МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ІВАНА ФРАНКА

Факультет прикладної математики та інформатики
(повне найменування назви факультету)

Кафедра інформаційних систем
(повне назва кафедри)

Магістерська робота

Розробка системи автоматизації процесу складання розкладу. Серверна частина.

Студента 6 курсу, групи ПМІМ-22

Спеціальності 122 «Комп'ютерні науки»

Стецьків О.Р.
(підпис) (прізвище та ініціали)

Керівник Горлач В.М.
(підпис) (прізвище та ініціали)

Рецензент Мельничук А. В.
(підпис) (прізвище та ініціали)

Львів-2022

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет *прикладної математики та інформатики*

Кафедра *інформаційних систем*

Освітньо-кваліфікаційний рівень *магістр*

Спеціальність *122 – комп'ютерні науки*

«ЗАТВЕРДЖУЮ»

Зав. кафедрою *проф. Шинкаренко Г.А.*

« *16* » листопада 2022 р.

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Стецьків Остап Романович

(прізвище, ім'я, по батькові)

1. Тема роботи
Розробка системи автоматизації процесу складання розкладу. Серверна частина.

керівник роботи *доц. Горлач В.М.*
затверджена Вченою радою факультету від « *16* » *листопада* 20 *22* р., № *15*

2. Строк подання студентом роботи *12.12.2022 р.*

3. Вихідні дані до роботи
Література та інтернет-ресурси за тематикою роботи

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Огляд сучасного стану проблеми
2. Обрані методи та технології
3. Програмна реалізація
4. Результати апробації
5. Висновки та підсумки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Схеми, діаграми, скріншоти
Презентація дипломної роботи

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання *05.09.22 р.*

КАЛЕНДАРНИЙ ПЛАН

№	Найменування етапів дипломної (кваліфікаційної) роботи	Строк виконання етапів роботи	Примітка
1.	*Огляд існуючих систем та технологій*	*Вересень*	
2.	*Постановка задачі*	*Вересень*	
3.	*Розробка схем та алгоритмів*	*Жовтень*	
4.	*Програмна реалізація*	*Листопад*	
5.	*Апробація*	*Листопад*	
6.	*Оформлення роботи*	*Грудень*	

Студент _____ *Стецьків О.Р.*
підпис

Керівник роботи _____ *доц Горлач В.М.*
підпис

Львів-2022

Зміст

Вступ

Кожен з нас у своєму житті користувався розкладом, будь-то розклад пар в університеті, чи розклад уроків в школі чи навіть заплановані справи. Ми усвідомлюємо наскільки важко складати такі розклади, особливо коли до них залучена велика кількість людей.

Якість надання освітніх послуг навчальним закладом, перш за все, залежить від чіткого планування та організації навчально-виховного процесу в освітньому закладі. Складання розкладу занять є одним із найважливіших і діючих видів планування навчальної роботи освітнього закладу. Ця робота дуже відповідальна і кропітка, адже необхідно брати до уваги і програму студентів, і кількість студентів під час лекцій та практичних занять, і побажання колег.

Складання розкладу в українських університетах не автоматизоване; робить це певна людина вручну. Тому в даній роботи ми будемо розглядати застосунок, який значноспростить складання розкладу та буде менш затратний по часу.

Мета і задачі роботи

Метою магістерської роботи є розробка застосунку побудови розкладу, що дозволяє автоматично формувати освітні програми ВНЗ з урахуванням вимог викладачів та наукової спільноти щодо рівноправних параметрів.

Предметом дослідження є процес створення навчальних програм для ВНЗ. В результаті роботи розроблено веб-застосунок.

Практичне значення отриманих результатів. Реальна робота розробленої системи підтверджує, що веб-застосунок може гарантувати реалізацію функції побудови розкладів.

Аналіз сучасного стану предметної області

На даний момент університет користується паперовим варіантом розкладу, та розклад складається вручну. З цього випливає що студенти чи викладачі мають іти в університет та запам’ятовувати або ж фотографувати розклад своїх пар, проте фотографії можуть легко загубитись, а запам’ятований розклад забутись, тому необхідно перенести їх у онлайн формат.

Перше що спадає на думку - поширення фотографій розкладу в месенджерах, де всі матимуть доступ завантажити їх та відкрити на власному гаджеті, проте ці фотографії теж можуть легко загубитись, також є можливість закріпити фотографії розкладу в групах у вайбері чи телеграмі, але і цей варіант нам не підходить, тому що при появі нової, актуальнішої інформації, розклад буде прибраний з закріплених повідомлень та знову загубиться. Тому нам необхідно розробити програму. Розглянемо уже готові програми якими користується університет:

- Teams
- Google Calendar
- Zoom

Перший варіант – Teams є дуже зручним для проведення конференцій та має власний календар, проте він не має змоги бронювати аудиторії, також Teams не покаже вже готовий розклад для групи, оскільки кожен викладач змушений сам створювати пару та додавати до неї студентів, на жаль, може трапитись, що не всіх студентів доєднають до пари відповідно до розкладу на паперовому варіанті, який можливо, не всі і мають.

Дуже схожа ситуація з Google Calendar, проте він має можливість бронювати аудиторії, але в нього постає інша проблема – проблема доступу. Ця проблема полягає в тому, що необхідно кожному викладачеві додати доступ до свого календаря, що може зашкодити приватності та і не всі мають змогу це

зробити.

Останній варіант – Zoom, цей варіант геть не підходить для складання розкладу, оскільки не може забронювати аудиторію, бачити розклад декількох груп.

Також варто розглянути варіант у системі https://dekanat.lnu.edu.ua/, на жаль, це розклад не є робочим, тому було вирішено створити веб-застосунок, який вирішуватиме наступні завдання.

Завдання, які вирішує веб-застосунок

Основна мета цього додатку очевидна з назви: спростити процес складання розкладу. Відповідно до цього, сам процес буде менш затратний по часу та по людських можливостях. В залежності від приєднаної бази даних, можна буде створювати розклад для будь-якого навчального закладу.

При створенні та редагуванні пари система коригує дані, які ми можемо змінювати, наприклад назва дисципліни залежить від обраного викладача, курсу та спеціалізації. Список аудиторій змінюється в залежності від типу заняття : лекція це чи практичне заняття.

Також система буде містити в собі підказки для користувача, який складатиме розклад, наприклад побажання лектора чи вибір дисципліни в залежності від раніше обраного викладача, групи та спеціальності. Також буде можливість завантажити розклад для певного курсу.

Крім додавання пари, користувач з легкістю може редагувати певну пару, натиснувши на неї. Вибір опцій при редагуванні також буде обмежений, як і при додаванні пари.

Обрані технології

Веб-додаток написано на мові .NET 6. NET — це безкоштовна керована комп’ютерна платформа з відкритим вихідним кодом для операційних систем Windows, Linux і macOS. З . NET, ви можете створювати веб-додатки, мобільні додатків, десктопні додатки, ігри та Інтернет речей.

Microsoft SQL Server — це система керування реляційною базою даних, розроблена компанією Microsoft. Як сервер баз даних, це програмний продукт, основною функцією якого є зберігання та отримання даних за запитом інших програмних додатків, які можуть працювати як на тому ж комп’ютері, так і на іншому комп’ютері в мережі.

Azure Active Directory (Azure AD) — це хмарна служба керування ідентифікацією та доступом. Ця служба допомагає вашим співробітникам отримувати доступ до зовнішніх ресурсів, таких як Microsoft 365, портал Azure та тисячі інших програм SaaS.

Архітектура проекту

Проект складається з таких підпроектів:

- **ScheduleSystem**
- **ScheduleSystem.Data**
- **ScheduleSystem.Domain**
- **ScheduleSystem.Tests**
- **ScheduleSystem.Data.Tests**
- **ScheduleSystem.Domain.Tests**

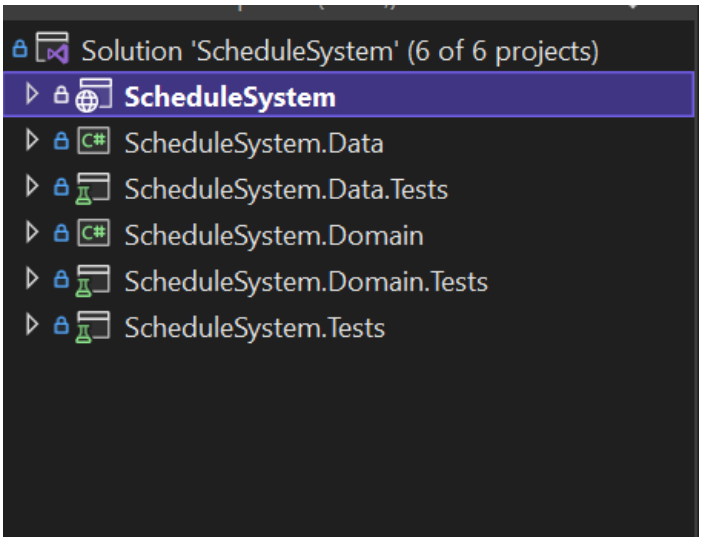

Рис.1 Архітектура проекту

ScheduleSystem.Data - містить в собі базу даних, міграції бази даних, сутності бази даних, та репозиторії до більшості сутностей.

