштування структури БД.
- Контрольованість версіями.
- Снепшоти даних до міграції.
- Портативність.

3.4. Аутентифікація та авторизація

Найголовнішим компонентом роботи складних систем є видача доступу та його перевірка під час роботи. Оскільки від цього залежить ефективність та гнучкість системи до керування дозволами, треба знайти правильний підхід та реалізувати в такому форматі, щоб це було просто швидко і надійно.

3.4.1. Аутентифікація

Аутентифікація – це процес перевірки користувача на ідентичність. До цього входить перевірка ідентифікатора (це може бути як пошта чи мобільний телефон так і юзернейм чи системний ID) і пароля. Також аутентифікація можлива по відбиткам пальців, розпізнаванні обличчя чи цифрового ключа. Після успішної перевірки своїх даних користувач отримує доступ до ресурсів.

В даній системі аутентифікація може проводитись декількома способами:

- Логін (пошта) та пароль: базова аутентифікація.
- DeviceId: аутентифікація, в якій проводиться аналіз пристрою, якщо на цьому пристрої попередньо був здійснений вхід, то система автоматично входить.
- ThirdParty сервіси аутентифікації: аутентифікація при якій для входу використовується стороні авторизаційні сервіси (наприклад Google чи Microsoft).

Сам процес входу відбувається в декілька етапів. При передачі логіну та паролю відправляється ще додаткові системні дані про застосунок та девайс. Це треба для ідентифікації типу застосунку та зберіганні інформації в сесії. Після перевірки застосунку йде перевірка локації пристрою. Згодом після усіх перевірок та отриманням всіх даних видається JWT-токен [17] користувачу, який зберігається у клієнта, а сам клієнтський застосунок пізніше його використовує для авторизації (див. рис. 3.4.1).

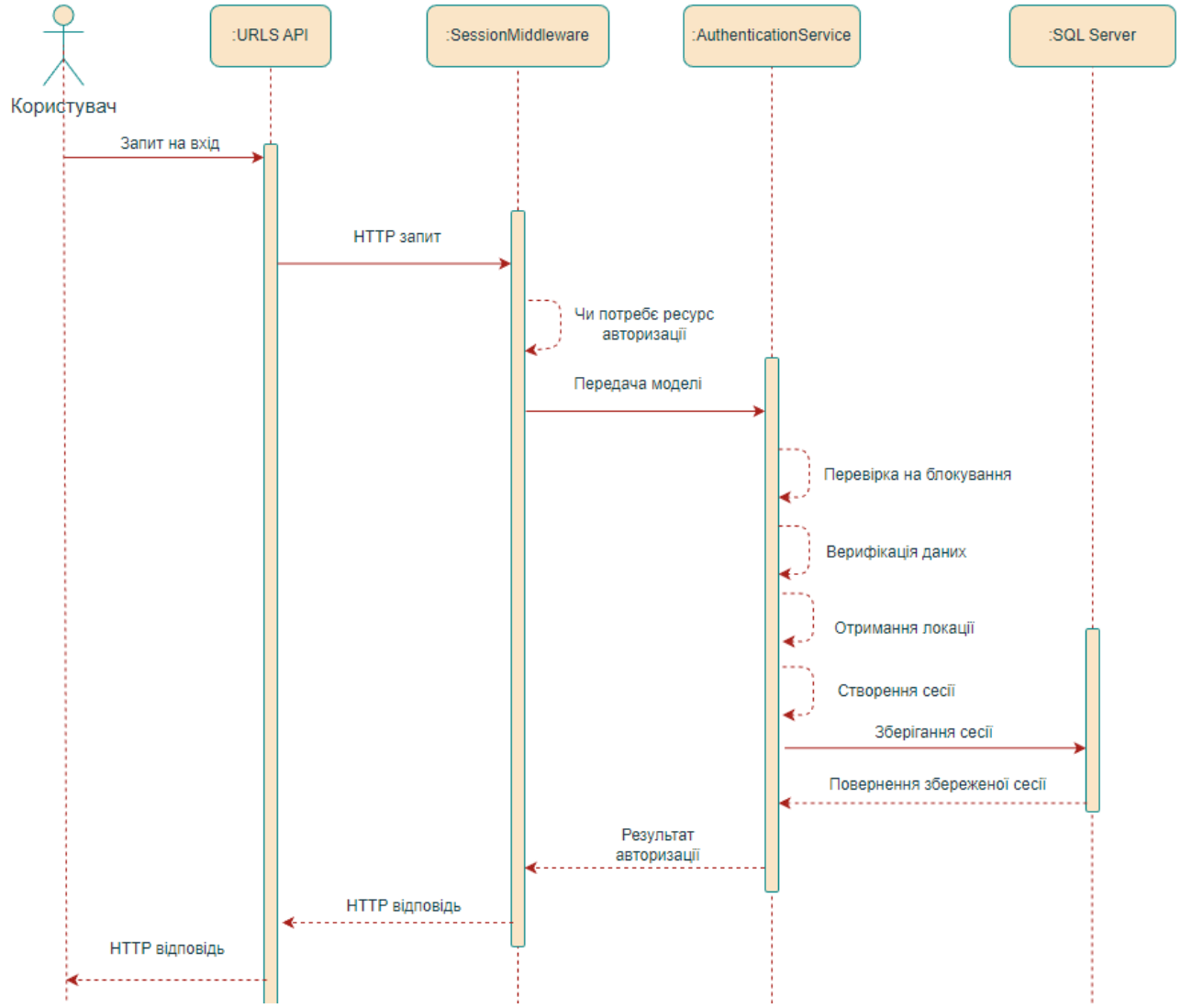
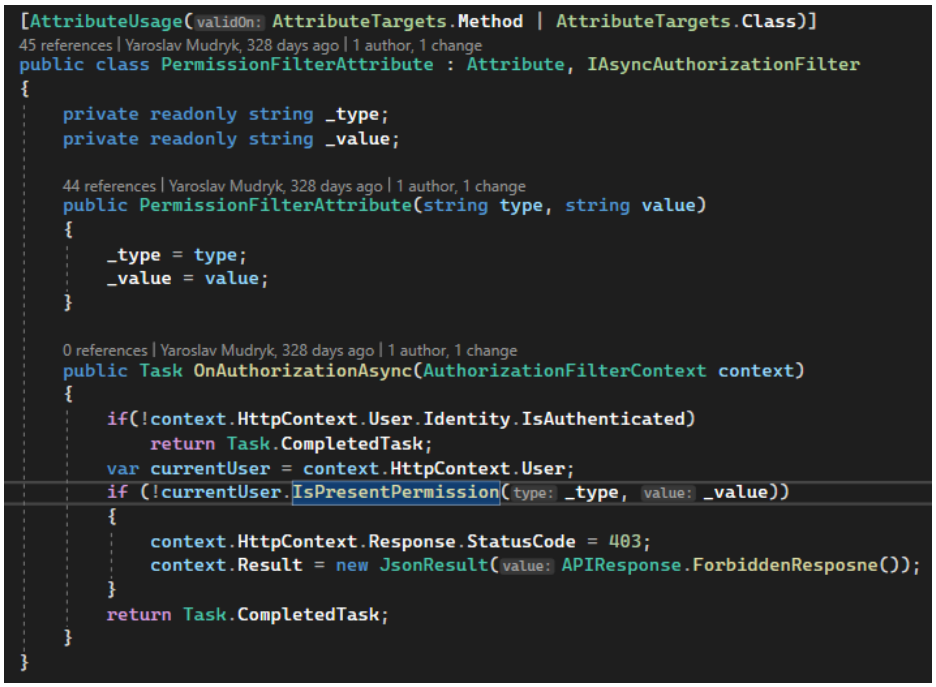


Рисунок 3.4.1 – Діаграма послідовності – вхід

3.4.2. Авторизація

Авторизація – це процес перевірки прав доступу користувача. Саме цей процес відповідає за перевірку доступності того чи іншого ресурсу користувача. Оскільки дану систему можна назвати гнучкою цей процес реалізовано достатньо ефективно. Усі базові дані (наприклад, логін, Id, юзернейм, роль, sessionId) з інформацією доступу записані в JWT-токен нема необхідності постійно робити запит до баз даних, таким чином можна зекономити багато часу та ресурсів. Проте для верифікації токену необхідний сервіс, який зберігає усі відкриті сесії у пам’яті, щоб перевірка займала мінімум часу. Для гнучкості доступу був створений

