# **ВИБІР ЗАСОБІВ ДЛЯ РЕАЛІЗАЦІЇ**

## **2.1 Перелік технологій та пояснення вибору**

Всі використані технолої були вибрані опираючись на досвід використання в попередніх комерційних проектах. Технології показали максимальну швидкодію, мінімальний час на розробку а також легкість використання.

Короткий перелік технологій та місця їх використання:

1. Клієнтська частина

* TypeScript – об‘єктно-орієнтована мова програмування, яка компілюється в JavaScript. Виконується в браузері та допомагає анімувати веб-сторінку.
* Angular – фреймворк, який спрощує роботу з TypeScript і зменшує час, який потрібен на розробку програмного рішення.
* HTML – декларативна мова для опису веб-сторінок. Відповідає за наповнення web-сторінки.
* SCSS – декларативна мова, яка компілюється в CSS. Відповідає за відображення web-сторінки.

1. Серверна частина

* C# - флагманська мова програмування компанії Microsoft. Відповідає за обробку даних та взаємодію з базою даних.
* ASP.NET Core – фреймворк, який спрощує роботу з C#. В ньому імплементований функціонал для роботи з HTTP.
* EntityFramework Core – фреймворк який спрощує взаємодію з базою даних. API між базою даних і C# кодом.
* SQLite – компактна СУБД. Використвоується для збереження даних.

## **2.2 Короткий опис використаних технологій**

### **2.2.1 TypeScript**

TypeScript [5] - це мова програмування, в якій виправлено багато недоліків JavaScript. Код на TypeScript виглядає майже так само, як і код на JS, і якщо у вас є досвід frontend-розробки, вивчити TypeScript досить просто. Особливо з огляду на те, що ви можете писати JS-код прямо в TS-скриптах.

Код на TypeScript компілюється в JS і підходить для розробки будь-яких проектів під будь-які браузери - тим більше, що можна вибрати версію JS, в яку компілюватиметься код.

TypeScript - проект з відкритим вихідним кодом, тому він швидко розвивається. Багато що з'являється в TS пізніше переходить і в JavaScript: наприклад, let і const, стрілочні функції і так далі.

**Переваги TypeScript:**

1. Строга типізація

Багато проблем у JavaScript виникають через динамічну типізацію і в цілому дивну поведінку типів даних. У TypeScript типізація статична, що позбавляє багатьох проблем. Є числовий тип, рядковий, логічний та інші.

1. Поліпшене ООП

І в JS, і в TS є підтримка об'єктно-орієнтованого програмування: класи, об'єкти, спадкування. Однак TypeScript зробив крок далі і використовує більше можливостей ООП. У тому числі, наприклад, інтерфейси. Інший великий плюс – модифікатори доступу. Їх у TypeScript три: public, private та protected.

### **1.2.2 Angular**

Angular [6] – це платформа розробки, побудована на TypeScript. Як платформа, Angular включає:

1. Компонентний фреймворк для створення масштабованих веб-додатків.
2. Колекція добре інтегрованих бібліотек, які охоплюють широкий спектр функцій, включаючи маршрутизацію, керуування формами, зв‘язок клієнт-сервер тощо.
3. Набір інструментів для розробників, які допоможуть вам розробляти, створювати, тестувати та оновлювати код.

З Angular розробник користується перевагами платформи, яка може масштабуватися від проектів окремого розробника до програм корпоративного рівня.

### **2.2.3 HTML**

HTML (Hypertext Markup Language) [7] – це код, який використовується для структурування та відображення веб-сторінки та її вмісту. Наприклад, вміст може бути структурованим всередині множини параграфів, маркованих списків або з використанням зображень та таблиць даних.

### **2.2.4 SCSS**

SCSS (Syntacically Awesome Style Sheet) [8] – надмножина CSS. SCSS є більш просунутою версією CSS. SCSS розроблений Хемптоном Кетліном, Крісом Еппштейном і Наталі Вайзенбаум.

Переваги SCSS над CSS:

1. SCSS містить усі функції CSS. А також ті, яких немає в CSS, що робить його хорошим вибором для розробників.
2. SCSS наповнений розшириними функціями.
3. SCSS пропонує змінні. Це велика перевага над звичайним CSS.
4. SCSS дозволяє використовувати вкладений синтаксис.

### **2.2.5 C#**

C# [9] - це строго типізована об‘єктно-орієнтована мова програмування.

C# - це мова програмування, розроблена та запущена компанією Microsoft у 2001 році. C# - проста, сучасна об‘єктно орієнтована мова програмування, яка надає сучасним розробникам гнучкість та можливості створення програмного забезпечення, яке не тільки працюватими сьогодні, але й буде застосовуватися протягом багатьох років.

**Основні переваги C#:**

1. C# сучасний та простий

C# — це проста, сучасна й об'єктно-орієнтована мова програмування. Метою C# була розробка мови програмування, яку не тільки легко вивчити, але й підтримувати сучасну функціональність для всіх видів розробки програмного забезпечення

1. C# є кросплатформним

C# — це кросплатформна мова програмування. Можна створювати програми .NET, які можна розгорнути на платформах Windows, Linux і Mac. Програми C# також можна розгортати в хмарі та в контейнерах.

### **2.2.5 ASP.NET Core**

ASP.NET Core [10] — це нова версія веб-фреймворку ASP.NET, яка в основному призначена для роботи на платформі .NET Core.

ASP.NET Core — безкоштовний, з відкритим вихідним кодом та кросплатформна платформа для створення хмарних додатків, таких як веб-програми, програми Інтернету речей і мобільні серверні програми. Він призначений для роботи як у хмарі, так і локально.

Так само, як і .NET Core, він був розроблений модульно з мінімальними накладними витратами, а потім інші більш розширені функції можуть бути додані як пакети NuGet відповідно до вимог програми. Це призводить до високої продуктивності, потребує менше пам’яті, меншого розміру розгортання та легкого обслуговування.

### **2.2.6 EntityFramework Core**

Entity Framework Core [11] — це нова версія Entity Framework після EF 6.x з відкритим вихідним кодом, легка, розширювана і кросплатформна версія технології доступу до даних Entity Framework.

Entity Framework — це структура об’єктного/реляційного відображення (O/RM). Це вдосконалення ADO.NET, яке надає розробникам автоматизований механізм доступу та зберігання даних у базі даних.

EF Core призначений для використання з додатками .NET Core. Однак його також можна використовувати зі стандартними додатками на основі .NET 4.5+.

### **2.2.7 SQLite**

SQLite [12] — це бібліотека, яка реалізує автономний, безсерверний, транзакційний механізм баз даних SQL з нульовою конфігурацією. Код для SQLite є загальнодоступним і, таким чином, безкоштовний для будь-яких цілей, комерційних чи приватних. SQLite — це найпоширеніша база даних у світі з більшою кількістю додатків, ніж ми можемо розрахувати, включаючи кілька високопрофільних проектів.

## **2.3 Висновоки**

Після розгляду та порівняння сучасних технологій та мов програмування мною були вибрані C# та TypeScript. C# - флагманська мова компанії Microsoft. Вона зручна, швидка та зменшує час на розробку. Використовується в серверній частині програмного рішення. TypeScript – сучасна мова програмування, яка використовується на клієнтській частині. Компілюється в JavaScript. Основна перевага TypeScript полягає в статичній типізації. Це допомагає уникнути більшості помилок на етапі компіляції. Відштовхуючись від вибраних мов програмування я вибрав зручні та функціональна фреймворки.

