Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

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

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

Магістерська кваліфікаційна робота

другий (магістерський)
(рівень вищої освіти)

на тему: Розроблення інформаційної системи "Менеджер компанії"

Виконав: студент 6 курсу групи КН-61м
спеціальності
122 "Комп'ютерні науки"
(шифр і назва напряму підготовки, спеціальності)

Антоник А. В.
(прізвище та ініціали)

Керівник Дендюк М. В.
(прізвище та ініціали)

Рецензент Сторожук О. А.
(прізвище та ініціали)

Львів – 2024

Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

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

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ Борецька І.Б.
"05" січня 2024 року

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

Антоник Андрій Вікторович
(прізвище, ім'я, по батькові)

1. Тема роботи Розроблення інформаційної системи "Менеджер компанії"

керівник роботи *Дендюк Михайло Володимирович, кандидат технічних наук, доцент*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від "13" лютого 2023 року № С-49
2. Термін подання студентом роботи "05" січня 2024 року
3. Вихідні дані до роботи Розробити веб-орієнтовану систему для спрощення, уніфікації та автоматизації процесів менеджменту компанії. Реалізувати всю необхідну інфраструктуру даного сервісу за допомогою стеку технологій .NET та Angular.

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
Стан проблемної області
Інформаційне забезпечення
Математичне забезпечення
Програмне та технічне забезпечення
Висновки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Підготовка матеріалу до доповіді.

6. Дата видачі завдання "15" лютого 2023 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1.	Огляд літератури згідно досліджуваної теми. Збір необхідних матеріалів.	10.9.23-22.9.23	вик
2.	Постановка задачі і її формалізація	23.9.23-29.9.23	вик
3.	Виконання вхідного етапу технологіями .Net та Angular (Розроблення архітектури).	30.9.23-13.10.23	вик
4.	Розроблення шаблону застосунку. Зовнішній вигляд.	14.10.23-21.10.23	вик
5.	Реалізація основного функціоналу. Тестування застосунку.	22.10.23-24.11.23	вик
6.	Оформлення записки до дипломного проекту.	24.11.23-29.11.23	вик
7.	Здача пояснювальної записки на рецензування.	5.12.23	вик
8.	Підготовка доповіді.	10.12.23-1.01.24	вик

Студент _____ **Антоник А. В.**
(підпис) (прізвище та ініціали)

Керівник роботи _____ **Дендюк М. В.**
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дипломна робота містить 69 сторінки пояснювальної записки, 73 рисунків, 1 додаток, 9 джерел.

В даній роботі, реалізовано веб-орієнтовану, інформаційну систему на базі стеку технологій .Net та Angular. Реалізовано інфраструктуру для менеджменту компанії, автоматизації робочих процесів.

Ключові слова:.*Net, Asp.Net 6, Entity Framework, Identity, JWT, MSSQL, Angular, REST API, SPA.*

ABSTRACT

The thesis contains 69 pages of explanatory note, 73 figures, 1 appendix, 9 used literary sources.

In this work, a web-oriented information system based on the .Net and Angular technology stack is implemented. The infrastructure for company management, automation of work processes has been implemented.

Keywords:.Net, Asp.Net 6, Entity Framework, Identity, JWT, MSSQL, Angular, REST API, SPA.

ТЕХНІЧНЕ ЗАВДАННЯ

Необхідно реалізувати інфраструктуру сервісу, що дозволить автоматизувати процеси менеджменту компанії:

1. Реалізувати архітектурну концепцію “Клієнт-Сервер”;
2. Спроектувати базу данних;
3. Створити модуль Domain, що міститиме всі моделі даних даної системи;
4. Реалізувати модуль Repository, що міститиме методи доступу до даних з бази даних;
5. Розробити модуль Infrastructure, що містить логіку опрацювання даних;
6. Реалізувати REST API;
7. Створити автентифікацію на основі JWT;
8. Реалізувати Клієнтську частину на основі принципів роботи SPA, за допомогою технології Angular;

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ 7
ВСТУП 8
РОЗДІЛ 1. ОГЛЯД ПРОБЛЕМНОЇ ОБЛАСТІ 10
1.1 1. Актуальність розробки серверних застосункув за допомогою технології ASP.NET. 10
1.1.2. Приклади застосування ASP.NET. 11
1.1.3. Недоліки та переваги застосування ASP.NET 12
1.2.1. Актуальність технології Angular та принципу SPA. 13
1.2.2. Основні переваги технології Angular 13
Висновки до розділу 14
РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ТА МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ 16
Висновки до розділу 14
РОЗДІЛ 3. ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ 22
3.1. Проектування бази даних. 22
3.2. Архітектура системи. 29
3.3. Серверна частина. 29
3.4. Domain. 30
3.5. Repository. 32
3.6. Infrastructure. 33
3.7. API. 33
3.8. Клієнтська частина. 38
Висновки до розділу 38
РОЗДІЛ 4. РОЗРОБЛЕННЯ СТАРТАП-ПРОЕКТУ 58
ВИСНОВКИ 61
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 62
ДОДАТКИ 63

Перелік скорочень та умовних позначень

API - Інтерфейс програмування додатків (Application Programming Interface) - програмне забезпечення, яке дозволяє іншому програмному забезпеченню взаємодіяти з ним.

UI - Графічний інтерфейс користувача (User Interface) - спосіб взаємодії користувача з програмою або пристроєм через графічні елементи, такі як кнопки та меню.

ВСТУП

Основна мета управління полягає в тому, щоб організувати спосіб діяльності суб'єкта, що залежить від продукту управління. Чим ефективніше управління продуктом, тим краще організація (компанія) і її результати.

Менеджмент - як галузь науки почав розвиватися в результаті необхідності зрозуміти, чому певні організації досягли успіху або зазнали невдачі. Вчені в галузі менеджменту провели експерименти, які привели до практики експериментів з управління, заснованих на методі проб і помилок. Ці експерименти відповідають на питання, наскільки успішною є організація. Відповідь на це запитання дозволить вам знайти більш практичне запитання: «Яка роль менеджера в забезпеченні успіху організації?».

Менеджер відіграє життєво важливу роль у забезпеченні тріумфу організації. Розглянуто причини існування менеджменту в організаціях, які є основою світу менеджменту. Розуміння основних принципів сучасного менеджменту є складним і різноманітним завданням, яке піддається ретельному розгляду експертів у різних галузях. Деякі люди стверджують, що природна схильність до управлінської діяльності є важливою, тоді як інші порівнюють менеджмент із такими предметами, як фізика чи біологія, стверджуючи, що для того, щоб стати успішним менеджером, перш за все потрібно постійно навчатися та особистісне зростання. Згідно з даними, лише невелика кількість людей, які володіють організаторськими здібностями, володіють необхідними знаннями, щоб плекати свій талант і використовувати його для розвитку суспільства.

Люди, які спеціалізуються на менеджменті, повинні володіти навичками планування, організації, управління та контролю за роботою, яку виконують співробітники організації для досягнення конкретної мети. Основним обов’язком керівництва є встановлення культури, яка сприяє інноваціям і творчості, створюючи атмосферу, у якій співробітників заохочують мислити нестандартно та висувати нові ідеї.

Об’єктом дослідження є розроблення веб-орієнтованої, інформаційної системи для ведення менеджменту компанії.

Предметом дослідження є розробка веб-сервісу з використанням стеку технологій .Net для створення серверної частини та Angular для проектування SPA-застосунку як клієнтської частини.

Метою роботи є створення сервісу що відповідає темі роботи, описаній вище, з використанням .Net для створення backend частини, та Angular для реалізації частини frontend.

Для досягнення поставленої мети необхідно виконати наступні **завдання**:

1. Реалізувати архітектурну концепцію “Клієнт-Сервер”;

2. Спроектувати базу данних;
3. Створити модуль Domain, що міститиме всі моделі даних даної системи;
4. Реалізувати модуль Repository, що міститиме методи доступу до даних з бази даних;
5. Розробити модуль Infrastructure, що містить логіку опрацювання даних;
6. Реалізувати REST API;
7. Створити автентифікацію на основі JWT;
8. Реалізувати Клієнтську частину на основі принципів роботи SPA, за допомогою технології Angular.

Практичне значення роботи полягає у легкого у користуванні, зручного, та технологічног сервісу для менеджменту крмпанії, що дасть можливість спростити, уніфікувати та автоматизувати процеси, які пов’язані із менеджментом підприємства.

Результати роботи апробовано на п’ятій науково-практичній конференції студентів, аспірантів та молодих вчених «Комп’ютерне моделювання та інформаційні технології» (Львів, 20-22 жовтня 2022 р.):

1. Андрій Антоник. Розробка інформаційної системи "Менеджер компанії" / А. Антоник, М.Дендюк // Комп’ютерне моделювання та інформаційні технології: матеріали п’ятої науково-практичної конференції студентів, аспірантів та молодих вчених (Львів, 19-21 жовтня 2023 р.). – Львів: ННІ КНІТ НЛТУ України, 2023. – С. 146-149.

РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

1.1. Огляд проблемної області.

1.1.1. Актуальність розробки серверних застосункув за допомогою технології ASP.NET.

ASP.NET — це серверна веб-інфраструктура з відкритим вихідним кодом, створена корпорацією Майкрософт, яка працює на Windows і була випущена на початку 2000-х років.

ASP.NET надає багато функцій, включаючи повне керування, безпеку, автентифікацію та авторизацію та інтеграцію бази даних. З його допомогою можна створювати невеликі сайти і великі корпоративні програми. Версія ASP.NET Core — це перероблена кросплатформна версія фреймворку, яка також підтримує розробку веб-додатків у Linux і macOS.

ASP.NET побудовано на загальномовному середовищі виконання (CLR), механізмі виконання .NET Framework. Це дозволяє розробникам писати код, використовуючи підтримувані мови .NET - C# і VB.NET. CLR надає такі функції, як керування пам’яттю, безпека та обробка винятків, щоб забезпечити надійну та ефективну роботу програм ASP.NET.

Операюсичь на статистику від сервісу Dou.ua можна з впевненістю сказати що актуальність стеку технологій .Net є на дуже високому рівні.

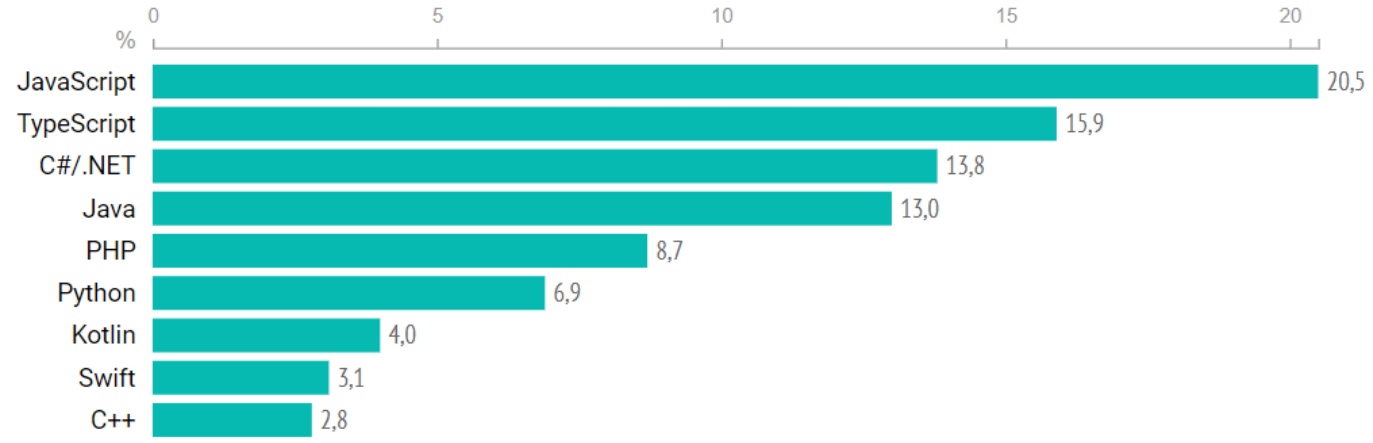


Рисунок 1.1.1. Графік актуальності стеку технологій .Net

Наразі дані показують, що все більше компаній віддають перевагу технології .NET, що є чітким свідченням того, що індустрія програмного забезпечення швидко рухається до прийняття культури розробки програмного забезпечення з відкритим кодом. Згідно з опитуванням розробників Stack Overflow 2021, ASP.NET Core є одним із найпопулярніших веб-фреймворків.

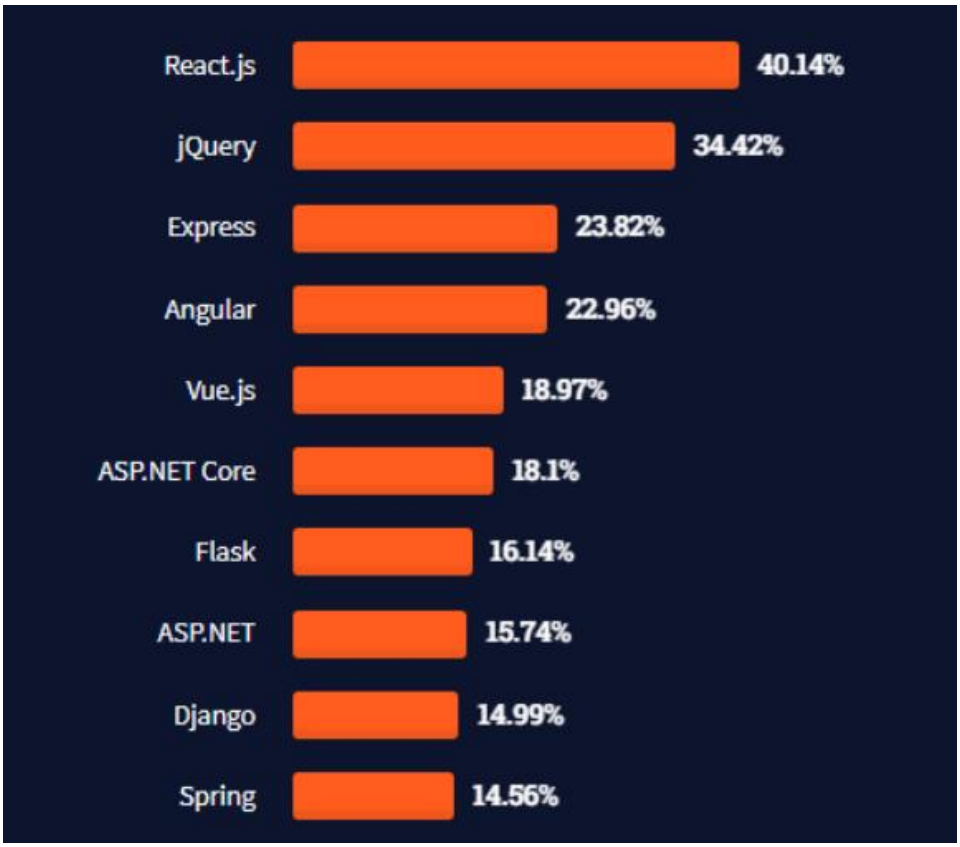


Рисунок 1.1.2. Графік актуальності технології ASP.NET Core серед веб-технологій

1.1.2. Приклади застосування ASP.NET.

ASP.NET — це потужне та універсальне середовище розробки, яке дозволяє професіоналам створювати універсальні, масштабовані та безпечні веб-додатки. ASP.NET залишається популярним вибором для створення веб-рішень у різноманітних галузях і варіантах використання завдяки широкому набору інструментів, потужній моделі програмування та можливостям інтеграції.

Найчастіше ця технологія використовується в наступних сферах:

✓ **Корпоративні веб-програми:** багато корпоративних програм і порталів розроблено за допомогою ASP.NET. Ця структура надає інструменти для створення складних систем із контролем доступу, обробкою даних та інтеграцією з іншими корпоративними системами.

✓ **Онлайн-продажі:** багато онлайн-магазинів і платформ електронної комерції використовують ASP.NET для створення динамічних і безпечних онлайн-магазинів.

✓ Сайти: **ASP.NET можна використовувати для створення статичних і динамічних сайтів, включаючи блоги. Багато систем керування вмістом (CMS) на основі ASP.NET пропонують зручні інструменти керування вмістом.**

✓ **Внутрішні інструменти та системи:** багато компаній використовують ASP.NET для створення внутрішніх програм, таких як системи управління персоналом, системи відстеження завдань та інші корпоративні інструменти.

✓ **Веб-служби:** ASP.NET також широко використовується для створення веб-служб, які надають дані та функціональність іншим програмам через Інтернет.

✓ **Освітні та державні установи:** ASP.NET зазвичай використовується освітніми установами та державними організаціями для розробки програм, систем управління інформацією та інших рішень.

✓ **Охорона здоров'я:** у сфері охорони здоров'я ASP.NET можна використовувати для створення веб-додатків для керування медичними даними, планування прийомів, управління лікарнями та інших завдань.

1.1.3. Недоліки та переваги застосування ASP.NET.

Деякі з переваг технології ASP.NET варто виділити.

1) **Гнучкість мови програмування**

Основними перевагами ASP.NET є його гнучкість і об'єктно-орієнтований характер. Назва ASP.NET походить від старішої технології Microsoft ASP. Однак нові .NET Framework і CLR забезпечують бездоганний інтерфейс для інших мов програмування, таких як Visual Basic.NET і C#. Це означає, що ви можете змінити мову програмування проекту в середині проекту без необхідності переписувати все. Це також означає, що кілька розробників можуть працювати над одним проектом, використовуючи інші мови, такі як Visual Basic.NET і C#.

2) **зручна структура**

Ще однією перевагою є те, що бібліотека ASP.NET базується на завданнях. Бібліотеки організовані в успадковані класи, пов'язані з конкретними завданнями (наприклад, XML або редагування зображень), що економить час розробників на багатьох поширених завданнях розробки.

3) **відкритий код**

Технології з відкритим кодом, такі як ASP.NET, корисні з двох важливих причин. По-перше, використовувати технологію з відкритим кодом можна безкоштовно. Це допомагає зменшити витрати на розробку, які можуть бути досить дорогими. Крім того, оскільки технологія безкоштовна, більше людей можуть її використовувати та покращувати.

Існує велика спільнота підтримки, включаючи команду Microsoft, що розробляє ASP.NET, яка може надати вказівки, документацію та нові ідеї. Крім того, люди з усього світу можуть зробити свій внесок у технології з відкритим кодом, такі як ASP.NET.

4) **Підтримка хмарних обчислень**

ASP.NET легко інтегрується з платформою хмарних обчислень Microsoft Azure, що дозволяє розгортати та масштабувати веб-додатки в хмарі.

5) **Масштабованість та продуктивність проекту**

ASP.NET є високопродуктивним і масштабованим, що дозволяє створювати невеликі сайти та великі корпоративні програми, які можуть обробляти велику кількість запитів.

Основними недоліками ASP.NET і Visual Studio є їх висока вартість і ресурсомістке обслуговування. Більшість програм ASP.NET працюють на IIS. Крім того, ASP.NET використовує більше ресурсів веб-сервера, ніж PHP або інші мови, тому потрібен кращий сервер або більше серверів. Нарешті, Windows і IIS записують історію помилок і вразливостей у програмах, які в минулому були вразливими до експлуатації.

6) **Хостинг сайту ASP.NET**

Щоб розмістити сайт ASP.NET, вам потрібно спеціально розмістити проект ASP.NET на базі операційної системи Windows, звичайний хостинг на Linux не працюватиме. Отже, якщо ви шукаєте послугу хостингу для розміщення, вам слід перевірити, чи підтримує вона ASP.NET перед покупкою.

1.2.1. Актуальність технології Angular та принципу SPA.

Angular є одним з найбільш потужних та розширюваних фреймворків для веб-розробки. Він був створений компанією Google і використовується багатьма великими компаніями, включаючи Microsoft, IBM та Deutsche Bank. Angular дозволяє розробникам створювати складні веб-додатки з високою продуктивністю та зручним інтерфейсом. Він також володіє потужною системою маршрутизації та безліччю інструментів для тестування. Іншими словами, технологію Angular визначають як - потужний JavaScript фреймворк із відкритим вихідним кодом для розробки односторінкових застосунків (SPA).

1.2.2. Основні переваги технології Angular.

• Двостороння прив'язка даних;

• При використанні MVC логіка відокремлена від зображень, і розробникам це подобається;

• Використовуйте DOM для простих операцій;

• Компоненти можна використовувати кілька разів.

Повернення податку:

• Деякі функції важко налаштувати;

• Можуть існувати складні сценарії.

Характеристика рогів:

• Зв'язування даних для синхронного оновлення представлень і моделей;

• Додати інтерактивні команди;

• Реалізація HTTP-запитів і залежностей передачі даних між компонентами;

• Шаблони для створення елементів UI;

• Дозволяє створювати прогресивні веб-програми з сучасними веб-можливостями;

• Механізм виявлення змін, який може оновлювати певні частини програми без перезавантаження всіх компонентів за допомогою механізмів за замовчуванням і OnPush.

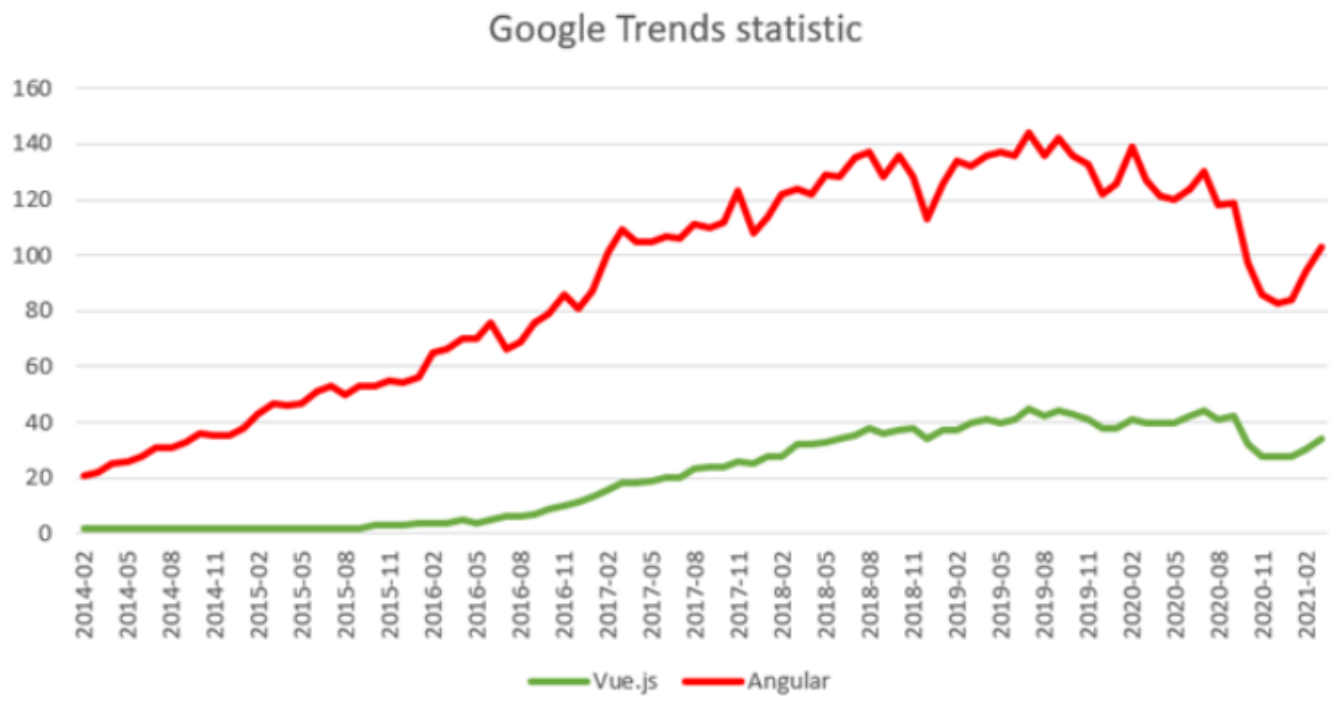


Рисунок 1.2.1. Графік росповсюдженості технології Angular від сервісу Google Trends

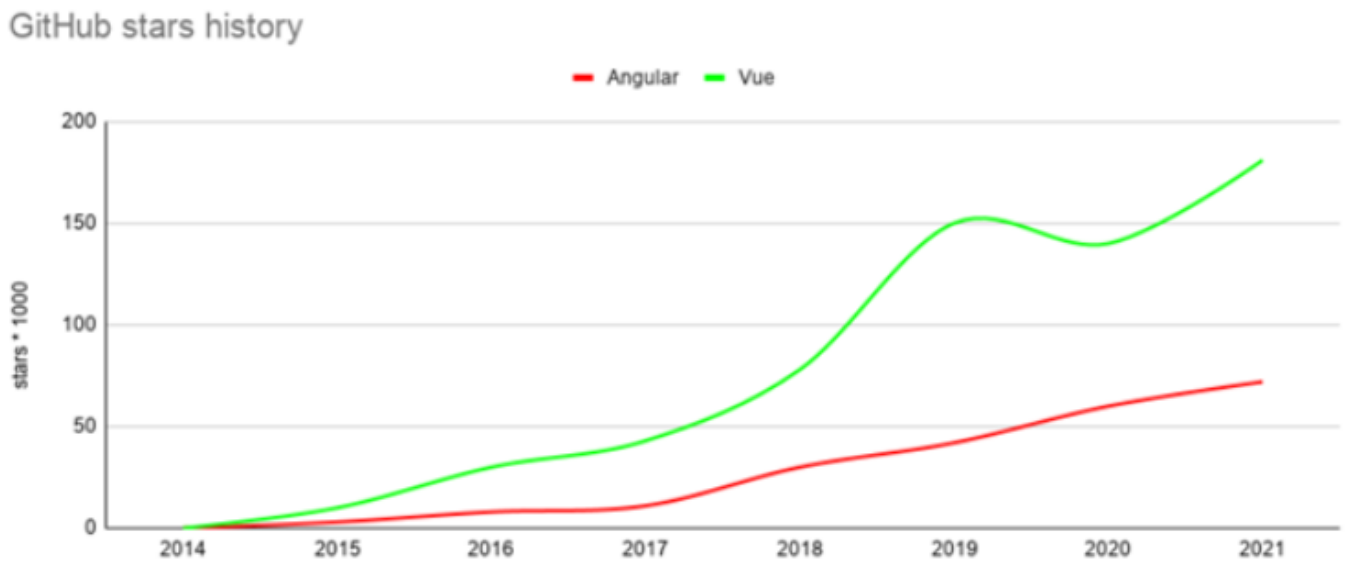


Рисунок 1.2.2. Графік кількості Angular-репозиторіїв створених за допомогою сервісу GitHub

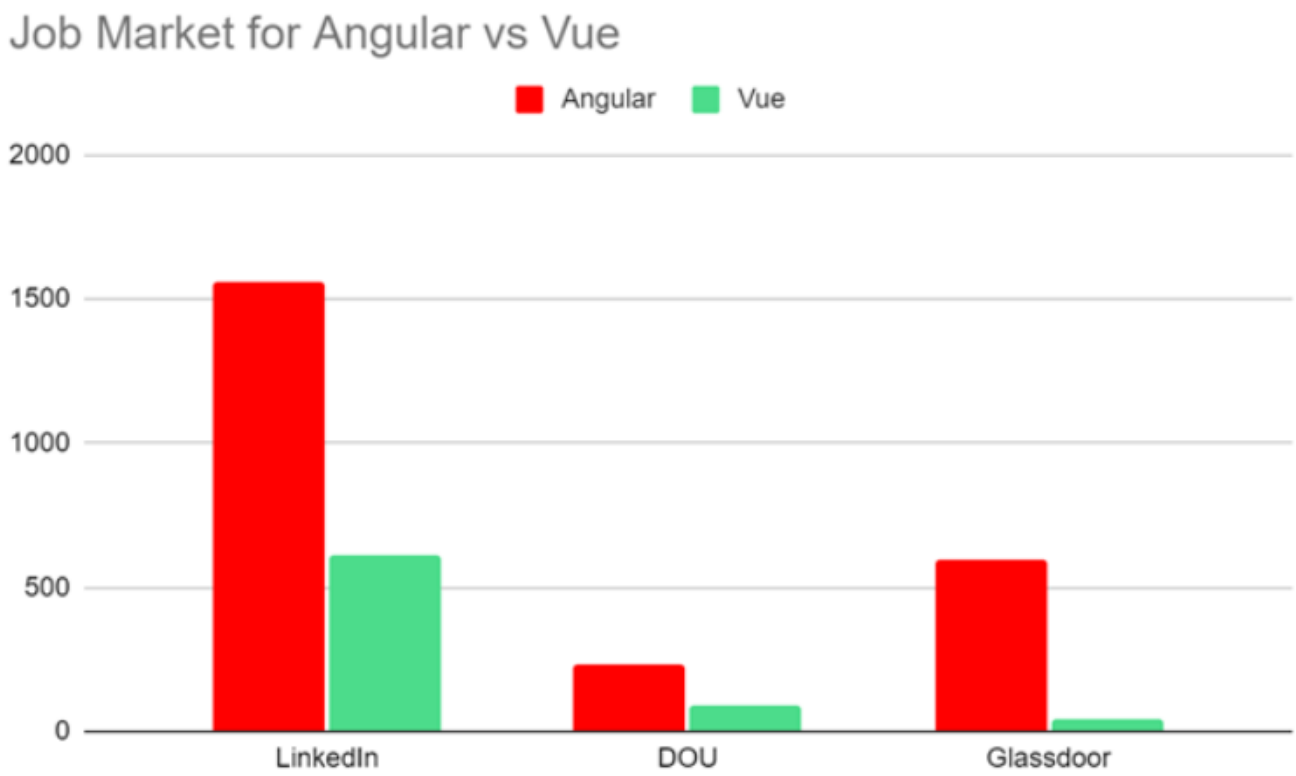


Рисунок 1.2.3. Графік порівняння популярності технологій Angular та Vue, опираючись на дані сервісів LinkedIn, DOU, Glassdoor.