спеціальний фільтр, який обмежений доступ до ресурсу якщо внутрішня перевірка не пройде (див. рис. 3.4.2).



```
[AttributeUsage(validOn: AttributeTargets.Method | AttributeTargets.Class)]
45 references | Yaroslav Mudryk, 328 days ago | 1 author, 1 change
public class PermissionFilterAttribute : Attribute, IAsyncAuthorizationFilter
{
    private readonly string _type;
    private readonly string _value;

    44 references | Yaroslav Mudryk, 328 days ago | 1 author, 1 change
    public PermissionFilterAttribute(string type, string value)
    {
        _type = type;
        _value = value;
    }

    0 references | Yaroslav Mudryk, 328 days ago | 1 author, 1 change
    public Task OnAuthorizationAsync(AuthorizationFilterContext context)
    {
        if(!context.HttpContext.User.Identity.IsAuthenticated)
            return Task.CompletedTask;
        var currentUser = context.HttpContext.User;
        if (!currentUser.IsPresentPermission(type: _type, value: _value))
        {
            context.HttpContext.Response.StatusCode = 403;
            context.Result = new JsonResult(value: APIResponse.ForbiddenResposne());
        }
        return Task.CompletedTask;
    }
}
```

Рисунок 3.4.2 – Атрибут перевірки доступу

Проте сама авторизація відбувається по наступній схемі (див. рис. 3.4.3).

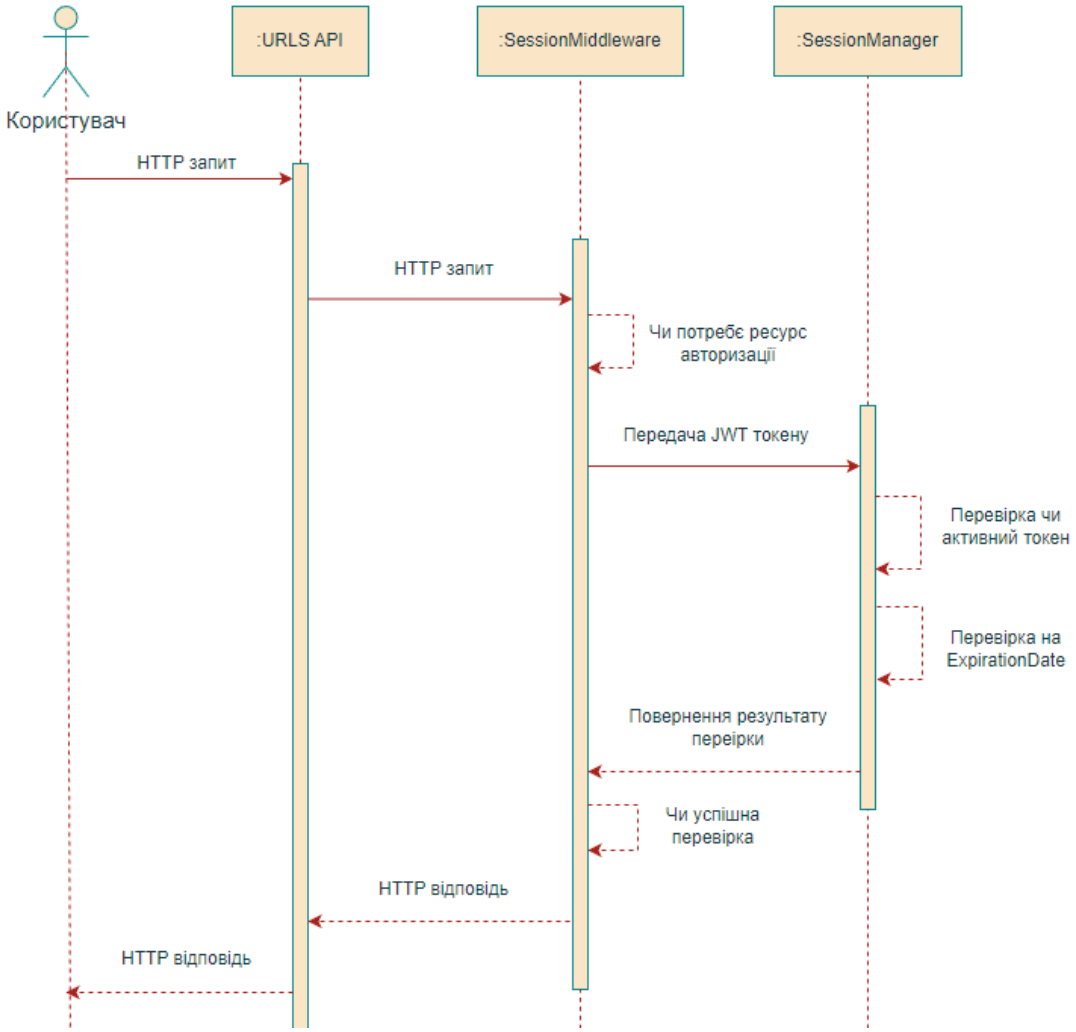


Рисунок 3.4.3 – Діаграма послідовності – виконання запиту

3.5. Сповіщення

В системі реалізовано можливість надсилання миттєвих та запланованих сповіщень засобами Push та електронною поштою. Для відправлення Push сповіщень використовується Firabase Cloud Messaging [18], а надсилання листа електронно поштою використовується бібліотека MailKit [19].

Для коректної обробки сповіщень по технології push необхідно, щоб кожен девайс був під’єднаний до сервісу Firebase, тому при вході в застосунок має виконуватися запит на отримання push токену після чого цей токен має отримати сервер URLS.

Для більш чіткого розуміння розглянемо приклад з сповіщенням під час входу на іншому пристрої: користувач вводить логін і пароль -> відправляє дані на сервер -> йде перевірка даних -> сповіщення додається в чергу -> користувач входить на новому пристрої -> сповіщення про новий вхід відправляється на інші девайси (див. рис. 3.5.1).

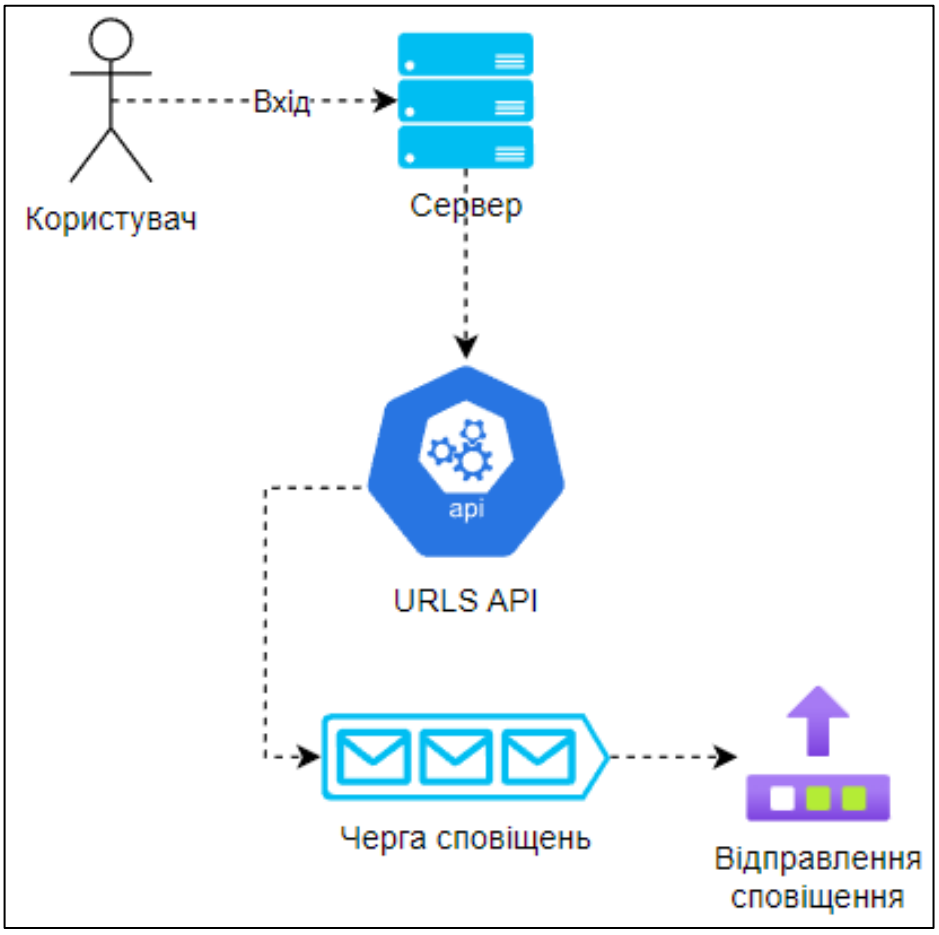


Рисунок 3.5.1 – Схема відправки сповіщень

4. ТЕСТУВАННЯ СИСТЕМИ

В програмних продуктів в звичайному форматі фінальною частиною розробки є тестування. Його можна розділити на наступні типи:

- Написання юніт тестів
- Написання інтеграційних тестів
- Ручне тестування

Написання юніт тестів та ручного тестування є необхідною умовою для випуску якісного продукту.