# **РЕАЛІЗАЦІЯ**

## **3.1 Випадки використання**

В користувача є 11 випадків використання програмного продукту:

1. Створити проект

Проект це абстактна сутність, до якої можуть прив‘язуватися інші сутності. Він слугує одиницею розмежування роботи. Різні проекти можуть містити свої унікальні дисципліни, викладачів тощо.

1. Вибрати проект для роботи

Після вибору проекту користувач може створювати інші сутності. Вони будуть використовуватись для збереження і обрахунку даних.

1. Створити університет

Сутність універститету допомагає досягнути універсальності. В майбутньому можна використовувати аплікацію для різних кафедр та університетів.

1. Створити активність

Під активністю мається на увазі «Лаборатна», «Курсова», «Димпломна» робота тощо. Теж допомагає досягти універсальності.

1. Створити лектора

Сутність лектора потрібна для співставленням з реальним лектором.

1. Створити дисципліну

Сутність дисципліни потрібна для співставлянням з реальною дисципліною. Вона повинна містити кількість студентів, груп тощо.

1. Приєднати лектора до університету

Сутність зв‘язку Лектор-Університет уникає дублювання записів в базі даних. Один лектор може викладати в декількох університетах або на різних кафедрах.

1. Приєднати лектора до активності і дисципліни.

Сутність зв‘язку Лектор-Активність-Дисципліна теж уникає дублювання записів в базі даних. Один лектор може проводити як лабораторні так і лекції в декількох дисциплінах.

1. Запустити обрахунок даних

Запуск обрахунку даних розподіляє навантаження між викладачами опираючись на попередньо додані дані.

10) Експортувати дані

Експорт даних це міграція даних з бази даних до xlsx або csv для подальшого використання деканатом або іншими керуючими органами.

1. Переглянути логи

Перегляд логів це системна функція. В логах буде зберігатися інформація про роботу програми або помилки, які виникли під час роботи. Опираючись на логи буде простіше пояснити розробнику про проблеми, які могли виникнути під час експлуатації програмного продукту. Таким чином розробник витратить менше часу на виправлення помилок в коді програми.

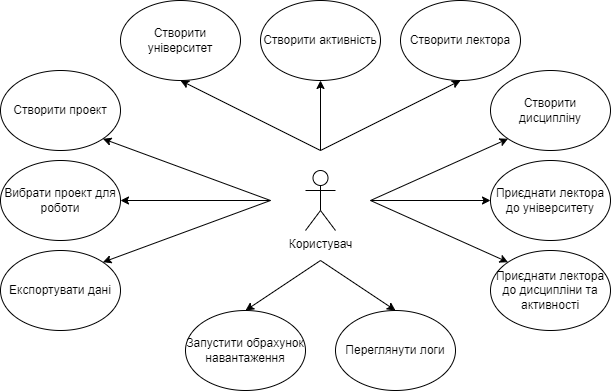
****

Рис. 3.1. Випадки використання

## **3.2 Алгоритм роботи користувача**

Всі блоки алгоритму були детально пояснені в попередньому підрозділі.



Рис. 3.2. Алгоритм роботи користувача

## **3.3 Архітектурне рішення**

Покладаючись на висунуті технічні вимоги я побудував артіхектуру, яка складається з двох частин. Перша – клієнтська частина, друга – серверна частина. В майбутньому серверну частину можна розгорнути на будь-якому сервері або в хмарі. Внаслідок цього користувач може працювати з аплікацією з будь-якого пристрою, знаходячись в будь-якій точці світу де є доступ до Інтернету. Клієнтська частина(Браузер) обмінюється даними з REST API. Який в свою чергу оперує даними з бази даних. В базі даних зберігаються потрібні користувачу дані.

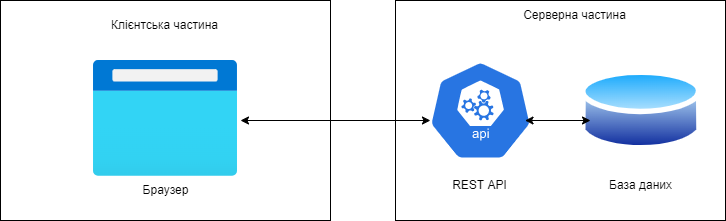


Рис. 3.3. Архітектура аплікації

## **3.4 Структура бази даних**

Спроектована база даних складається з 10 таблиць:

1. \_EFMigrationsHistory

Використовується для збереження даних про міграції для простого звязку коду та таблиці. Допомогає співставити існуючий відбиток бази та класів, описаних в коді.

1. Logs

Таблиця, в якій буде зберігатися технічна інформація і помилки. Допоможе розробнику виявити помилки та виправити їх.

1. Lecturer

В цій таблиці буде збережено дані про кожного з викладачів. Для майбутнього їх викристання в обчисленнях.

1. Activity

Буде використана для зберігання всіх існуючих активностей: лабораторна робота, курсова робота, практична робота, тощо...

1. Project

Є верхнім рівнем ієрархії, для можливості використання програмного продукту декількома кафердами паралельно.

1. University

Зберігається інформація про поточний унівеситет.

1. Discipline

В цій таблиці буде збережена вся інформація про дисципліну, кількість студентів, курсів, потоків, тощо...

1. LecturerDisciplineActivityMap

Допоміжна таблиця, для збереження звязку між лекторами, дисциплінами та активностями.

1. DisciplineActivityMap

Допоміжна таблиця для збереження звязку між дисциплінами та активностями.

10) UniversityLecturerMap

Допоміжна таблиця для збереження звязку між університетами і лекторами.

Всі таблиці містять первинні, вторинні ключі, індекси для збільшення швидкодії виконання запитів. База даних була нормалізована та оптимізована.