Висновки до розділу

Операюсичь на дані різноманітних ресурсів та сервісів, можна прийти до висновку, що розробка Клієнт-серверної системи на основі стеку технологій .NET та Angular є релевантним рішенням. Даний стек технологій показує велику степінь актуальності на сучасному ринку IT.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ТА МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

.Net платформа та Angular

.NET — це платформа від Microsoft, яка дозволяє створювати програмні додатки. Перша версія .NET Framework була випущена в 2002 році. Вважається, що .NET Framework був створений як заміна платформі Java від Sun Microsystems. Основна відмінність полягає в тому, що .NET Framework офіційно розроблено для роботи з операційними системами сімейства Microsoft Windows. З тих пір платформа пройшла довгий шлях, починаючи з версії 1.0, і сьогодні, незважаючи на появу платформи нового покоління (.NET Core), вона все ще досить популярна: існує велика кількість програмних продуктів, бібліотек і фреймворків. написаний і розроблений під .NET Framework.

У 2016 році, крім .NET Framework, також була випущена модульна платформа .NET Core, сумісна з різними операційними системами. Іншими словами, це кросплатформенність. Кросплатформна природа .NET Core відкриває багато нових сценаріїв і можливостей для його програм.

Багато людей думають, що мова C# і платформа .NET - це одне і те ж. Звичайно, це не так. Безсумнівно, вони розвивалися, озираючись один на одного, але вони не є строго взаємозалежними. Наприклад, крім офіційно випущеної реалізації .NET, існують такі альтернативи, як Mono, .NET Compact Framework, .NET Micro Framework тощо. Ми можемо використовувати C# на всіх цих платформах, але до певної міри. .NET, з іншого боку, сумісний не лише з C#, але й з іншими мовами: F#, VB.NET і навіть C++.

Розробники, які знають різні мови, можуть об'єднатися для написання спільних програмних продуктів для конкретної платформи .NET. Елементи цього продукту, написані різними мовами, зможуть без проблем спілкуватися один з одним. До речі, це пояснює, чому спільнота .NET така велика й різноманітна: вона об’єднує програмістів, які пишуть різними мовами.

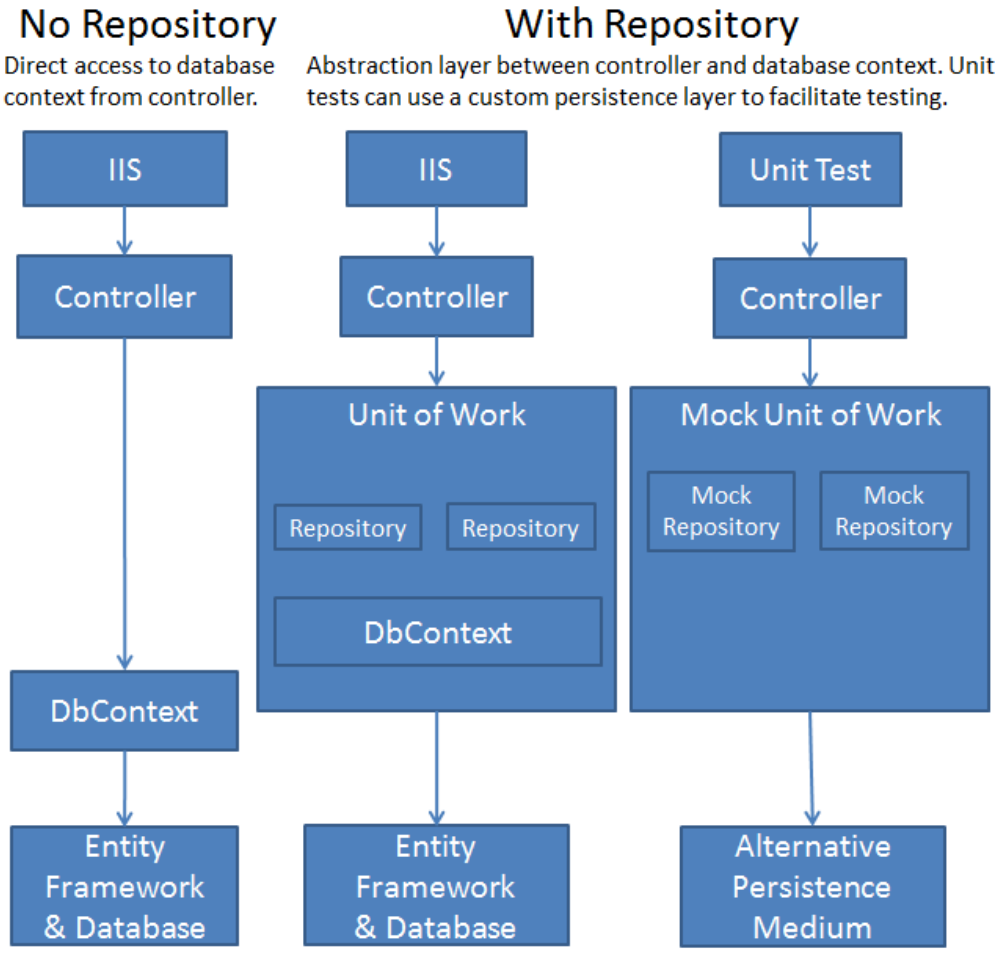


Рисунок 2.1. «Загальна діаграма моделей даних»

Entity Framework Core (EF Core) — це об'єктно-орієнтована, легка та розширювана технологія доступу до даних, запущена Microsoft. EF Core — це інструмент ORM (Object Relational Mapping — зіставлення даних із реальними об'єктами). Тобто EF Core дозволяє використовувати базу даних, але з більш високим рівнем абстракції: EF Core дозволяє абстрагуватися від самої бази даних і її таблиць і працювати з даними незалежно від типу зберігання. Якщо фізично ми працюємо з використанням таблиць, індексів, первинних ключів і зовнішніх ключів, але на концептуальному рівні, наданому Entity Framework, ми вже працюємо з об'єктами.

ASP.NET Identity — це система автентифікації та авторизації, вбудована в ASP.NET. Система дозволяє користувачам створювати облікові записи, перевіряти, керувати обліковими записами або входити на веб-сайт, використовуючи облікові записи від зовнішніх постачальників, таких як Facebook, Google, Microsoft, Twitter та інші.

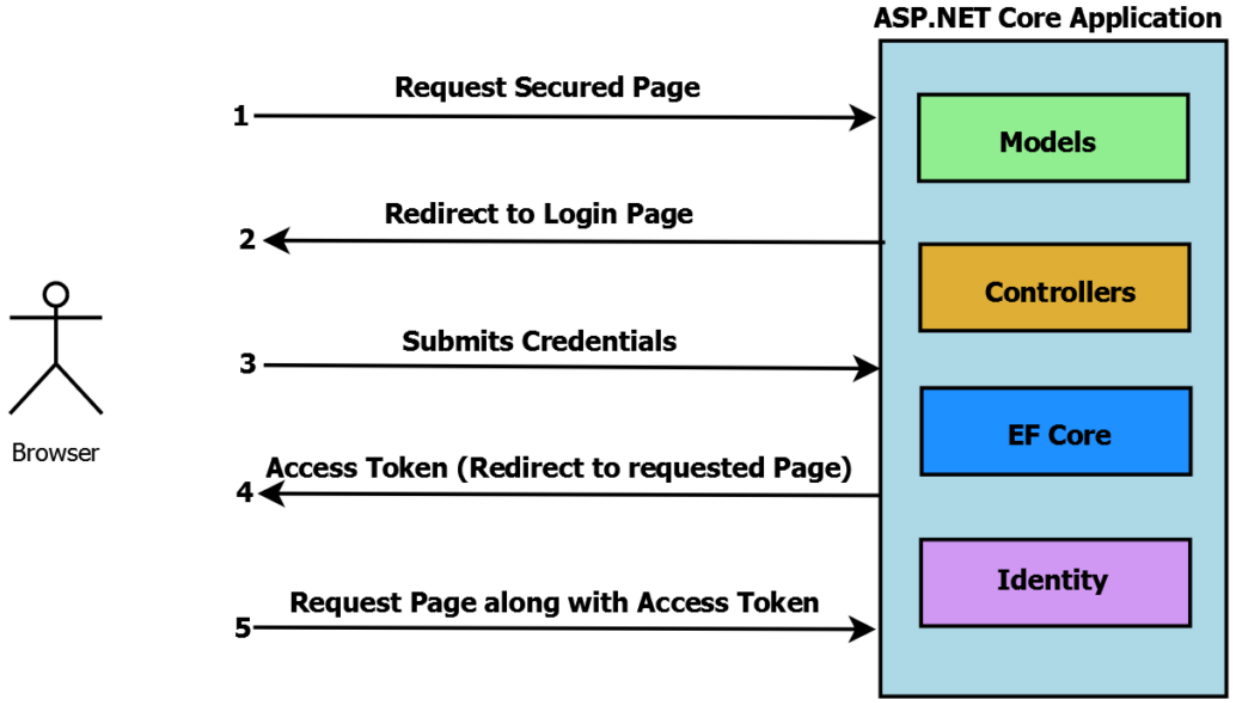


Рисунок 2.2. «Загальна діаграма моделей даних»

Автентифікація за допомогою JWT.

Загальні методи авторизації та автентифікації в ASP.NET Core Web API дещо відрізняються від методів у MVC. Зокрема, механізм авторизації Web API значною мірою покладається на токени JWT.

JWT (або JSON Web Token) — це веб-стандарт, який визначає метод передачі даних користувача в зашифрованому форматі JSON.

Токен JWT складається з трьох частин:

1) **Заголовок** - об'єкт JSON, що містить інформацію про тип маркера та його алгоритм шифрування.
2) **Payload** - об'єкт JSON, що містить дані, необхідні для авторизації користувача.
3) **Підпис** – рядок, створений за допомогою пароля, заголовка та корисного навантаження. Цей рядок призначений для перевірки маркер.

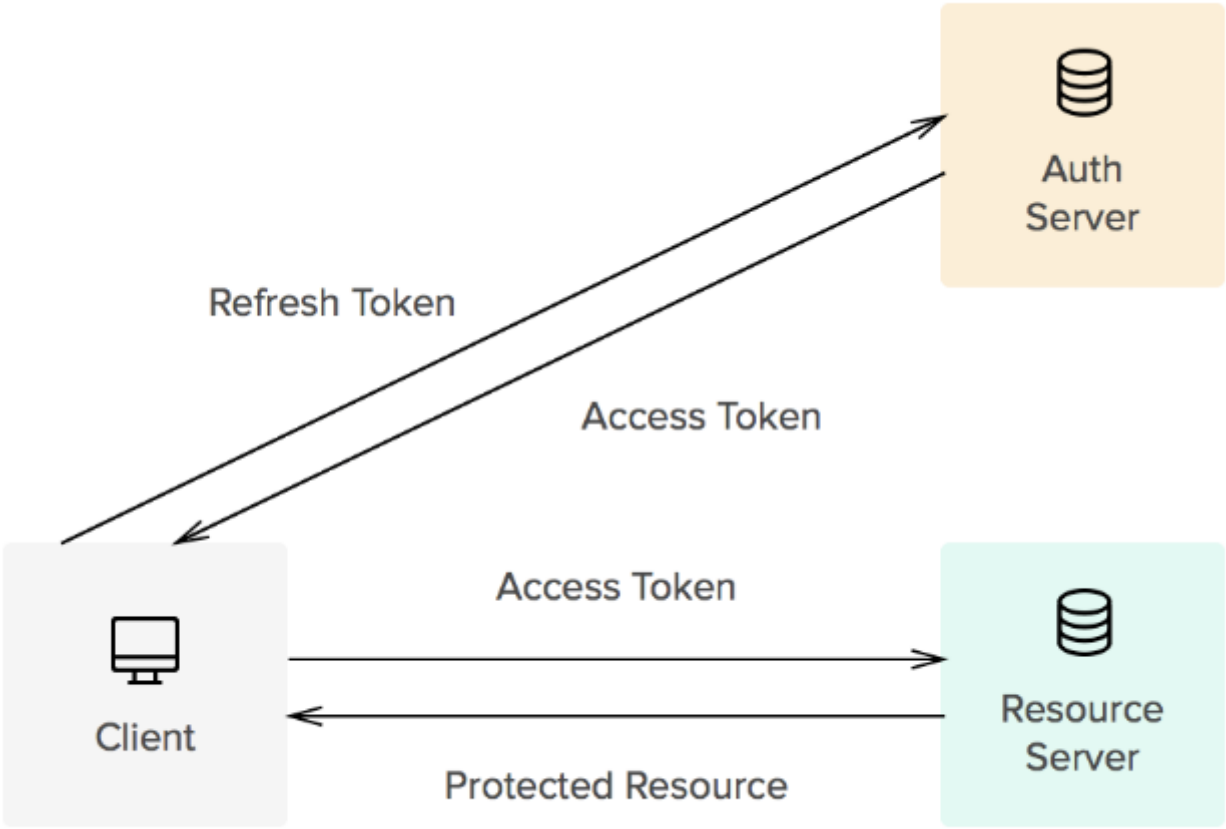


Рисунок 2.3. «Загальна діаграма моделей даних»

Angular — це платформа розробки, побудована на TypeScript.

Платформа Angular включає:

1) Компонентна структура для створення масштабованих веб-додатків
2) Добре інтегрований набір бібліотек, що охоплює широкий спектр функцій, включаючи маршрутизацію, керування формами, зв'язок клієнт-сервер тощо
3) Набір інструментів розробника, які допоможуть вам розробляти, будувати, тестувати й оновлювати свій код

Angular використовує платформу, яка може масштабуватися від одного проекту розробника до програм корпоративного рівня. Найкраще те, що екосистема Angular складається з різноманітної групи з понад 1,7 мільйона розробників, авторів бібліотек і творців контенту.

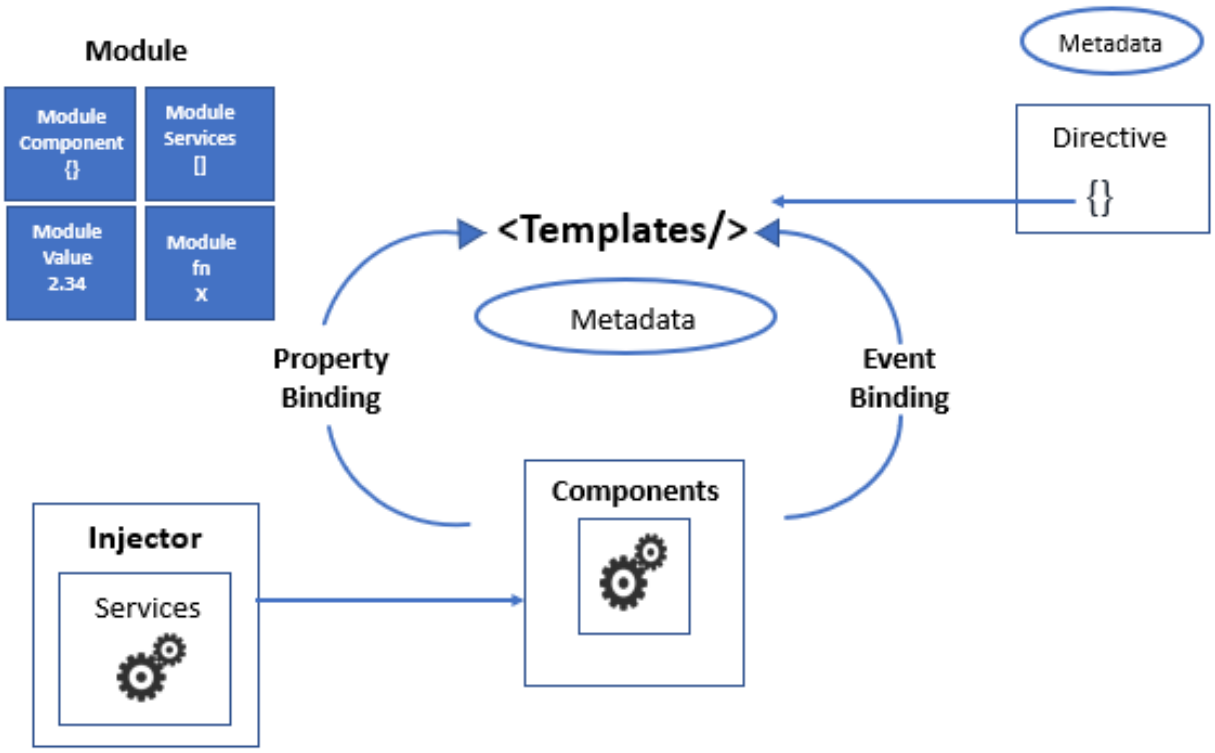


Рисунок 2.4. «Загальна діаграма моделей даних»

REST — це архітектурний стиль написання серверної системи, а не протокол чи стандарт. Принципи даного стилю можуть бути реалізовані різноманітними методами.

Коли клієнтський запит робиться через RESTful API, він передає представлення стану ресурсу запитувачу або кінцевій точці. Ця інформація або представлення надається через HTTP в одному з кількох форматів: JSON (нотація об'єктів Javascript), HTML, XLT, Python, PHP або звичайний текст. JSON є найпопулярнішим форматом файлів, оскільки, незважаючи на свою назву, він не залежить від мови та може бути прочитаний як людьми, так і машинами.

Також слід пам'ятати, що заголовки та параметри також важливі в HTTP-запитах RESTful API, оскільки вони містять важливу інформацію про метадані запиту, авторизацію, уніфіковані ідентифікатори ресурсів (URI), кешування, файли cookie тощо. Ідентифікаційна інформація. Є header запиту та відповіді, які містять власну інформацію про з'єднання HTTP та статус-код

Щоб API вважався RESTful, він має відповідати цим критеріям:

1) Архітектура клієнт-сервер, що складається з клієнтів, серверів і ресурсів із запитами, керованими через HTTP.
2) Зв'язок клієнт-сервер без збереження стану, тобто інформація про клієнта не зберігається між запитами на отримання, і кожен запит є окремим і не пов'язаним.

3) Кешовані дані, які спрощують взаємодію клієнт-сервер.

4) Уніфікований інтерфейс між компонентами, щоб інформація передавалась у стандартній формі. Це вимагає, щоб:

   - запитані ресурси можна ідентифікувати та відокремити від представлень, надісланих клієнту.
   - ресурсами клієнт може маніпулювати через представлення, яке вони отримують, оскільки представлення містить достатньо інформації для цього.
   - самоописові повідомлення, які повертаються клієнту, містять достатньо інформації, щоб описати, як клієнт повинен їх обробити.
   - доступний гіпертекст/гіпермедіа, тобто після доступу до ресурсу клієнт повинен мати можливість використовувати гіперпосилання для пошуку всіх інших доступних на даний момент дій, які він може виконати.

5) Багаторівнева система, яка організовує кожен тип серверів (відповідальних за безпеку, балансування навантаження тощо), передбачала пошук необхідної інформації в ієрархії, невидимі для клієнта.

6) Код на вимогу (необов'язково): можливість надсилати виконуваний код із сервера клієнту за запитом, розширюючи функціональність клієнта.

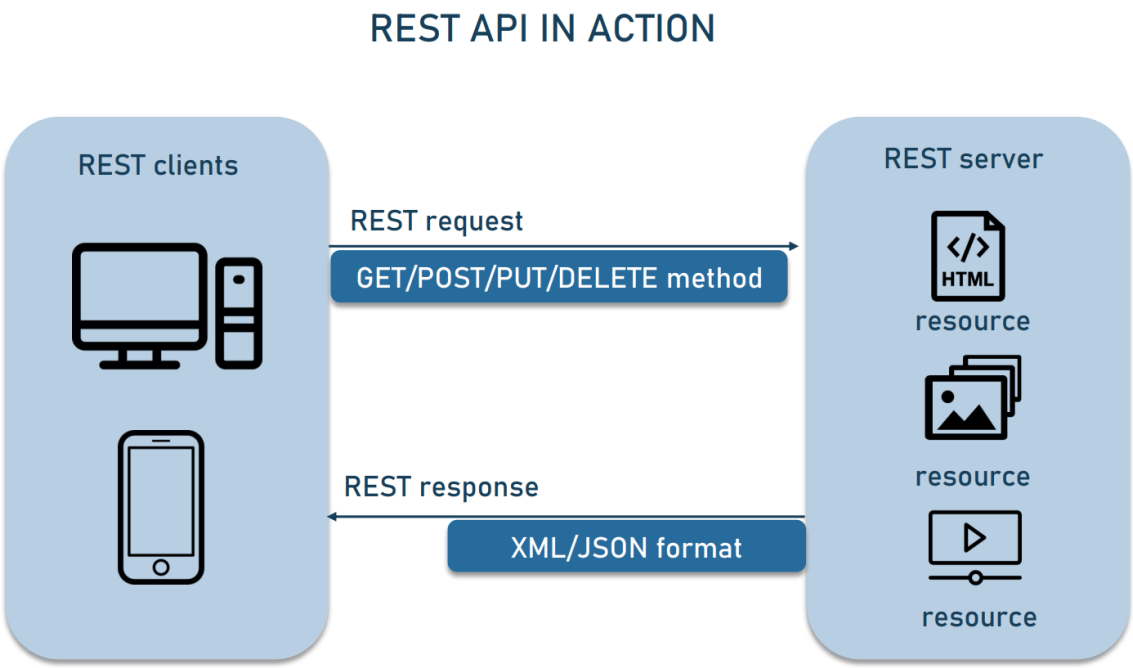


Рисунок 2.5. «Загальна діаграма моделей даних»

Висновки до розділу

З описаного вище, можна прийти до висновку – вибраний інформаційний та математичний апарат прекрасно покриває усі потреби які можуть виникати у інженера з програмного забезпечення для реалізації даної системи, її архітектури. Дані технологіє містять безліч фінкціоналу з доступною документацією та показують чудовий результат роботи.

РОЗДІЛ 3. ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

Технології та інструменти які були використані у процесі створення даної веб-орієнтованої системи:

Серверна сторона (Server-side):

1. Asp.Net 6
2. Entity Framework Core
3. Microsoft Asp.Net Core Authentication JwtBearer
4. Microsoft.Identity (IdentityModel.Tokens, IdentityModel.Tokens.Jwt)
5. Newtonsoft.Json
6. Swagger
7. C#

База данних:

1. MSSQL
2. SQL-server
3. T-SQL

Система контролю версій:

1. Git
2. Git-hub (віддалений git-репозиторій)

Сторона клієнту (Client-side):

1. Angular 17
2. Node.js
3. Type-script
4. Html, CSS, SCSS, SAAS
5. NPM, Angular-CLI
6. Material
7. Materialize.CSS
8. Bootstrap
9. RXJS
10. I18N

Інструменти:

1. Visual Studio 2022
2. WebStorm
3. MS SQL Server Management Studio
4. Source Tree

Основні вимоги до програмного забезпечення для запуску даного вебзастосунку:

1. CLR
2. .Net 6
3. Node.js: v14
4. Angular: 17
5. MSSql

3.1. Проектування бази даних.