4.1. Unit тести

Юніт тести [20] є головним інструментом у розробці. Їх пишуть для перевірки деяких шматочків коду, котрі можна назвати “юніт”. Основна мета написання цих тестів є перевірка коректності роботи системи.

Юніт тести пишуться програмістами для перевірки правильності виконання модулів, для їх автоматизації. Вони забезпечують перевірку функціональності окремих частин програми перед повноцінною інтеграцію з головним продуктом. Кожен тест включає в себе вхідні дані, виконання коду і аналіз отриманого результату. Головна їх перевага використання це покращення якості програмного коду. Вони допомагають виявити і фіксити помилки, некоректності на всіх стадіях розробки. Також частково тести і є документацією по коду. Один тест може служити прикладом використання певного методу чи функціоналу і стати розумінням для чого потрібен.

В даному проекті використовувалась бібліотека xUnit, яка підтримує всі кейси і є достатньо простою у використанні, проте вона також вмію працювати і з складеними тестами.

Для її написання тестів необхідно встановити наступні пакети: Microsoft.NET.Test.Sdk, xunit, xunit.runner.visualstudio.

Щоб писати локальні тести інколи необхідно запрограмувати деякий

функціонал на власну відповідь, це називається мок (Mock). Це перш за все спросить обробку подій та функцій. Більшість функціоналу який тестується має багаторівневі залежності. Тому можливість замокати є життєвою необхідністю.

Для прикладу розглянемо один з тест-кейсів, який включає в себе конфігурування сервісу, мокання даних та виконання тесту з подальшими перевірками на коректність (див. рис. 4.1.1).

```
public class GroupServiceTests
{
    private IGroupService _groupService;

    public GroupServiceTests()
    {
        var groupContext = new Mock<IGroupContext>();

        groupContext.Setup(x => x.GetGroupBySpec(It.IsAny<string>())
            .Returns(() => new GroupWithRandomName());

        _groupService = new GroupService(groupContext);
    }

    [Test]
    public async Task GetGroupsBySpecialtyIdAsync_ReturnsResultWithCorrectGroupViewModelList()
    {
        // Arrange
        int specialtyId = 1;
        var expectedGroups = new List<GroupViewModel>
        {
            new GroupViewModel { Id = 1, Name = "ПД-44 (2019)" },
            new GroupViewModel { Id = 2, Name = "ПД-34 (2020)" },
            new GroupViewModel { Id = 3, Name = "ПД-24 (2021)" },
            new GroupViewModel { Id = 4, Name = "ПД-14 (2022)" },
        };

        // Act
        var result = await _groupService.GetGroupsBySpecialtyIdAsync(specialtyId);

        // Assert
        Assert.NotNull(result);
        Assert.AreEqual(3, result.Data.Count);
        Assert.AreEqual(expectedGroups, result.Data);
    }
```

Рисунок 4.1.1 – Unit test

Звичайно є можливість тестувати усі дії через одним метод передаючи різні параметри, проте інколи це дуже складно, а також порушує цілісність перевірки, через різність параметри. Тому краще створювати окремі тести для кожного модулю окремо (див. рис. 4.1.2).

```
[Test]
public async Task GetGroupsBySpecialtyIdAsync_ReturnsResultWithCorrectGroupViewModelList()
{…
}

[Test]
public async Task GetGroupsBySpecialtyIdAsync_ReturnsResultWithEmptyGroupViewModelListForNone
{…
}

[Test]
public async Task GetGroupsBySpecialtyIdAsync_ReturnsResultWithErrorForInvalidSpecialtyId()
{…
}

[Test]
public async Task GetGroupByIdAsync_ReturnsResultWithCorrectGroupViewModel()
{…
}

[Test]
public async Task GetGroupByIdAsync_ReturnsResultWithEmptyGroupViewModelListForNonexistentGro
{…
}

[Test]
public async Task GetGroupByIdAsync_ReturnsResultWithErrorForInvalidGroupId()
{…
}

[Test]
public async Task CreateGroupAsync_ReturnsResultWithCorrectGroupViewModel()
{…
}

[Test]
public async Task CreateGroupAsync_ReturnsResultWithEmptySpecialtyListForNonexistentGroupId(
{…
}

[Test]
public async Task CreateGroupAsync_ReturnsResultWithErrorForInvalidSpecialtyId()
{…
}
```

Рисунок 4.1.2 – Окремі тести GroupService

4.2. Ручне тестування

Ручне тестування є процесом, який виконується вручну тестувальником, без використання автоматизованих інструментів або скриптів. Його основною метою є виявлення дефектів, помилок та недоліків у програмному продукті, а також перевірка його відповідності вимогам та очікуванням користувачів.

Під час ручного тестування тестувальник вручну виконує тестові сценарії, які охоплюють різні можливі випадки використання програмного продукту. Він активно взаємодіє з інтерфейсом, вводить дані, виконує операції та аналізує результати, з'ясовуючи, чи працює програма належним чином.

Ручне тестування вимагає високого рівня уваги до деталей, точності та систематичності. Тестувальник повинен бути досконалим спостерігачем та аналітиком, щоб виявляти навіть найменші аномалії або відхилення в роботі програми. Важливо також документувати та звітувати про знайдені проблеми, вказуючи на їхні особливості та шляхи відтворення.

Незважаючи на широке застосування автоматизованого тестування, ручне тестування залишається важливою складовою частиною процесу тестування програмного забезпечення. Воно дозволяє виявляти проблеми, які можуть залишитися непоміченими автоматизованими засобами, а також забезпечує більш гнучкий підхід до перевірки функціоналу та взаємодії програми з користувачем.

Для проходження системи на коректність роботи необхідно сформувати тест-кейси які стануть основою стабільної роботи системи.

Тест-кейс 1. Реєстрація студента і приєднання в групу відбувається наступним чином (див. табл. 4.2.1) (див. рис. 4.2.1).

Таблиця 4.2.1 – Тест-кейс, реєстрація студента

Дія	Реєстрація студента в системі з унікальною поштою
Умови	Надано код-приєднання до групи та облікові дані студента

Очікуваний результат	Студент зареєстрований в системі, чекає на дозвіл від викладача.