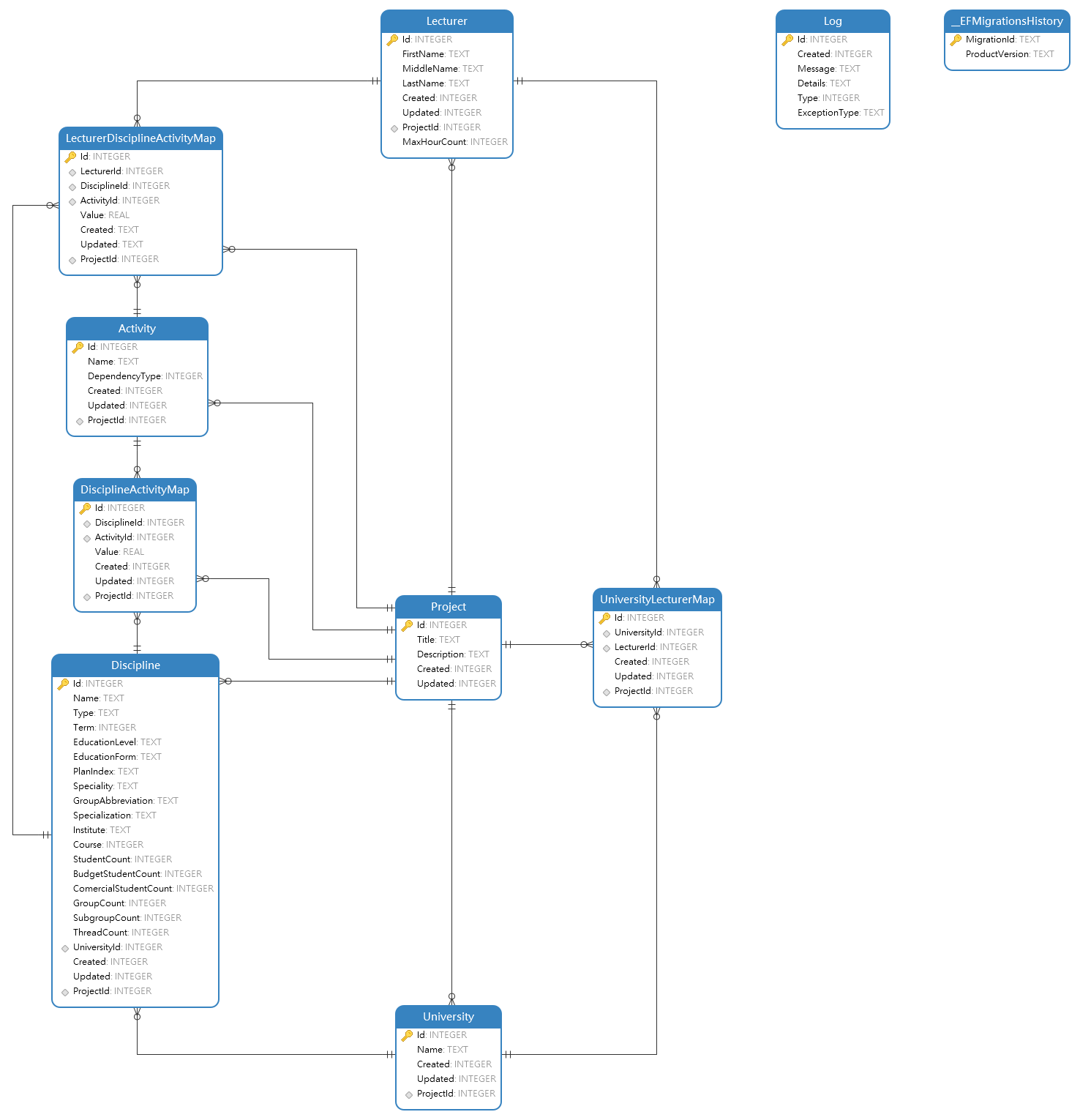
****

Рис. 3.4. Структура бази даних

## **3.5 Ієрархія та звязки між класами серверної частини**

Загалом було створено 3 інтерфейса, 2 абстрактні класи та 9 доменних класів. Всі доменні класи відповідають таблицям в базі даних, які були описані в розділі 3.1.1. Опис абстрактних класів та інтерфейсів:

1. IEntity

Інтерфейс для поліморфної поведінки доменних моделей. Містить ідентифікатор. Використовується в абстракному репозиторію для взаємодії з базою даних.

1. ICreateble

Інтерфейс для поліморфної поведінки доменних моделей. Містить дату створення запису в базі даних. Використовується для сортування по даті створення в абстрактному репозиторію.

1. IUpdateble

Інтерфейс для поліморфної поведінки доменних моделей. Містить дату оновлення записів в базі даних. Використовується для збереження дати останньої зміни доменної моделі.

1. BaseEntity

Абстрактний клас для уникання копіювання властивостей в доменних моделях.

1. BaseProjectRelatedEntity

Абстрактний клас для уникання копіювання навігаційних властивостей в доменних моделях повязаних з проектом.