Проектування бази даних було здійснено за допомогою Entity Framework Core, який реалізує ORM(Object-Relational Mapping). В якості СУБД було використано MS SQL Server.

MS SQL Server – продукт корпорації Microsoft, який використовує мову Transact-SQL для здійснення запитів і часто використовується в якості СУБД в сукупності з іншими технологіями Microsoft, включно з .Net, в даному випадку.

Entity Framework Core – це кросплатформна версія Entity Framework, яка, в свою чергу, дає можливість реалізувати ORM в .Net застосунках. Даний принцип дає можливість розробнику працювати з базою даних в об’єктно-орієнтованому стилі, за допомогою об’єктів, та надає велику кількість готового функціоналу для роботи з базою даних. Ця технологія є open source проектом.

Отже, для створення Data-слою даної системи, згідно з ORM, спершу, створюються моделі (класи які будуть представляти таблиці в базі даних), на основі яких за принципом Code First і будуть генеруватись таблиці в базі данних. Дана генерація відбувається за допомогою системи міграцій, яку надає пакет Microsoft.EntityFrameworkCore.Migrations.

Міграції бази даних - це інструмент для керування структурою бази даних. Дана технологія дає можливість швидкої та контрольованої зміни таблиць бази даних. Всі зміни чітко фіксуються та піддаються редагуванню.

У даній архітектурі, кожна модель таблиці успадковується від базового абстрактного класу **Entity** яка місить данні які повинен мати кожен запис у базі данних, кируючись принципами нормалізації бази даних та ACID.

```
7 references | 0 changes | 0 authors, 0 changes
public abstract class Entity
{
    24 references | 0 changes | 0 authors, 0 changes
    public Guid Id { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public DateTime Created { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public DateTime LastUpdate { get; set; }
}
```

Рисунок 3.1.1. «Базовий клас моделей Entity»

Поле “**LastUpdate**” слугує інфраструктурою для реалізації принципів **Optimistic-Pessimistic locking.**

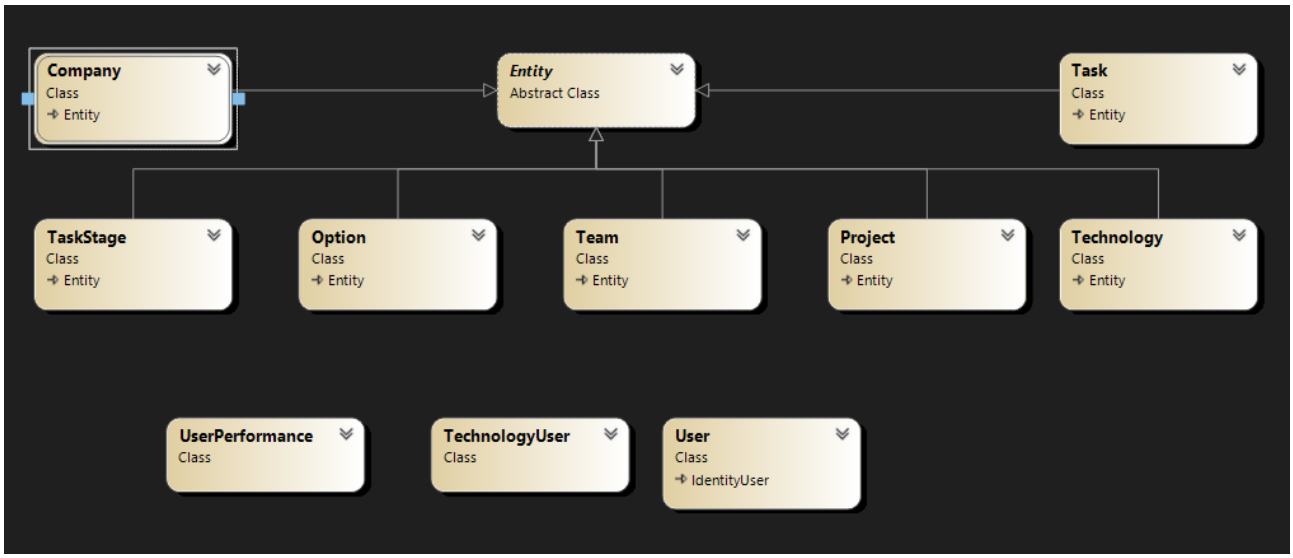
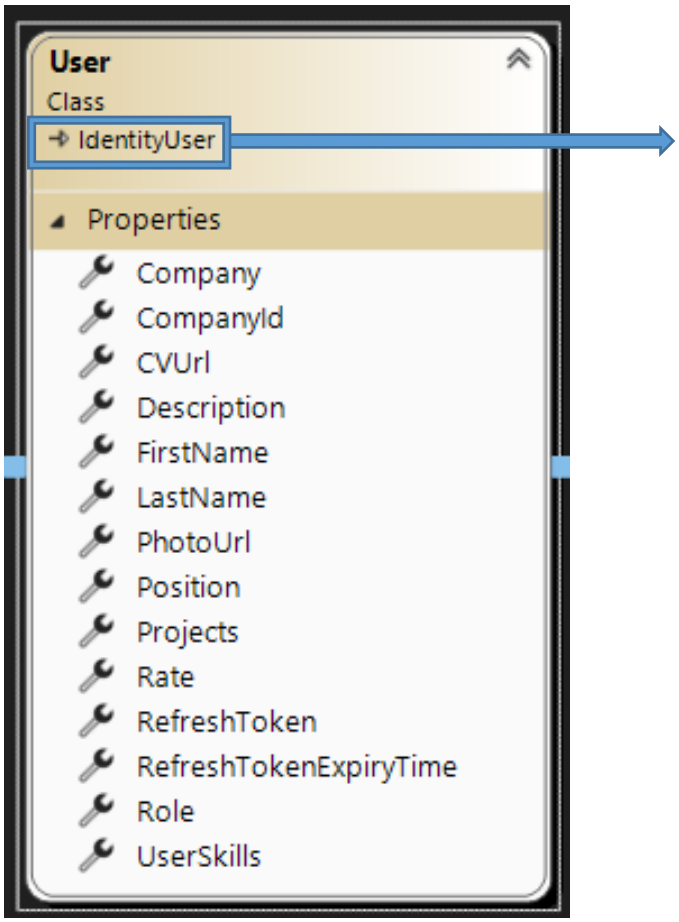


Рисунок 3.1.2. «Загальна діаграма моделей даних»

Основні моделі та їх таблиці на стороні бази даних:



Базовий клас технології **Identity**, який включає поля для роботи з користувачем, нормалізоване ім'я пароль, перевірки електронної пошти то що…

Место для уравнения.

Рисунок 3.1.3. «Діаграма моделі користувача»

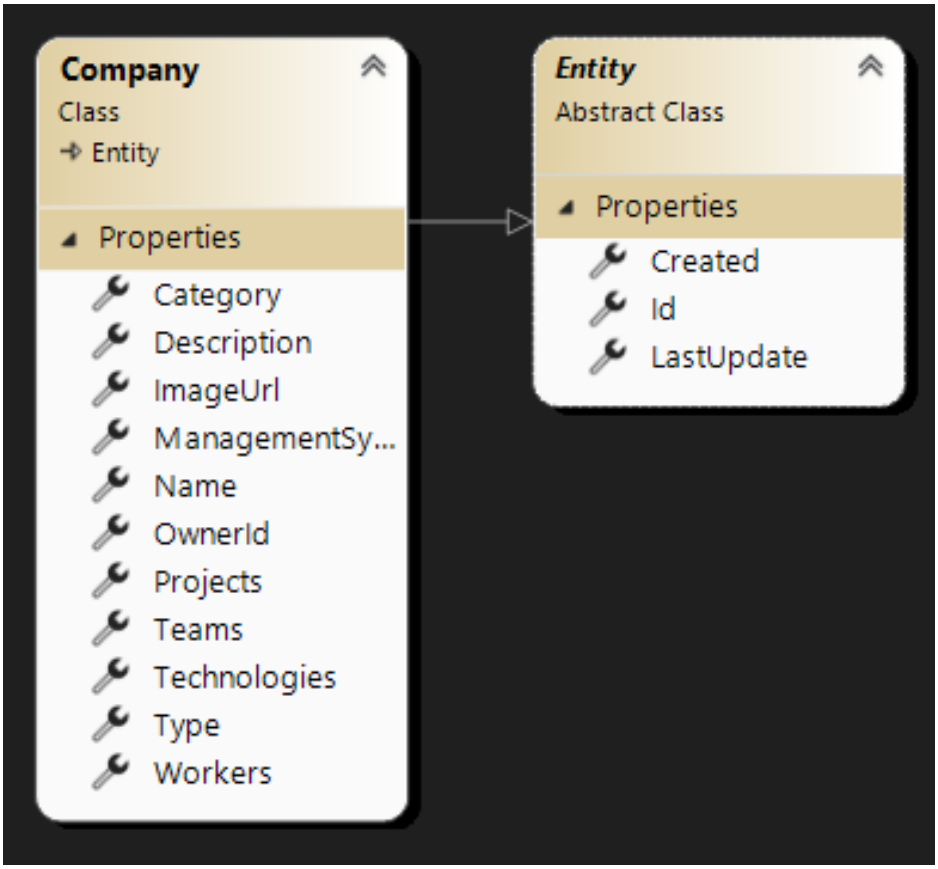


Рисунок 3.1.4. «Діаграма моделі компанії»

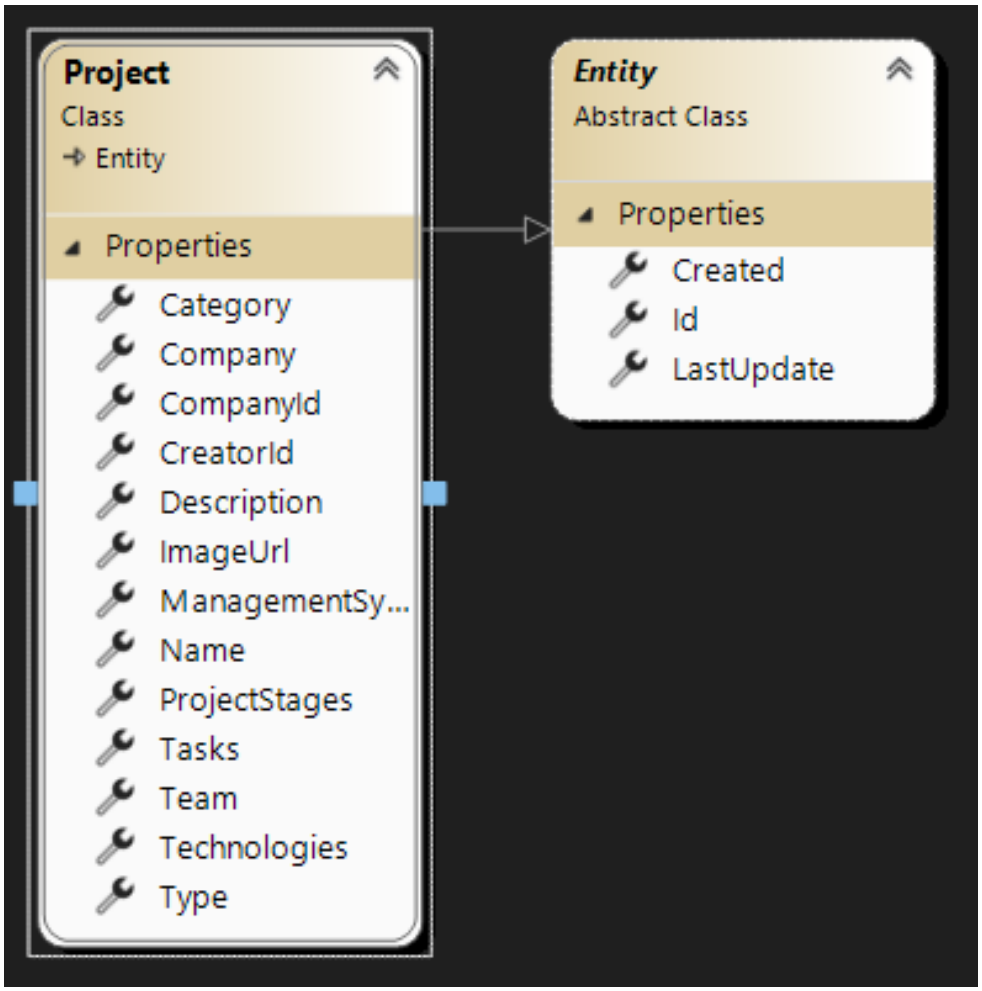


Рисунок 3.1.5. «Діаграма моделі проекту»

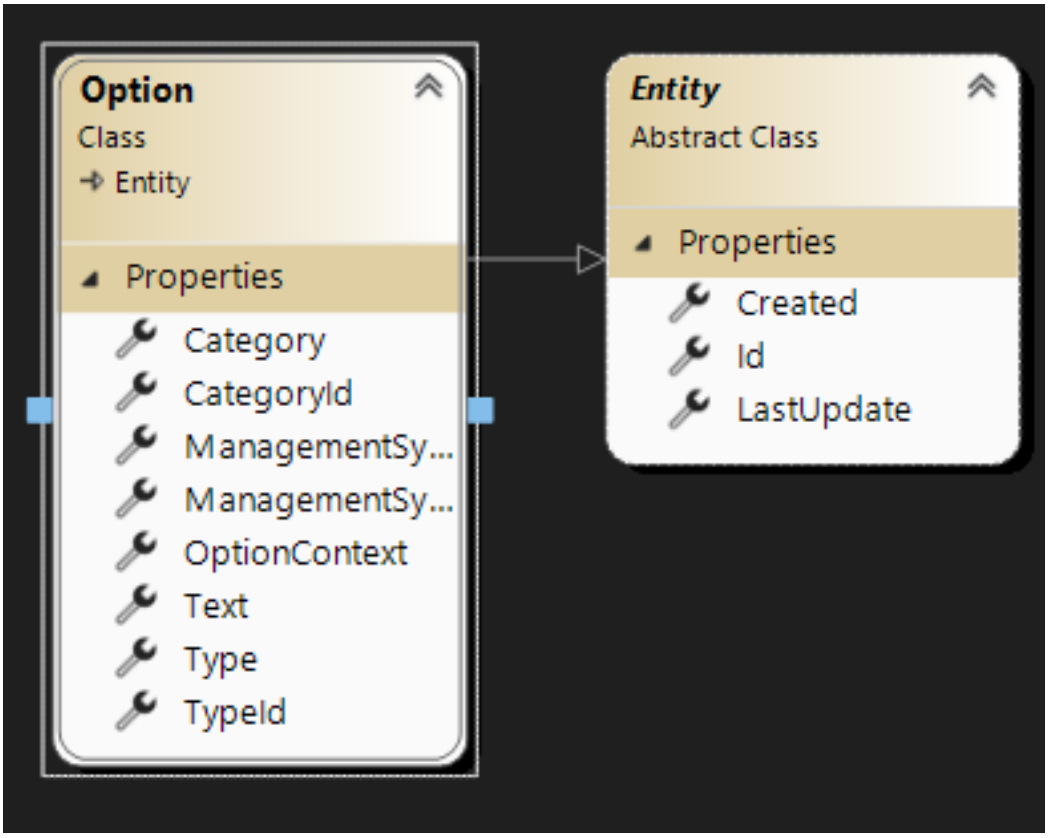


Рисунок 3.1.6. «Діаграма моделі опцій»

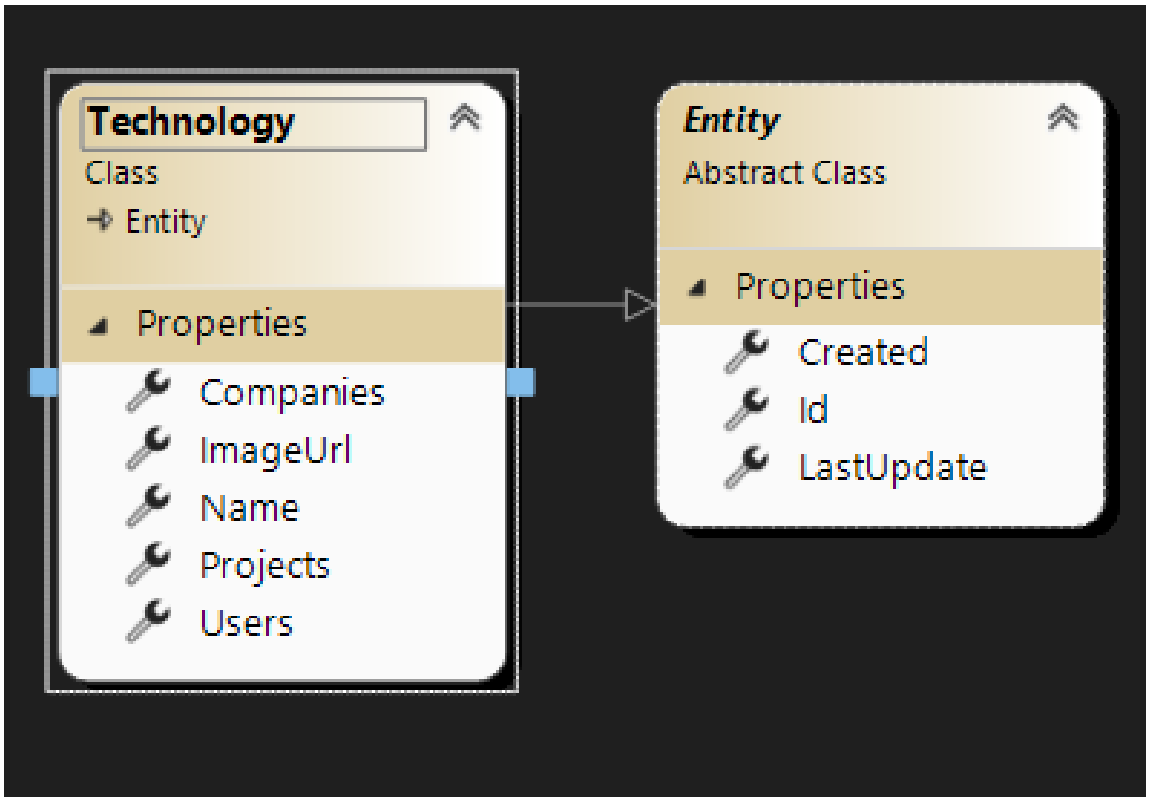


Рисунок 3.1.7. «Діаграма моделі технологій»

Реалізація зв’язків між таблицями у базі даних.

1. Зв'язок компанії та її опцій (зв'язок багато до багатьох)

```
modelBuilder.Entity<Company>()
    .HasMany(e => e.Category)
.WithMany(e => e.Category);

modelBuilder.Entity<Company>()
    .HasMany(e => e.Type)
.WithMany(e => e.Type);

modelBuilder.Entity<Company>()
    .HasMany(e => e.ManagementSystem)
.WithMany(e => e.ManagementSystem);
```

Рисунок 3.1.8. Конфігурація зв’язків компанії та її опцій

2. Зв’язок компанії до її працівників (один до багатьох)

```
modelBuilder.Entity<User>()
    .HasOne(e => e.Company)
.WithMany(e => e.Workers)
.HasForeignKey(ur => ur.CompanyId);
```

Рисунок 3.1.9. Конфігурація зв’язку компанії до її працівників

3. Зв'язок працівника до навичок (багато до багатьох)

```
modelBuilder.Entity<User>()
    .HasMany(e => e.UserSkills)
.WithMany(e => e.Users);
```

Рисунок 3.1.10. Конфігурація зв’язку користувача та його навичок

4. Зв'язок проекту до працівників (багато до багатьох)

```
modelBuilder.Entity<User>()
    .HasMany(e => e.Projects)
.WithMany(e => e.Team);
```

Рисунок 3.1.11. Конфігурація зв’язку користувачів до проектів

5. Зв'язок компанії до технологій (багато до багатьох)

```
modelBuilder.Entity<Company>()
    .HasMany(e => e.Technologies)
.WithMany(e => e.Companies);
```

Рисунок 3.1.12. Конфігурація зв’язку компаній до технологій

6. Зв'язок компанії до проектів (один до багатьох)

```
modelBuilder.Entity<Project>()
    .HasOne(e => e.Company)
.WithMany(e => e.Projects)
.HasForeignKey(ur => ur.CompanyId);
```

Рисунок 3.1.13. Конфігурація зв’язку компаній до проектів

3.2. Архітектура системи.

На найвищому рівні масштабування, у даній системі можна виділити 4 компоненти.

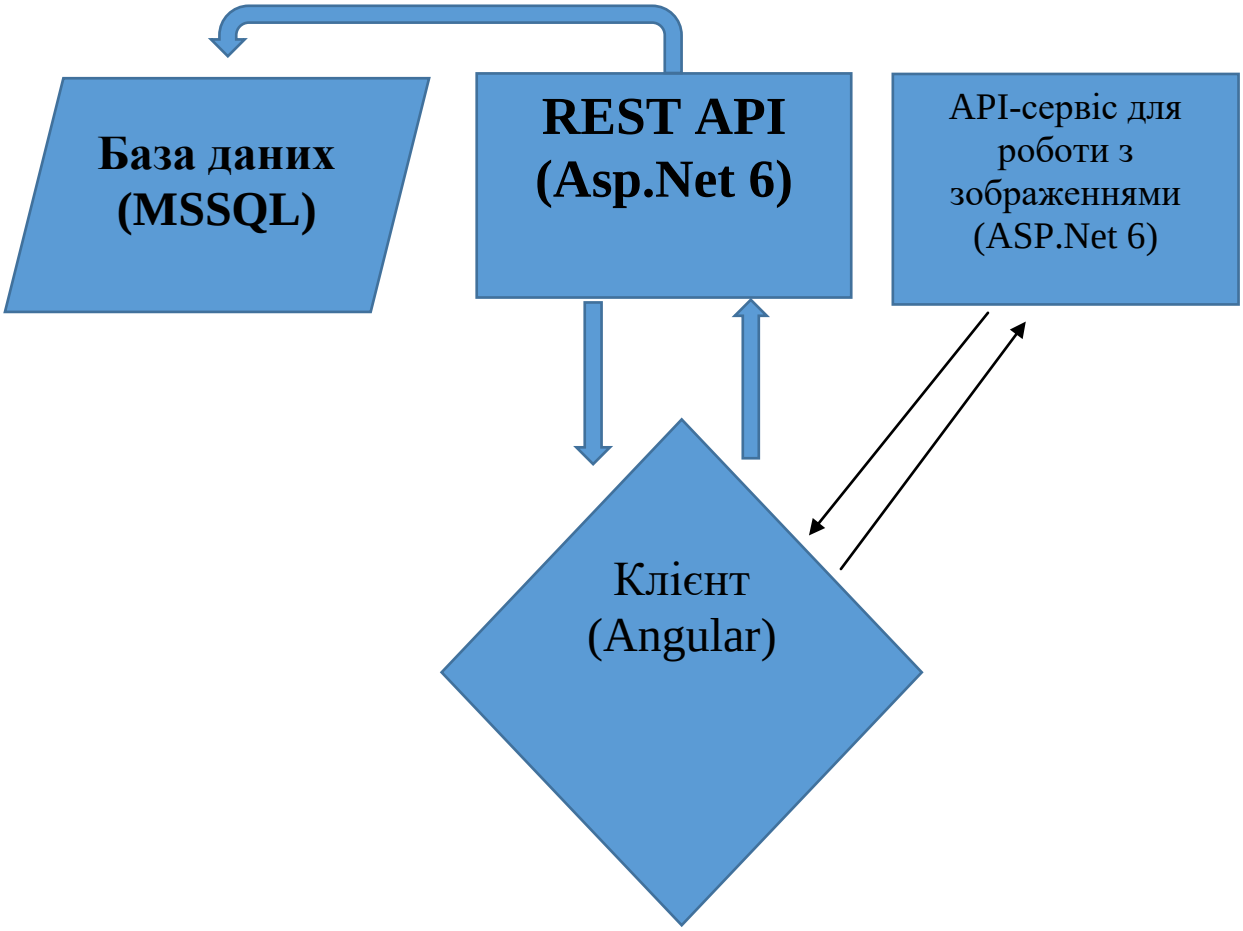


Рисунок 3.2.1. Схематичне відображення архітектури у абстрактному вигляді

Наступний рівень маштабізації це безпосередньо розробка кожного з цих компонентів.

3.3. Серверна частина.

Архітектура backend-частини даної системи реалізована за принципом 'Моноліт', за виключенням декомпозиції функціоналу для роботи із зображенням, який поміщено в окрему API. Даній архітектурі властиві 2 види масштабування “**Горизонтальне Дублювання**” та “**Сегментація”**.

- **Переваги монолітної архітектури:**
    - Простота розробки і розгортання.
    - Легкість у відлагодженні та тестуванні.

- **Недоліки монолітної архітектури:**
    - Складнощі масштабування, оскільки всі компоненти пов'язані.
    - Важкість в оновленні окремих частин системи без перевірки всього моноліту.