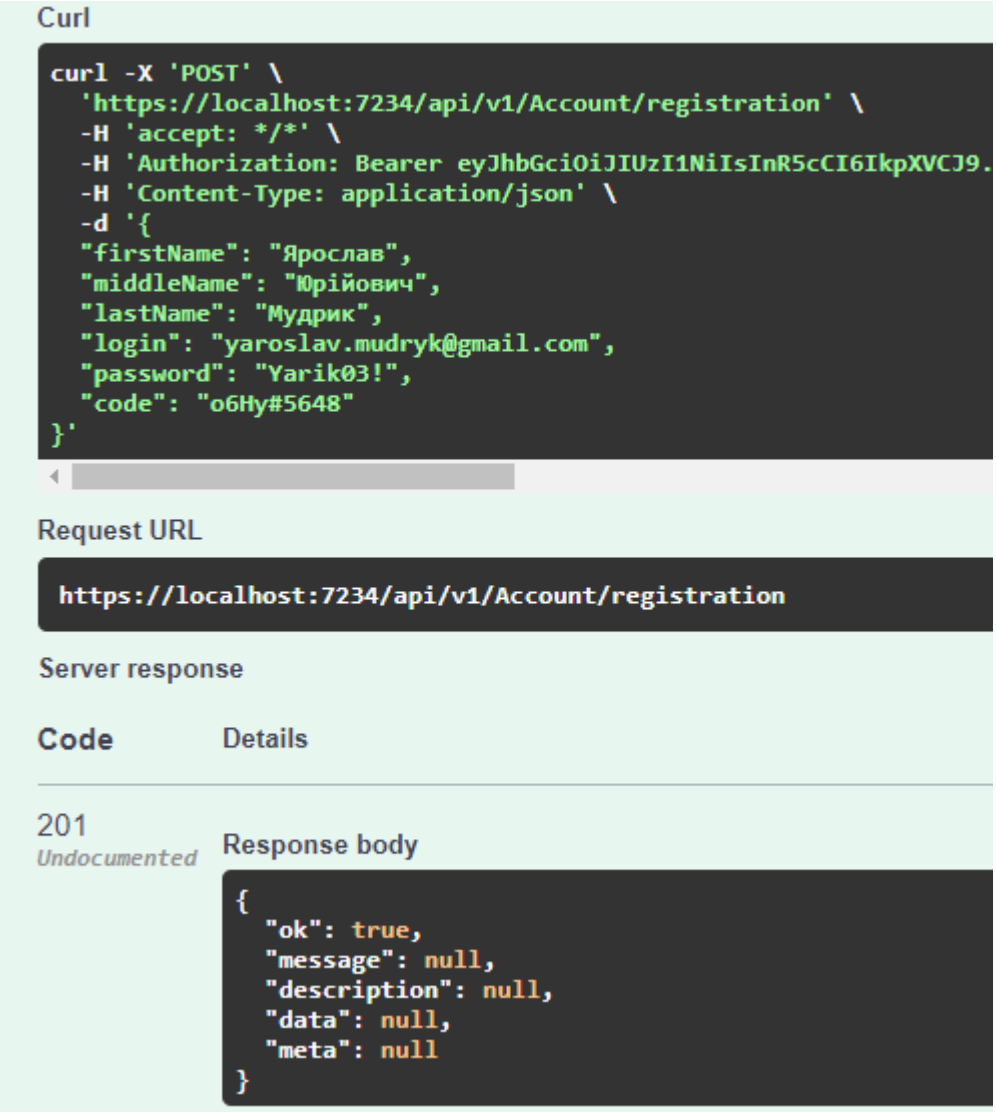


Рисунок 4.2.1 – Форма реєстрації

Тест-кейс 2. Вхід в систему користувачами відбувається за допомогою логіну, паролю та даних застосунку, після їх успішної перевірки надається токен доступу (див. рис. 4.2.2) (див. табл. 4.2.2).

Таблиця 4.2.2 – Тест-кейс, вхід користувача

Дія	Вхід користувача
Умови	Ведення облікових даних користувача з роллю студент

Очікуваний результат	Успішний вхід і перенаправлення на головну сторінку.

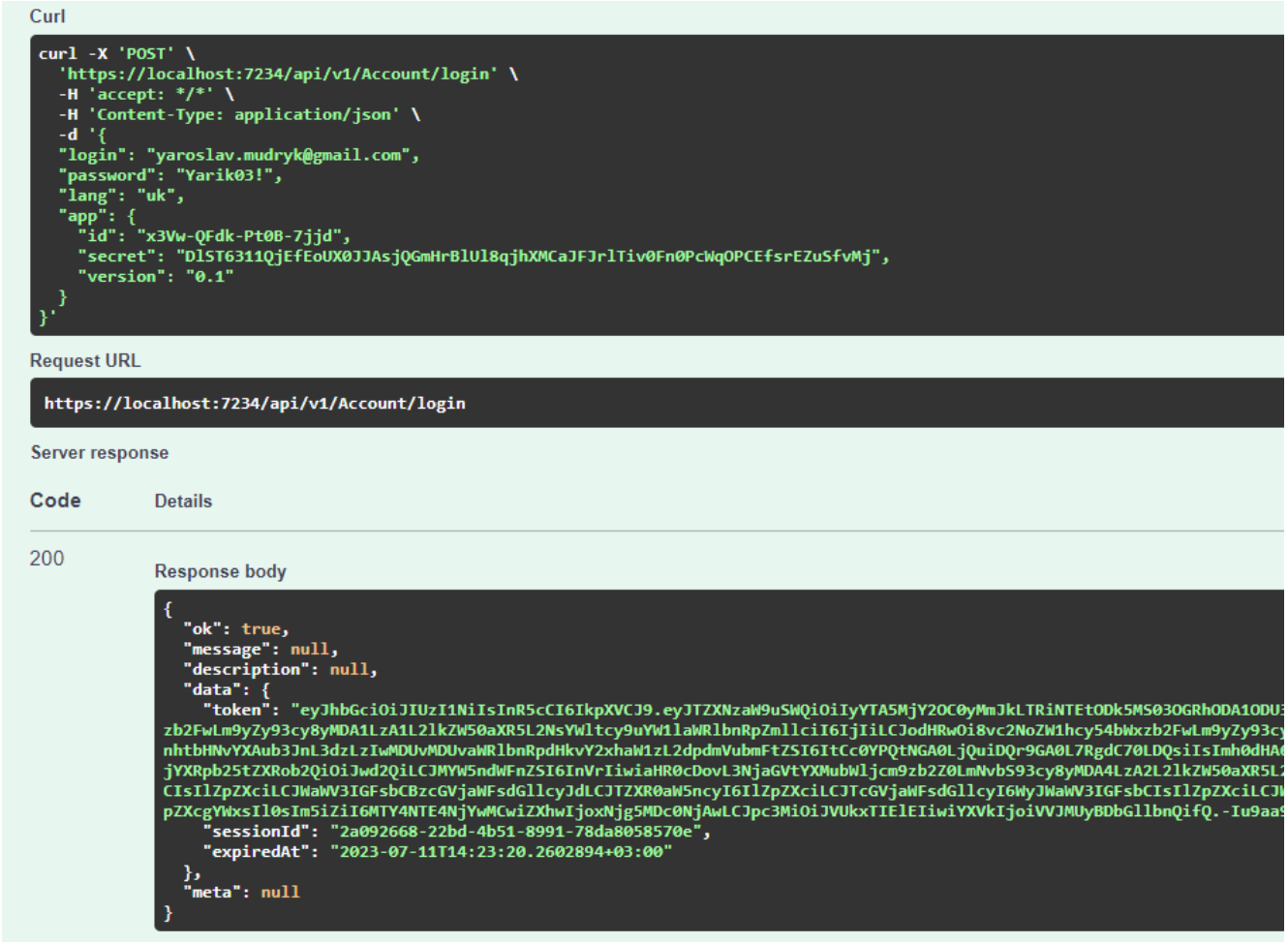


Рисунок 4.2.2 – Токен доступу

Тест-кейс 3. Перегляд структури університету відбувається після запиту на інститути (див. рис. 4.2.3) та їх кафедр (див. рис. 4.2.4) (див. табл. 4.2.3).

Таблиця 4.2.3 Тест-кейс, перегляд структури

Дія	Перегляд структури університету
Умови	Авторизований користувач
Очікуваний результат	Отримання інформації про структуру університету.