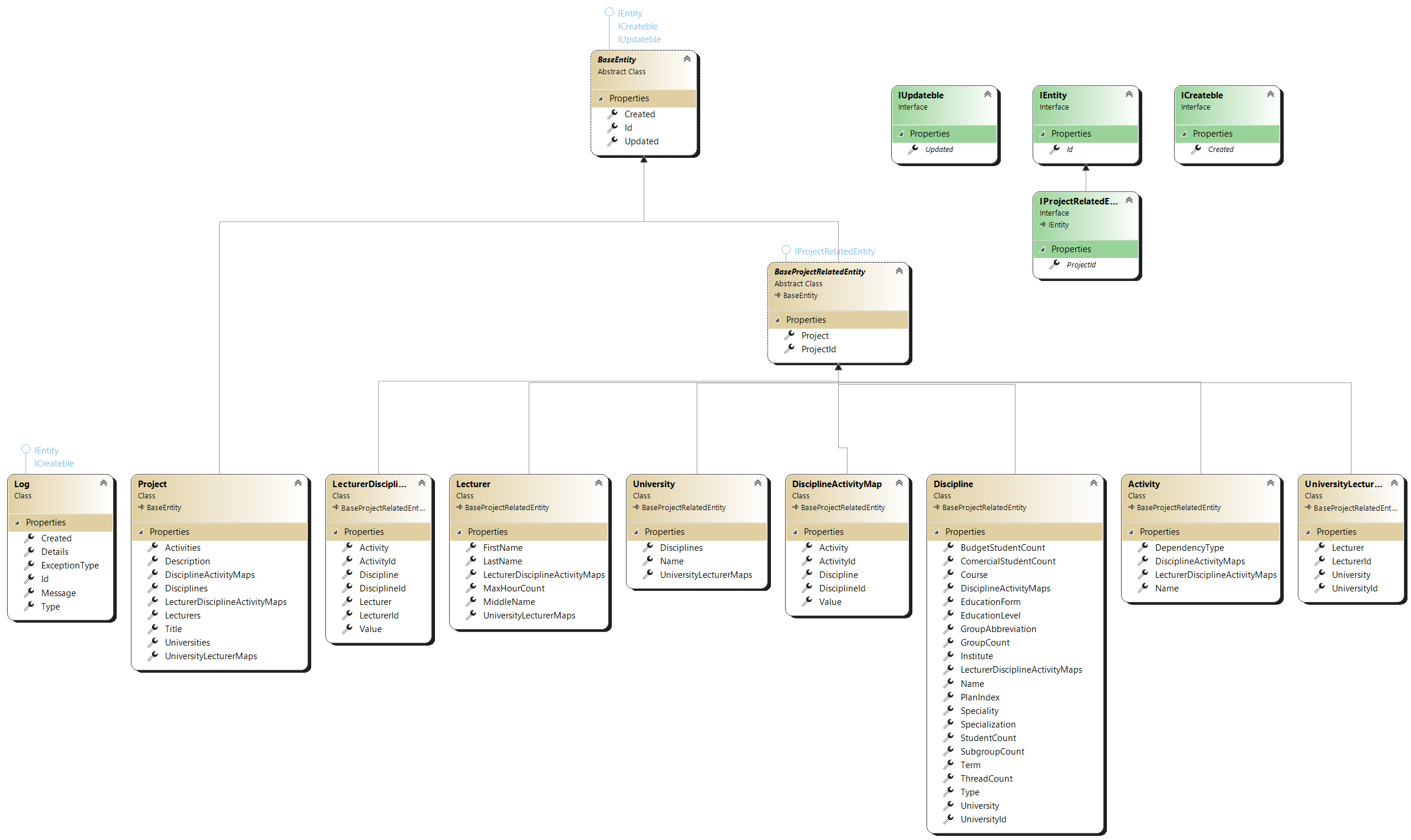
****

Рис. 3.5. Ієрархія та звязки між доменними класами

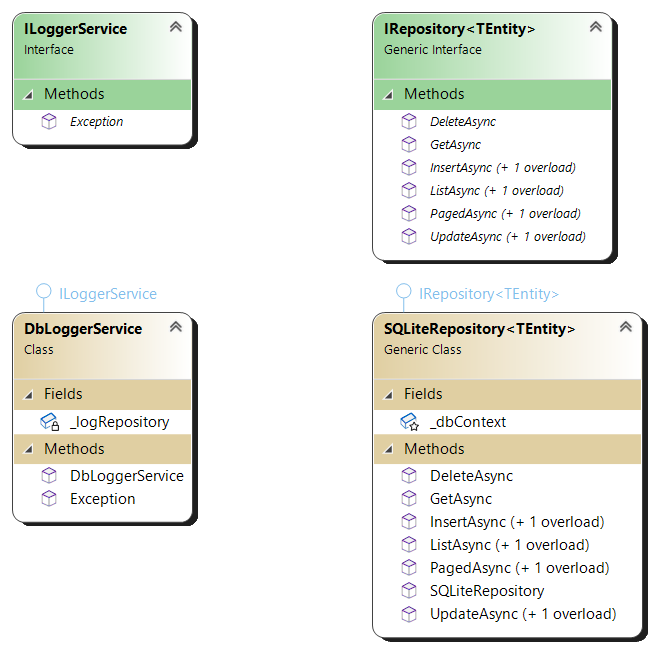


Рис. 3.6. Ієрархія та звязки між сервісами

Було створено 1 сервіс та 1 шаблонний репозиторій, а також інтерфейси для них. Інтерфейси будуть використані для швидкої зміни реалізацій сервісів або репозиторіїв в разі потреби.

1. ILoggerService

Інтерфейс, в якому описані методи логування помилок.

1. DbLoggerService

Клас, який використовується для збереження логів в базі даних.

1. IRepository

Шаблонний інтерфейс, в якому описані методи для взаємодії з базою даних.

1. SQLiteRepository

Шаблонний клас, який використовується для взаємодії з базою даних.

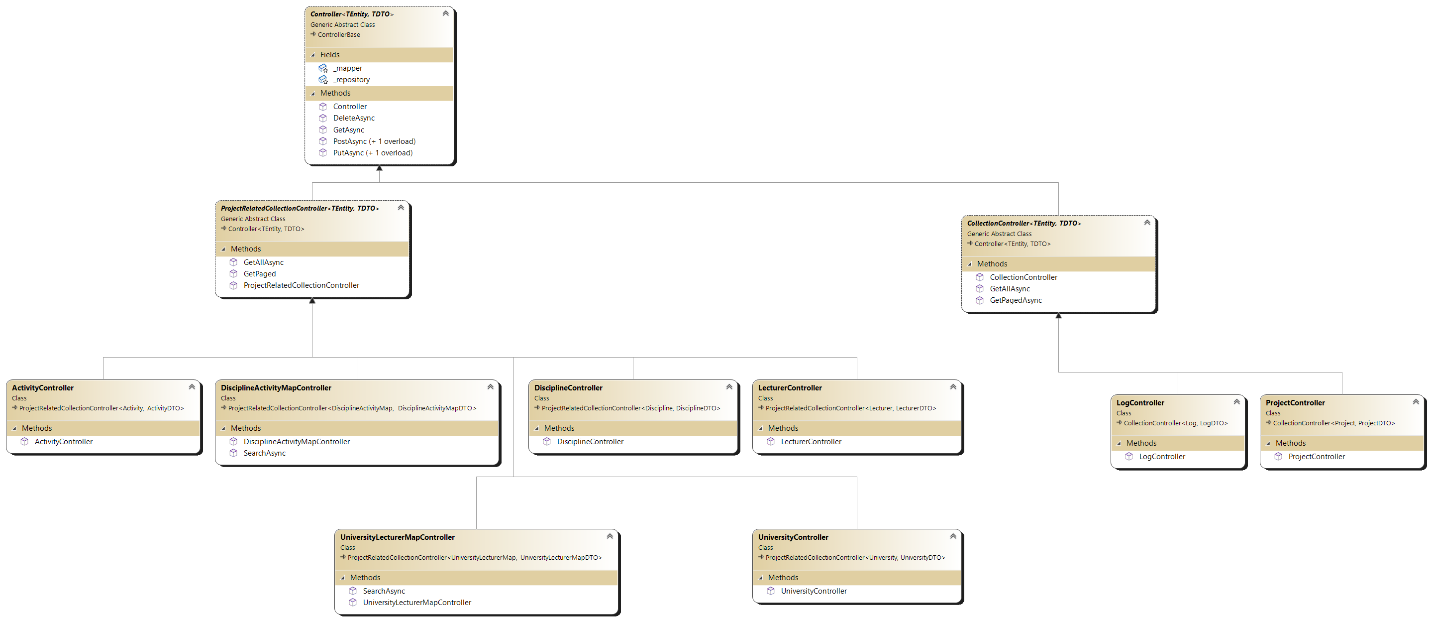
****

Рис. 3.7 Ієрархія та звязки між контроллерами

Контроллери служать для трансферу даних з бази даних до клієнтської частини. Було створено 11 класів, 3 з яких абстрактні.

1. Controller

Абстрактний шаблонний клас. Створений для базових CRUD операцій і зменшення кількості коду.

1. CollectionController

Абстрактний шаблонний клас. Створений для маніпуляції колекціями даних не повязаних з проектом.

1. ProjectRelatedCollectionController

Абстрактний шаблонний клас. Створений для маніпуляції колекціями даних повязаних з проектом.

Всі інші контроллери наслідуються від абстрактних і можуть перевантажувати методи цих контроллерів. Перевантаження залежить від особливостей реалізації кінцевого контролера.

## **3.6 Ієрархія та звязки між класами клієнтської частини**

Для абстрагування та зменшення дубляжу коду було створено 6 допоміжних класів:

1. CUComponent

Клас використовується для зберігання базових даних компонетів. В ньому реалізовано методи обробки подій а також оновлення даних на веб сторінці. Цей клас забезпечує 2 базові операції: створення та оновлення даних.

1. OperationManager

Використовується для агрегації базовий операцій для колекцій даних. Такі як: оновлення, видалення і створення.

1. PaginatorI18n

Служиь для локалізації елементу управління «Paginator».

4) ProjectHanlder

Використовуэться для реактивного вибору проекту.

5) TableDataSource

Агрегує операцї, потрібні для відображення колекцій даних.

1. TableComponent

Базовий компонент для роботи з таблицями на клієнтській стороні.