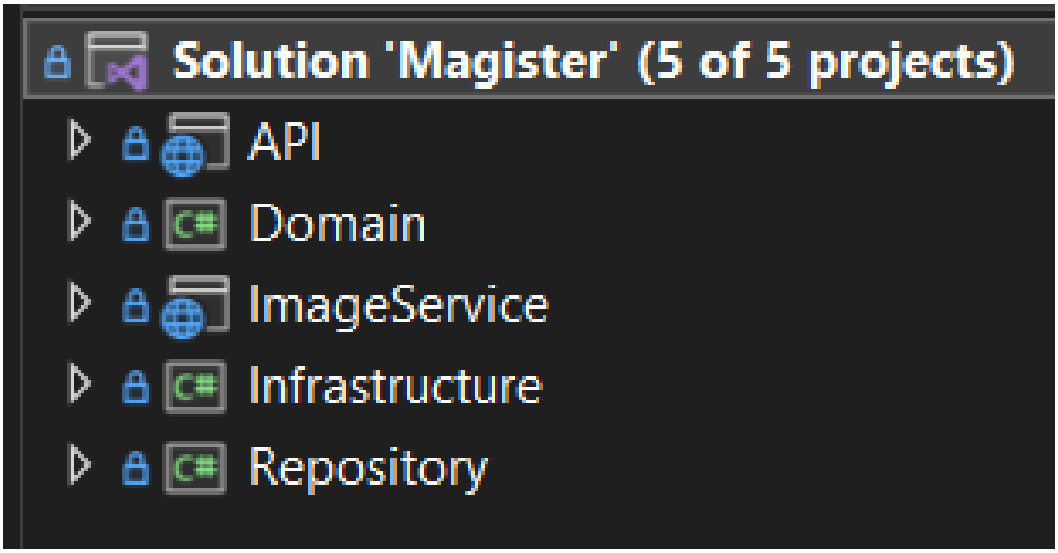


Рисунок 3.3.1. Відображення проекту у Visual Studio 2022 »

Модулі серверу:

1. API
2. Domain
3. Repository
4. Infrastructure
5. ImageService

3.4. Domain.

Модуль Domain – це модуль якій містить всі моделі даних, які використовує даний застосунок.

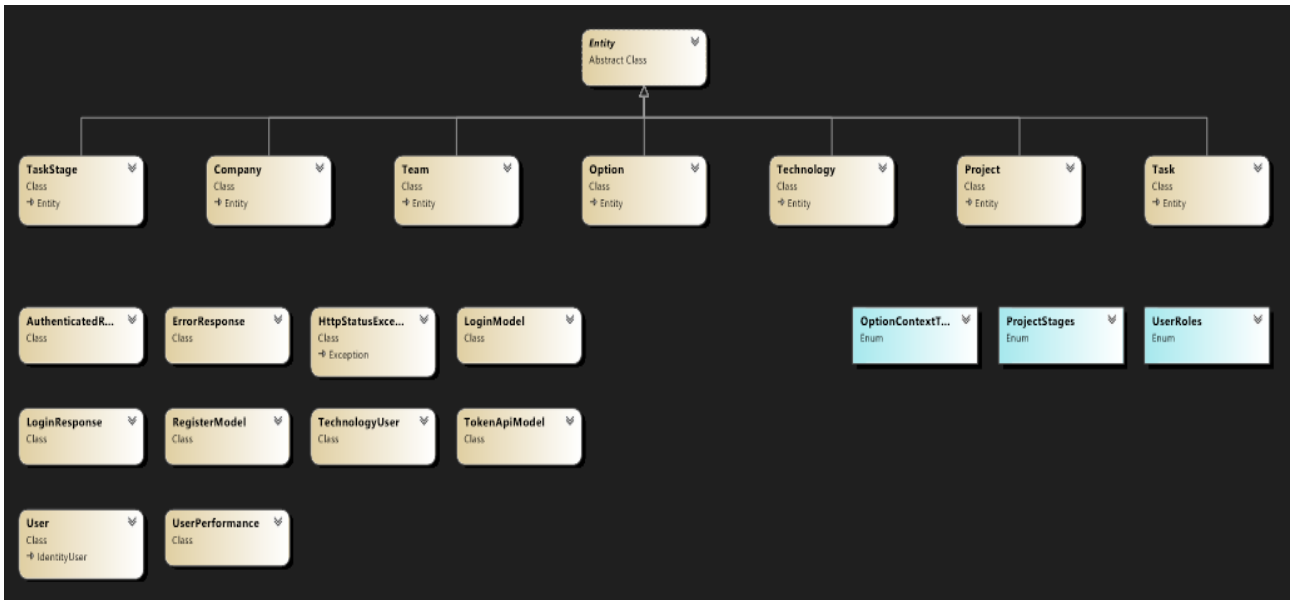


Рисунок 3.4.1 Діаграма класів модуля Domain

3.5. Repository.

Модуль Repository – педставляє собою набір інструментів для доступу до даних із бази даних з використанням моделей та конфігурація зазначених у модулі Domain.

Рисунок 3.5.1. Діаграма класів модуля Repository

3.6. Infrastructure.

Модуль Infrastructure – містить функціонал для обробки даних інкапсульованих за допомогою моделей(Domain), які в свою чергу отримуються за допомогою модуля Repository.

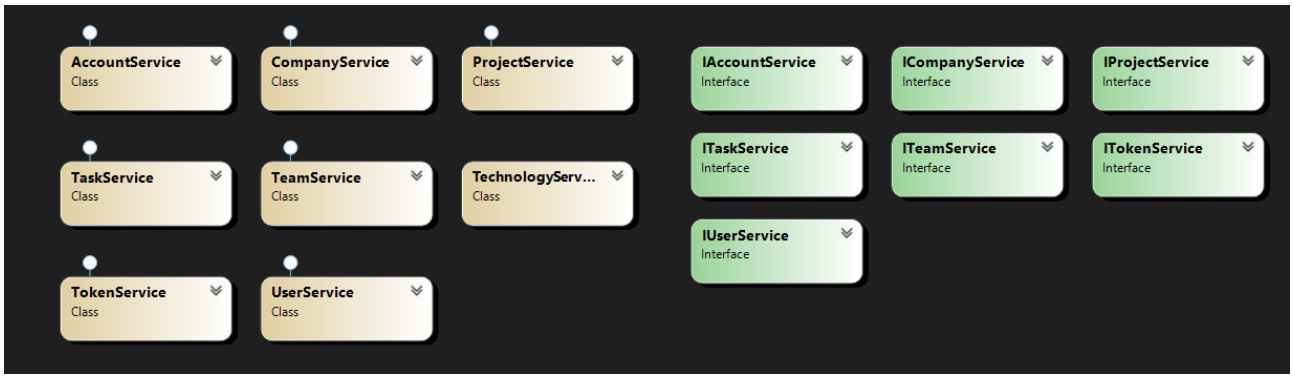


Рисунок 3.6.1. Діаграма класів модуля Infrastructure

3.7.API.

Модуль API – це REST-full Api, створений за допомогою технології ASP.Net 6, який сперається на матрицю **Richardson Maturity Model (RMM)**, яку описав Леонард Річардсон 2008му році, та яка визначає степінь реалізації REST API та приналежність до “ідеального” REST API .

Запити до бази даних відбуваються за допомогою принципу **Eager Loading**.

Eager Loading - один з принципів роботи з даними та побудови транзакцій що надає Entity Framework core (існуютьс також **Lazy Loading** та **Explicit Loading**). Даний принцип базується на використанні транзакцій, що забезпечує атомарність запитів, консистентність даних, ступінь ізоляції запитів та довговічність системи. Рівень ізоляції транзакцій який було використано у даному проекті є Read Commited, що є стандартним рівнем ізоляції транзакцій за замовчуванням у функціоналі який надає Entity Framework та провайдер LINQ – LINQ-to-Entity. Суть цього принципу полягає у тому що данні завантажуються за допомогою використанням навігаційної властивості та методу Include(), який використовує Inner Join для завантаження даних з таблиць бази даних по зв’язках між ними в об’єктно-орієнтовану форму.

Приклад запиту з декількома операціями **Eager Loading** (Inner Join):

```
var companyToUpdate = await _context.Companies
    .Include(x => x.Category)
    .Include(x => x.ManagementSystem)
    .Include(x => x.Type)
    .Include(x => x.Workers)
    .Include(x => x.Technologies)
    .Include(x => x.Projects)
    .Where(x => x.Id == company.Id)
    .FirstOrDefaultAsync();
```

Рисунок 3.7.1. Приклад запису з використанням Eager Loading

Оновлення зв’язків між даними таблиць компанії та працівників.

```
if(company.Workers != null)
{
    IEnumerable<User> workers = await _context.Users
        .Where(x => company.Workers.Select(y => y.Id).Contains(x.Id)).ToListAsync();
    companyToUpdate.Workers = workers;
}
```

Рисунок 3.7.2. Оновлення зв’язків між даними таблиць компанії та працівників

3.7.1. Ін’єкції залежностей.

Залежності між модулями визначаються за допомогою механізму Dependency Injection, який в свою чергу є однією з можливих реалізацій принципу Invertion of Control. У даній системі, ін’єкції залежностей відбуваються за допомогою передачу залежності через конструктор, тип такої ін’єкції називається - **Constructor Injection.** Можливі також реалізації ін’єкцій через властивість (**Property Injection)**, та за допомогою методу **(Method Injection**).

3.7.2. Конфігурації сервісів.

1. Конфігурація контексту бази даних:

```
string connection = builder.Configuration.GetConnectionString("DefaultConnection");
builder.Services.AddDbContext<ApplicationDbContext>(options =>
    options.UseSqlServer(connection, x => x.MigrationsAssembly("Repository")));
```

Рисунок 3.7.3. Конфігурація контексту бази данних

2. Конфігурація Identity (налаштування політики паролів):

```
builder.Services.AddIdentity<User, IdentityRole>(options =>
{
    // Password settings
    options.Password.RequireDigit = false;
    options.Password.RequireLowercase = false;
    options.Password.RequireUppercase = false;
    options.Password.RequireNonAlphanumeric = false;
    options.Password.RequiredLength = 1;
    options.Password.RequiredUniqueChars = 1;
})
                .AddEntityFrameworkStores<ApplicationDbContext>().AddDefaultTokenProviders(); ;
```

Рисунок 3.7.4. Конфігурація Identity

3. Конфігурація автентифікації за допомогою JWT:

```
builder.Services.AddAuthentication(opt =>
{
    opt.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
    opt.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
})
            .AddJwtBearer(options =>
            {
                options.TokenValidationParameters = new TokenValidationParameters
                {
                    ValidateIssuer = true,
                    ValidateAudience = true,
                    ValidateLifetime = true,
                    ValidateIssuerSigningKey = true,
                    ValidIssuer = "https://localhost:44304",
                    ValidAudience = "https://localhost:44304",
                    IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes("superSecretKey@345"))
                };
            });
```

Рисунок 3.7.5. Конфігурація автентифікації за допомогою JWT

4. Конфігурація ін’єкцій репозиторіїв та сервісів інкапсолюється в окремо виділений статичний клас AppHelper.

```
2 references | zadrot1995, 4 days ago | 2 authors, 3 changes
public static class AppHelper
{
    1 reference | zadrot1995, 4 days ago | 2 authors, 3 changes
    public static void InjectServices(IServiceCollection services)
    {
        services.AddTransient<ICompanyService, CompanyService>();
        services.AddTransient<IProjectService, ProjectService>();
        services.AddTransient<IAccountService, AccountService>();
        services.AddTransient<ITokenService, TokenService>();
        services.AddTransient<IUserService, UserService>();
    }

    1 reference | zadrot1995, 4 days ago | 1 author, 1 change
    public static void InjectRepositories(IServiceCollection services)
    {
        services.AddTransient<ICompanyRepository, CompanyRepository>();
        services.AddTransient<IProjectRepository, ProjectRepository>();
        services.AddTransient<IUserRepository, UserRepository>();
    }
}
```

Рисунок 3.7.6. Ін’єкція залежностей сервісів та репозиторіїв

```
AppHelper.InjectRepositories(builder.Services);

AppHelper.InjectServices(builder.Services);
```

Рисунок 3.7.7. Використання AppHelper

5. ASP.Net Pipeline (Конвеєр):

Рисунок 3.7.8. Pipeline даної системи

3.8 Клієнтська частина.

Front-end даного застосунку розроблено за допомогою технології Angular 17, яка дає можливість реалізовувати клієнтські веб застосунки за принципом Single Page Application (SPA).

1. Загальна діаграма компонентів клієнтської частини застосунку

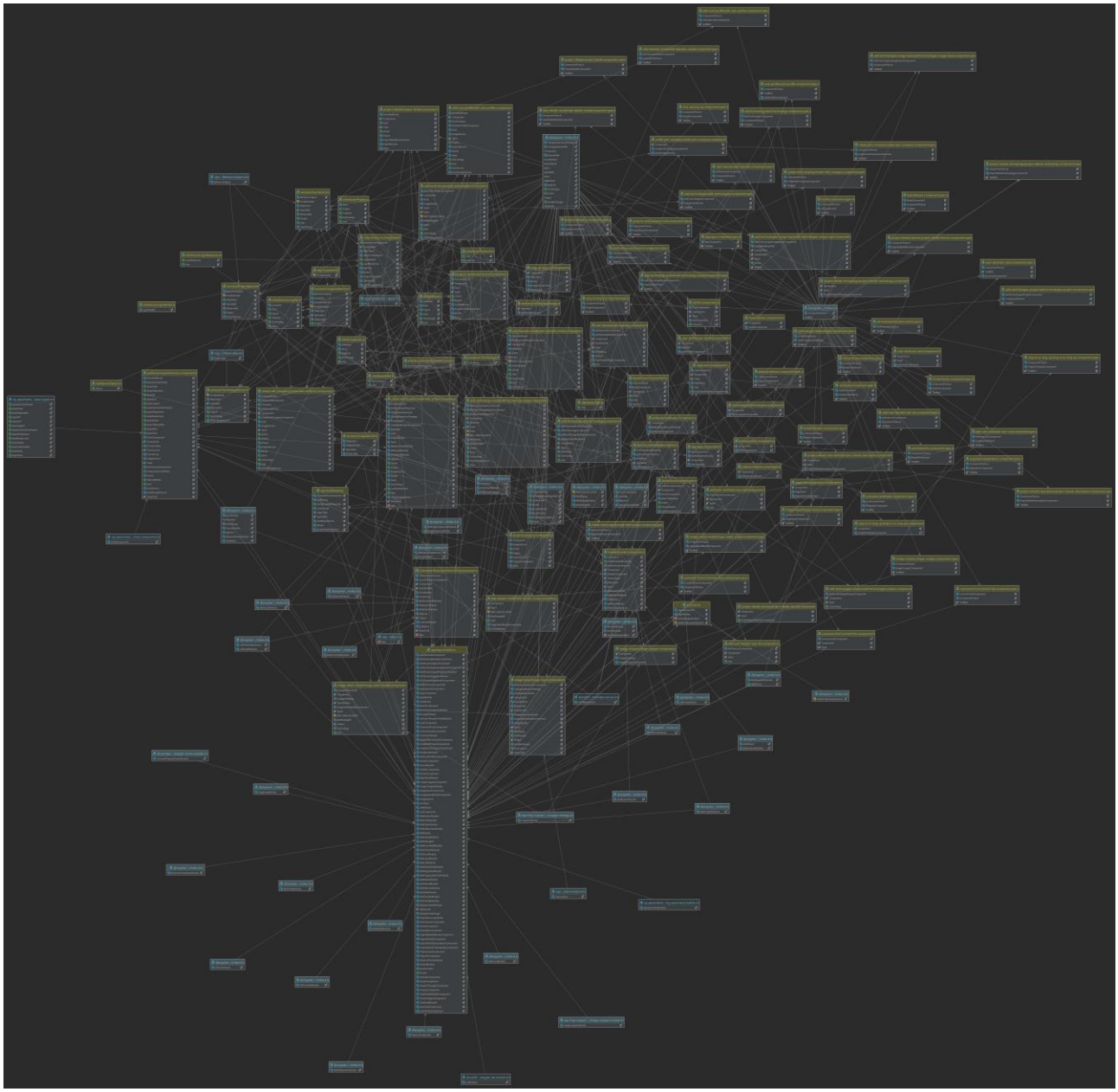


Рисунок 3.8.1. Діаграма клієнтської частини

MatButton

Компонент кнопки Material Design. Користувачі взаємодіють за допомогою кнопки, щоб виконати дію.

Клас MatButton застосовується до власних елементів кнопки та фіксує зовнішній вигляд «text button», «outlined button» та «вміщеної кнопки» відповідно до специфікації Material Design. MatButton додатково фіксує додатковий «flat» вигляд, який відповідає «вміщеному», але без висоти.

Селектор: button[mat-button] button[mat-raised-button] button[mat-flat-button] button[mat-stroked-button]

2. Автентифікація, авторизація та реєстрація.

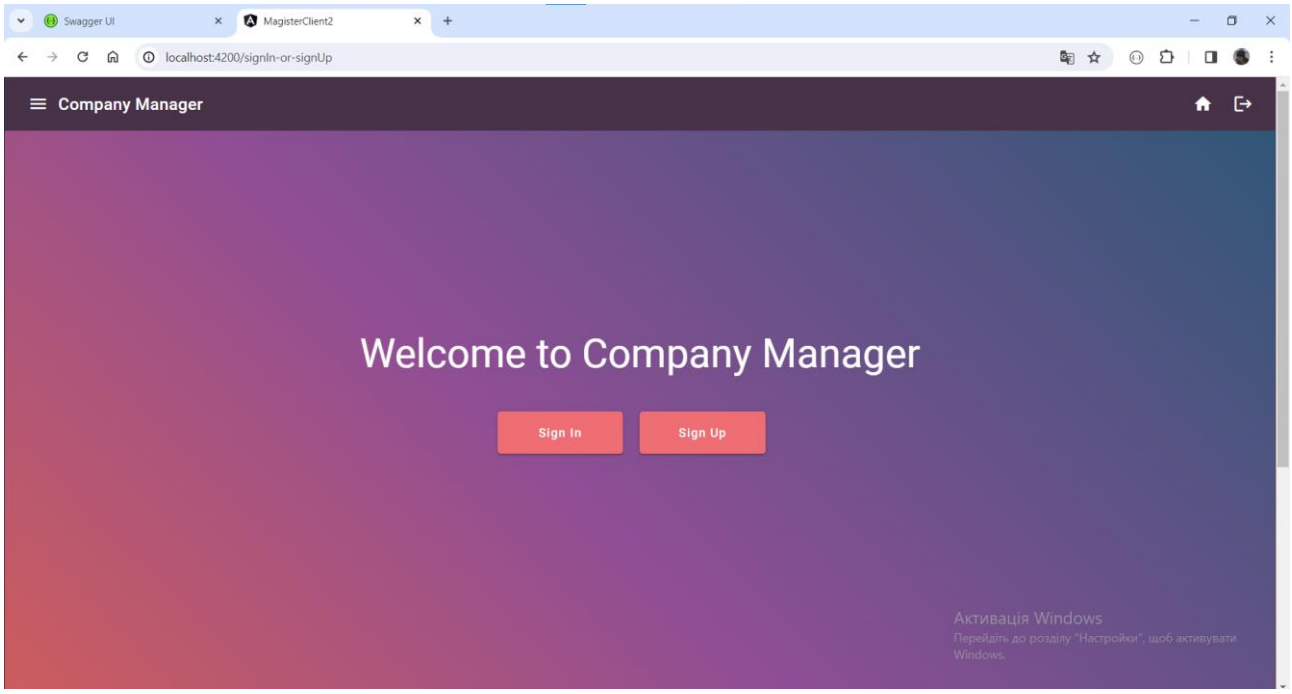


Рисунок 3.8.2. Стартове вікно

MatCard

Компонент картки Material Design. Картки містять вміст і дії щодо одного предмета.

MatCard не забезпечує поведінки, натомість слугує суто візуальним лікуванням.

Селектор: **mat-card**

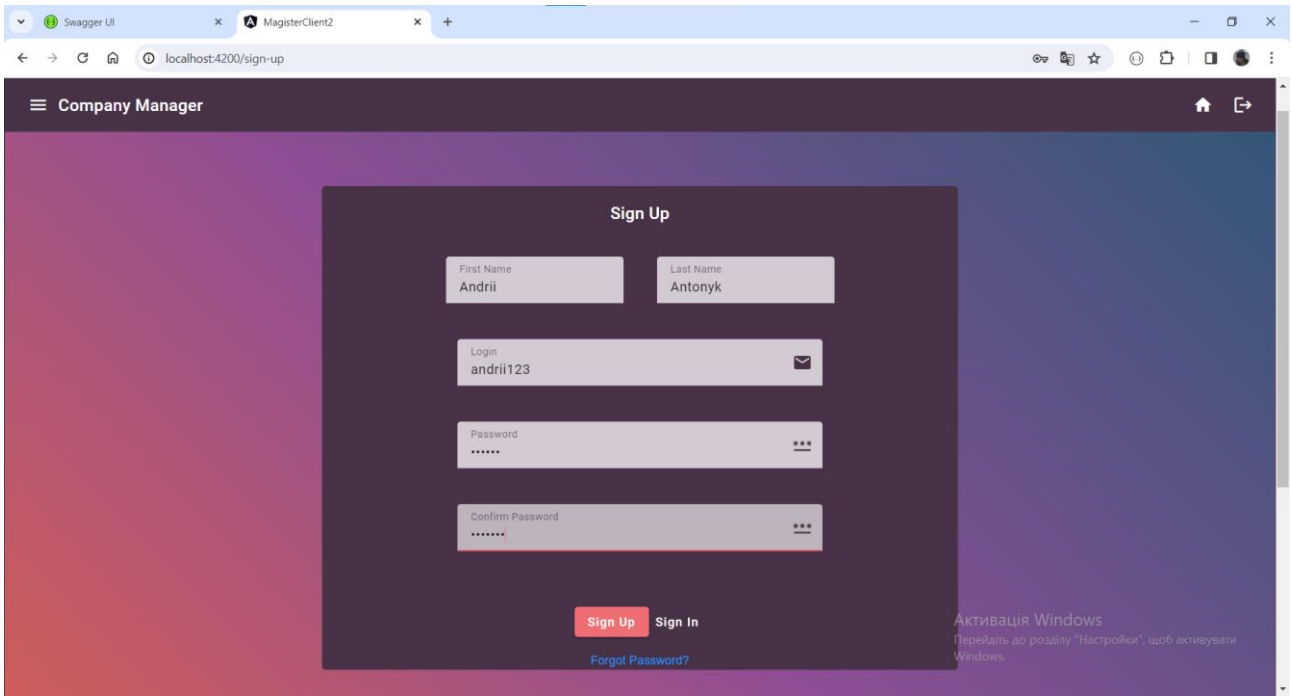


Рисунок 3.8.3. Вікно реєстрації

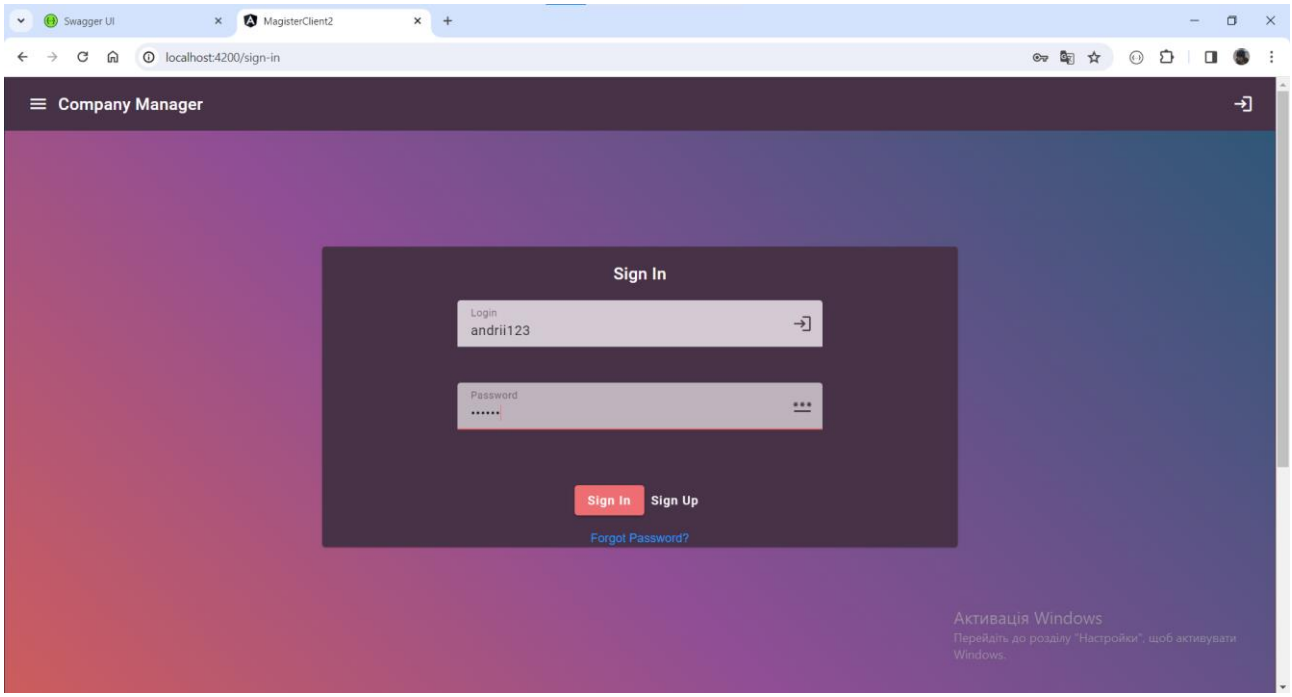


Рисунок 3.8.4. Вікно автентифікації

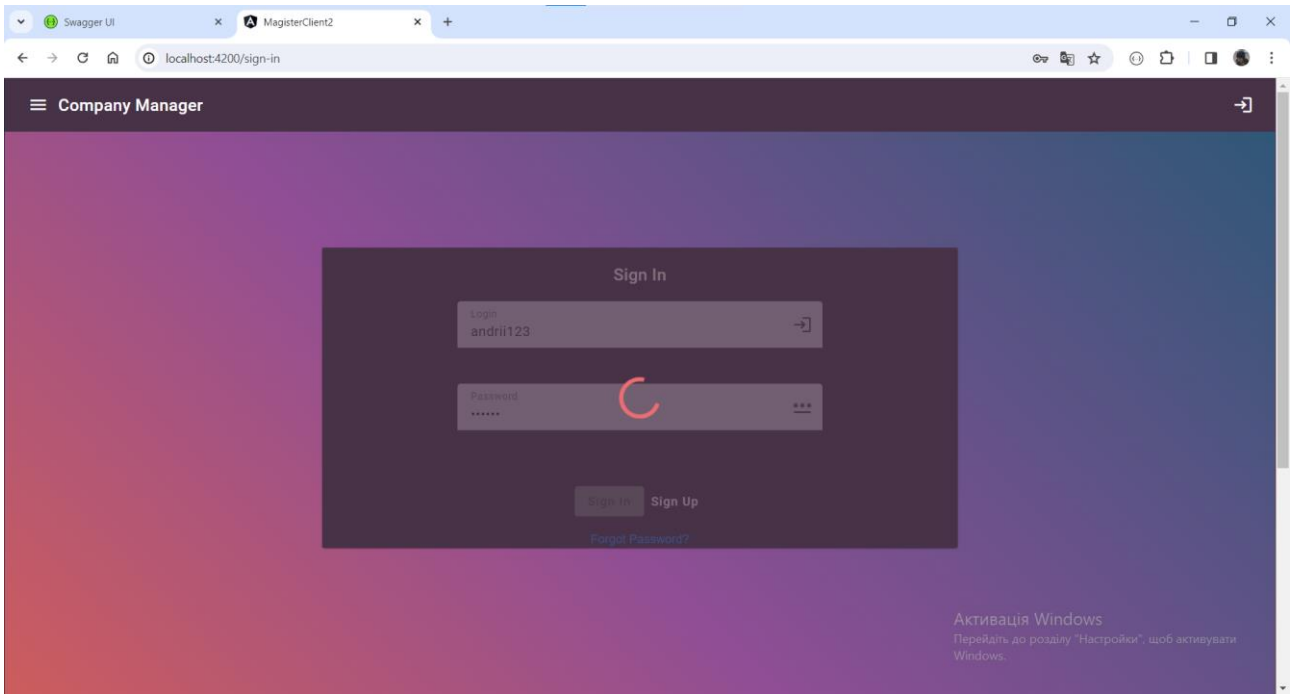


Рисунок 3.8.5. Завантаження даних під час автентифікації(preloader/circular progress-barr)