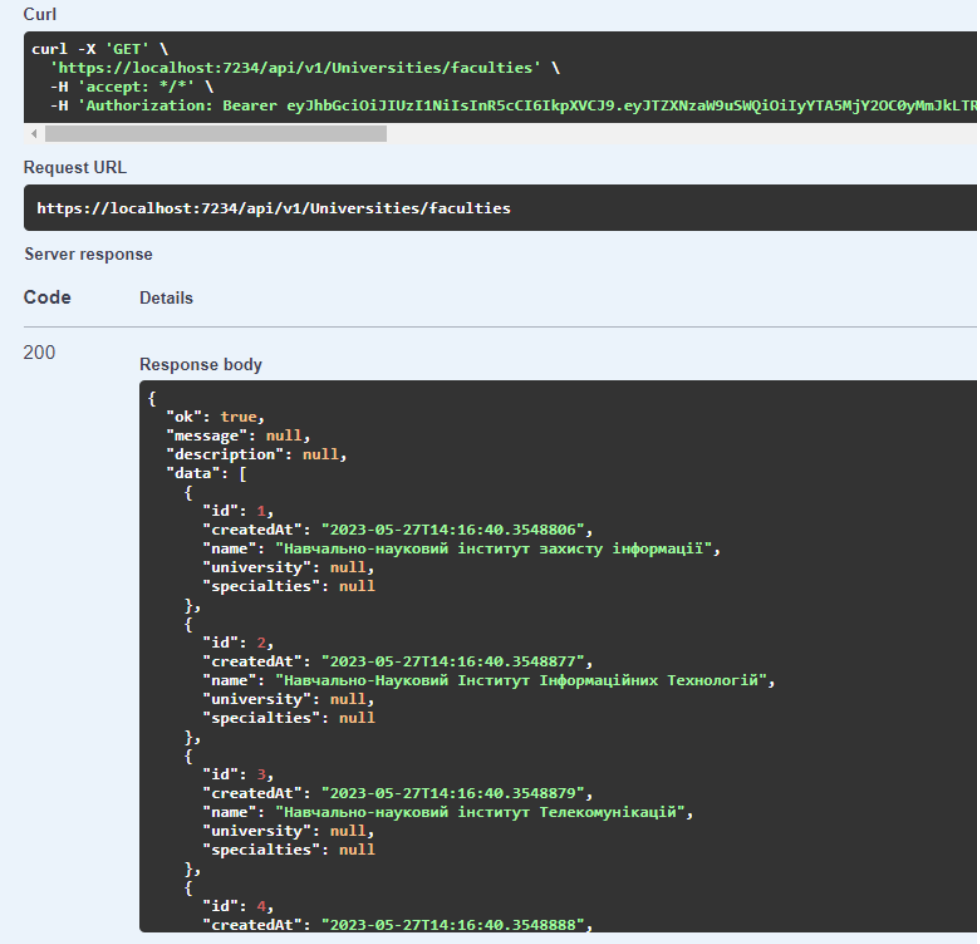


Рисунок 4.2.3 – Інформація про інститути

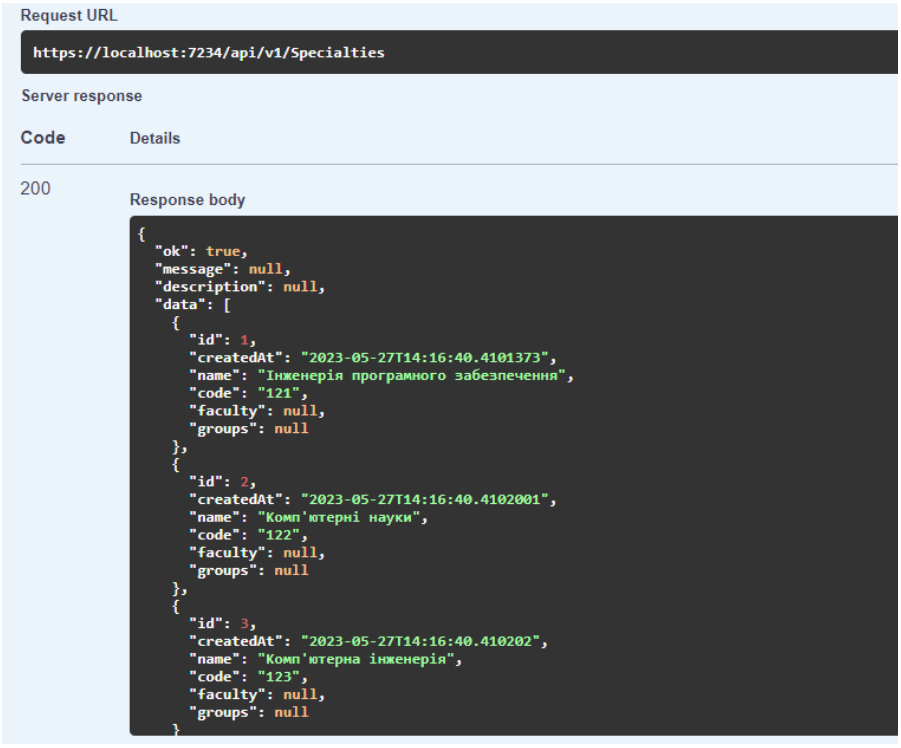


Рисунок 4.2.4 – Інформація про кафедри

Тест-кейс 4. Щоб переглянути сесії необхідно зробити запит на сесії (див. рис. 4.2.5) (див. табл. 4.2.4).

Таблиця 4.2.4 Тест-кейс, перегляд сесій

Дія	Перегляд сесій
Умови	Авторизований користувач
Очікуваний результат	Список активних сесій

Рисунок 4.2.5 – Список сесій

Тест-кейс 5. Щоб переглянути сповіщення необхідно зробити запит до них (див. рис. 4.2.6) (див. табл. 4.2.5).

Таблиця 4.2.5 – Тест-кейс, перегляд сповіщень

Дія	Перегляд сповіщень
Умови	Авторизований користувач
Очікуваний результат	Список сповіщень

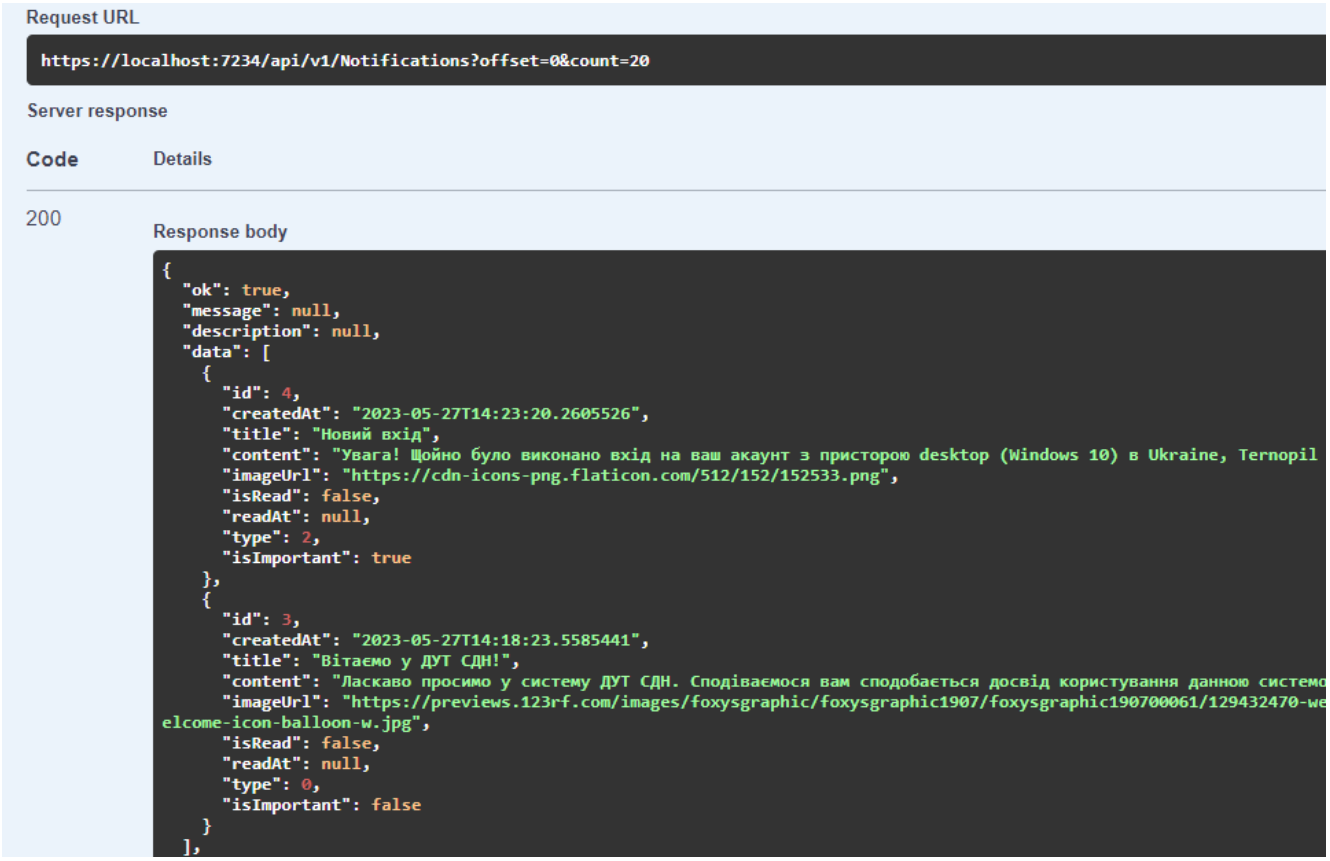


Рисунок 4.2.6 – Список сповіщень

ВИСНОВКИ

Дана робота є підтвердженням розробки веб-сервісу, який забезпечує дистанційне навчання студентам та викладачам.