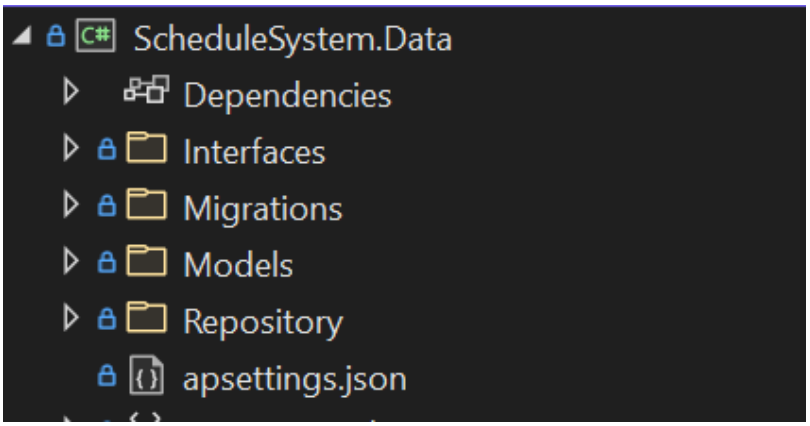

Рис.2 Архітектура проекту ScheduleSystem.Data

ScheduleSystem.Domain - відповідає за бізнес-логіку проекту, та активно комунікує з базою даних та контролерами, де обробляє запити з контролерів, та за необхідності робить зміни у базі даних.

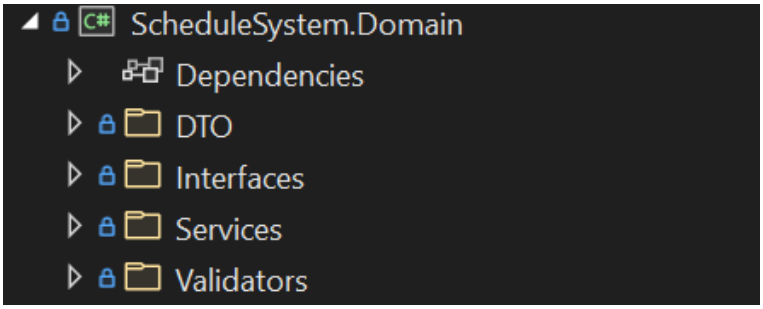

Рис.3 Архітектура проекту

ScheduleSystem - містить в собі налаштування проекту у класі Startup.cs, контролери та папку ClientApp - де зберігається фронт-енд функціонал.

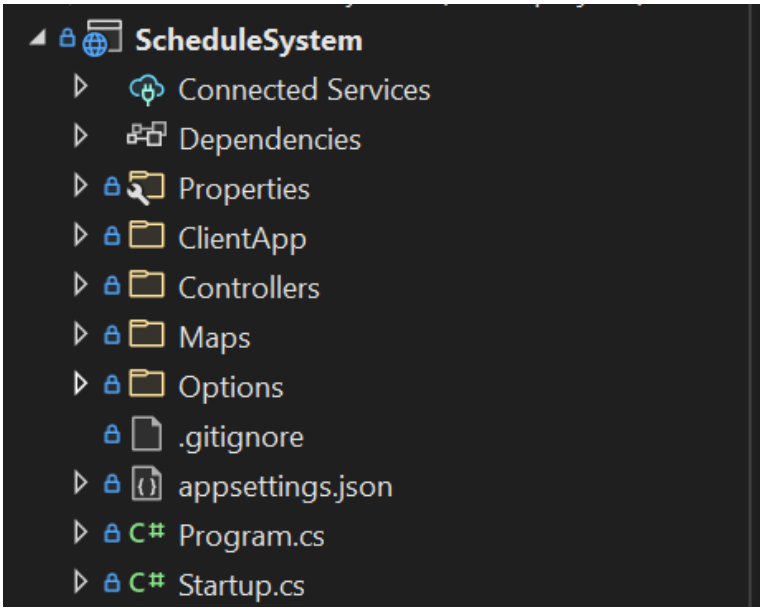

Рис.4 Архітектура проекту ScheduleSystem

Як можемо бачити проект має трирівневу архітектуру.

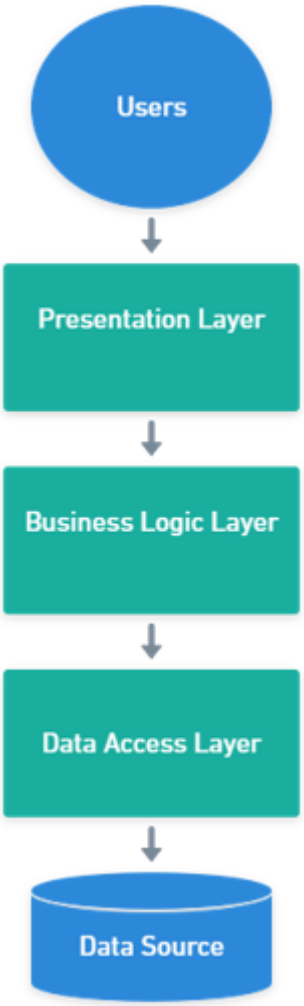

Рис. 5 UML діаграма проекту

Трирівнева архітектура — це усталена архітектура програмного додатка, яка організовує програми на три логічні та фізичні обчислювальні рівні:
рівень презентації або інтерфейс користувача – ScheduleSystem,
рівень програми, на якому обробляються дані – ScheduleSystem.Domain
і рівень даних, де зберігаються й керуються дані – ScheduleSystem.Data.
Додаток складається з шарів, які є абстрактними горизонтальними частинами. Ці рівні представляють різні рівні та типи абстракції програмного забезпечення. Рівні допомагають розділити програму на більш керовані одиниці та підтримують кілька реалізацій. Зазвичай, коли користувач робить веб-запит, потік даних пропускає всі ці рівні. Я вважаю, що кожен шар повинен мати окремий набір моделей, які не можна отримати з іншого шару. Крім того, кожен шар повинен мати можливість досягати лише шару під ним або на тому ж рівні.

Рівень ScheduleSystem.Data

Розглянемо детальніше рівень **ScheduleSystem.Data**.

Почнемо з бази даних, де зберігаються наступні сутності **Rooms**, **SharedRooms**, **FacultySharedRooms**, **Faculties**, **Groups**, **Subjects**, **Pairs**.

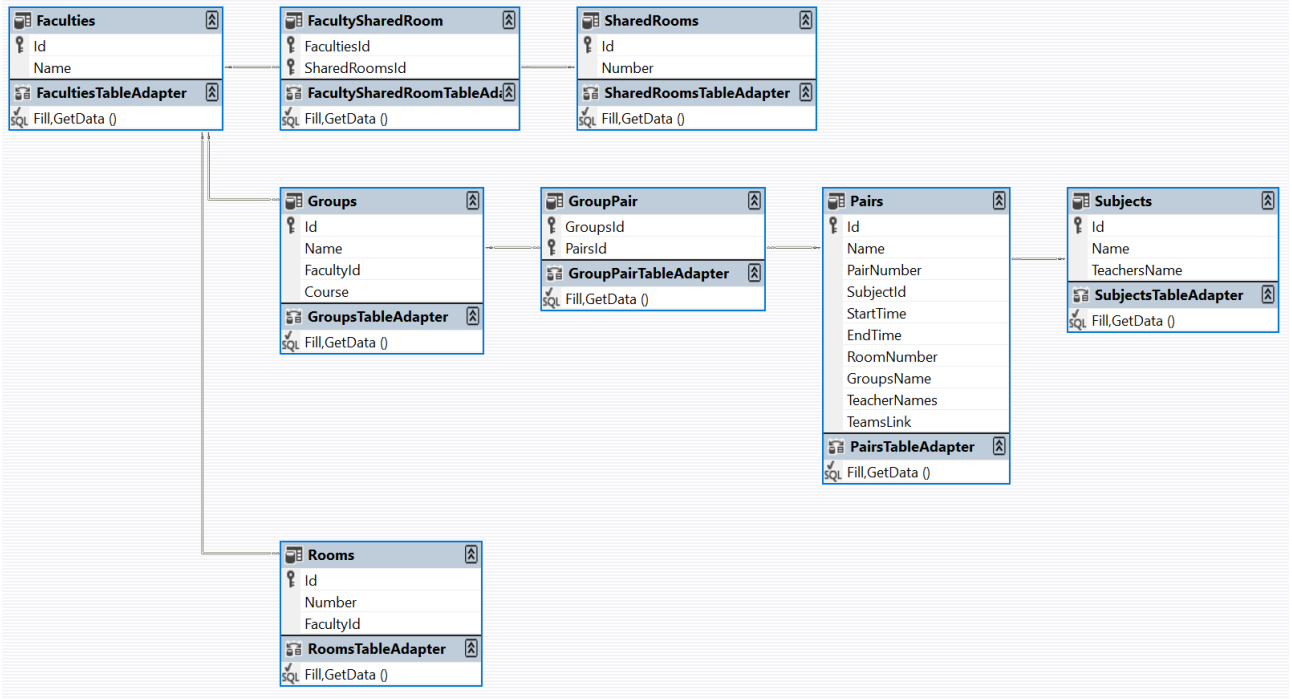

Рис.6 Діаграма Баз Даних проекту.

На цій діаграмі зображені наступні моделі:

Модель **Rooms** має такі поля:

1. Id - унікальний ідентифікатор
2. Number - номер аудиторії
3. FacultyId - зовнішній ключ(посилання на таблицю Faculties)

Модель **SharedRooms**:

1. Id - унікальний ідентифікатор
2. Number - номер аудиторії

Модель **Faculties**:

1. Id - унікальний ідентифікатор
2. Name - назва факультету

Модель **FacultySharedRooms**:

1. FacultiesId - посилання на унікальний ідентифікатор моделі Faculty
2. SharedRoomsId - посилання на унікальний ідентифікатор моделі SharedRooms

Модель **Subjects**:

1. Id - унікальний ідентифікатор
2. Name - назва предмету
3. TeacherName - ім’я викладача

Модель **Maps**:

1. Id – унікальний ідентифікатор
2. Name – Назва мапи
3. Path – шлях зберігання
4. FacultyId – посилання на факультет

Модель **Groups**:

1. Id - унікальний ідентифікатор
2. Name - назва групи
3. FacultyId - зовнішній ключ(посилання на таблицю Faculties)

Модель **Pairs**:

1. Id - унікальний ідентифікатор
2. Name - назва пари
3. TeachersNames – імена викладачів
4. PairNumber – номер пари
5. SubjectId - зовнішній ключ(посилання на таблицю Subjects)
6. StartTime - початок пари
7. EndTime - кінець пари
8. RoomNumber - номер аудиторії
9. GroupId - зовнішній ключ(посилання на таблицю Groups)

А також відповідні зв’язки:

- Один до багатьох між Faculties і Rooms
- Один до багатьох між Faculties і Groups
- Один до багатьох між Groups і Pairs
- Один до багатьох між Subjects і Pairs
- Багато до багатьох між Faculties і SharedRooms через проміжну таблицю FacultySharedRooms.
- Один до багатьох між Faculties і Maps

Відношення «один до багатьох» є типом потужності, який відноситься до відносин між двома сутностями A і B, у яких елемент A може бути пов'язаний з багатьма елементами B, але член B пов’язаний лише з одним елементом A.