3. Профіль користувача:

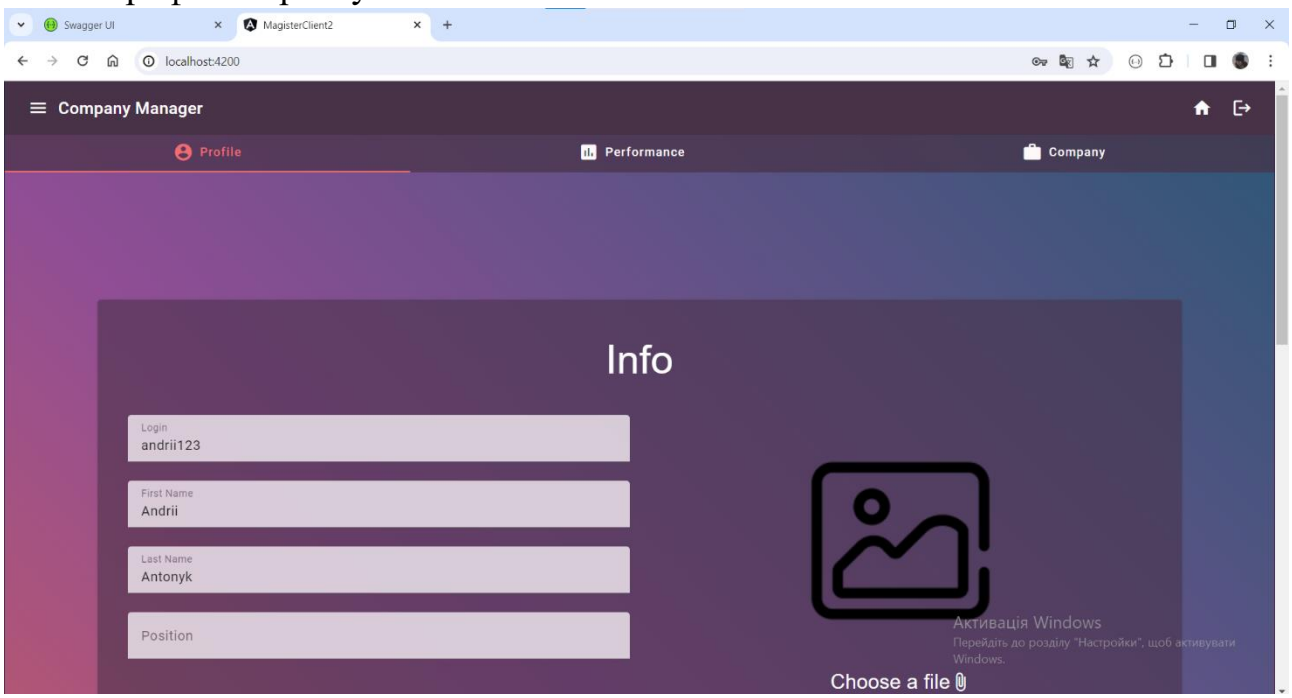


Рисунок 3.8.6. Вікно профілю користувача

4. Вибір фото користувача:

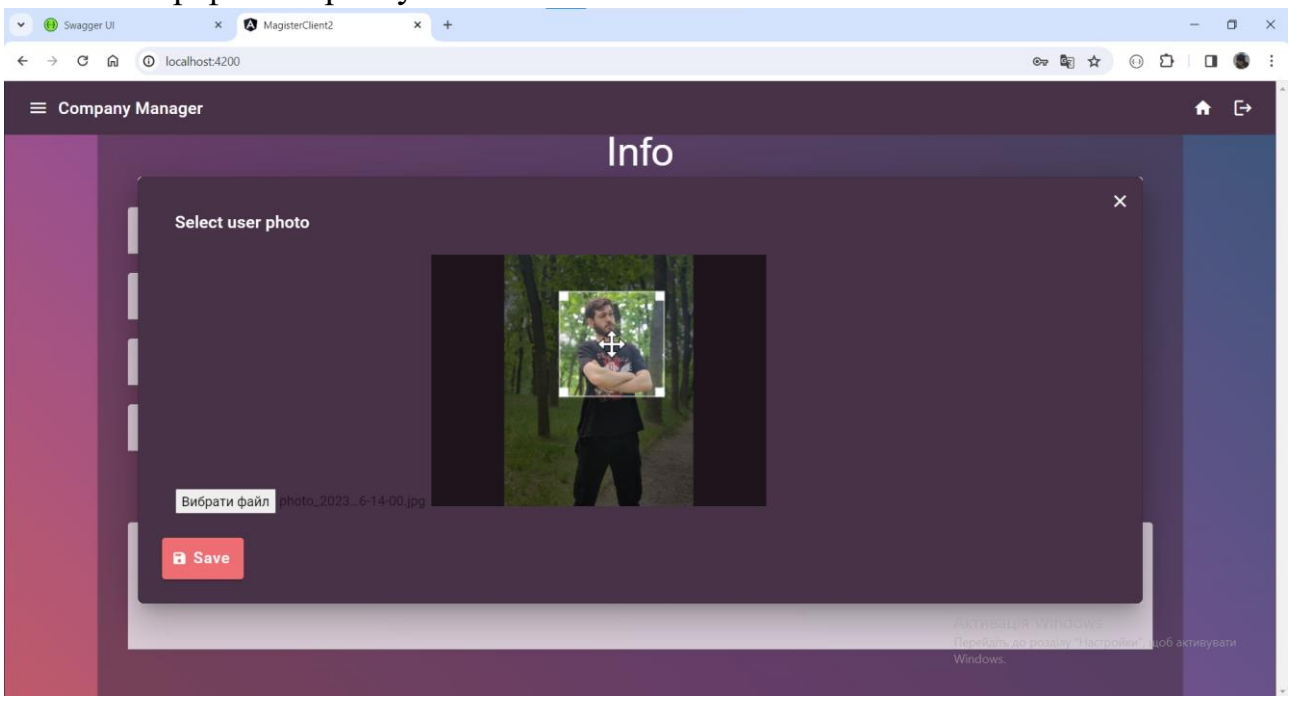


Рисунок 3.8.7. Встановлення зображення користувача

MatCardTitle

Заголовок картки, призначеної для використання в <mat-card>. Цей компонент є додатковим зручним для одного різновиду заголовка картки; будь-який настроюваний елемент заголовка може бути використаний замість нього.

MatCardTitle не забезпечує поведінки, натомість слугує суто візуальним лікуванням.

Селектор: **mat-card-title [mat-card-title] [matCardTitle]**

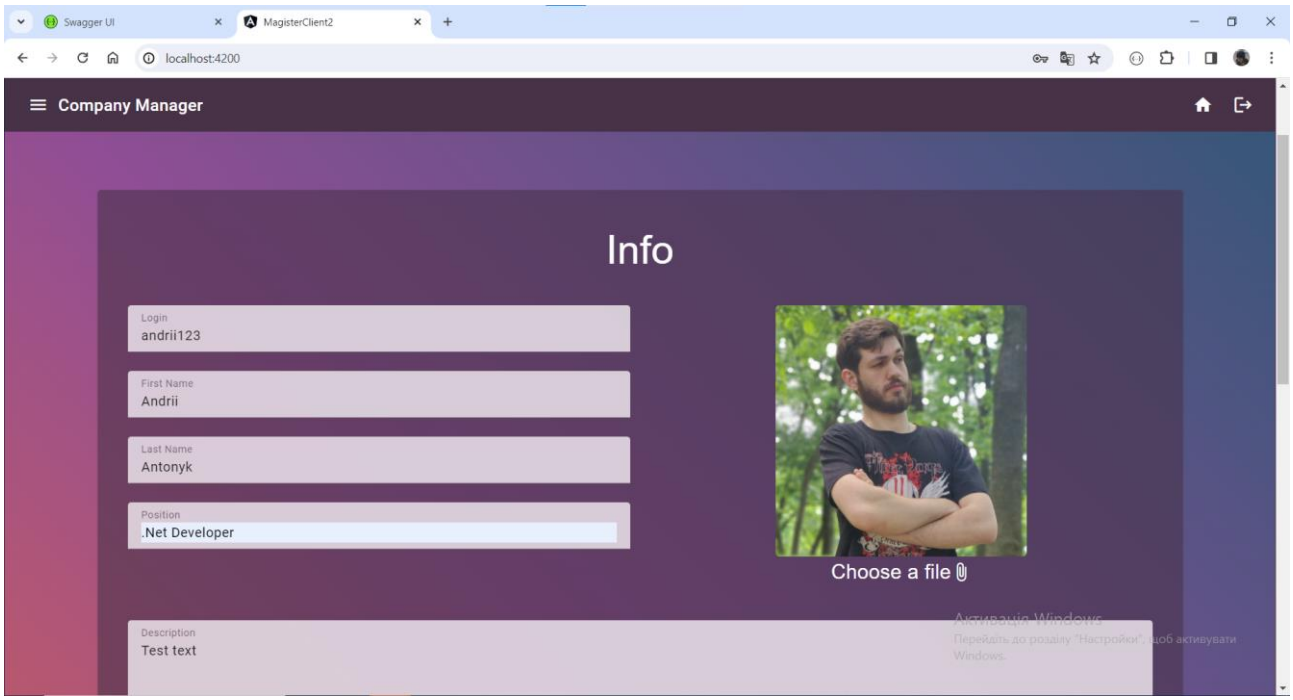


Рисунок 3.8.8. Профіль користувача

5. Вибір навичок користувача:

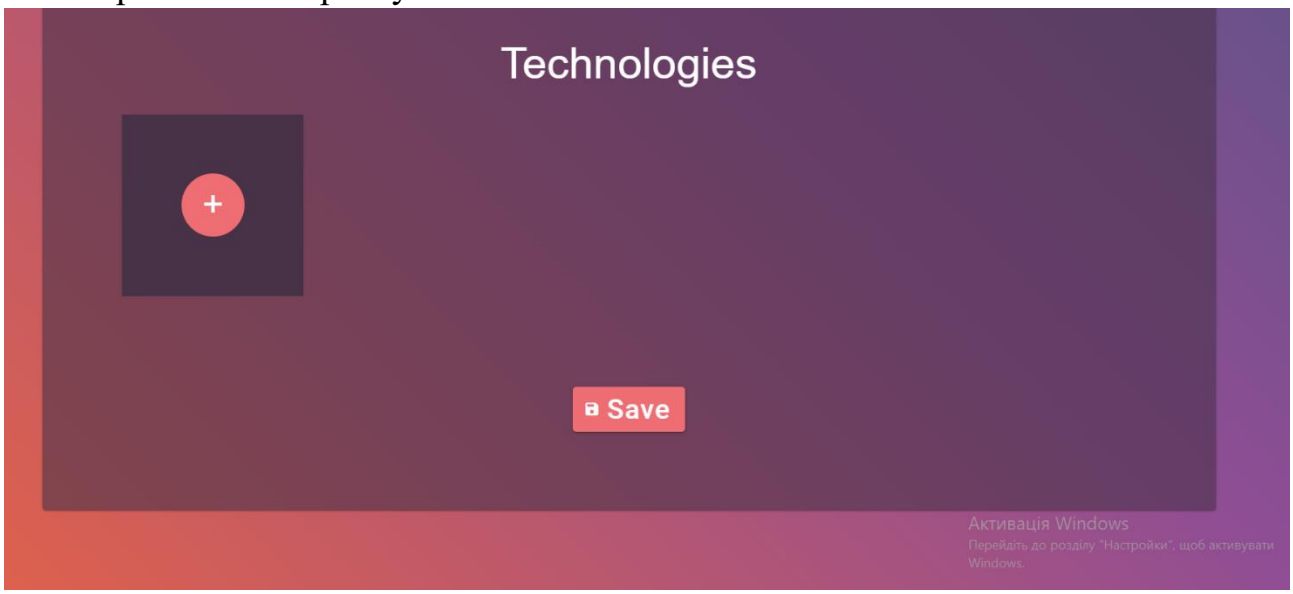


Рисунок 3.8.9. Список технологій користувача

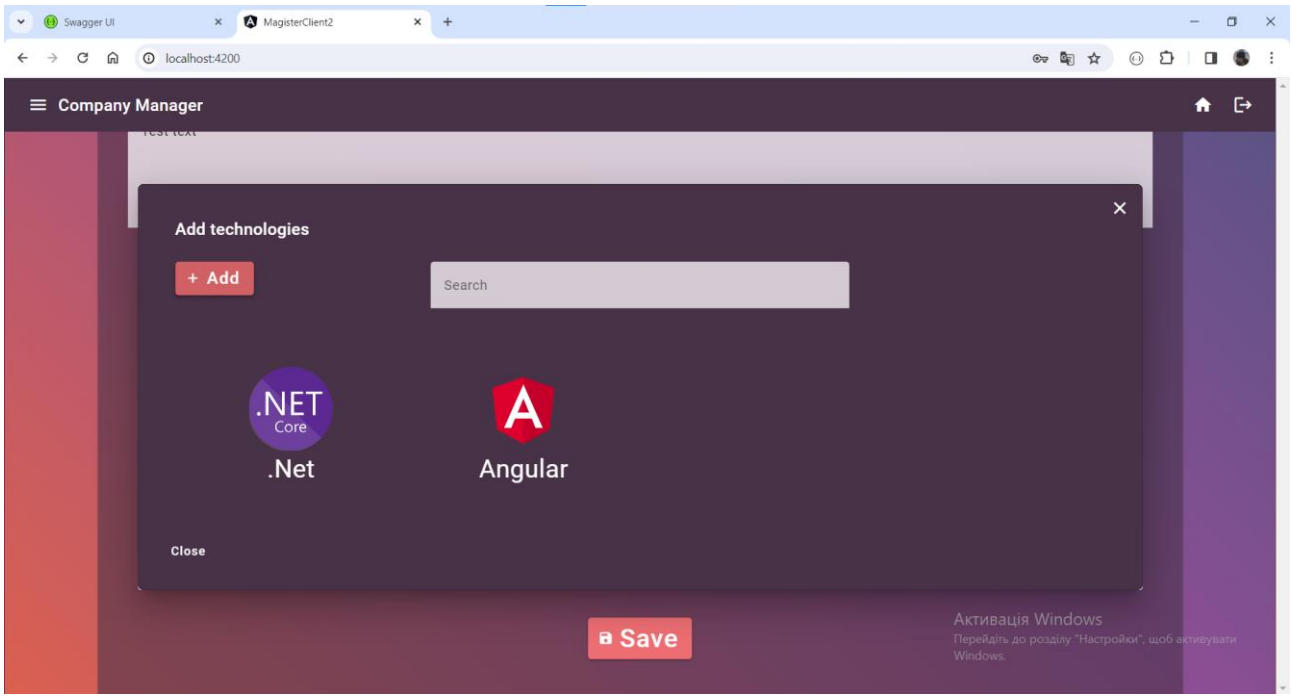


Рисунок 3.8.10. Вікно вибору технологій якими володіє користувач

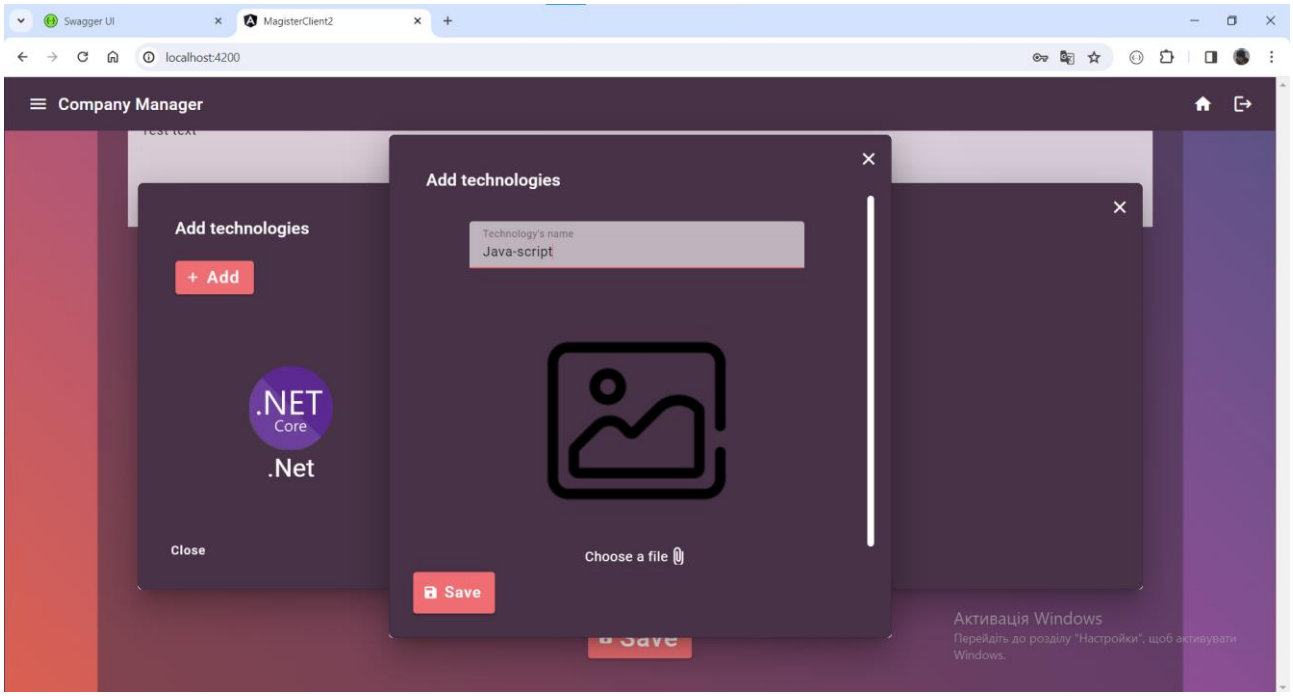


Рисунок 3.8.11. Створення нової технології

MatCardTitleGroup

Контейнер призначений для використання в компоненті <mat-card>. Може містити точно один <mat-card-title>, один <mat-card-subtitle> і одне зображення вмісту будь-якого розміру (наприклад, <img matCardLgImage>).