1. Актуальність розробленої системи була обґрунтована шляхом аналізу продуктів, що є подібними за функціоналом. Досліджено недоліки та проблеми, які попадаються студентам і викладачам під час навчального процесу та з’ясовано, аналоги котрі існують не можуть повністю задовільнити вимоги користувачів які з ними працюють.

2. Було розроблено веб-сервіс з використанням сучасних фреймворків, архітектури, сторонніх сервісів типу Azure, GitHub Pipelines та Firebase Cloud Messaging від Google. Головним інструментом було обрано мову програмування C# у 9 версії та фреймворк для розробки веб-додатків ASP.NET у 7 версії. Для сховища даних було використано SQL Server в парі з PostgreSQL. В зберіганні файлів великих обсягів було виконано інтеграцію з сервісом Google Drive та Azure Blob Storage.

3. Було спроектовано архітектуру та алгоритми обробки даних в системи у вигляді діаграм та схем. Створені схеми та діаграми було надано на перегляд колегам з досвідом та була ними схвалена з маленькими поправками.

4. Також було досліджено такий важливий пункт роботи системи, як обробка великих файлів при завантажені, збережені, відкриті чи редагуванні. За рахунок розділених сховищ для зберігання великих об’ємів даних було розділено їх зберігання, файли, які треба читати зберігаються на Google Drive, файли які треба для того, щоб можна було тільки відкривати зберігаються на Azure Blob Storage.

Результати дослідження бакалаврської роботи апробовані на всеукраїнських науково-практичних конференції: Особливості сучасної платформи . ASP.NET CORE для створення серверних застосунків:

Матеріали iii Всеукраїнської науково-практичної конференції «Сучасні інтелектуальні інформаційні технології в науці та освіті»

ПЕРЕЛІК ПОСИЛАНЬ

1. Що таке Standalone застосунок? [Електронний ресурс] // Geek for geeks. – 2022. – Режим доступу до ресурсу: https://www.geeksforgeeks.org/what-is-standalone-application/.
2. Що таке sql-ін'єкція? [Електронний ресурс] // Acode. – 2023. – Режим доступу до ресурсу: https://acode.com.ua/sql-injection/.
3. Що таке rate limit? [Електронний ресурс] // Axway. – 2022. – Режим доступу до ресурсу: https://blog.axway.com/learning-center/apis/basics/api-rate-limits.
4. Що таке Azure Pipelines? [Електронний ресурс] // CodeFresh. – 2023. – Режим доступу до ресурсу: https://codefresh.io/learn/azure-devops/azure-pipelines-the-basics-and-creating-your-first-pipeline/.
5. Що таке Restful? [Електронний ресурс] // Red Hat. – 2020. – Режим доступу до ресурсу: https://www.redhat.com/en/topics/api/what-is-a-rest-api.
6. Visual Studio 2022 [Електронний ресурс] // Microsoft. – 2023. – Режим доступу до ресурсу: https://visualstudio.microsoft.com/vs/.
7. Що таке MSIL? [Електронний ресурс] // Microsoft. – 2023. – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/dotnet/standard/managed-execution-process.
8. Що таке Azure DevOps? [Електронний ресурс] // QA Group. – 2023. – Режим доступу до ресурсу: https://qagroup.com.ua/publications/what-is-azure-devops/.
9. Що таке Agile? [Електронний ресурс] // Brain Rain. – 2021. – Режим доступу до ресурсу: https://brainrain.com.ua/uk/chto-takoe-agile-ua/.
10. Що таке Redis-server? [Електронний ресурс] // NEX-Software. – 2023. – Режим доступу до ресурсу: https://uk.nex-software.com/sho-take-redis-serverexe.
11. Що таке монолітна архітектура? [Електронний ресурс] // QA Light. – 2023. – Режим доступу до ресурсу: https://qalight.ua/baza-znaniy/shho-take-monolitna-arhitektura/.

12. Що таке gRPC і як він працює [Електронний ресурс] // Highload Today. – 2022. – Режим доступу до ресурсу: https://highload.today/uk/shho-take-grpc-i-yak-vin-pratsyuye/.

13. Що таке SOAP? [Електронний ресурс] // Education Wiki. – 2023. – Режим доступу до ресурсу: https://uk.education-wiki.com/9679948-what-is-soap.

14. Kestrel web server [Електронний ресурс] // Microsoft. – 2023. – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/servers/kestrel?view=aspnetcore-7.0.

15. Що таке Docker і навіщо він? [Електронний ресурс] // QA Group. – 2023. – Режим доступу до ресурсу: https://qagroup.com.ua/publications/shcho-take-docker-i-navishcho-vin/.

16. ACID транзакції [Електронний ресурс] // Data Bricks. – 2023. – Режим доступу до ресурсу: https://www.databricks.com/glossary/acid-transactions.

17. Як використовувати JSON Web Tokens? [Електронний ресурс] // Code Guida. – 2022. – Режим доступу до ресурсу: https://codeguida.com/post/1567.

18. Що таке Firebase Cloud Messaging? [Електронний ресурс] // Microsoft. – 2023. – Режим доступу до ресурсу: https://learn.microsoft.com/ru-ru/xamarin/android/data-cloud/google-messaging/firebase-cloud-messaging.

19. MailKit інтеграція [Електронний ресурс] // Abp. – 2021. – Режим доступу до ресурсу: https://docs.abp.io/en/abp/latest/MailKit.

20. Що таке юніт-тестування? [Електронний ресурс] // Smart Bear. – 2023. – Режим доступу до ресурсу: https://smartbear.com/learn/automated-testing/what-is-unit-testing/.

ДОДАТОК А

Код від даної роботи можна знайти на платформі GitHub за посиланням - https://github.com/YaroslavMudryk/URLS

ДОДАТОК Б

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка серверної додатку для системи дистанційного навчання мовою C#

Роботу виконав студент 4 курсу
Групи ПД-44
Мудрик Ярослав Юрійович
Керівник роботи
Зав.кафедри, к.ф.-м.н, доцент
Поперешняк Світлана Володимирівна

Київ – 2023

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета** – покращення процесу взаємодії з клієнтом, за рахунок реалізації API на C#.
- **Об'єкт** – процес дистанційного навчання за допомогою серверного додатку.
- **Предмет** – серверний додаток для забезпечення доступності функцій дистанційного навчання.

2

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Дослідити та проаналізувати існуючі системи дистанційного навчання.
2. Розробити прикладний інтерфейс взаємодії для клієнтів.
3. Розробити систему аутентифікації та авторизації.
4. Розробити модуль підтримки інформації про структуру університету.
5. Розробити модуль підтримки навчального процесу.
6. Розробити систему сповіщень.
7. Протестувати систему.

3

АНАЛІЗ АНАЛОГІВ

Показник	Google Classroom	Blackboard	Moodle	Ukrainian Remote Learning System
Інтеграція в інші продукти	+	-	-	+
Запрошення до курсу	+	+	+	+
Налаштування сповіщень	-	-	-	+
Можливість експорту оцінок	-	-	+	+
Зберігання великих файлів	+	-	-	+
Відкритий код	-	-	+	+
Панель керування інформацією про студента	+	+	+	+
Панель адміністратора	-	+	+	+
Статистика досягнення студента	+	-	+	+
Відкритий API	-	-	+	+

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональна :

1. Забезпечити можливість аутентифікації та авторизації.
2. Забезпечити експортування оцінок.
3. Забезпечити можливість підтримки інформації про структуру університету.
4. Забезпечити підтримку навчального процесу.
5. Забезпечити керування доступом до акаунту.
6. Забезпечити початкову ініціалізацію даними.
7. Забезпечити надійність завантаження великих файлів .

Нефункціональні :

1. Стабільне функціонування на різних ОС (Windows, Linux, MacOS).
2. Забезпечити високу продуктивність роботи.
3. Забезпечити масштабованість при збільшенні користувачів.
4. Забезпечити надійність та безпеку даних.