Відношення «багато до багатьох» є типом потужності, який відноситься до відносин між двома сутностями, скажімо, A і B, де A може містити батьківський екземпляр, для якого є багато дочірніх у B і навпаки.

Патерн Репозиторій

Патерн - це типовий спосіб вирішення певної проблеми, що часто зустрічається при проектуванні архітектури програм. На відміну від готових функцій чи бібліотек, патерн не можна просто взяти й скопіювати в програму. Патерн являє собою не якийсь конкретний код, а загальний принцип вирішення певної проблеми, який майже завжди треба підлаштовувати для потреб тієї чи іншої програми. Патерни часто плутають з алгоритмами, адже обидва поняття описують типові рішення відомих проблем. Але якщо алгоритм — це чіткий набір дій, то патерн — це високорівневий опис рішення, реалізація якого може відрізнятися у двох різних програмах. Якщо провести аналогії, то алгоритм — це кулінарний рецепт з чіткими кроками, а патерн — інженерне креслення, на якому намальовано рішення без конкретних кроків його отримання.

Патерн репозиторію набув досить великої популярності з моменту його вперше представлення як частини доменно-орієнтованого дизайну в 2004 році. По суті, він забезпечує абстракцію даних, щоб ваша програма могла працювати з простою абстракцією, яка має інтерфейс, наближений до що з колекції. Додавання, видалення, оновлення та вибір елементів із цієї колекції здійснюється за допомогою ряду простих методів, без необхідності мати справу з проблемами бази даних, такими як з'єднання, команди, курсори чи читачі. Використання цього шаблону може допомогти досягти слабкого зв'язку та може ігнорувати збереження об'єктів домену.

В папці Interfaces зберігаються інтерфейси репозиторіїв.

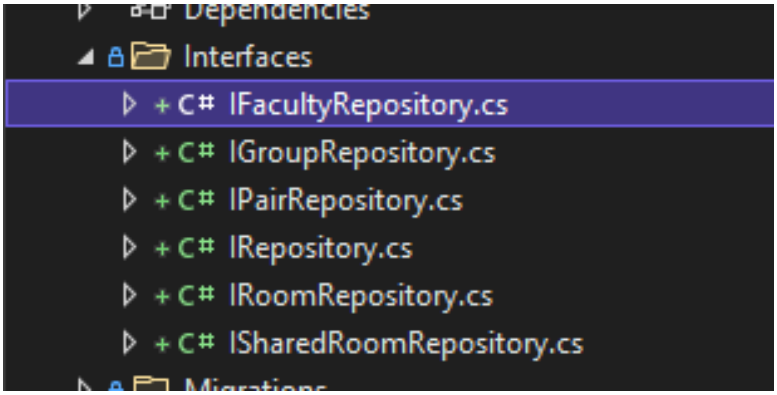

Рис.7.

В папці Repository зберігається реалізація репозиторіїв.

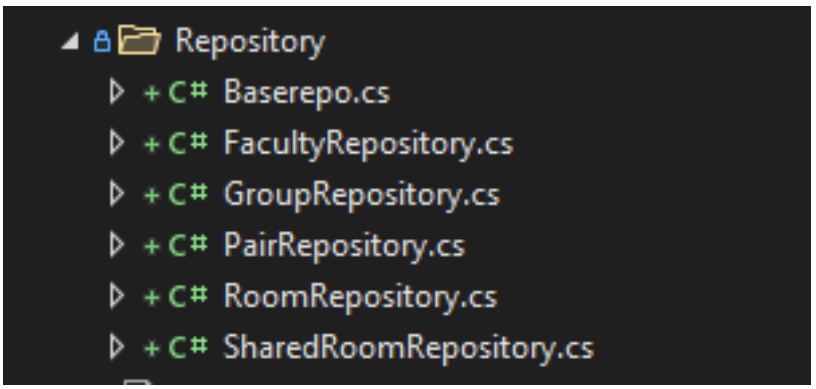

Рис.8.

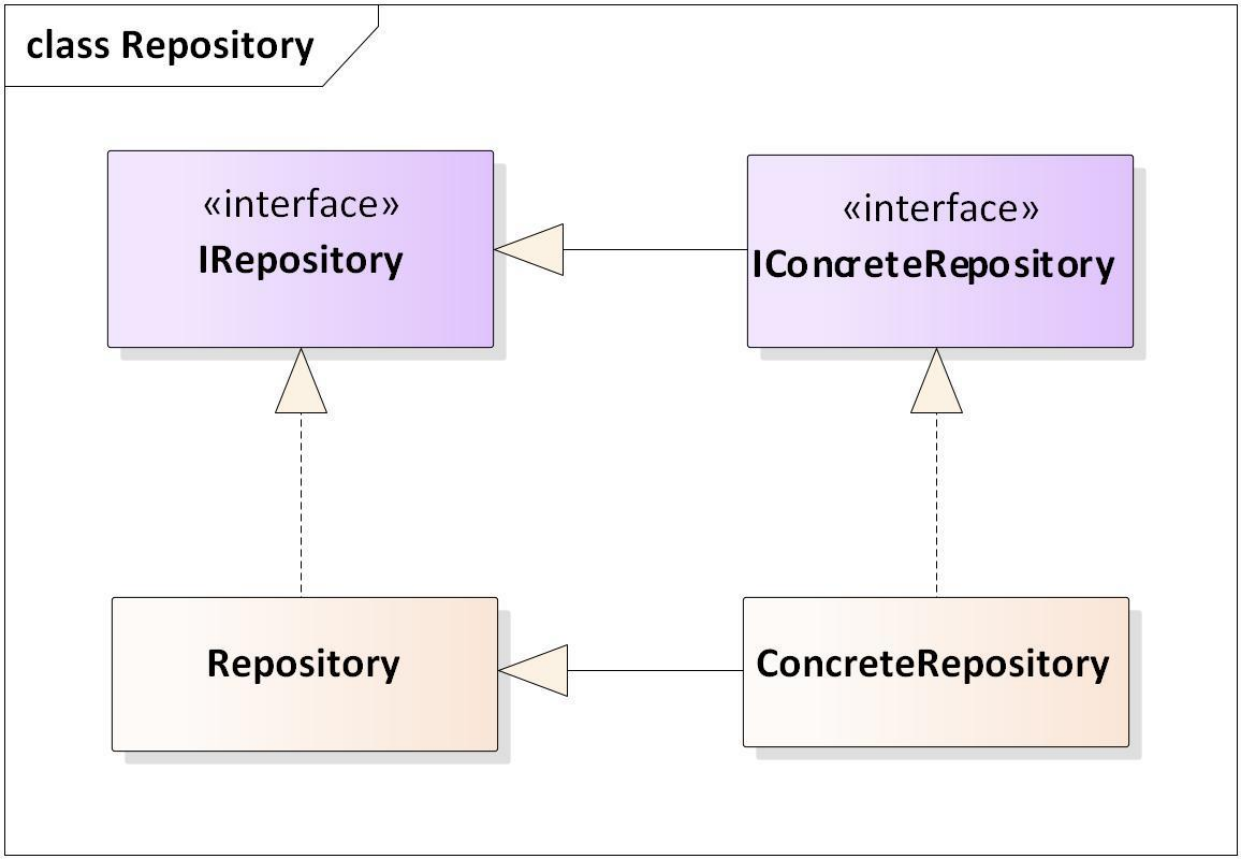

Рис.9. Діаграма патерну Repository

Переваги патерну Repository:

- Тестування контролерів стає легким, оскільки тестова структура не повинна запускатися з фактичним кодом доступу до бази даних.
- Патерн Repository відокремлює фактичну базу даних, запити та іншу логіку доступу до даних від решти програми.
- Бізнес-логіка може отримати доступ до об’єкта даних, не знаючи базової архітектури доступу до даних.
- Бізнес-логіка не знає, чи використовує програма LINQ to SQL чи ADO.NET. У майбутньому основні джерела даних або архітектуру можна буде змінити, не впливаючи на бізнес-логіку.
- Стратегію кешування для джерела даних можна централізувати.
- Централізація логіки доступу до даних, що полегшує обслуговування коду

Недоліки патерну Repository:

- Основним недоліком буде продуктивність. Чим абстрактнішою є бібліотека репозиторію, тим більше потрібно реалізувати рівнів абстракції, охоплюючи більше випадків. Однак спеціалізований Repository працюватиме краще.
- Іншим недоліком буде розробка та підтримка цього патерну. Для маленьких проектів ця вада майже не відчутна, проте для великих проектів буде дуже важко розробляти та підтримувати цей патерн.