Рис. 3.8. Ієрархія та звязки між допоміжними класами

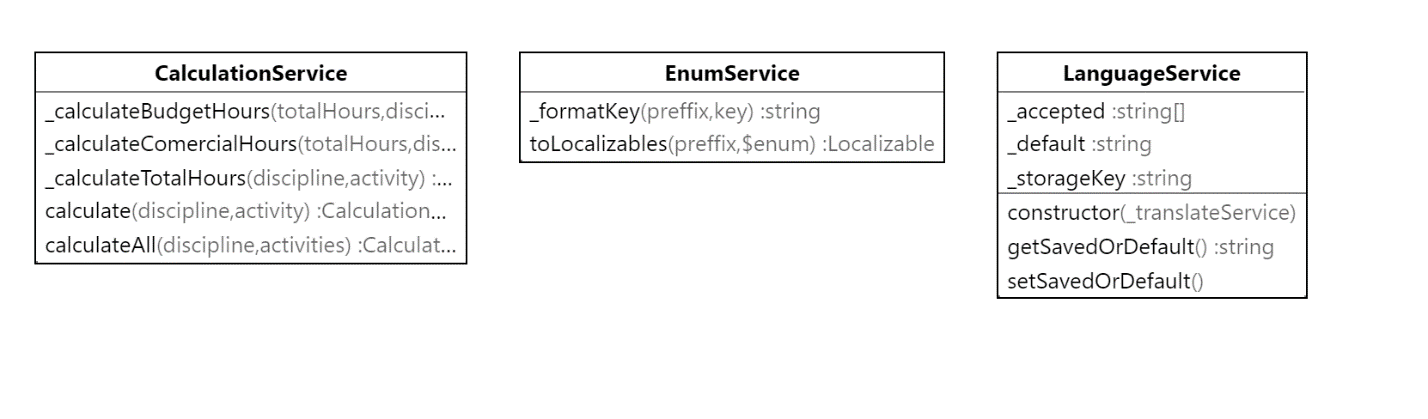


Рис. 3.9. Ієрархія та звязки між сервісами

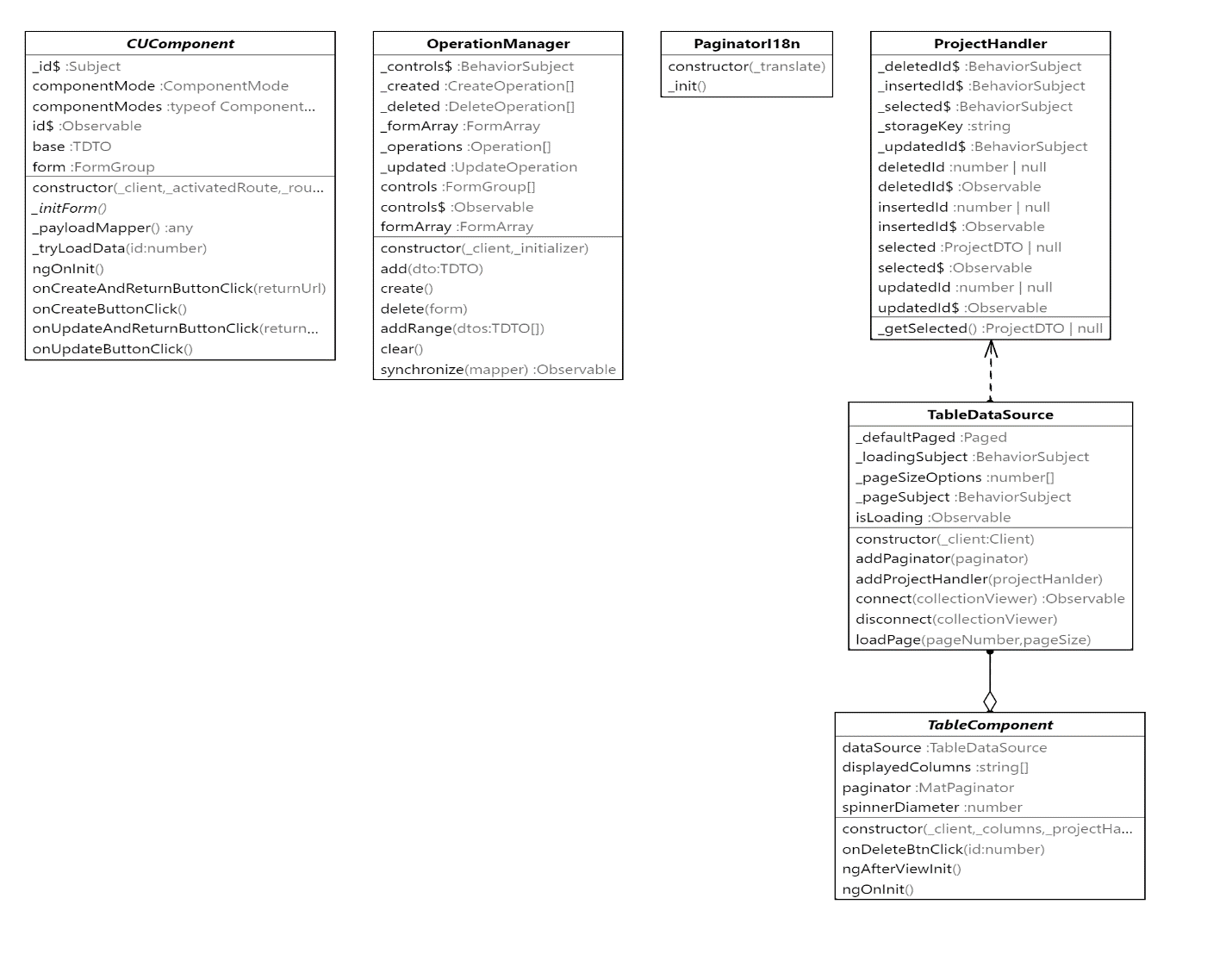
Сервіси були створені для зменшення дубляжу коду в компонентах. Наразі створено 3 сервіса:

1. CalculationService

Сервіс створений для обрахунку розподілу навантаження між викладачами кафедри.

1. EnumService

Створений для спрощення роботи з типом даних enum в TypeScript.

Рис. 3.10. Ієрархія та звязки між сервісами

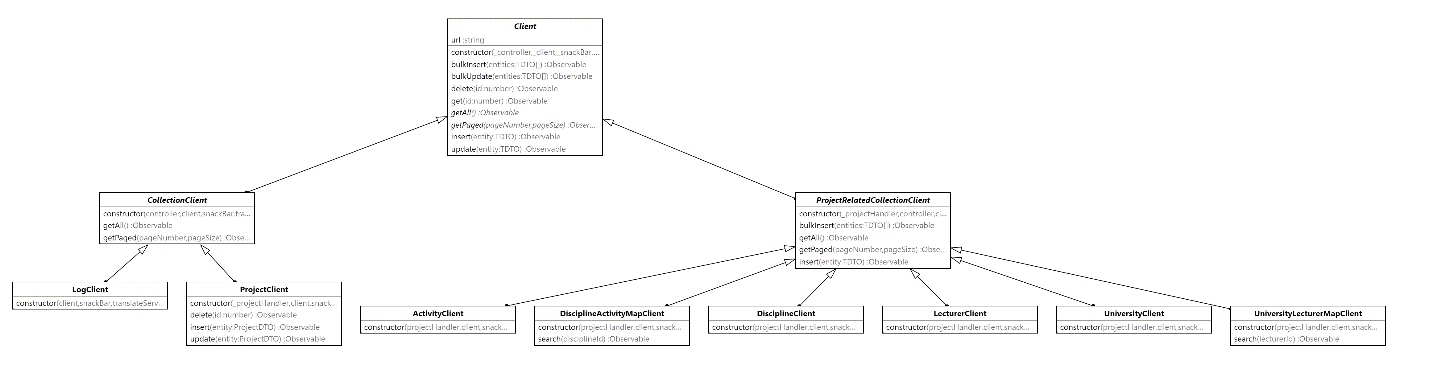


Рис. 3.11. Ієрархія та звязки між клієнтами

Клієнти це окремий вид сервісів, які служать для комунікації клієнтської частини з серверною. Вони виконують HTTP запити до сервера і конвертують результат. Клієнти використовуються в сервісах та компонентах для спрощення комунікації з сервером.

## **3.7 Результати роботи програми**

Результати роботи програми показані на рисунках нижче.