ПРОГРАМНІ ТА ТЕХНІЧНІ ЗАСОБИ РЕАЛІЗАЦІЇ

Identity Server 4

Dapper

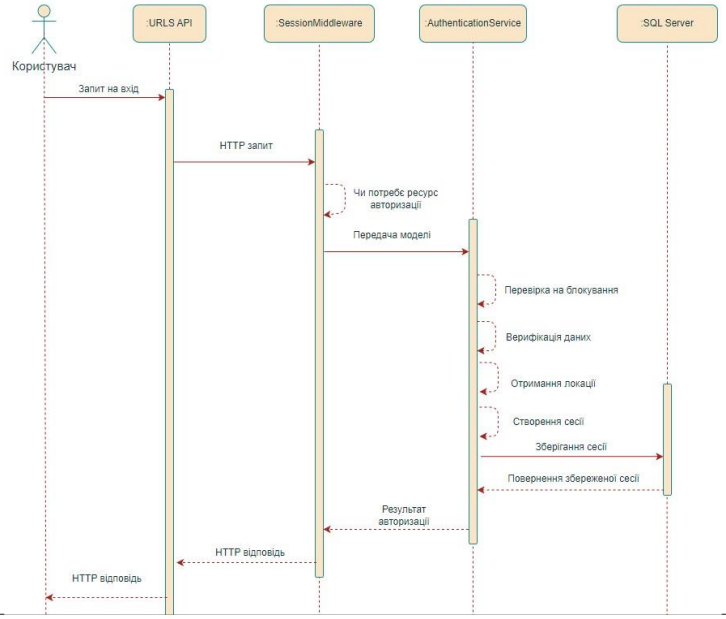
Entity Framework Core

ASP.NET Core

Visual Studio 2022

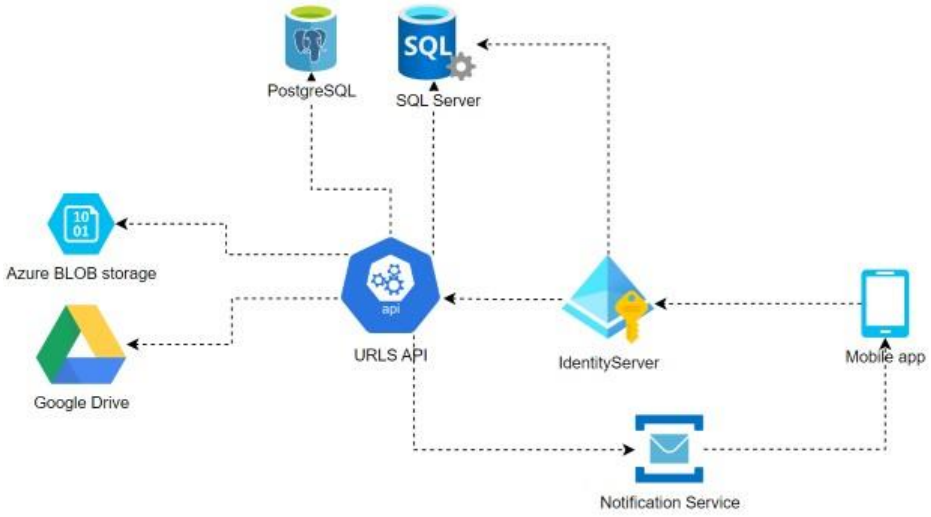
6

ДІАГРАМА ПОСЛІДОВНОСТІ ПРОЦЕСУ «Вхід»