Реалізація патерну Repository:

```
public interface IRepository<TEntity> where TEntity : class
{

    IQueryable<TEntity> GetAll();

    IQueryable<TEntity> GetAll(params Expression<Func<TEntity, object>>[] includes);

    Task<TEntity> GetById<TId>(TId id);

    Task<TEntity> Add(TEntity entity);
    4 references | Ostap Stetskiv, 9 days ago | 1 author, 1 change
    Task<TEntity> Update(TEntity entity);
    4 references | Ostap Stetskiv, 9 days ago | 1 author, 1 change
    Task<TEntity> Delete<TId>(TId id);
}
```

Рис. 10. Інтерфейс IRepository.

```
11 references | Ostap Stetskiv, 9 days ago | 1 author, 1 change
public class BaseRepository<TEntity, TContext> : IRepository<TEntity>
      where TEntity : class
      where TContext : DbContext
{
    protected readonly TContext _context;

    5 references | Ostap Stetskiv, 9 days ago | 1 author, 1 change
    public BaseRepository(TContext context)
    {
        _context = context;
    }

    4 references | Ostap Stetskiv, 9 days ago | 1 author, 1 change
    public async Task<TEntity> Add(TEntity entity)
    {
        _context.Set<TEntity>().Add(entity);
        await _context.SaveChangesAsync();
        return entity;
    }
```

Рис. 11. Клас BaseRepository та метод Add.

```
public async Task<TEntity> Delete<TId>(TId id)
{
    var entity = await _context.Set<TEntity>().FindAsync(id);
    if (entity == null)
    {
        return entity;
    }

    _context.Set<TEntity>().Remove(entity);
    await _context.SaveChangesAsync();

    return entity;
}

4 references | Ostap Stetskiv, 9 days ago | 1 author, 1 change
public async Task<TEntity> GetById<TId>(TId id)
{
    return await _context.Set<TEntity>().FindAsync(id);
}

8 references | Ostap Stetskiv, 9 days ago | 1 author, 1 change
public IQueryable<TEntity> GetAll()
{
    return _context.Set<TEntity>();
}
```

Рис. 12. Методи Delete, GetById, GetAll.

```
2 references | Ostap Stetskiv, 9 days ago | 1 author, 1 change
public IQueryable<TEntity> GetAll(params Expression<Func<TEntity, object>>[] includes)
{
    IQueryable<TEntity> query = _context.Set<TEntity>();
    if (includes != null)
    {
        foreach (Expression<Func<TEntity, object>> include in includes)
        {
            query = query.Include(include);
        }
    }

    return query;
}

4 references | Ostap Stetskiv, 9 days ago | 1 author, 1 change
public async Task<TEntity> Update(TEntity entity)
{
    _context.Entry(entity).State = EntityState.Modified;
    await _context.SaveChangesAsync();
    return entity;
```

Рис. 13. Методи GetAll та Update.

Контролери та методи

Для Функціонування програми було реалізовано наступні контролери:

- FacultyController
- GroupContrroller
- ScheduleController
- RoomController
- MapController

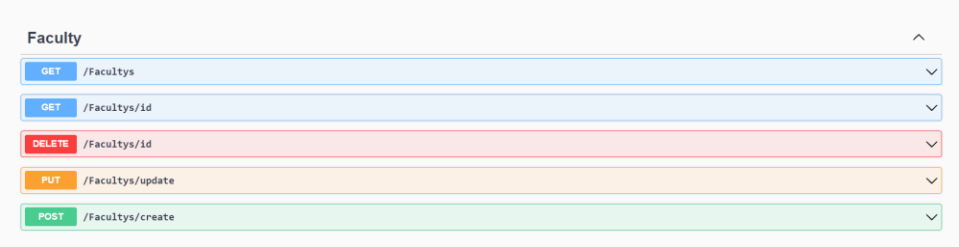

Рис. 14. FacultyController.

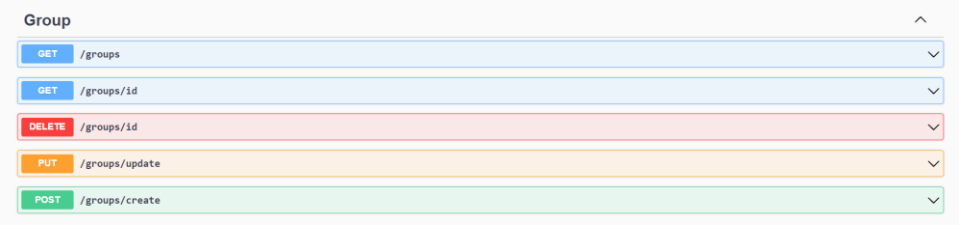

Рис. 15. GroupController

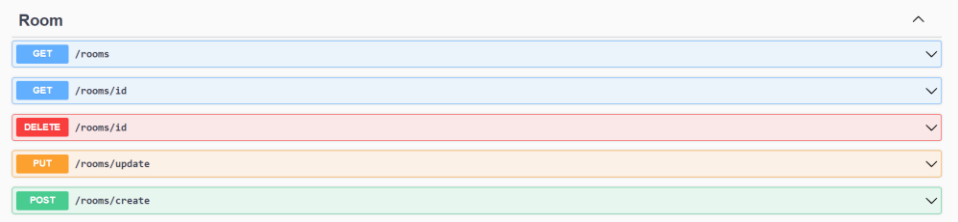

Рис. 16. RoomController

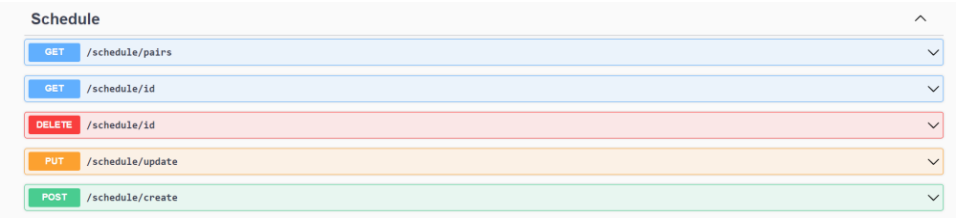

Рис. 17. ScheduleController

CRUD – це абревіатура, що означає CREATE, READ, UPDATE та DELETE. Ці чотири команди бази даних є основою для терміну CRUD.

Методи **ScheduleController**:

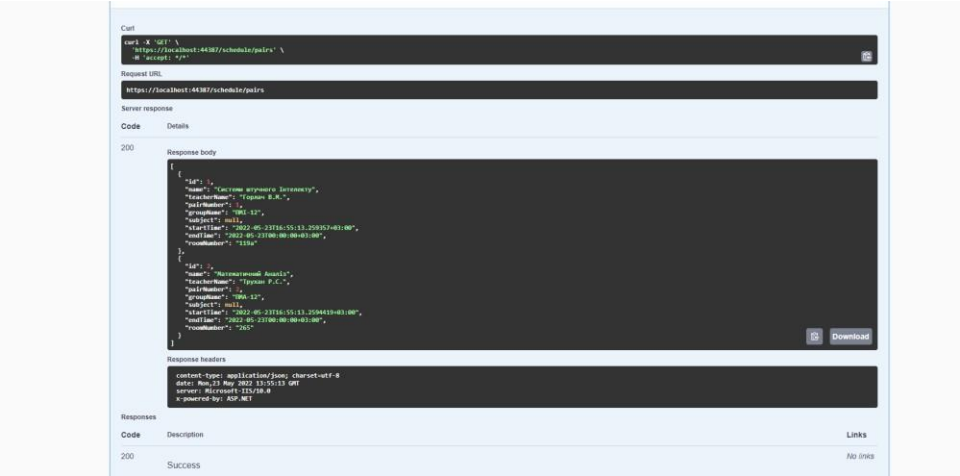

Рис. 18. Метод GET schedule/pairs

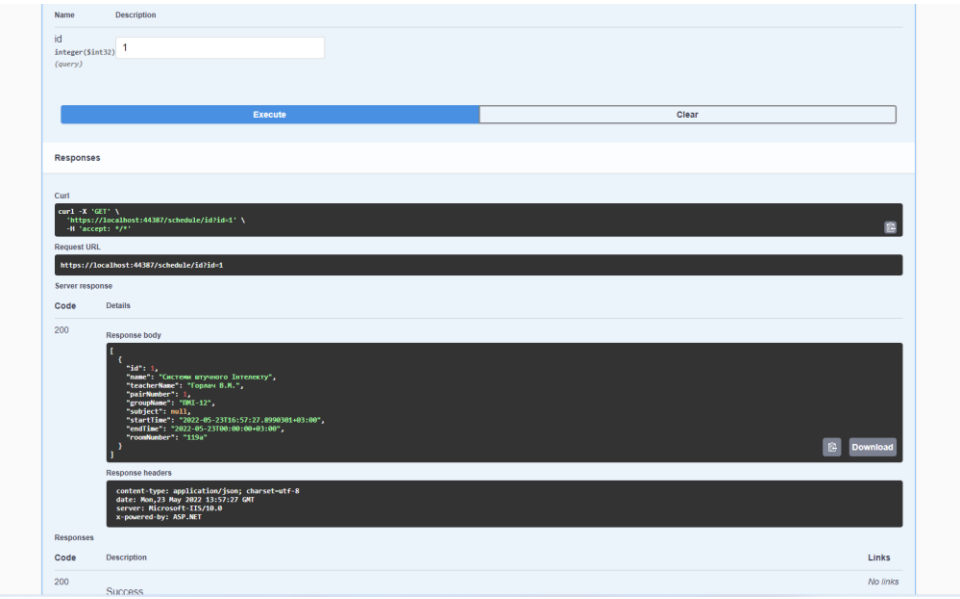

Рис. 19. Метод GET schedule/pairs/id

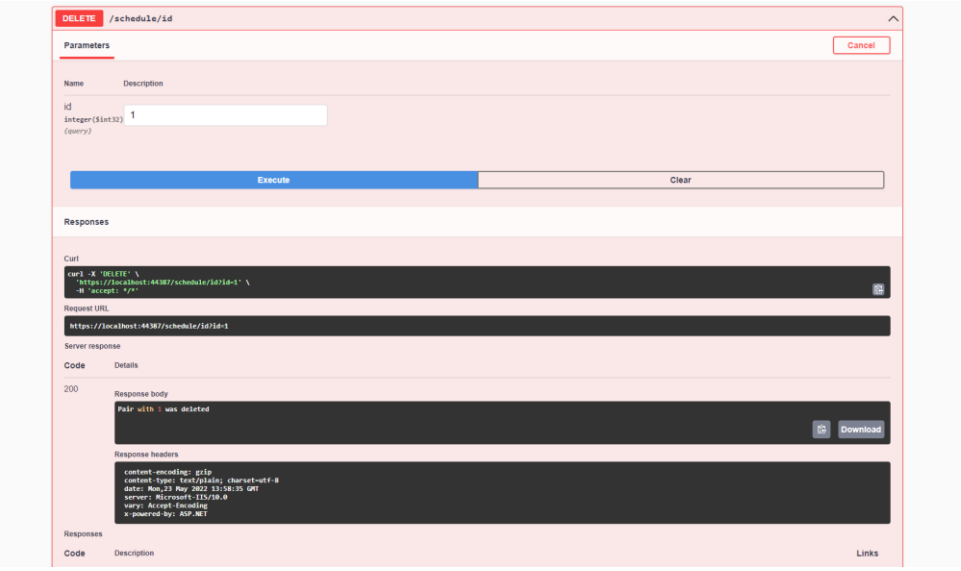

Рис. 20. Метод DELETE schedule/pairs/id

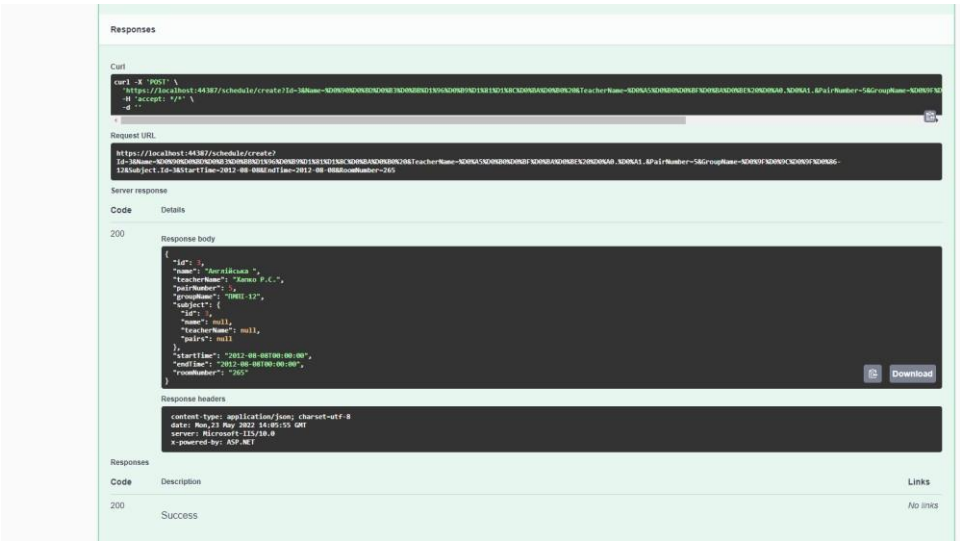

Рис. 21. Метод POST schedule/create

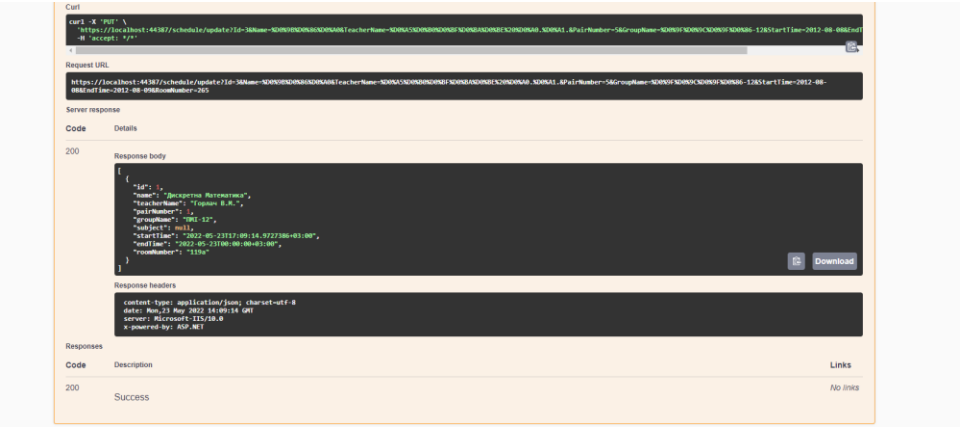

Рис. 22. Метод PUT schedule/update/id

Методи RoomController:

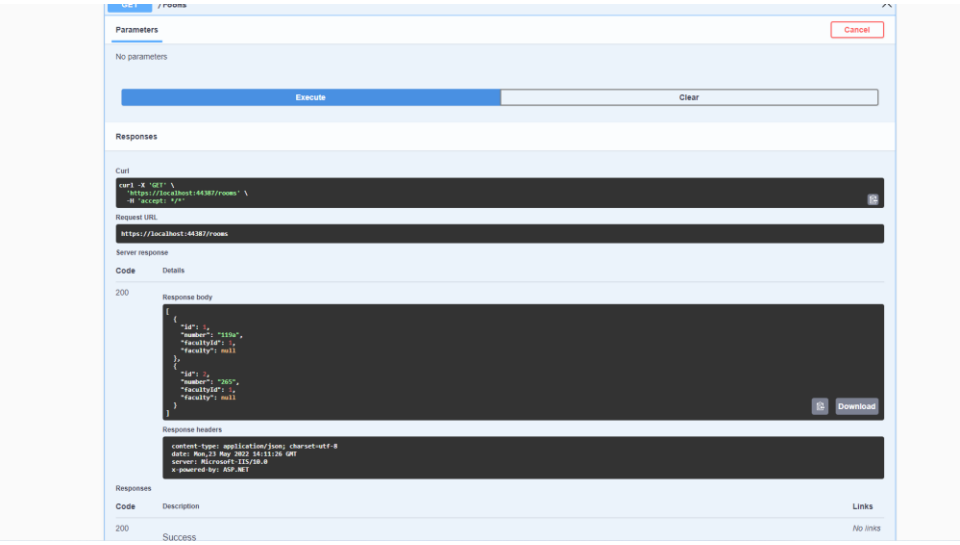

Рис. 23. Метод GET rooms/

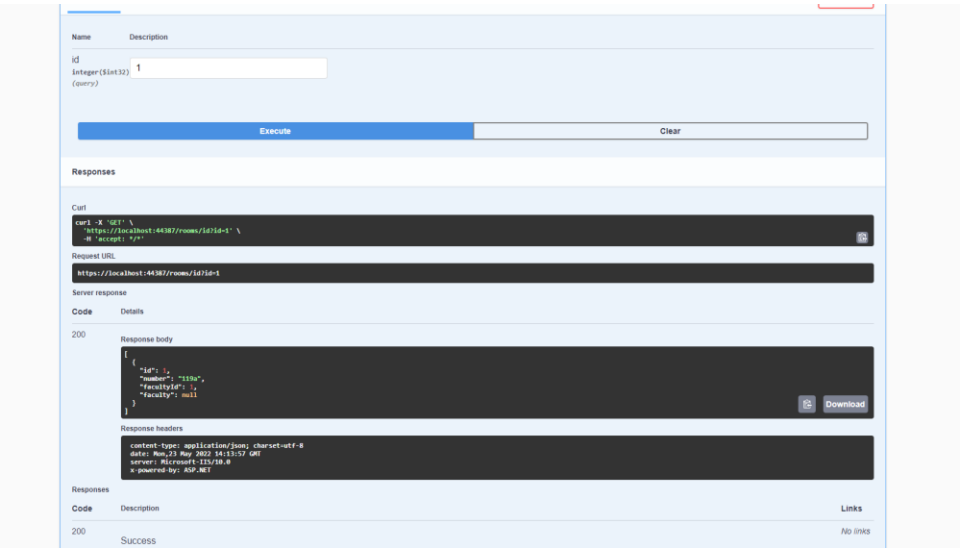

Рис. 24. Метод GET rooms/id

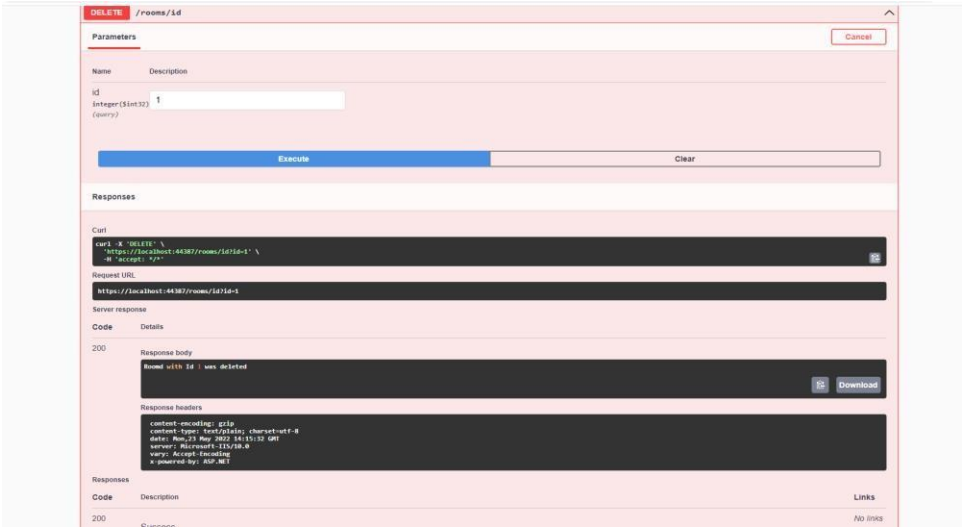

Рис. 25. Метод DELETE rooms/id

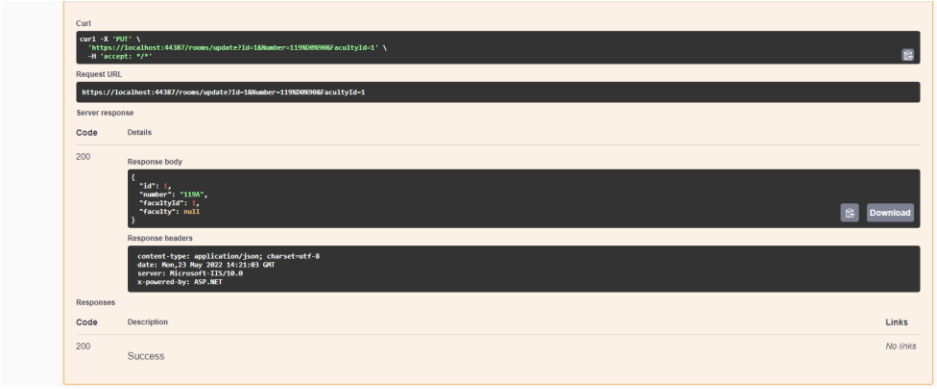

Рис. 26. Метод PUT rooms/update/id

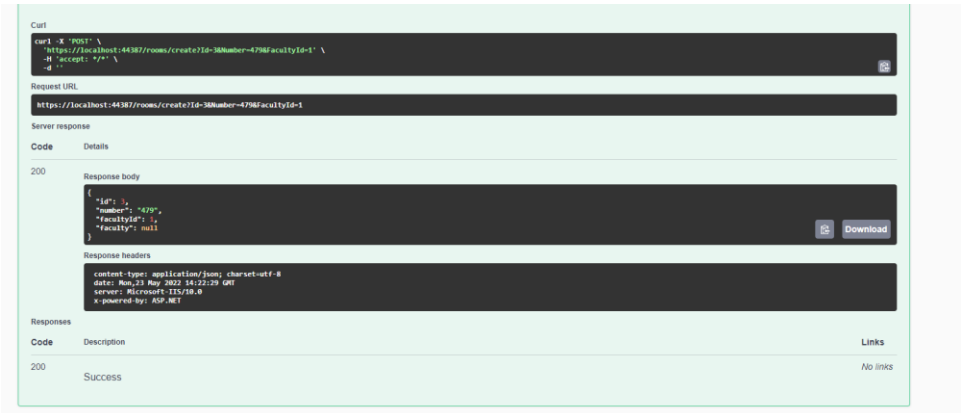

Рис. 27. Метод POST rooms/create

Методи **FacultyController**:

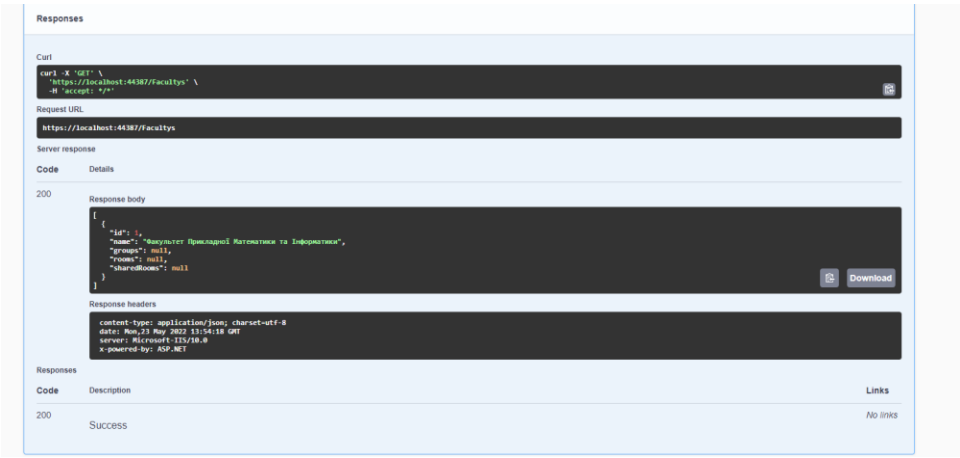

Рис. 28. Метод GET facultys/

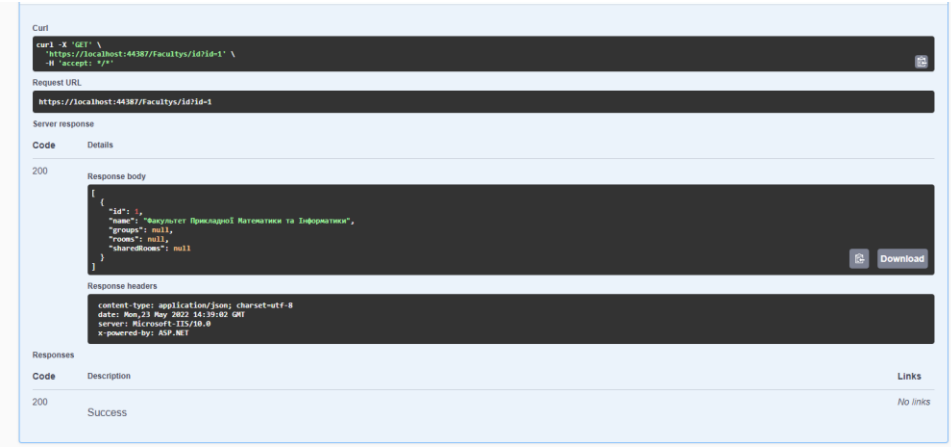

Рис. 29. Метод GET facultys/id

Рис. 30. Метод DELETE facultys/id

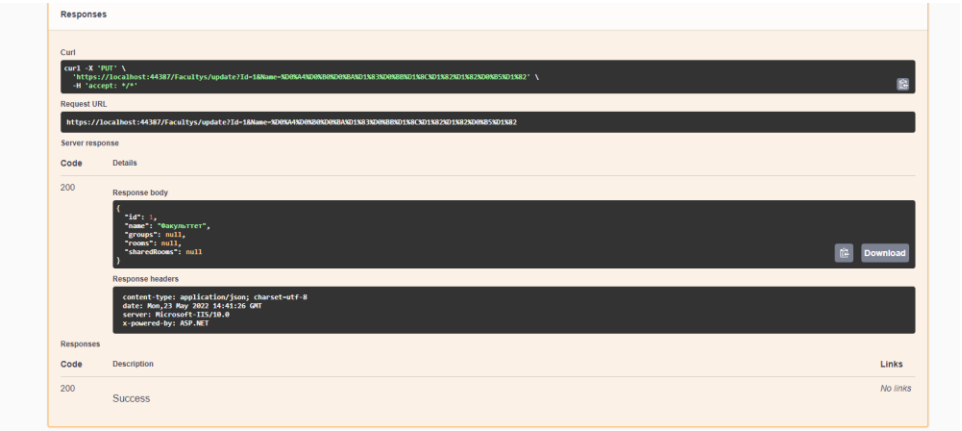

Рис. 31. Метод UPDATE facultys/update/id

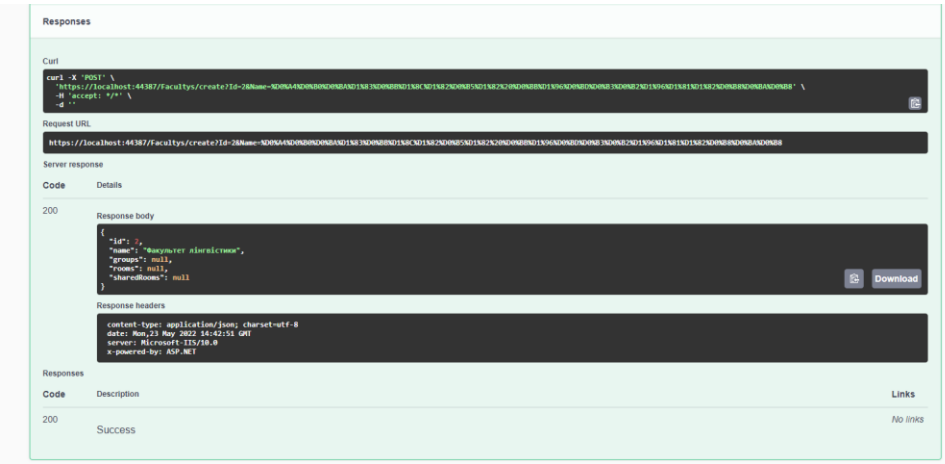

Рис. 32. Метод POST facultys/create

Методи **GroupController**:

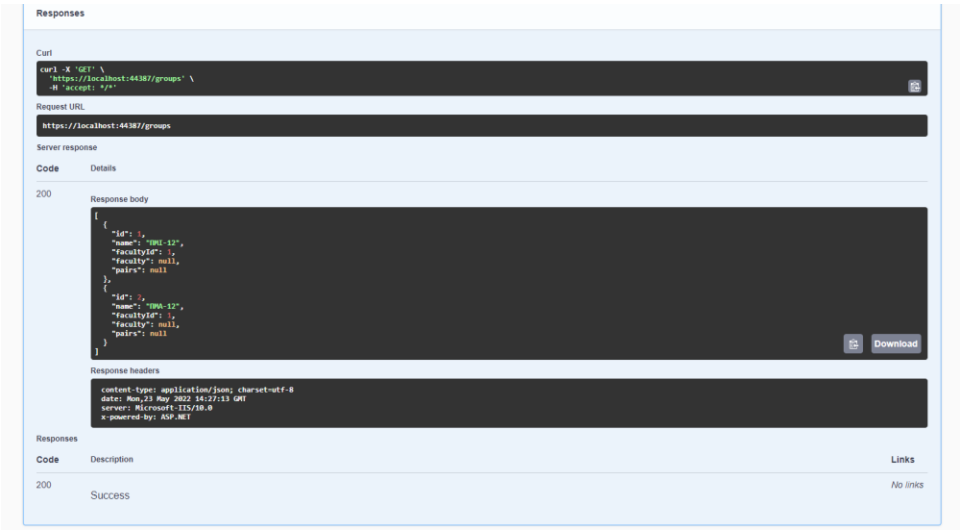

Рис. 33. Метод GET groups

Рис. 34. Метод GET groups/id

Рис. 35. Метод DELETE groups/id

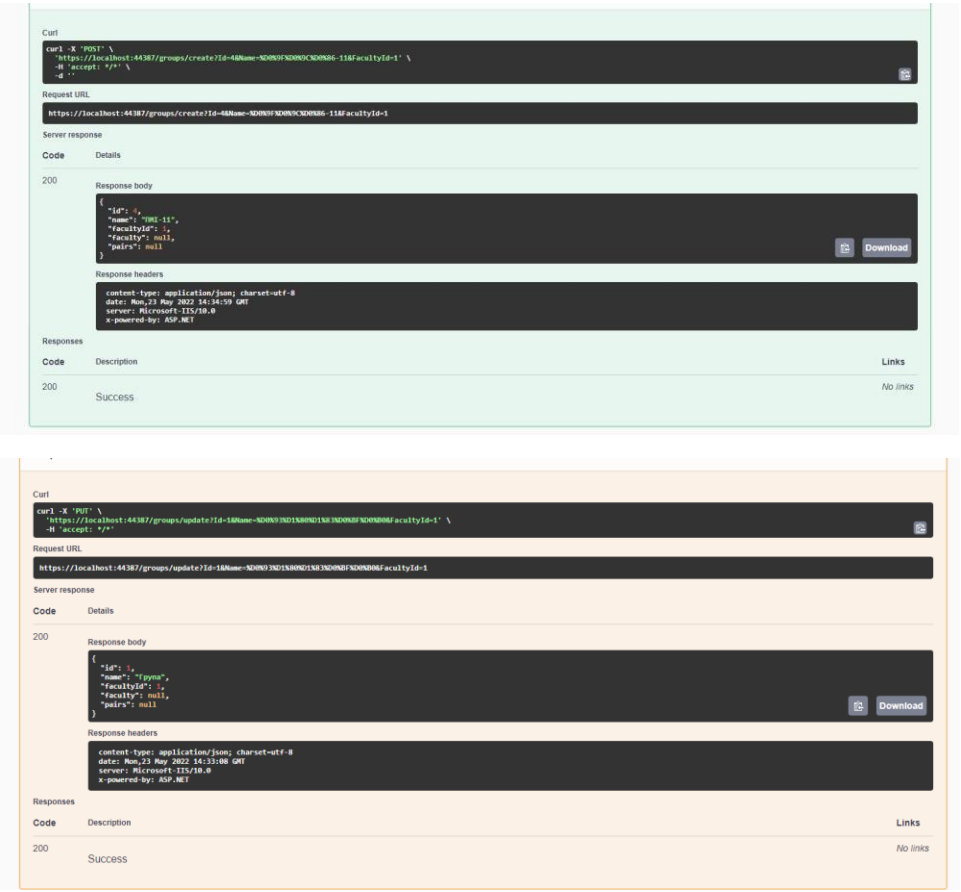

Рис. 36. Метод UPDATE groups/update

Методи **MapController**:

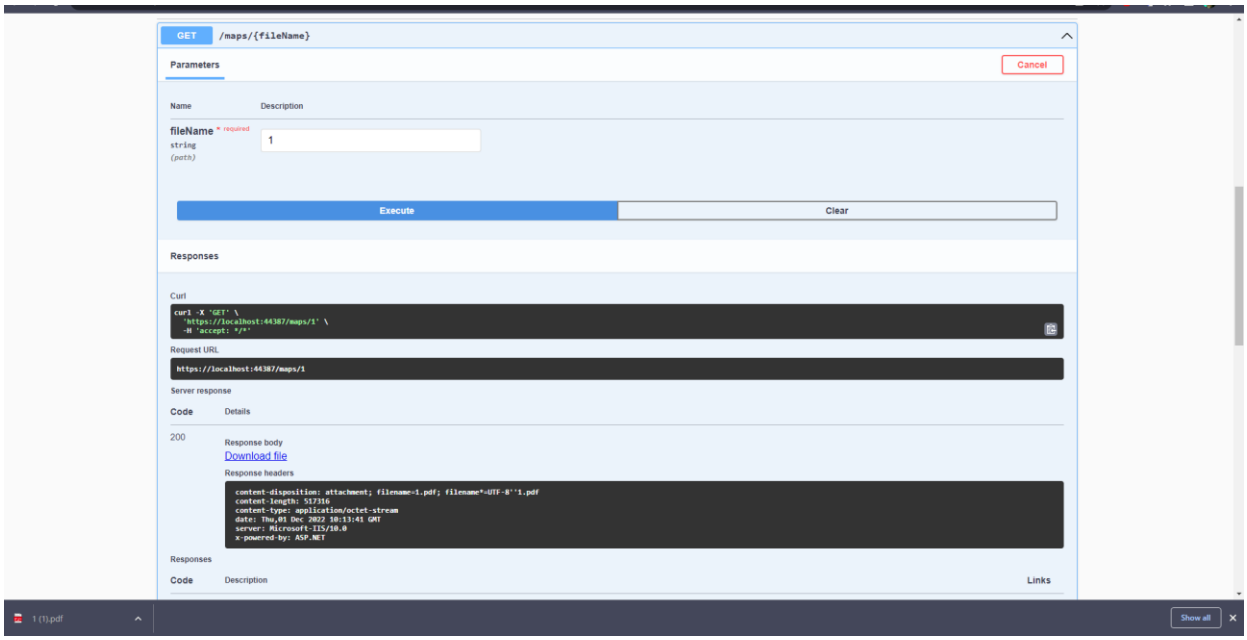

Рис. 38. Метод GET maps\filename

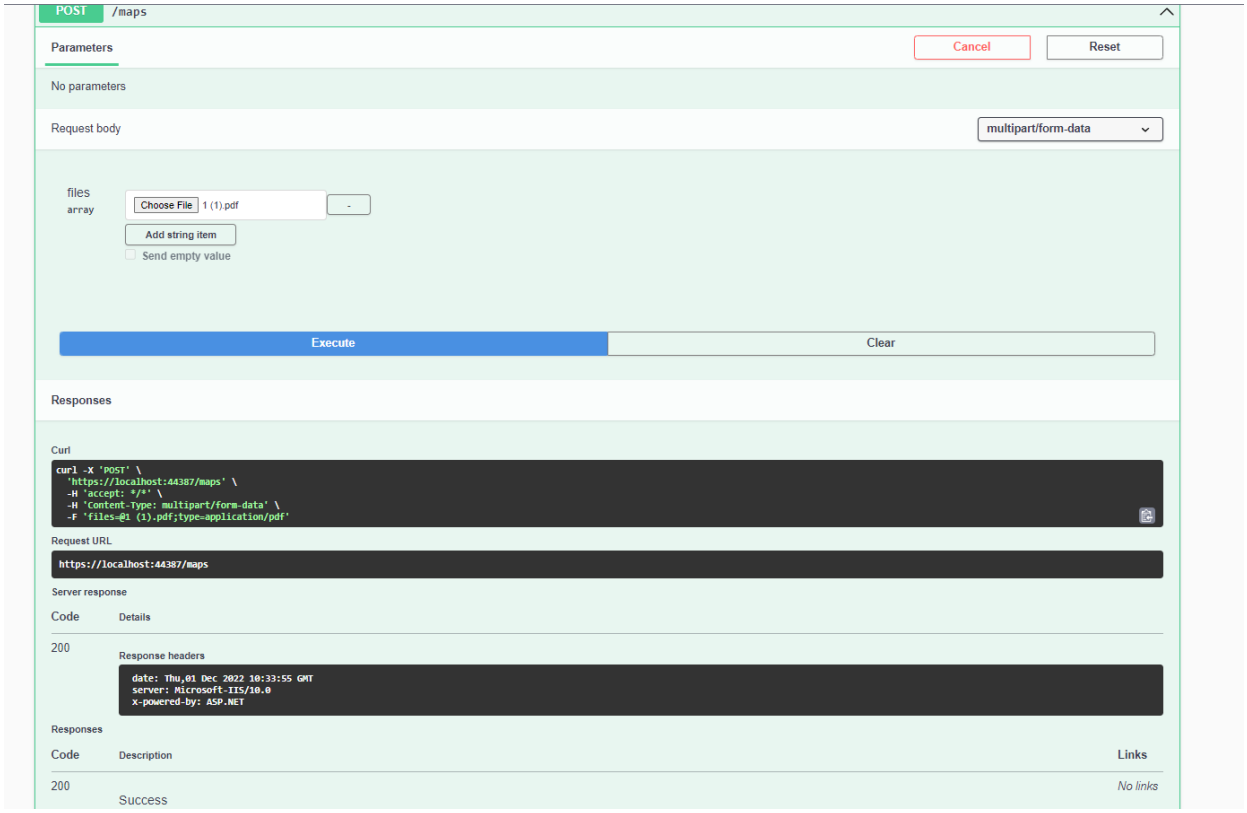

Рис. 39. Метод POST maps.

Валідація

Валідація на стороні клієнта може допомогти користувачам вирішити проблеми перед тим, як зробити запит, але валідації на стороні клієнта часто недостатньо для повної валідації стану. Що я маю на увазі під «недостатнім»? За задумом клієнти рідко мають всю інформацію, необхідну для визначення того, чи введені користувачем дані пройдуть валідацію на сервері. Типовий приклад — форма реєстрації користувача. Хоча ім’я користувача може відповідати критеріям на стороні клієнта для дійсного імені користувача, новачки не знатимуть, чи буде їхня реєстрація успішною, доки сервер не виконає валідацію, щоб визначити унікальність імені користувача щодо всіх існуючих користувачів.

Як правило, форми мають валідацію введення та валідацію стану. Валідація стану вимагає поєднання вхідних даних і зовнішніх даних для визначення дійсності. У цих більш складних сценаріях ви можете використовувати HTMX для надсилання форми та використовувати механізми валідації ASP.NET Core на стороні сервера, щоб виділити та відобразити проблеми. Цей підхід також означає, що ми можемо уникнути створення неповної валідації на стороні клієнта, яка може швидко вийти з синхронізації з нашими аналогами на стороні сервера.