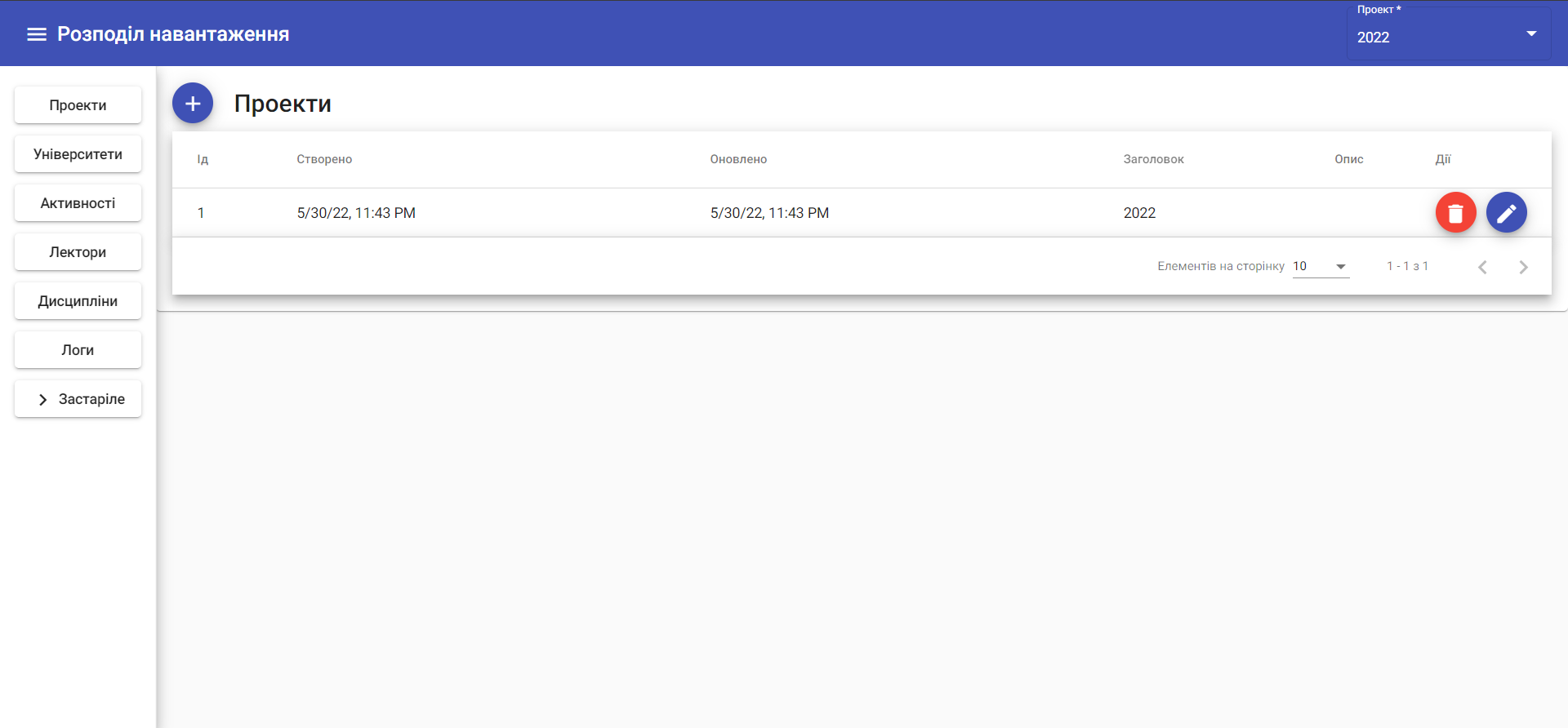


Рис. 3.12. Таблиця з проектами

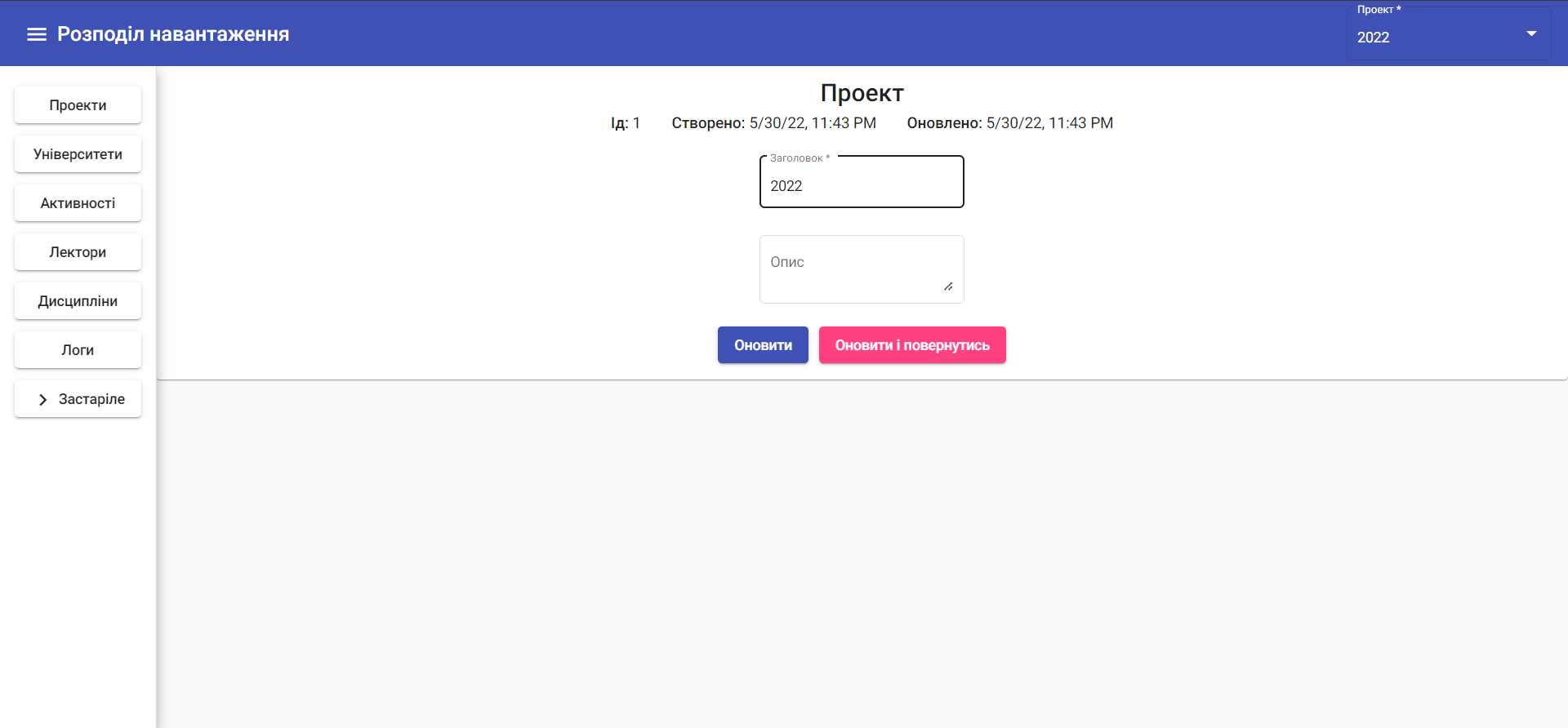


Рис. 3.13. Зміна проекту

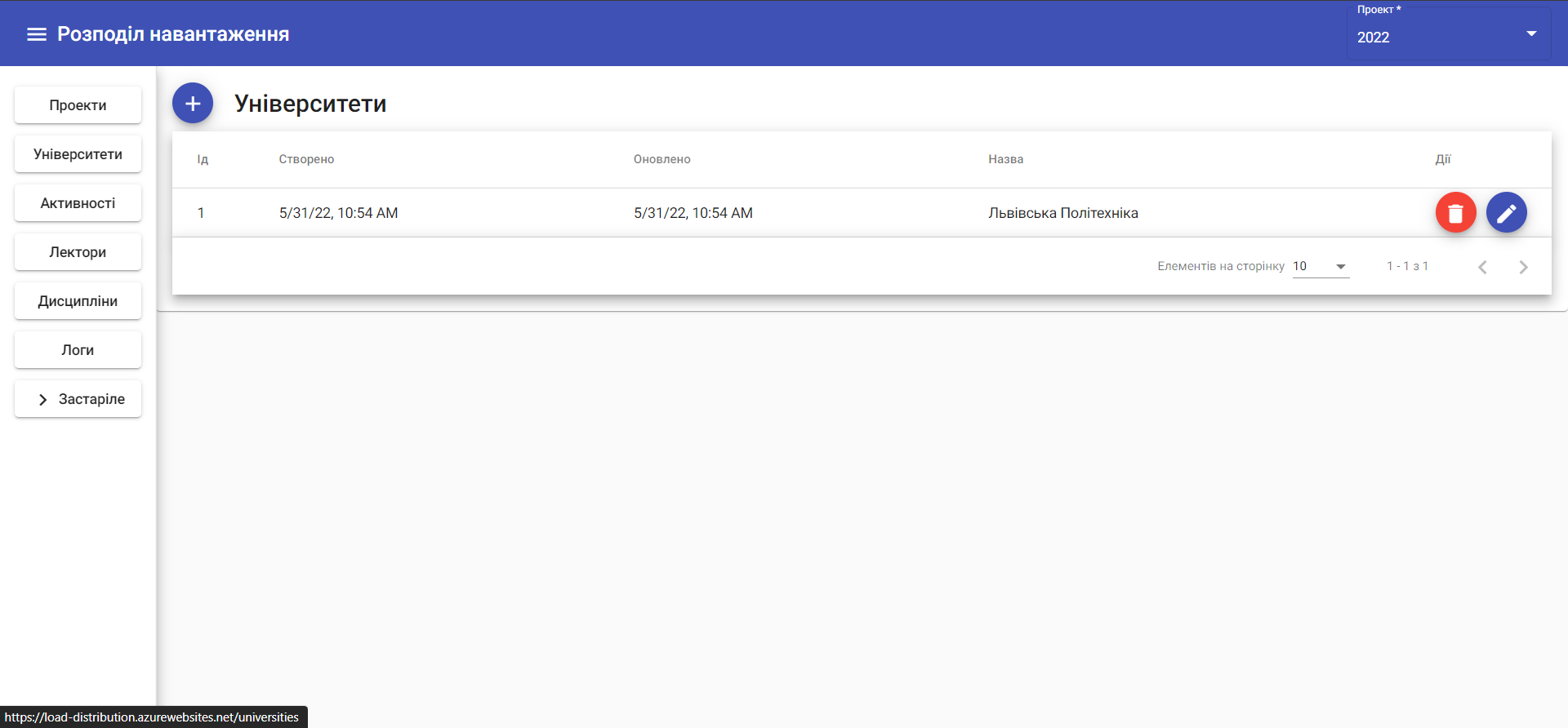


Рис. 3.13. Таблиця з університетами

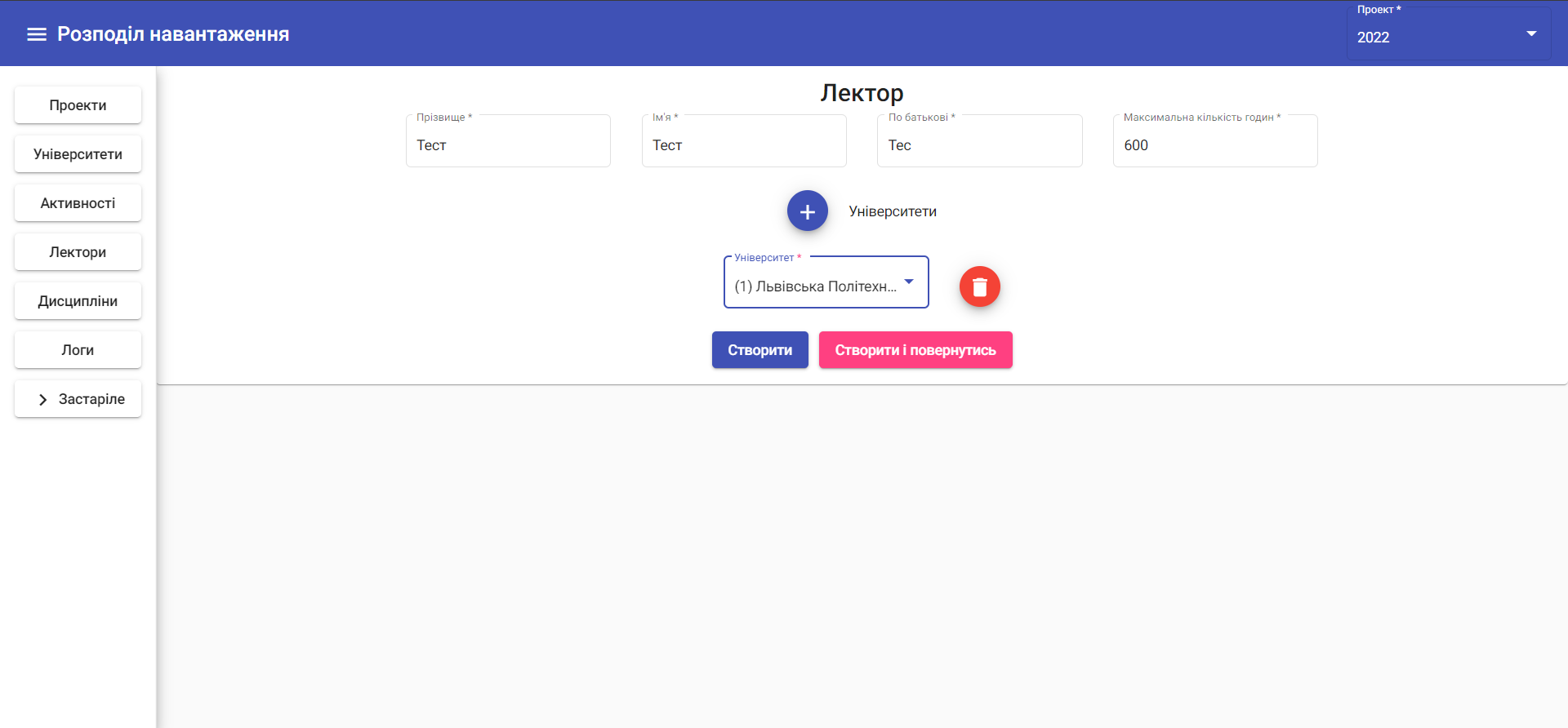


Рис. 3.15. Створення викладача

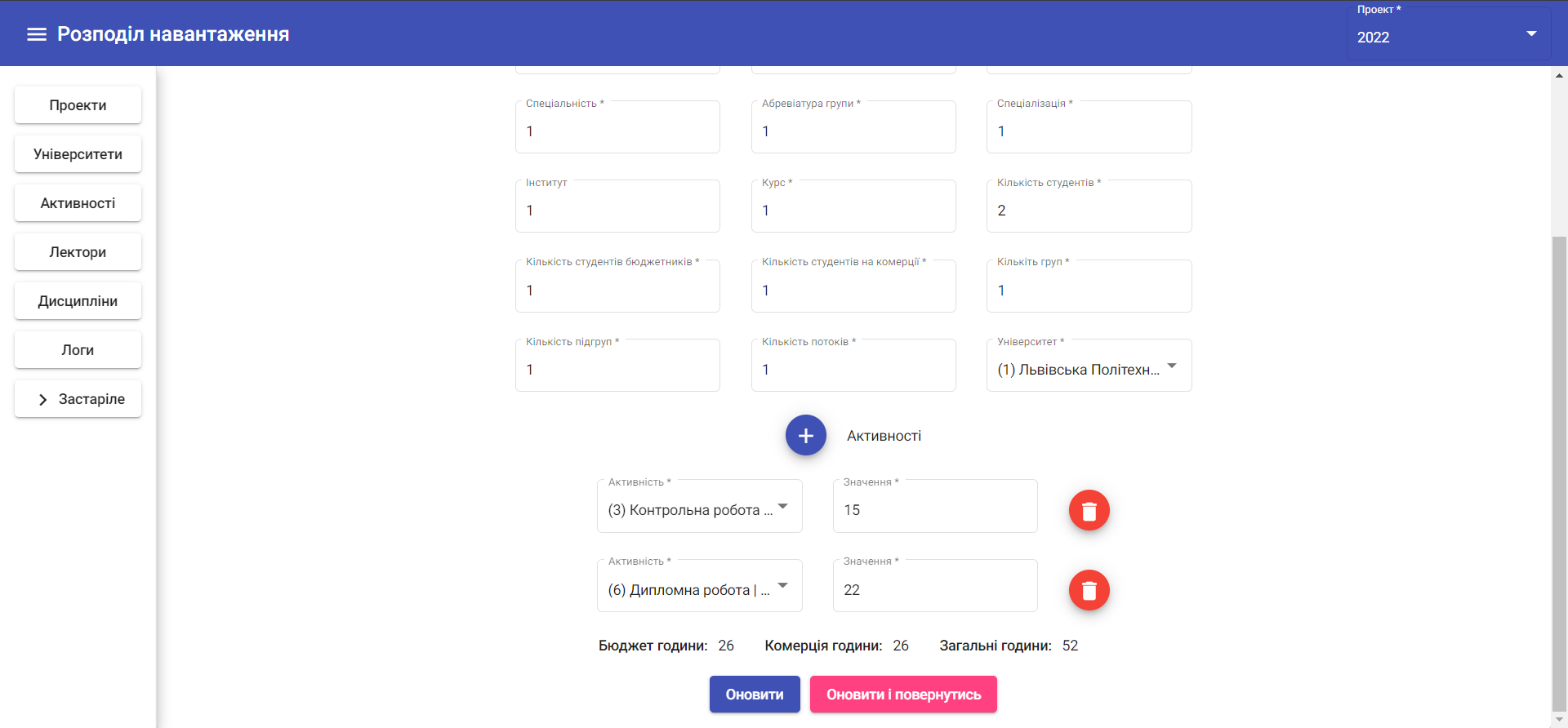