Селектор: **mat-card-title-grou**

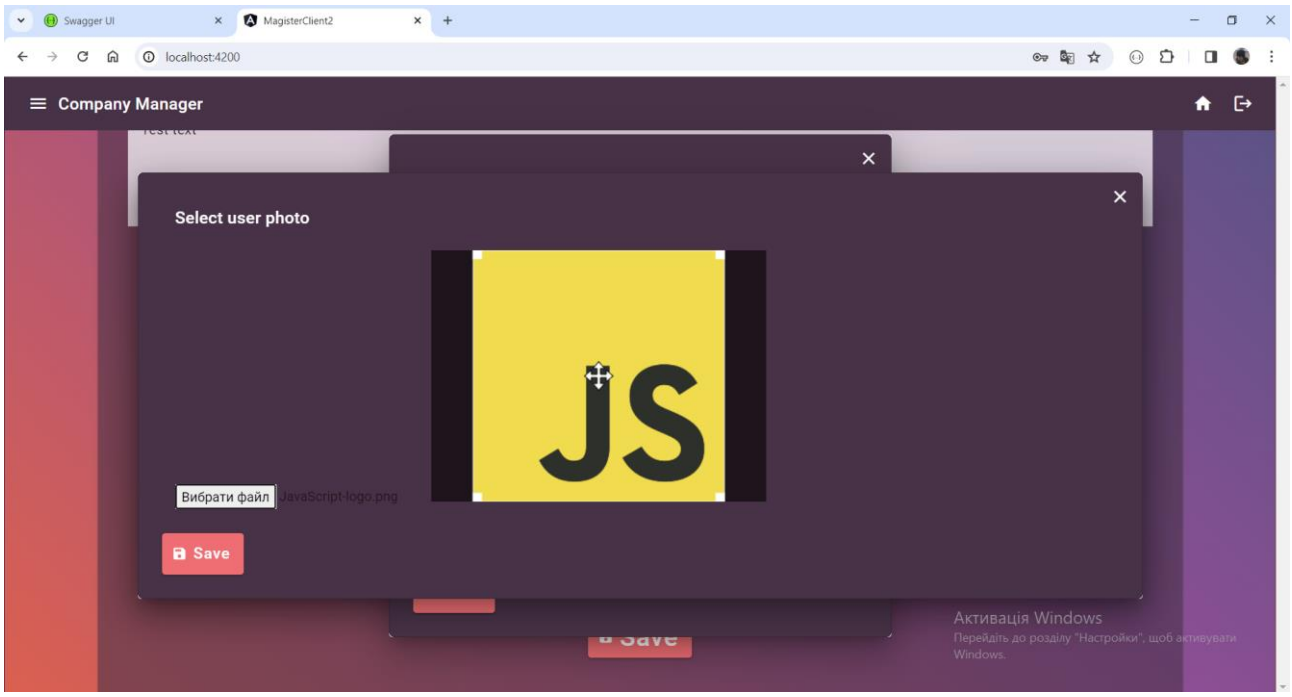


Рисунок 3.8.12. Додавання зображення до технології

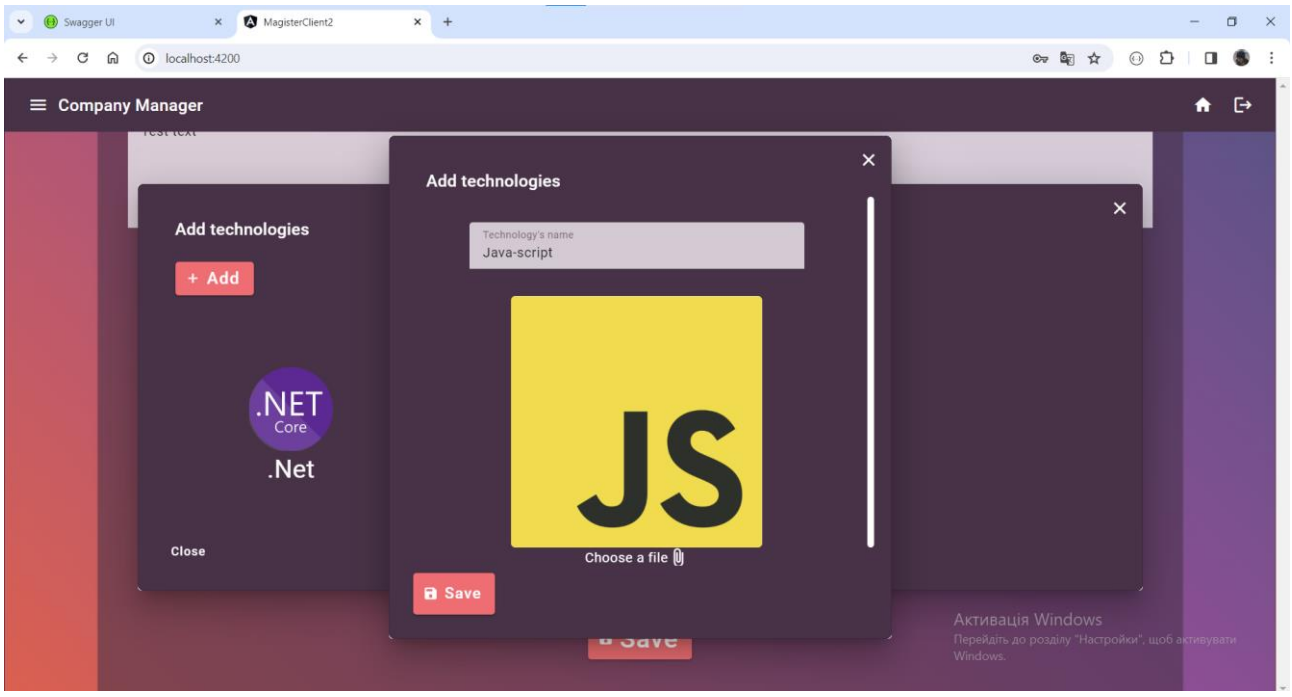


Рисунок 3.8.13.Зберігання нової технології

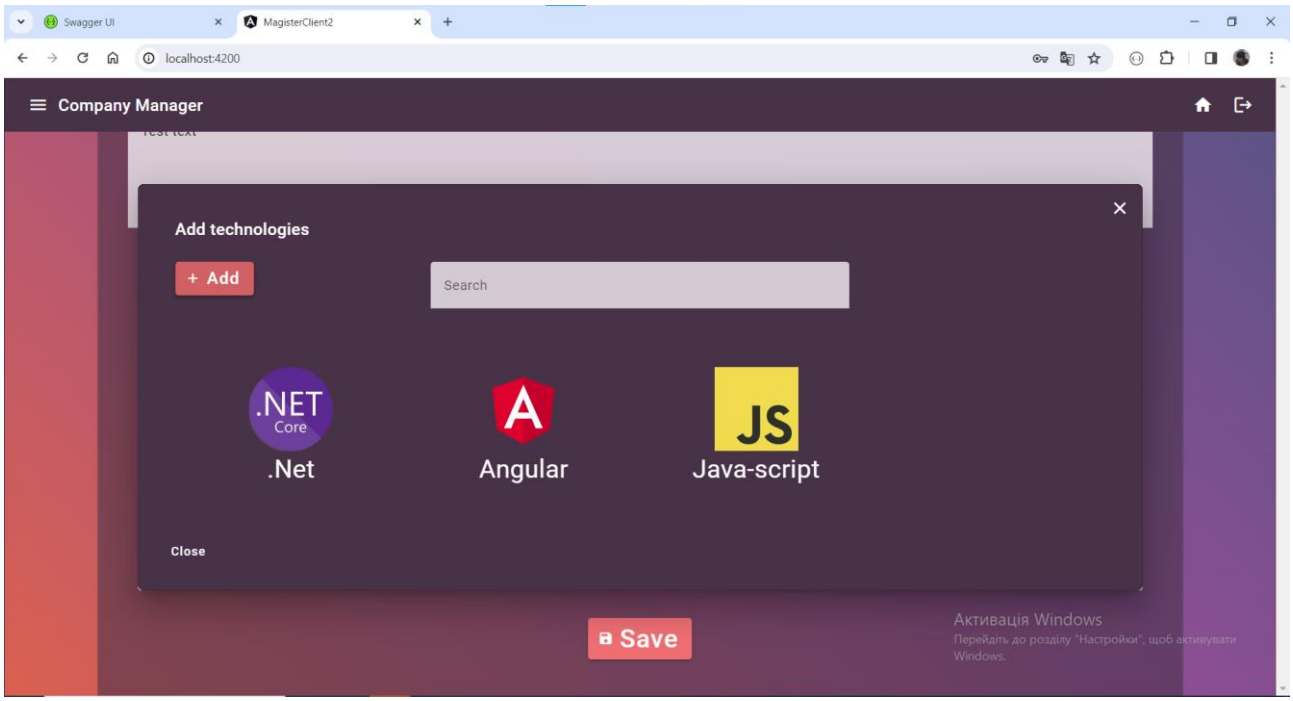


Рисунок 3.8.14. Вікно доступних технологій

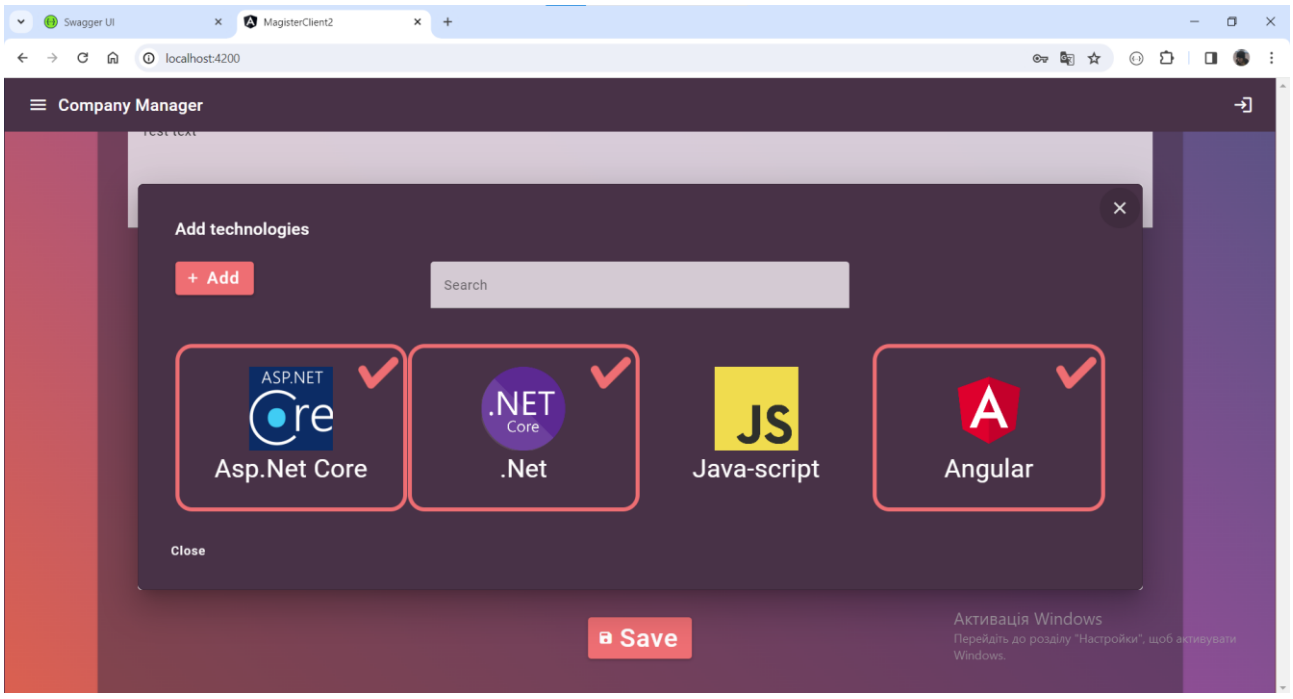


Рисунок 3.8.15. Додавання доступної технології до списку технологій користувача

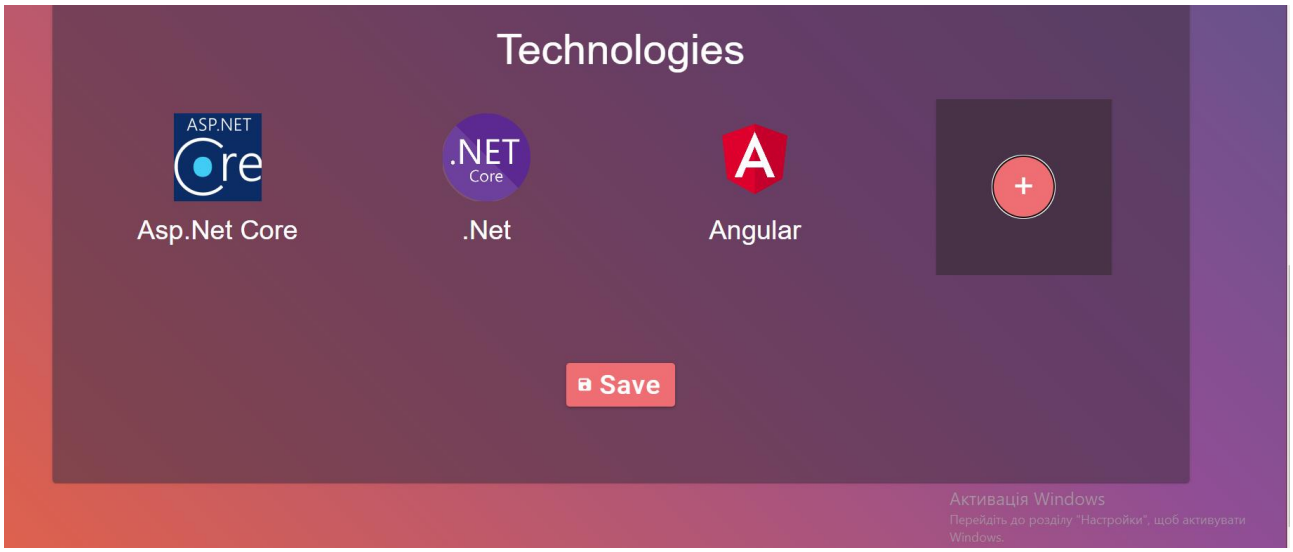


Рисунок 3.8.16. Список технологій користувача

6. Створення компанії:

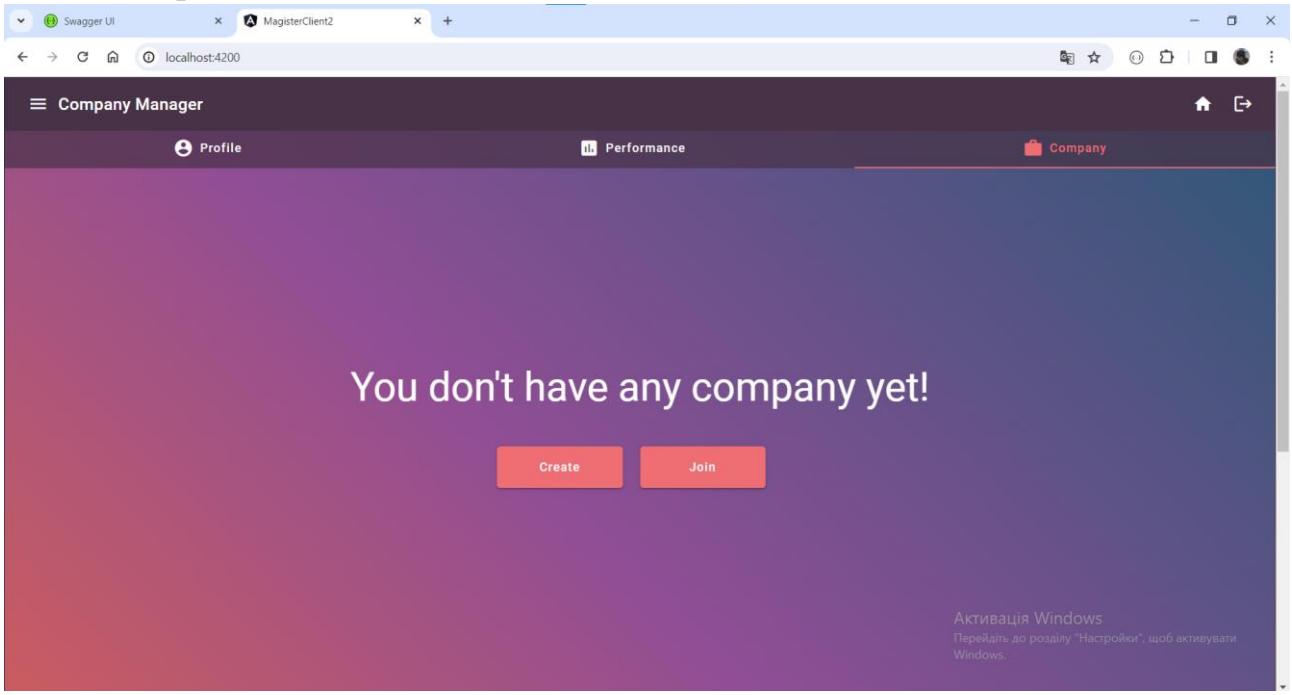


Рисунок 3.8.17. Вікно компанії(користувач не належить до жодної компанії)

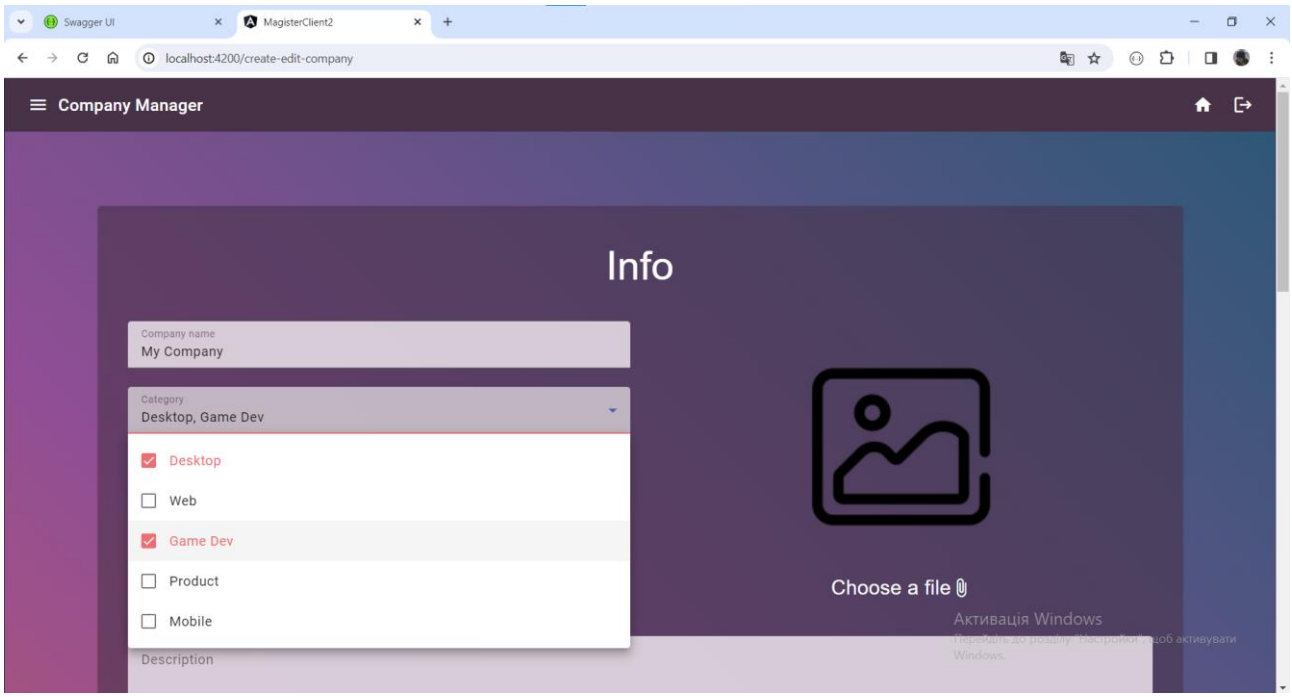


Рисунок 3.8.18. Вікно створення нової компанії

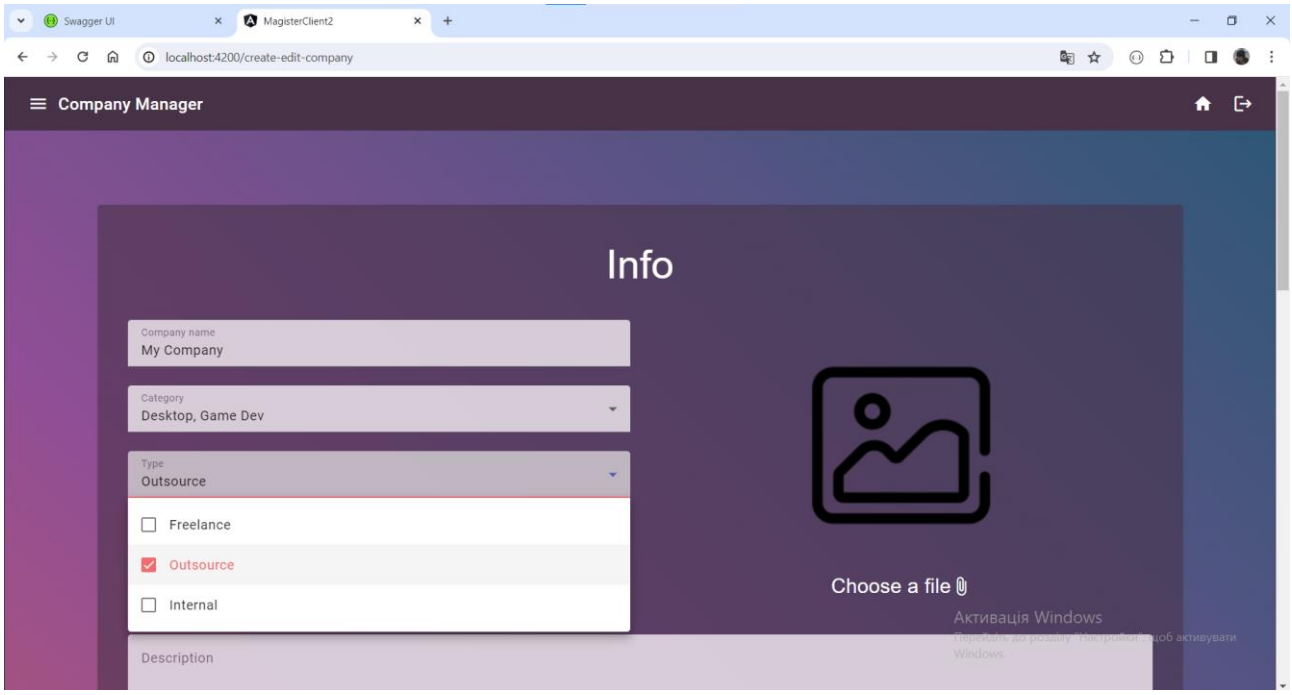


Рисунок 3.8.19. Вибір типу компанії

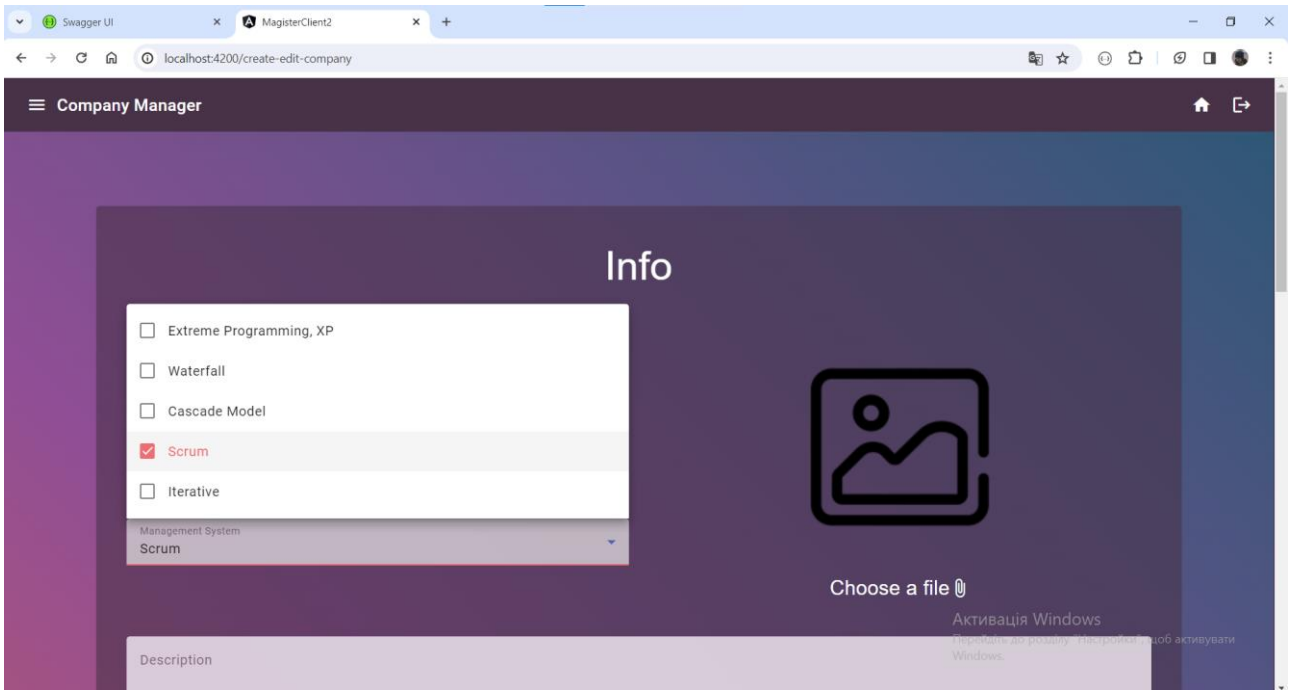


Рисунок 3.8.20. Вибір системи бенеджменту

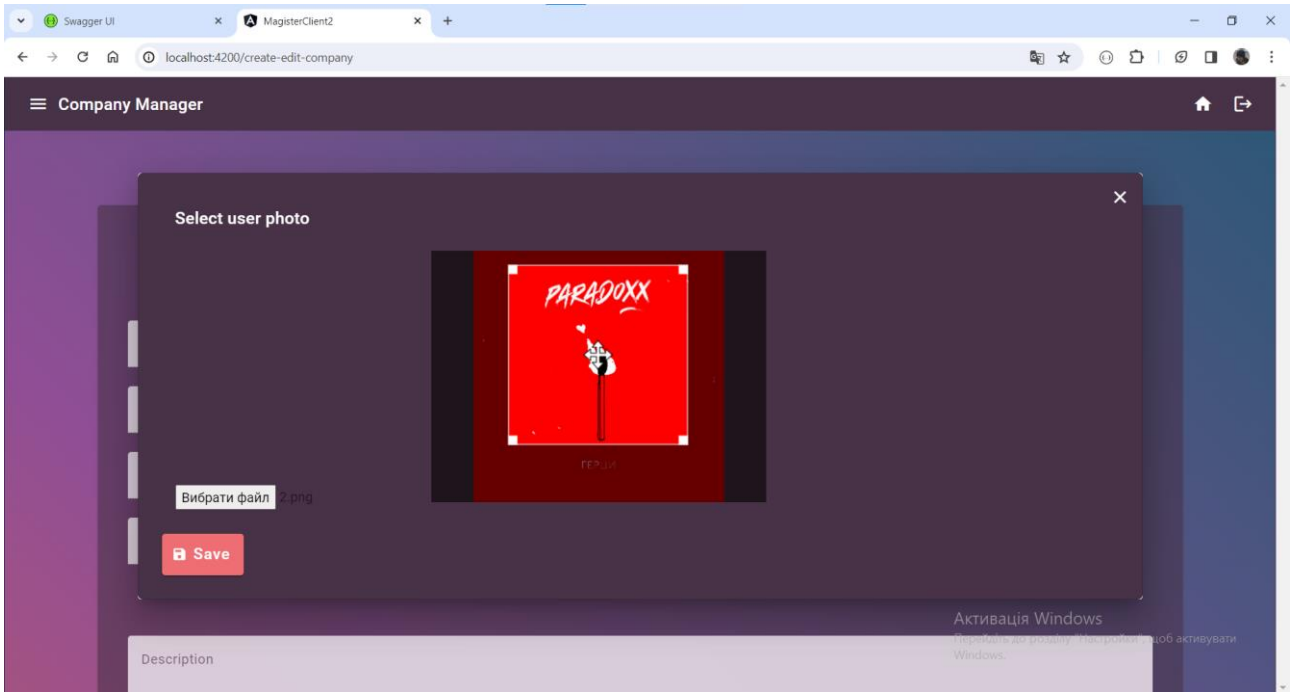


Рисунок 3.8.21. Додавання зображення компанії

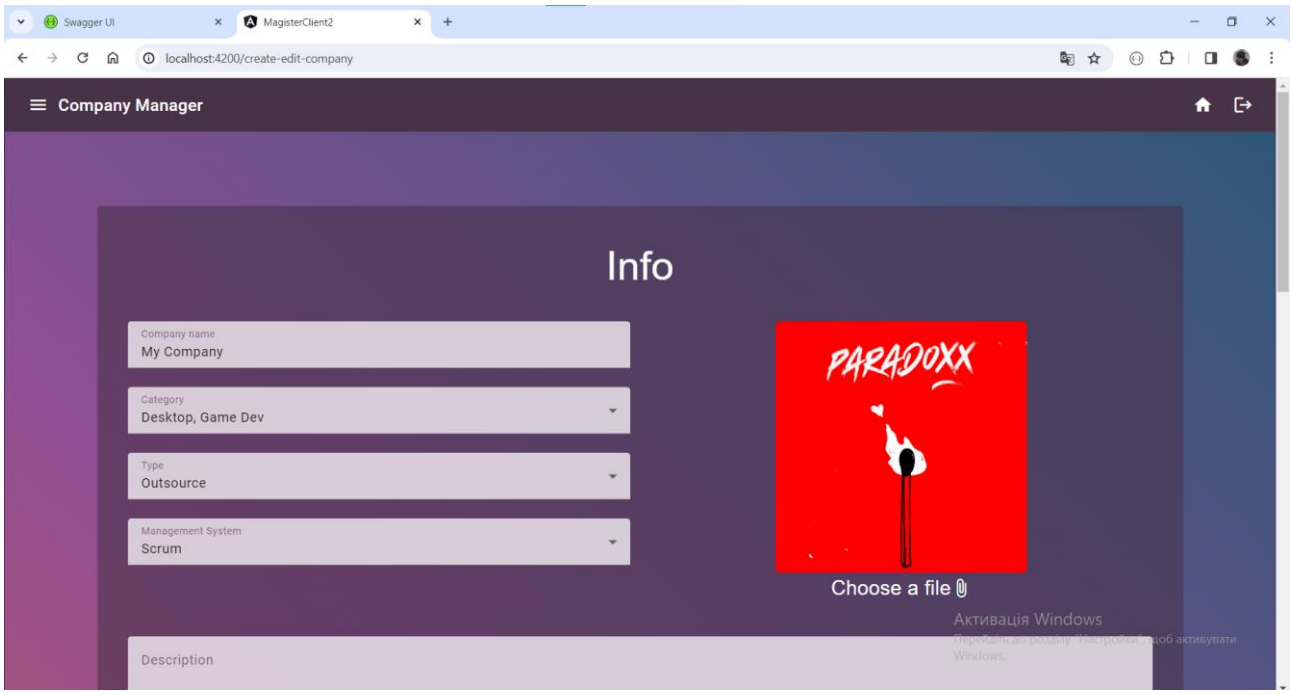


Рисунок 3.8.22. Сторінка редагування компанії

7. Створення проекту:

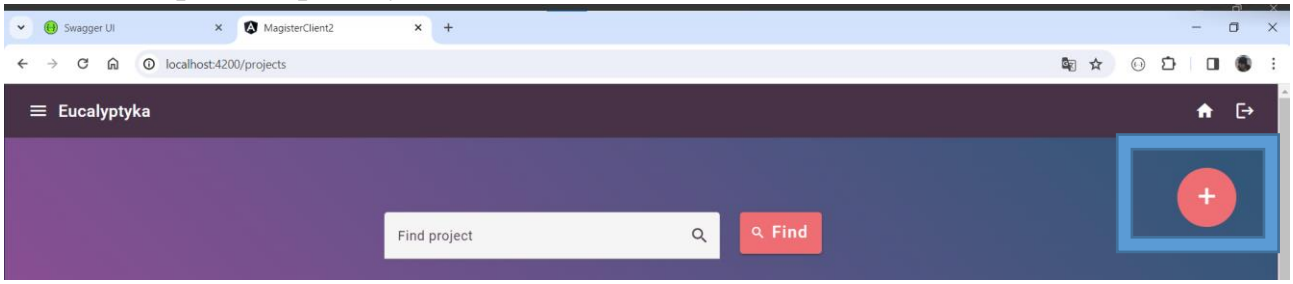


Рисунок 3.8.22. Кнопка додавання нового проекту до компанії

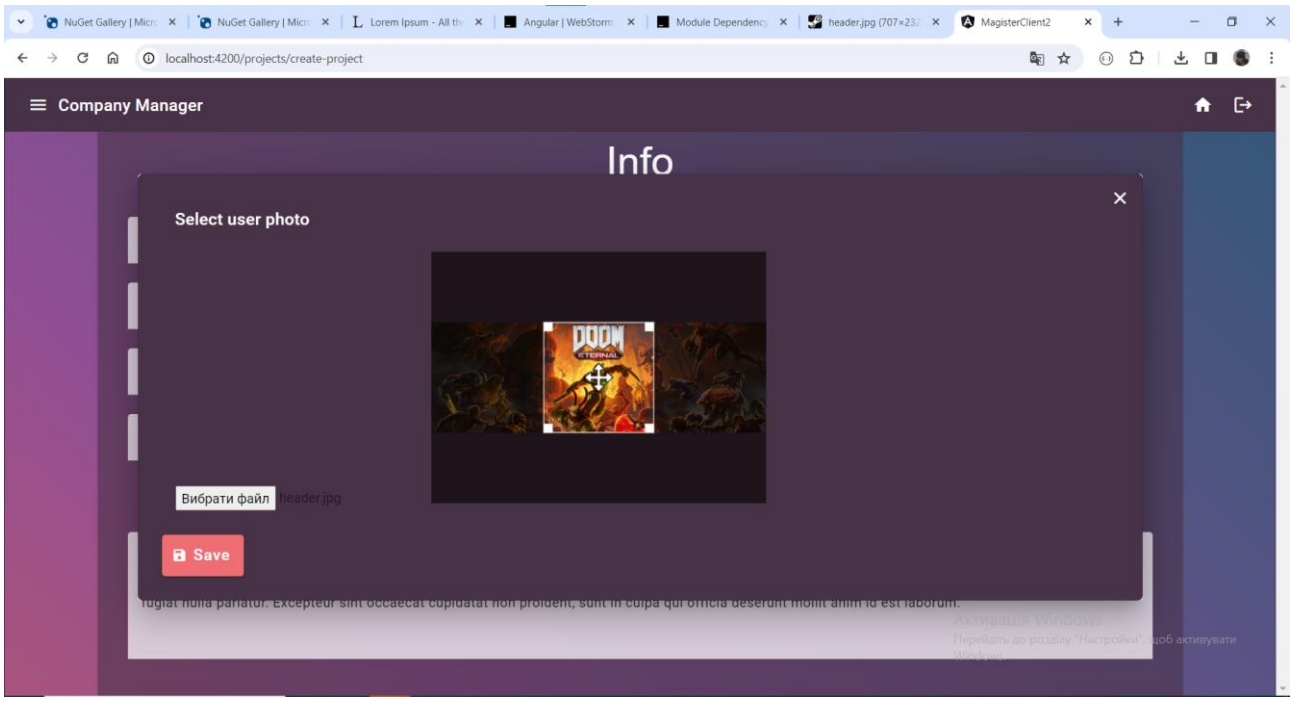


Рисунок 3.8.23. Додавання зображення проекту

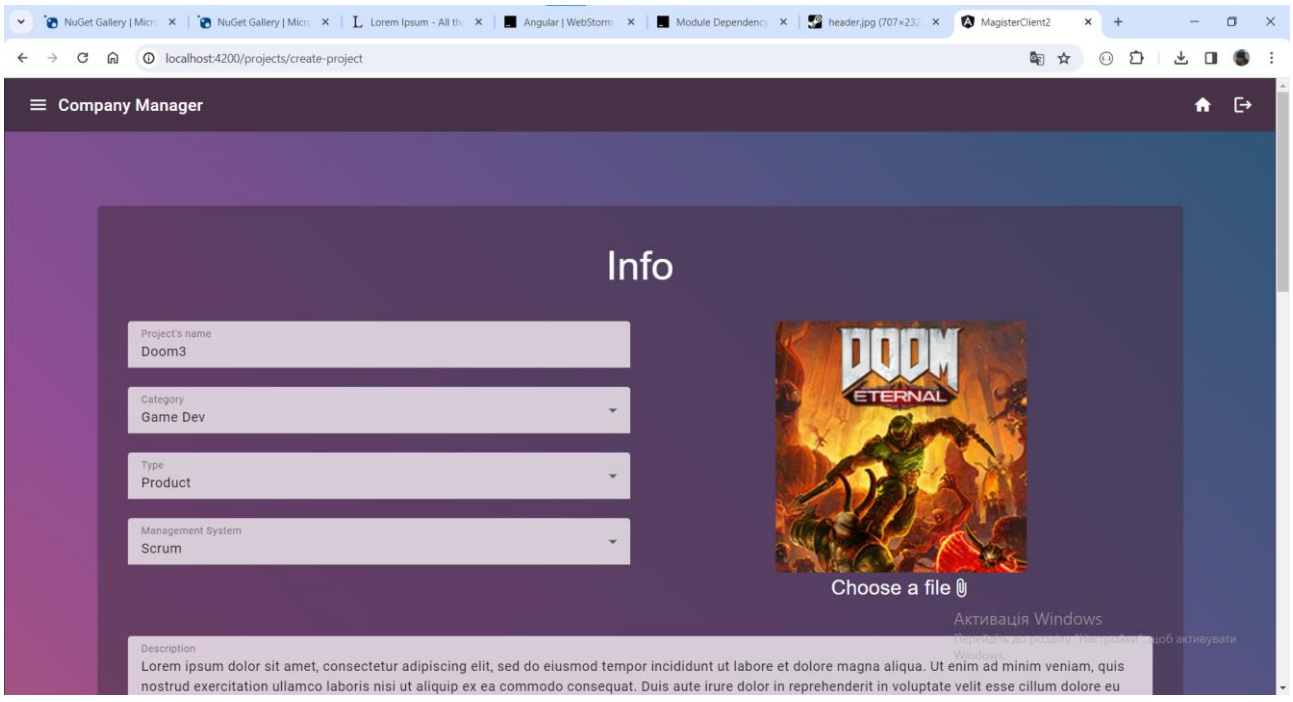


Рисунок 3.8.24. Вікно редагування проекту

Додавання учасників у команду роботи

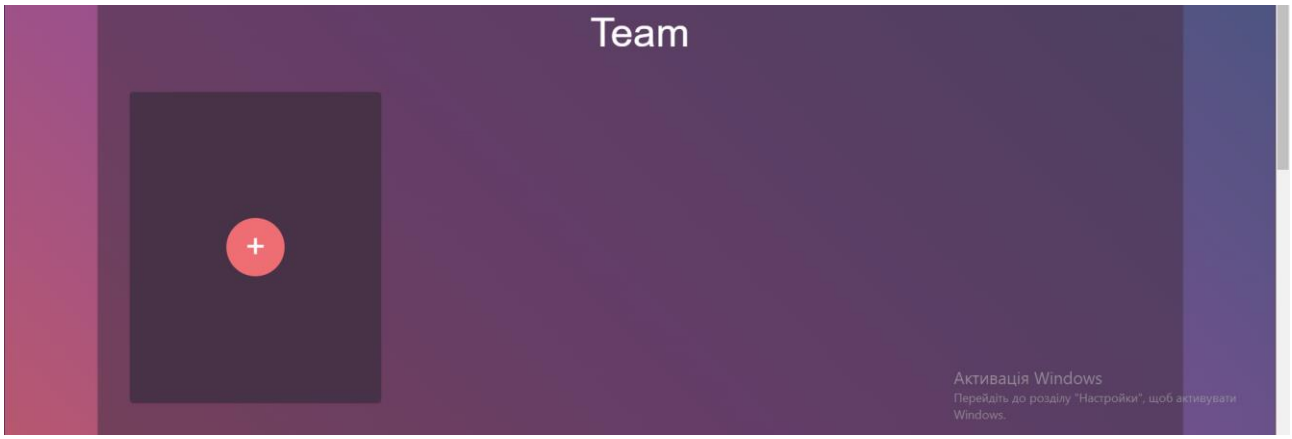


Рисунок 3.8.25. Блок залучення працівників до проекту

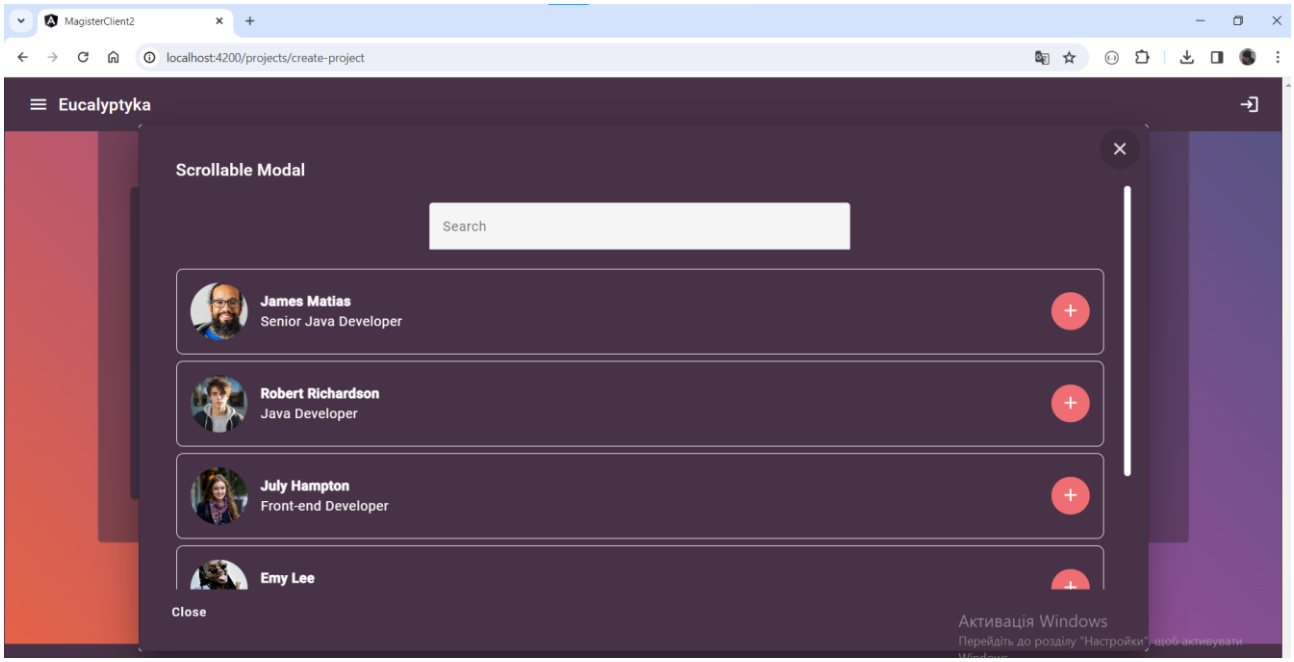


Рисунок 3.8.26. Список доступних працівників

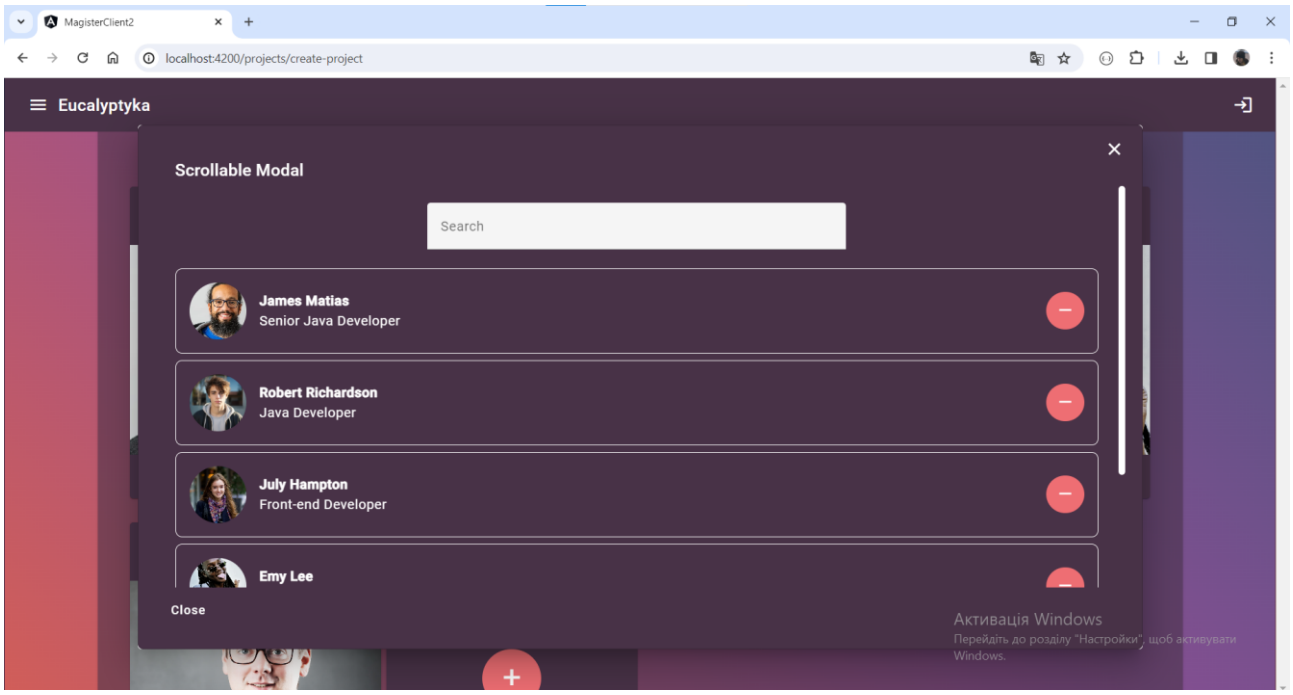


Рисунок 3.8.27. Додавання працівників до проекту

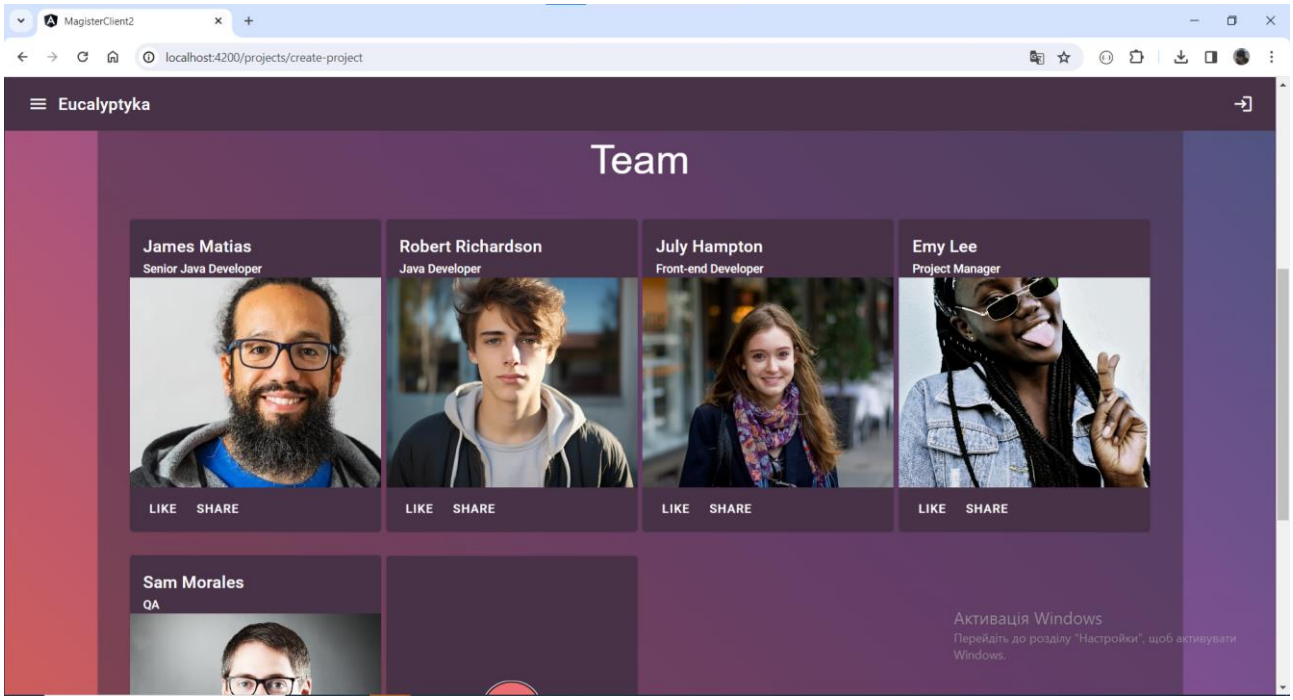


Рисунок 3.8.28. Список працівників, залучених до проекту

Додавання технологій на проект

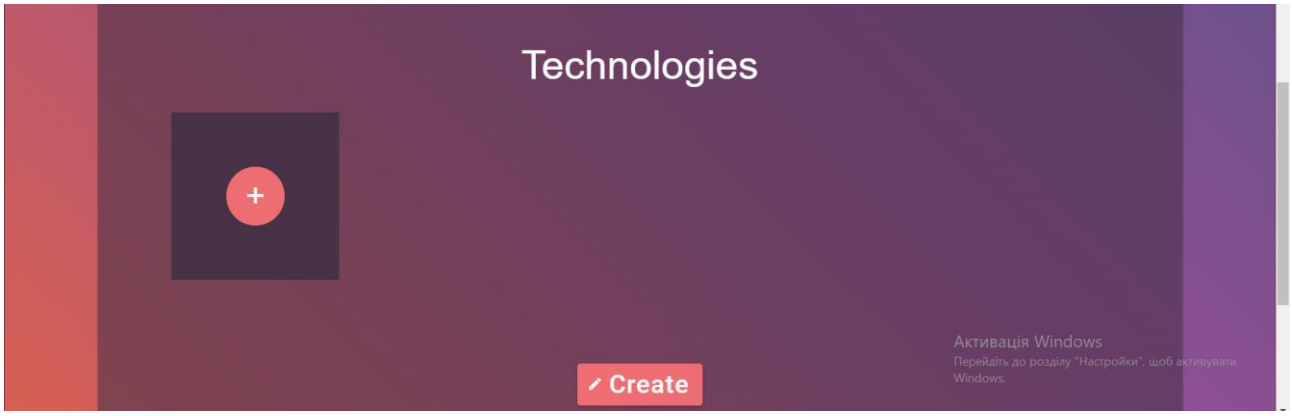


Рисунок 3.8.29. Блок додавання технологій до проекту

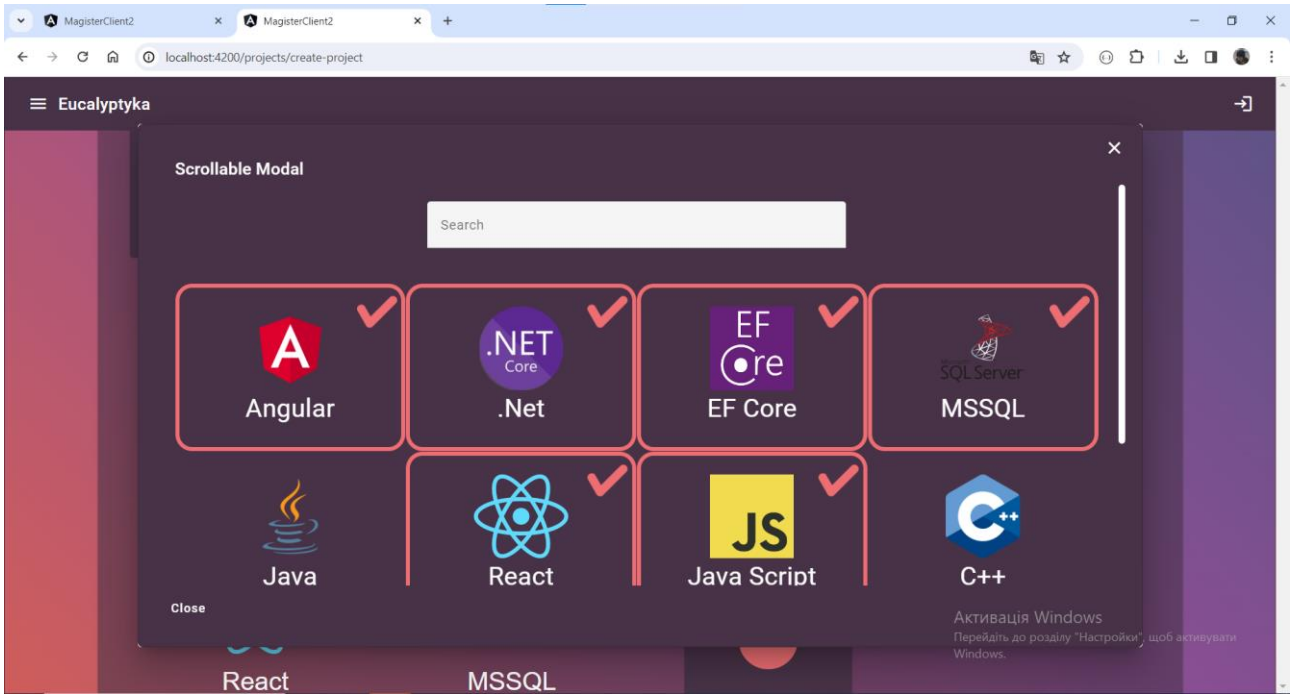


Рисунок 3.8.30. Додавання можливих технологій до проекту

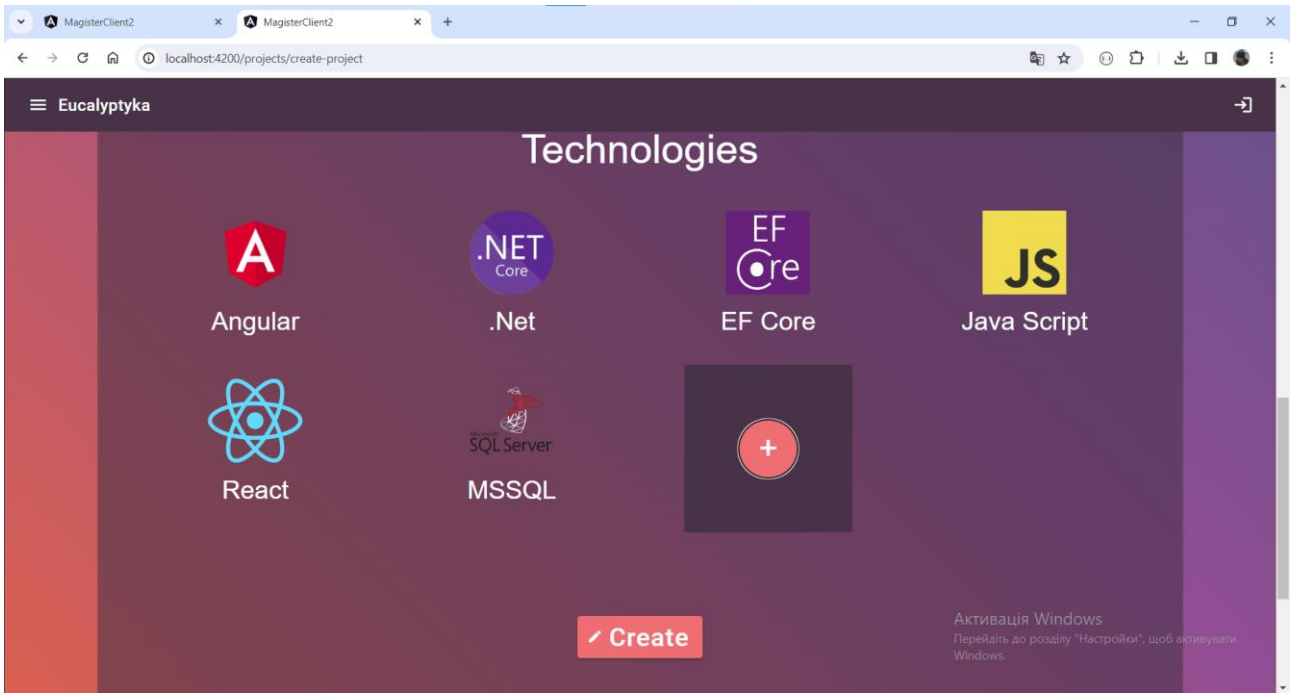


Рисунок 3.8.31. Список технологій на проекті

Сторінка проектів компанії.

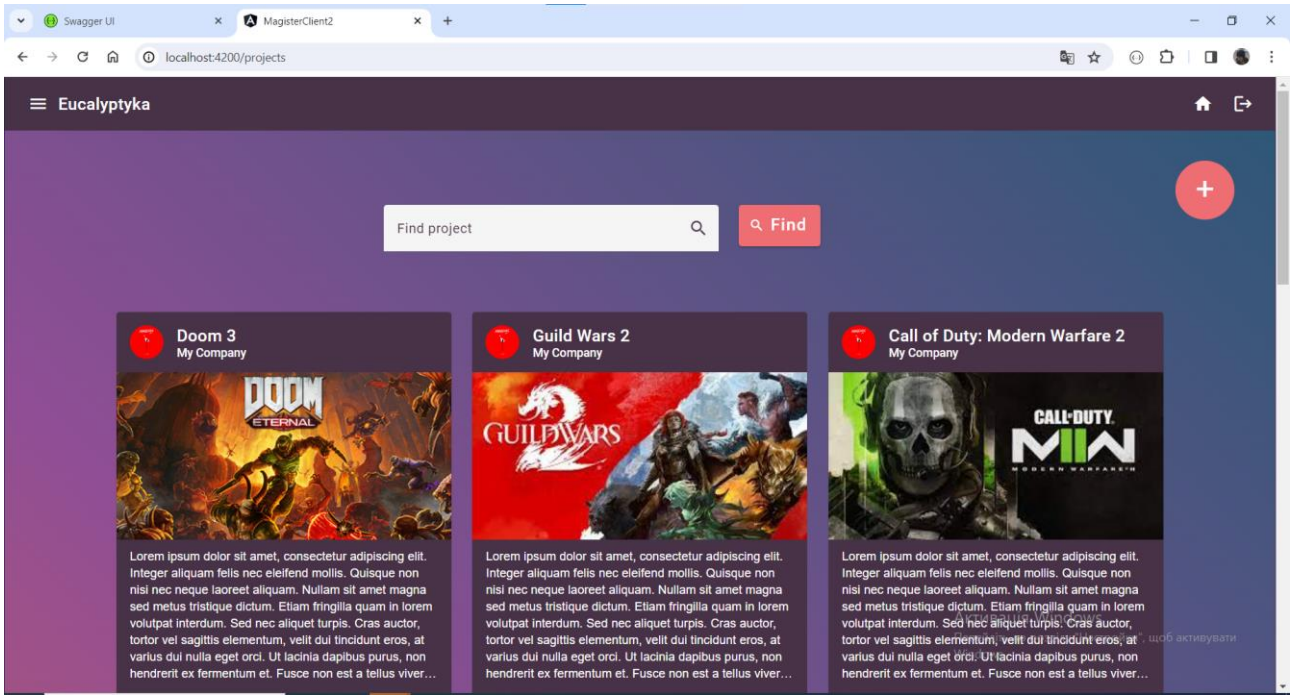


Рисунок 3.8.32. Вікно проектів компанії

Встановлення кількості колонок

Список mat-grid повинен визначати атрибут **cols**, який встановлює кількість стовпців у сітці. Кількість рядків буде автоматично визначено на основі кількості стовпців і кількості елементів.