Валідацію я виконуватиму у проекті ScheduleSystem.Domain за допомогою сервісів, які виконуватимуть відповідні перевірки називатимемо їх валідаторами.

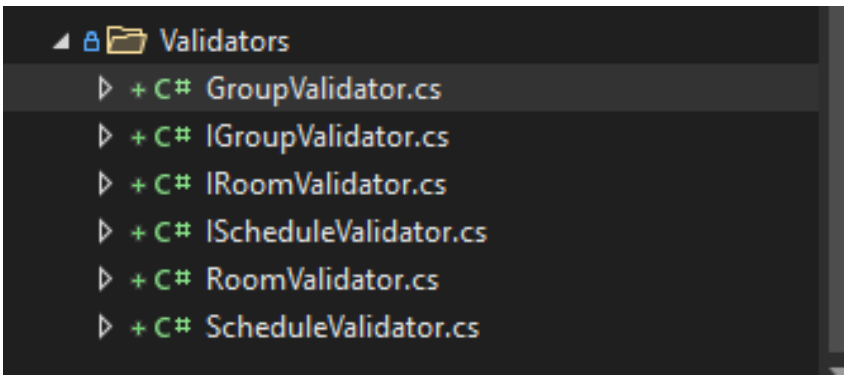

Рис. 40. Валідатори

Система логування

Система логування створена активно використовує Azure Active Directory яка має у собі OpenID Connect. Це дозволить нам використовувати базу даних університету та заходити під акаунтом майкрософт (замість копіювання, одних і тих самих даних).

Також в разі створення ще одної аплікації пов’язаної з майкрософт акаунтом, буде можливість синхронізувати вхід та вихід з майкрософт акаунту.

Рис. 41. Azure Active Directory

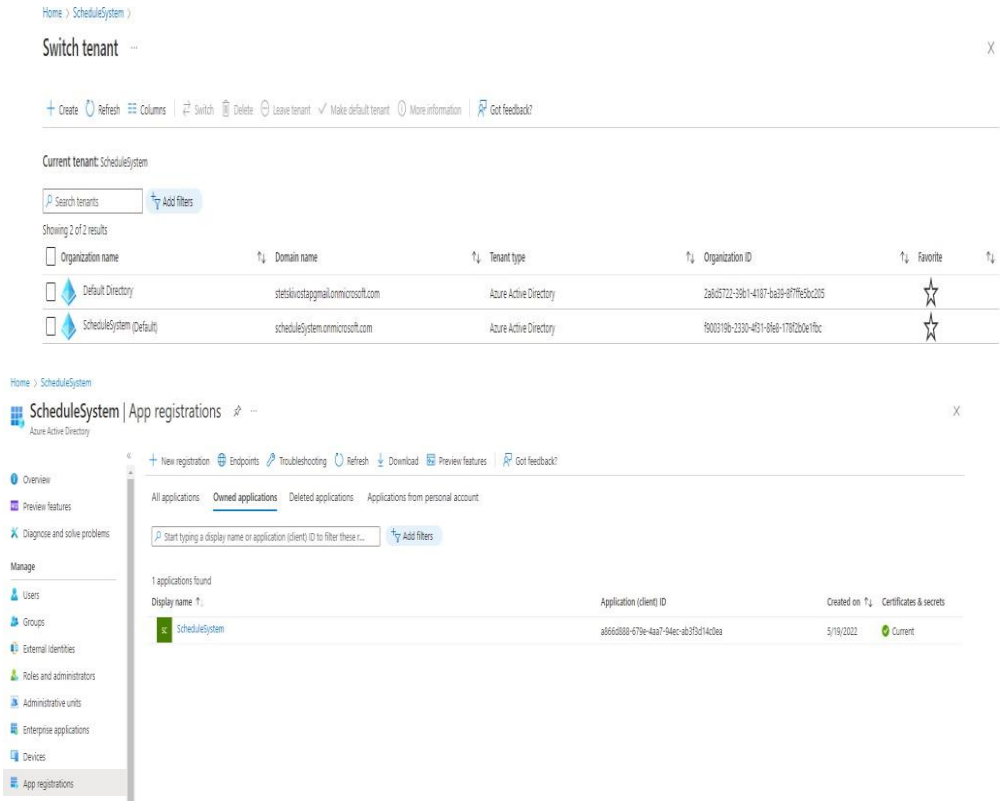

Рис. 42. Зареєстрована ScheduleSystem в Azure Active Directory

OpenID Connect — це простий рівень ідентифікації поверх протоколу OAuth 2.0. Це дозволяє Клієнтам перевіряти особу Кінцевого Користувача на основі аутентифікації, виконаної Сервером Авторизації, а також отримувати основну інформацію про профіль Кінцевого Користувача у взаємосумісний та REST-подібний спосіб.

OpenID Connect дозволяє клієнтам усіх типів, включаючи веб-, мобільні та JavaScript клієнти, запитувати й отримувати інформацію про автентифіковані сеанси та кінцевих користувачів. Набір специфікацій є розширюваним, дозволяючи учасникам використовувати такі додаткові функції, як шифрування ідентифікаційних даних, виявлення постачальників OpenID та керування сеансами, коли це має для них сенс. OpenID Connect виконує багато тих самих завдань, що й OpenID 2.0, але робить це у спосіб, який є зручним для API та придатним для використання рідними та мобільними програмами. OpenID Connect визначає додаткові механізми для надійного підпису та шифрування. Тоді як інтеграція OAuth 1.0 та OpenID 2.0 вимагала розширення, у OpenID Connect можливості OAuth 2.0 інтегровані з самим протоколом.

```
services.AddAuthentication(OpenIdConnectDefaults.AuthenticationScheme)
    .AddMicrosoftIdentityWebApp(Configuration.GetSection("AzureAd"));
```

Рис. 43. Додавання комунікації з OpenId Connect.

```
0 references | Ostap Stetskiv, 206 days ago | 1 author, 1 change
public class AdOptions
{
    0 references | Ostap Stetskiv, 206 days ago | 1 author, 1 change
    public string ClientId { get; init; }

    0 references | Ostap Stetskiv, 206 days ago | 1 author, 1 change
    public string ClientSecret { get; set; }

    0 references | Ostap Stetskiv, 206 days ago | 1 author, 1 change
    public string TenantId { get; init; }

    0 references | Ostap Stetskiv, 206 days ago | 1 author, 1 change
    public string Instance { get; init; }
}
```

Рис. 43. Додавання комунікації з OpenId Connect.

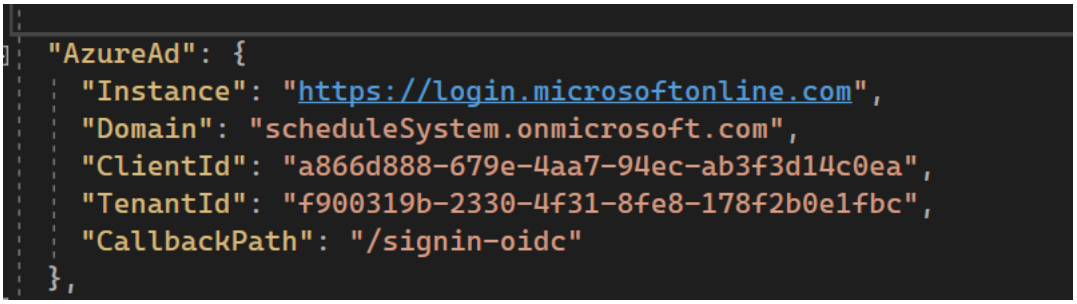

Рис. 44. Додавання комунікації з OpenId Connect.

Висновок

Як бачимо, зі збільшення доступу до інформації неабияку необхідність у студентів та викладачів викликала можливість переглядати розклад онлайн правильне формування навчальних планів у ВНЗ. В наш час розробка такого розкладу є дуже складним і трудомістким завданням.

Якщо при складанні розкладу враховано не всі зв’язки між сутностями, бізнес-логіку та можливостями студента та викладача то це позначається на якості навчального процесу, що явно знизить рівень підготовки студентів.

Проте ручна розробка паперового варіанту розкладу все ще досить поширена у ВНЗ, незважаючи на те що це займає багато часу та несе високий ризик помилки, не всі університети можуть дозволити собі використовувати програмне забезпечення, яке автоматизує планування навчання, тому існує нагальна потреба в розробці таких веб-застосунків.

Тому метою магістерської роботи була розробка веб-застосунку, який дасть доступ навчального плану. Як можемо бачити у магістерській роботі була досягнута ціль та розроблений веб-ресурс побудови розкладу.

Джерела

- Головна сторінка факультету Прикладної математика та інформатики https://ami.lnu.edu.ua/
- Інформація про викладачів факультету Прикладної математика та інформатики https://ami.lnu.edu.ua/about/staff
- Розклад факультету Прикладної математика та інформатики https://ami.lnu.edu.ua/students/rozklad-zanyat
- Документація по Azure Active Directory https://azure.microsoft.com/en-us/free/active-directory/
- Переваги та недоліки патерну репозиторій https://stackoverflow.com/questions/12846490/what-are-the-advantages-disadvantages-of-making-a-repository-persistence-ignoran
- Документація по .NET https://docs.microsoft.com/en-us/dotnet/core/introduction
- Документація по створенні міграцій та бази даних Microsoft SQL Server https://docs.microsoft.com/en-us/aspnet/mvc/overview/getting- started/getting-started-with-ef-using-mvc/migrations-and-deployment- with-the-entity-framework-in-an-asp-net-mvc-application
- Трирівненва архітектура https://medium.com/@deanrubin/the-three-layered-architecture-fe30cb0e4a6
- Патерн - https://refactoring.guru/uk/design-patterns/what-is-pattern