Рис. 3.16. Створення дисципліни та базові обрахунки

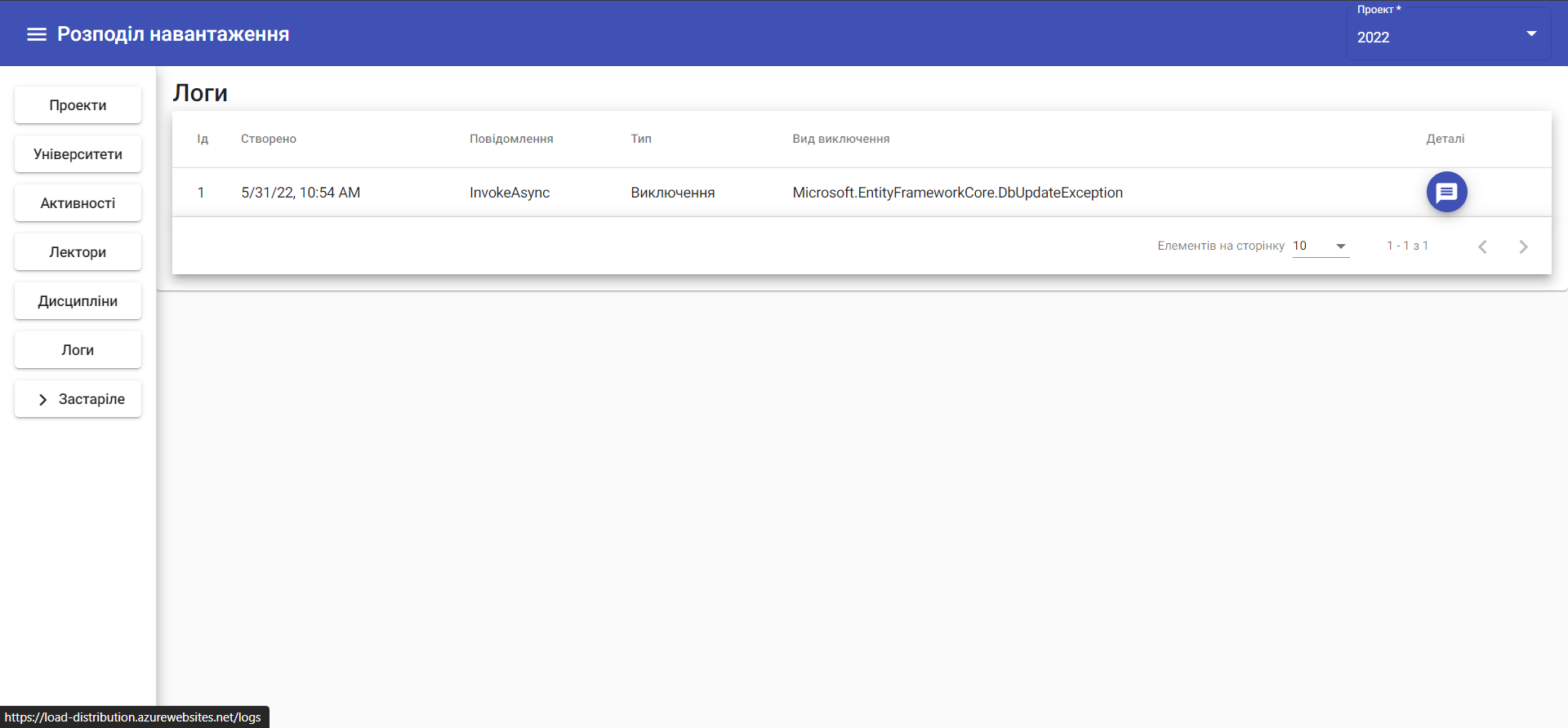


Рис. 3.17. Таблиця з логами

## **3.8 Висновоки**

Під час реалізації проекту було побудовано гнучку, функціональну та швидкодіючу систему. Використовуючи наслідування та поліморфізм створено базові класи, які мінімізують час на розробку та тестування. Спроектовано базу даних, яка була нормалізована та оптимізована. Внаслідок нормалізації, швидкість доступу до даних є максимальною. Програмний продукт показав хороші результати при тестуванні швидкодії. А саме: час запиту до API – 54 мс, час запиту до бази даних – 4 мс, час завантаження веб сторінки – 48 мс.