Встановлення висоти рядка

Висоту рядків у списку сітки можна встановити за допомогою атрибута **rowHeight**. Висоту рядка для списку можна обчислити трьома способами:

1) Фіксована висота: висота може бути в px, em або rem. Якщо одиниці вимірювання не вказано, приймаються одиниці px (наприклад, 100px, 5em, 250).

2) Співвідношення: це співвідношення ширина стовпця:висота рядка, яке має передаватися через двокрапку, а не десяткову дробу (наприклад, 4:3).

3) Fit: встановлення висоти рядка відповідно до розміру Цей режим автоматично ділить доступну висоту на кількість рядків. Будь ласка, зверніть увагу на висоту сітки-списку або його контейнера.

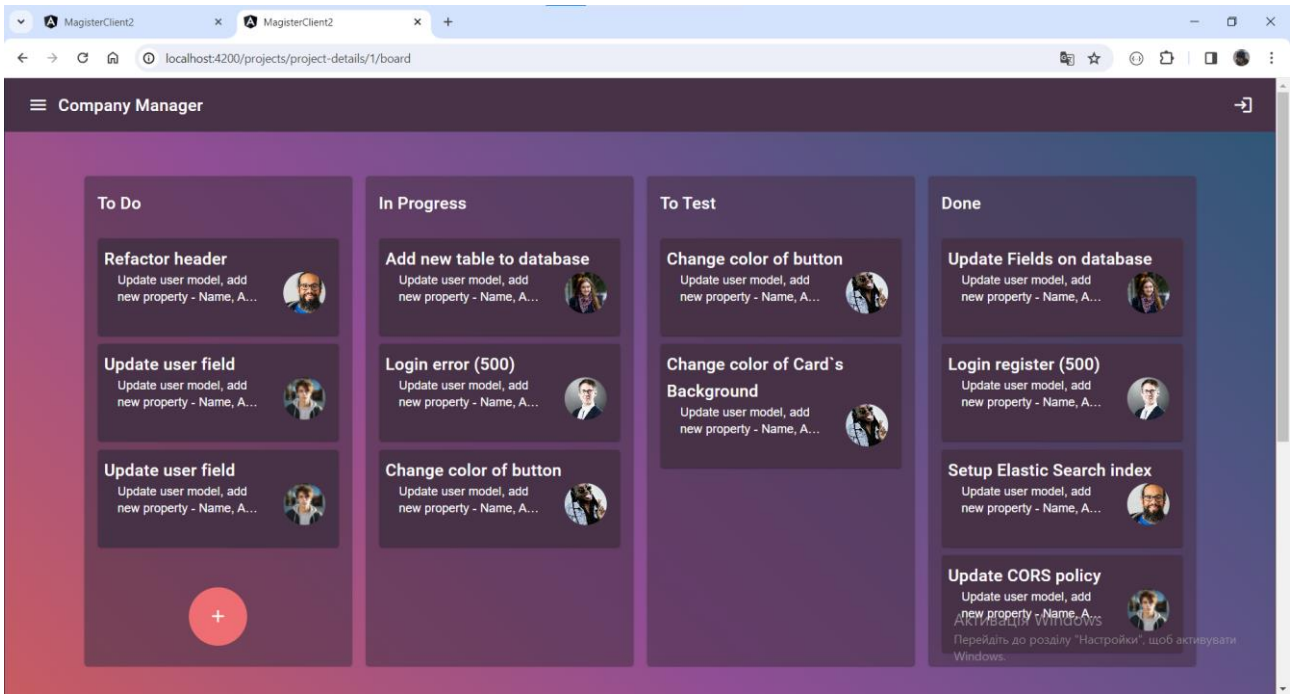


Рисунок 3.8.33. Дошка завдань на проекті

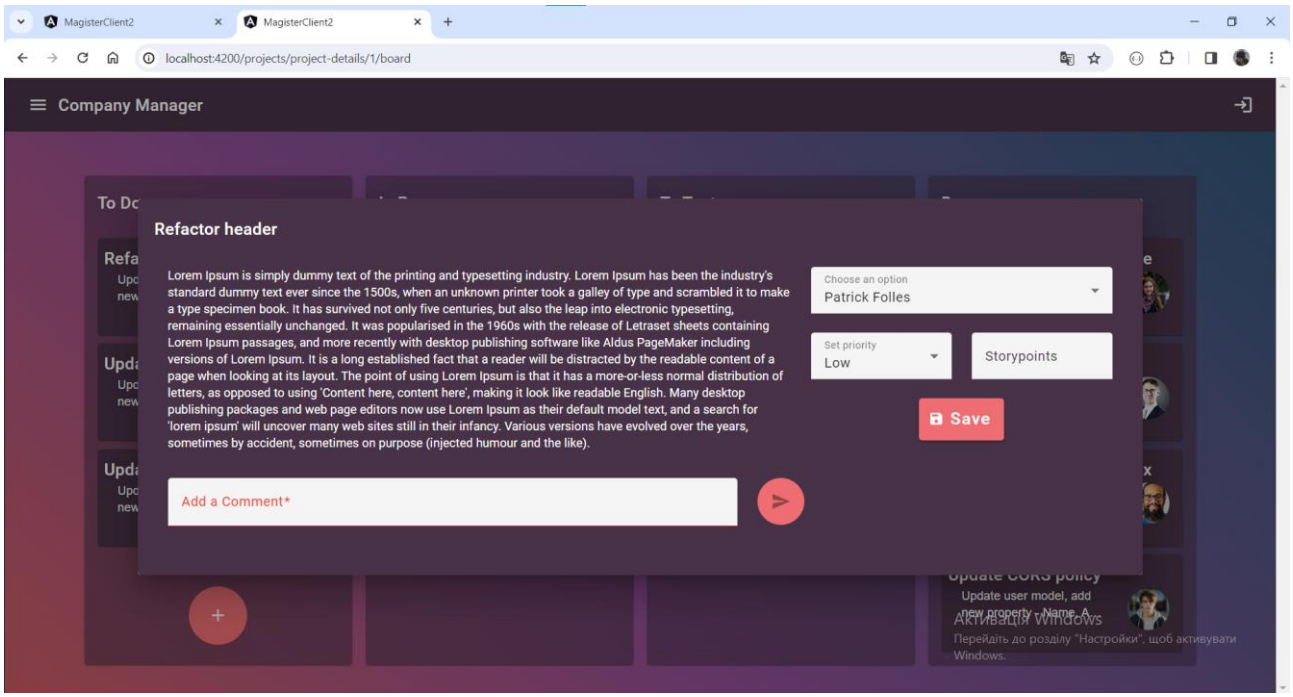


Рисунок 3.8.34. Вікно конкретного завдання на проекті

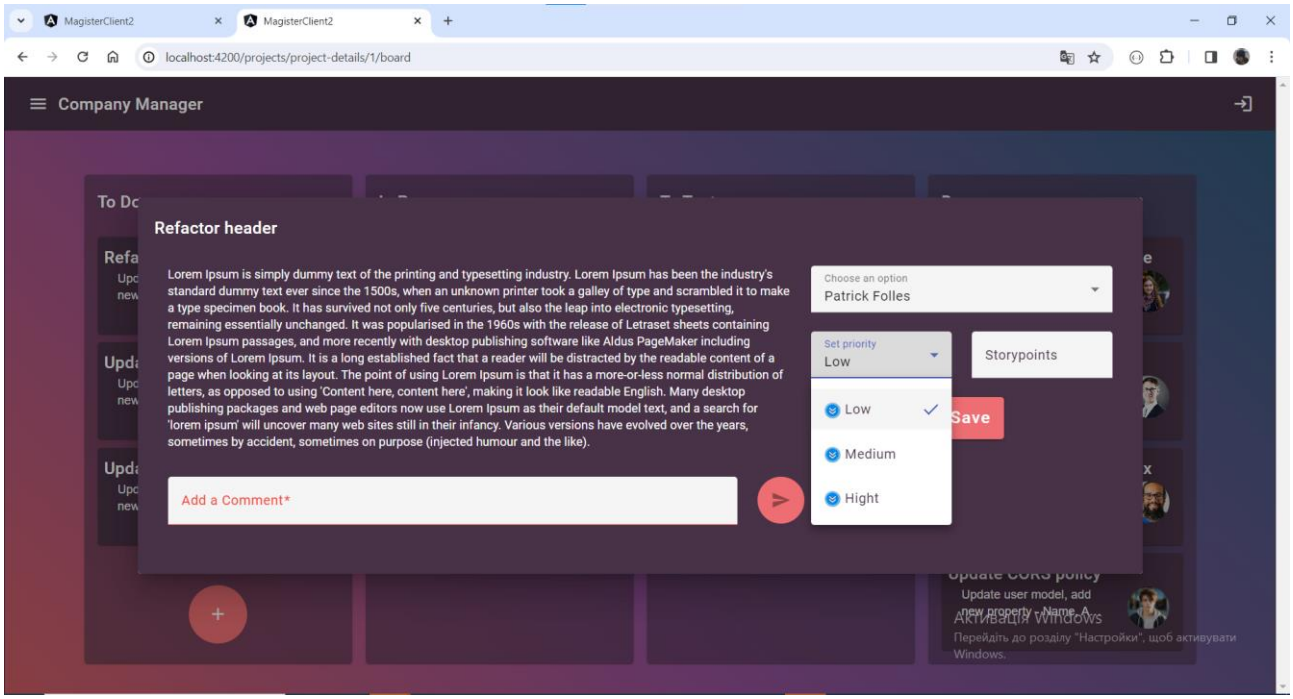


Рисунок 3.8.35. Виставляння пріоритетності даній задачі

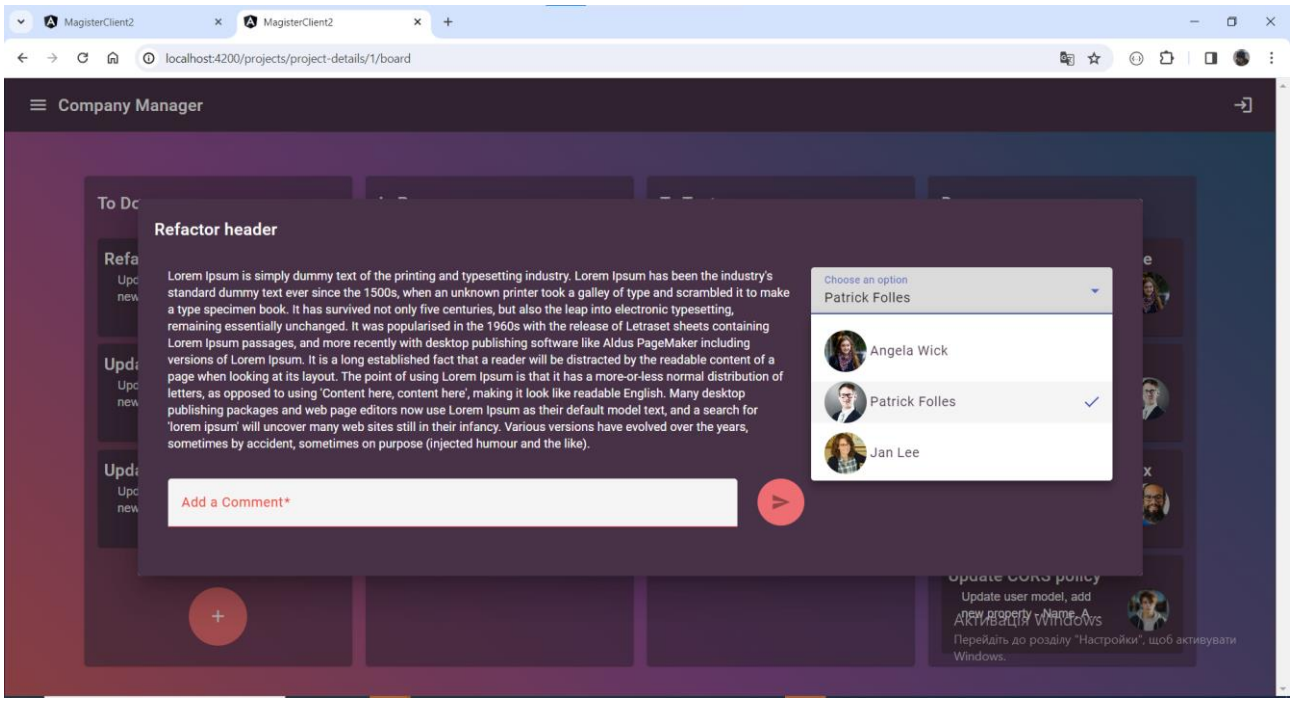


Рисунок 3.8.36. Залучення конкретного працівника до завдання

Висновки до розділу

На зважаючи не весь оверхед який несе за собою використання Entity Framework Core та Angular, даний вибір є цілком оправданий в умовах проектів типу ентерпрайз, за рахунок інкапсолювання готової інфраструктури, зрозумілих інтерфейсів розробки та здатності до різних видів поліморфізму.

РОЗДІЛ 4. Розроблення Стартап-проекту

Веб-сервіс для менеджменту компанії на базі .NET та Angular

1. **Опис ідеї проєкту:**

Зміст ідеї:

Стартап-проєкт розробляє веб-сервіс для ефективного управління компанією. Ідея полягає в створенні інтегрованої платформи, яка об'єднує функції управління завданнями, планування ресурсів та моніторингу виконання завдань, забезпечуючи комунікацію між командами в режимі реального часу.

Можливі напрямки застосування:

Даний веб-сервіс призначений для компаній різних розмірів, незалежно від галузі діяльності. Він спрощує планування проектів, ведення обліку завдань та підвищує ефективність командної роботи.

Основні вигоди для користувача:

- Зручне управління завданнями та ресурсами.
- Підвищення ефективності командної роботи.
- Прозорість та контроль над процесами.
- Можливість реагування на зміни в реальному часі.

Відмінність від існуючих аналогів:

Даний веб-сервіс вирізняється використанням технологій .NET для серверної частини, ASP.NET 6 та Entity Framework для роботи з базою даних, Identity для автентифікації, та Angular для клієнтської частини. Це гарантує високу швидкодію, безпеку та масштабованість системи.

2. **Аналіз технологічних можливостей реалізації ідей проєкту:**

Технологічна здійсненність ідеї:

Виготовлення товару: Використання технологій .NET для серверної частини, ASP.NET 6 та Entity Framework для роботи з базою даних. Використання Angular для реалізації клієнтської частини.

Технології виготовлення: Використання існуючих технологій .NET, ASP.NET та Angular, які є добре розповсюдженими та мають велику спільноту розробників.

Доступність технологій: Вибір технологій, таких як .NET, що є доступними та мають широку підтримку на ринку.

3. **Аналіз ринкових можливостей запуску стартап-проєкту:**

Визначення ринкових можливостей:

- Аналіз попиту та динаміки розвитку ринку для ефективного планування стратегії впровадження.

- Визначення потенційних груп клієнтів та їх характеристик.
- Аналіз ринкового середовища, факторів, що сприяють або перешкоджають впровадженню.

Розроблення ринкової стратегії:

Визначення стратегії охоплення ринку: концентрований, диференційований чи масовий маркетинг.

Розроблення маркетингової програми, враховуючи конкурентні переваги, цінності та потреби клієнтів.

Висновки:

Після проведеного аналізу визначається можливість ринкової комерціалізації проекту, перспективи впровадження, конкурентоспроможність та доцільність подальшого розвитку.

Рекомендована література:

"Інвестування стартап проектів в Україні" - Гук О.В., Мохонько Г.А.

"Технології розробки стартапів" - Подольчак Н. Ю., Шаповалова Т. В.

"Священна книга стартапера" - Стів Бланк, Боб Дорф.

"Startup Toolkit: 30 корисних посилань для стартапу" - https://uaspectr.com/2021/11/24/startup-toolkit-30-korysnyh-posylan/

https://platforma-msb.org/shho-neobhidno-vrahuvaty-dlya-uspishnogo/

https://octavacapital.ua/biznes-plan-dlya-startapu-vid-ideyi-do-uspishnoyi-kompaniyi/

ВИСНОВКИ

Було розроблено веб-орієнтовану систему для спрощення, уніфікації та автоматизації процесів менеджменту компанії. Реалізувано всю необхідну інфраструктуру даного сервісу за допомогою стеку технологій .NET та Angular, зокрема:

1) архітектурну концепцію “Клієнт-Сервер”;
2) База даних;
3) Модуль Domain, що міститить всі моделі даних даної системи;
4) Модуль Repository, що міститить методи доступу до даних з бази даних;
5) Модуль Infrastructure, що містить логіку опрацювання даних;
6) REST API;
7) Автентифікацію на основі JWT;
8) Клієнтську частину на основі принципів роботи SPA, за допомогою технології Angular;

Як бачимо, в умовах сучасного ринку IT, архітектурний стиль REST є невідємною частиною сучасного WEB за рахунос своєї гнучкості та, в певній мірі, доступності для розробника програмного забезпечення. Принципи на які він спирається добре підкреслюють тези які описував у своїх роботах немало відомий Роберт Мартін, який, на мою думку, якщо не сформував, то суттєво вплинув на формування сучасного бачення програмування з орієнтацією на низькозвязність, абстракцію та легкозамінність компонентів інфраструктури програмного забезпечення.

ЛІТЕРАТУРА

[1]. Роберт Мартін. Чиста Архітектура: Мистецтво створення програмного забезпечення. Видання друге / пер. з англ. І. Бондар-Терещенко. – Вид-во «Ранок»: Фабула, 2023. – 368 с.

[2]. Роберт Мартін. Чистий код: створення і рефакторинг за допомогою Agile. Видання друге / пер. з англ. І. Бондар-Терещенко. – Вид-во «Ранок»: Фабула, 2022. – 448 с.

[3]. Роберт Мартін. Чистий Agile: назад до основ/ пер. з англ. І. Бондар-Терещенко. – Вид-во «Ранок»: Фабула, 2021. – 416 с.

[4]. Jeffrey Richter CLR via C#, Publisher(s): Microsoft Press, 2012

[5]. Judith Bishop C# 3.0 Design Patterns O'Reilly Media, Inc, 2007

[6]. Vaskaran Sarcar Design Patterns in C#: A Hands-on Guide with Real-World Examples, Apress, 2018

[7]. Marinko Spasojevic JWT Authentication in ASP.NET Core Web API [Електронний ресурс]: [сайт]. – Режим доступу: https://code-maze.com/authentication-aspnetcore-jwt-1

[8]. Rick Anderson, Steve Smith ASP.NET Core Middleware [Електронний ресурс]: [сайт]. – Режим доступу: https://docs.microsoft.com/en-us/aspnet/core/fundamentals/middleware/?view=aspnetcore-6.0

[9]. Marinko Spasojevic JWT Authentication in ASP.NET Core Web API [Електронний ресурс]: [сайт]. – Режим доступу: https://code-maze.com/authentication-aspnetcore-jwt-1

ДОДАТКИ

Даний проект є open-source проектом. Весь код даного застосунку можна переглянути за посиланнями

1) Серверна частина: https://github.com/zadrot1995/Magister
2) Клієнтська частина: https://github.com/zadrot1995/MagisterClient2

```
using Domain.DTOs;
using Domain.Models;
using Infrastructure.Interfaces;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;

namespace API.Controllers
{
   [Route("api/[controller]")]
   [ApiController]
   public class AccountController : ControllerBase
   {
      private readonly IAccountService _accountService;

      public AccountController(IAccountService accountService)
      {
         _accountService = accountService;
      }

      [HttpGet]
      public async Task<ActionResult<User>> GetUser()
      {
         return Ok(_accountService.GetCurrentUser());
      }

      [HttpPost("register")]
      public async Task<ActionResult<User>> Register([FromBody] RegisterModel model)
      {
         if (ModelState.IsValid)
         {
            return await _accountService.Register(model);
         }
         return BadRequest();
      }

      [HttpPost("login")]
      public async Task<ActionResult<User>> Login([FromBody] LoginModel model)
      {
```

```
            if (ModelState.IsValid)
            {
                return Ok(await _accountService.Login(model));
            }
            return BadRequest();
        }

        [HttpPost("logout")]
        public async Task<IActionResult> Logout()
        {
            await _accountService.Logout();
            return Ok();
        }

    }
}
using Azure;
using Domain.DTOs;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Diagnostics;
using Microsoft.AspNetCore.Mvc;
using System.Collections.Generic;

namespace API.Controllers
{
    [AllowAnonymous]
    [ApiExplorerSettings(IgnoreApi = true)]
    public class ErrorsController : ControllerBase
    {
        [Route("error")]
        public ErrorResponse Error()
        {
            var context = HttpContext.Features.Get<IExceptionHandlerFeature>();
            var exception = context.Error; // Your exception
            var code = 500; // Internal Server Error by default

            if (exception is HttpStatusException)
                code = (int) ((HttpStatusException) exception).Status; // Not Found


            Response.StatusCode = code; // You can use HttpStatusCode enum instead

            return new ErrorResponse(exception, code); // Your error model
        }
    }
}
using Domain.Dtos;
```

```
using Infrastructure.Interfaces;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Repository.DbContexts;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace API.Controllers
{
   [Route("api/[controller]")]
   [ApiController]
   public class TokenController : ControllerBase
   {
      private readonly ApplicationDbContext _userContext;
      private readonly ITokenService _tokenService;

      public TokenController(ApplicationDbContext userContext, ITokenService
tokenService)
      {
         this._userContext = userContext ?? throw new
ArgumentNullException(nameof(userContext));
         this._tokenService = tokenService ?? throw new
ArgumentNullException(nameof(tokenService));
      }

      [HttpPost]
      [Route("refresh")]
      public IActionResult Refresh(TokenApiModel tokenApiModel)
      {
         if (tokenApiModel is null)
            return BadRequest("Invalid client request");

         string accessToken = tokenApiModel.AccessToken;
         string refreshToken = tokenApiModel.RefreshToken;

         var principal = _tokenService.GetPrincipalFromExpiredToken(accessToken);
         var username = principal.Identity.Name; //this is mapped to the Name claim by
default

         var user = _userContext.Users.SingleOrDefault(u => u.UserName == username);

         if (user is null  user.RefreshToken != refreshToken  user.RefreshTokenExpiryTime
<= DateTime.Now)
            return BadRequest("Invalid client request");
```

```
            var newAccessToken = _tokenService.GenerateAccessToken(principal.Claims);
            var newRefreshToken = _tokenService.GenerateRefreshToken();

            user.RefreshToken = newRefreshToken;
            _userContext.SaveChanges();

            return Ok(new AuthenticatedResponse()
            {
                Token = newAccessToken,
                RefreshToken = newRefreshToken
            });
        }

        [HttpPost, Authorize]
        [Route("revoke")]
        public IActionResult Revoke()
        {
            var username = User.Identity.Name;

            var user = _userContext.Users.SingleOrDefault(u => u.UserName == username);
            if (user == null) return BadRequest();

            user.RefreshToken = null;

            _userContext.SaveChanges();

            return NoContent();
        }
    }
}
using Domain.Dtos;
using Infrastructure.Interfaces;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Repository.DbContexts;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace API.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class TokenController : ControllerBase
```

```
    {
        private readonly ApplicationDbContext _userContext;
        private readonly ITokenService _tokenService;

        public TokenController(ApplicationDbContext userContext, ITokenService
tokenService)
        {
            this._userContext = userContext ?? throw new
ArgumentNullException(nameof(userContext));
            this._tokenService = tokenService ?? throw new
ArgumentNullException(nameof(tokenService));
        }

        [HttpPost]
        [Route("refresh")]
        public IActionResult Refresh(TokenApiModel tokenApiModel)
        {
            if (tokenApiModel is null)
                return BadRequest("Invalid client request");

            string accessToken = tokenApiModel.AccessToken;
            string refreshToken = tokenApiModel.RefreshToken;

            var principal = _tokenService.GetPrincipalFromExpiredToken(accessToken);
            var username = principal.Identity.Name; //this is mapped to the Name claim by
default

            var user = _userContext.Users.SingleOrDefault(u => u.UserName == username);
if (user is null  user.RefreshToken != refreshToken  user.RefreshTokenExpiryTime <=
DateTime.Now)
                return BadRequest("Invalid client request");

            var newAccessToken = _tokenService.GenerateAccessToken(principal.Claims);
            var newRefreshToken = _tokenService.GenerateRefreshToken();

            user.RefreshToken = newRefreshToken;
            _userContext.SaveChanges();

            return Ok(new AuthenticatedResponse()
            {
                Token = newAccessToken,
                RefreshToken = newRefreshToken
            });
        }

        [HttpPost, Authorize]
        [Route("revoke")]
```

```
        public IActionResult Revoke()
        {
            var username = User.Identity.Name;

            var user = _userContext.Users.SingleOrDefault(u => u.UserName == username);
            if (user == null) return BadRequest();

            user.RefreshToken = null;

            _userContext.SaveChanges();

            return NoContent();
        }
    }
}
```