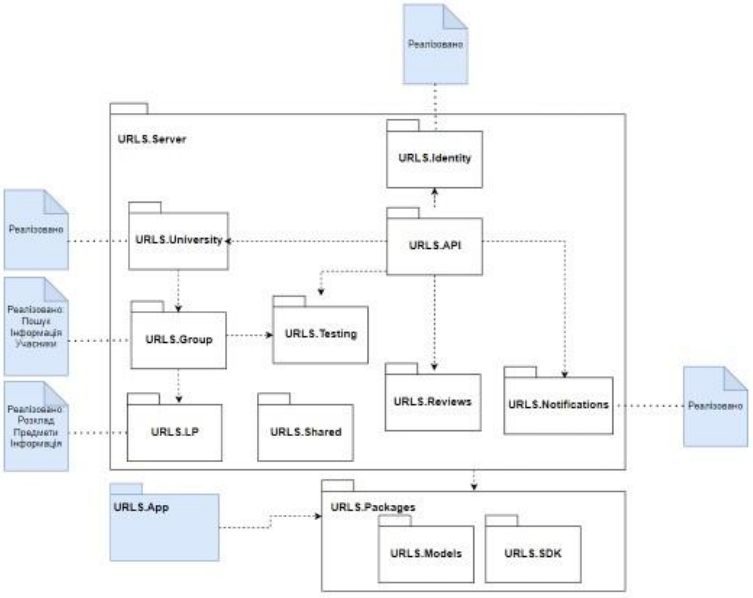
7

АРХІТЕКТУРА СИСТЕМИ



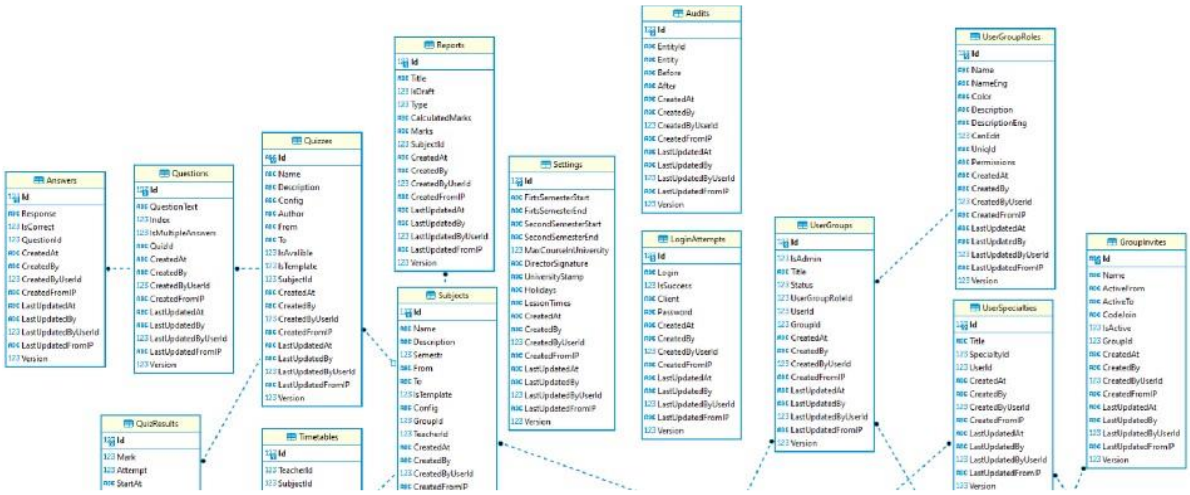
8

ДІАГРАМА ПАКЕТІВ



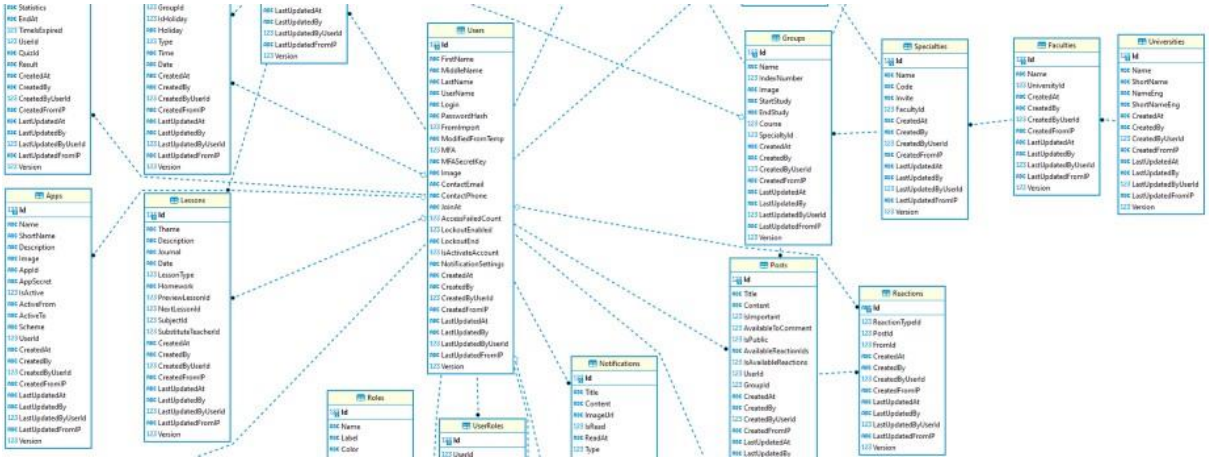
9

СХЕМА БАЗИ ДАНИХ



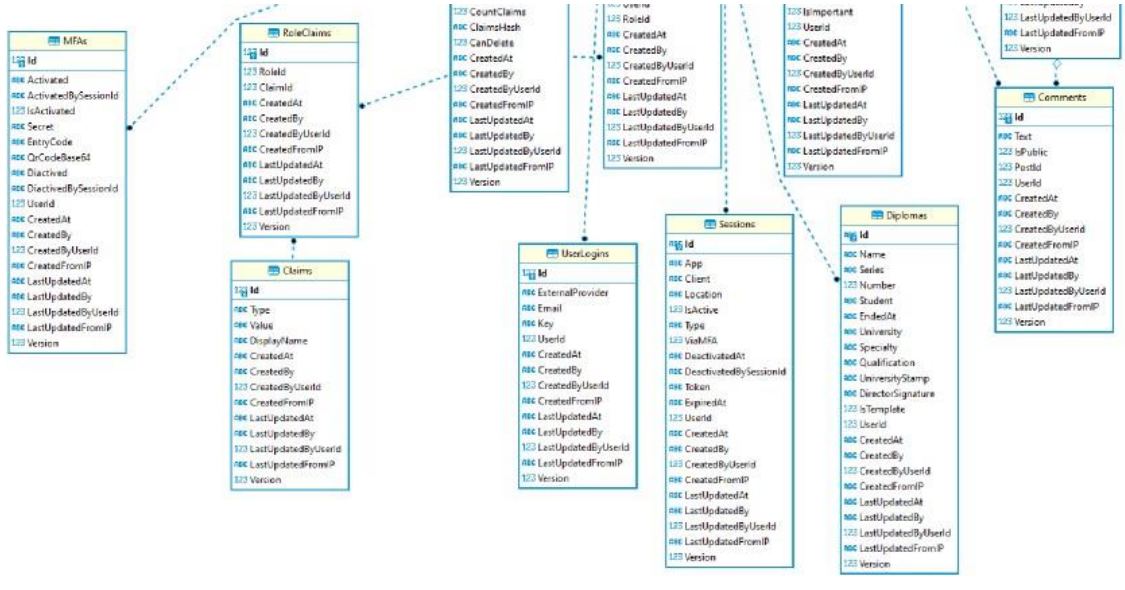
10

СХЕМА БАЗИ ДАНИХ



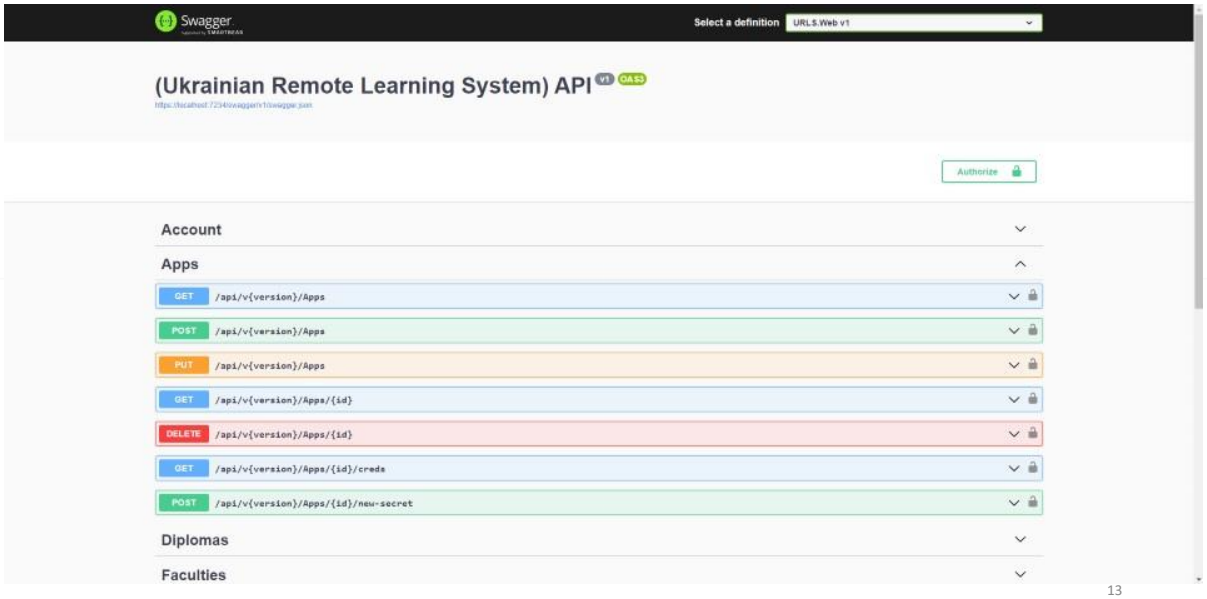
11

СХЕМА БАЗИ ДАНИХ



12

SWAGGER

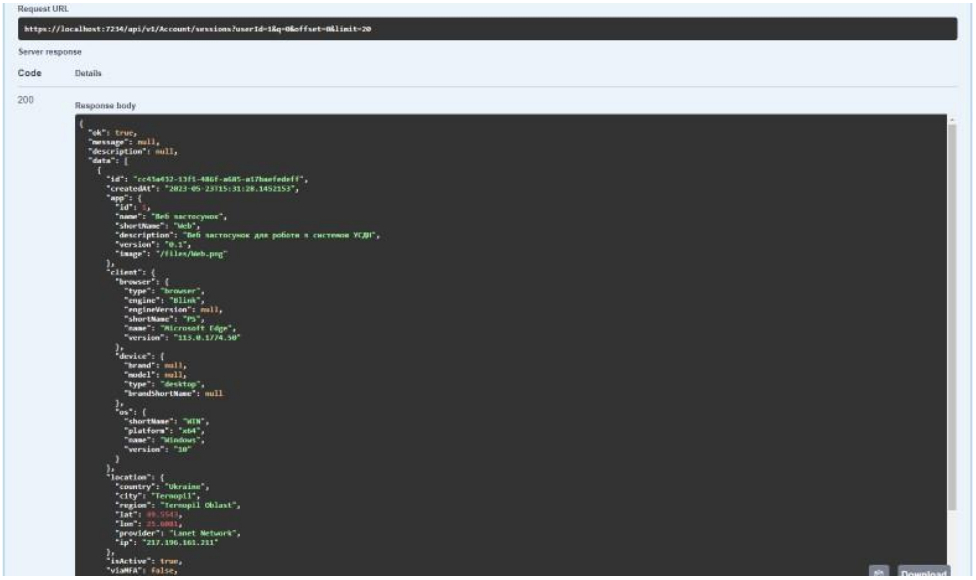


13

СТВОРЕННЯ ГРУПИ

14

ОТРИМАННЯ СЕСІЙ КОРИСТУВАЧА



15

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Мудрик Я.Ю. Актуальність систем дистанційного навчання в сучасному світі / Поперешняк С.В, Мудрик Я.Ю // Застосування програмного забезпечення в ІКТ: Матеріали всеукраїнської науково-технічної конференції. Збірник тез. 20 квітня 2023р., ДУТ, м. Київ – К: ДУТ, 2023. – С. 135.

2. Мудрик Я.Ю. Особливості сучасної платформи. ASP.NET CORE для створення серверних застосунків / Поперешняк С.В, Мудрик Я.Ю // Сучасні інтелектуальні інформаційні технології в науці та освіті: Матеріали всеукраїнської науково-практичної конференції. Збірник тез. 16 травня 2023р., ДУТ, м. Київ – К: ДУТ, 2023. – Подано до друку.

17

ВИСНОВКИ

1. Досліджено та проаналізовано існуючи аналоги систем дистанційного навчання.
2. Спроектовані та розроблені бази даних.
3. Розроблено прикладний інтерфейс взаємодії для клієнтів (API).
4. Розроблено механізм ініціалізації системи.
5. Розроблено систему авторизації.
6. Розроблено модуль підтримки інформації про структуру університету.
7. Розроблено модуль для підтримки навчального процесу.
8. Розроблено супутні бібліотеки для спрощення роботи клієнта.
9. Розроблено систему сповіщень.
10. Проведено тестування системи, в результаті якого виявлено помилки, які були усунуті

18

ДЯКУЮ ЗА УВАГУ!

19