Хмельницький національний університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

# КВАЛІФІКАЦІЙНА РОБОТА

Веб-застосунок для керування проєктами будівельної компанії
Назва теми

Рівень вищої освіти Перший (бакалаврський)

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр КвРІПЗ.190126.19.02.ПЗ

Виконав студент IV курсу, група ІПЗ-19-1 _____ Валігура О.Г.
Підпис Ініціали, прізвище

Керівник Канд. Тех. Наук, доцент _____ Яшина О.М.
Науковий ступінь, звання Підпис Ініціали, прізвище

Нормоконтролер Канд. Пед. Наук, доцент _____ Праворська Н.І.
Науковий ступінь, звання Підпис Ініціали, прізвище

**До захисту допускаю:**

Завідувач кафедри інженерії
програмного забезпечення _____ Л. П. Бедратюк
Підпис Ініціали, прізвище

5 червня 2023 р.

Хмельницький 2023

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій
Кафедра Інженерії програмного забезпечення
Рівень вищої освіти Перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Л. П. Бедратюк
02 01 2023 р.

# ЗАВДАННЯ
# НА КВАЛІФІКАЦІЙНУ РОБОТУ

Валігурі Олексію Григоровичу

Прізвище, ім'я, по батькові студента

1. Тема кваліфікаційної роботи Веб-застосунок для керування проєктами будівельної компанії

Керівник кваліфікаційної роботи Яшина Оксана Миколаївна, кандидат технічних наук, доцент

Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 05.02.2021 р. № 11

2. Строк подання студентом роботи на кафедру 01.06.2023 р.

3. Вихідні дані до роботи Матеріали переддипломної практики

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити) *Характеристика предметної області та постановка задачі, проектування програмного забезпечення, програмна реалізація та тестування.*

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень) Три креслення: діаграма відношень сутностей бази даних, діаграма класів запиту деталей проєкту, діаграма реактивної архітектури front-end частини застосунку

6. Консультанти розділів кваліфікаційної роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| Нормоконтроль | Праворська Наталья Іванівна, доцент | 01.06 | 01.06 |
| Антиплагіат | Гурман Іван Васильович, доцент | 02.06. | 02.06 |

7. Дата видачі завдання «_02_» __січня_ 2023 р.

КАЛЕНДАРНИЙ ПЛАН

| | | |
|---|---|---|
| 1. Збір матеріалу за темою кваліфікаційної роботи (КвР), дослідження предметної області, в якій планується використання програмного забезпечення (ПЗ), визначення здач та вимог, розробка технічного завдання | 02.01–31.01 2023 | |
| 2. Проектування програмного забезпечення | 01.02–28.02 2023 | |
| 3. Програмна реалізація програмного забезпечення | 01.03–10.04 2023 | |
| 4.Тестування програмного забезпечення | 11.04–30.04 2023 | |
| 5. Написання вступу, загальних висновків, оформлення переліку джерел та посилання на додатків. Оформлення пояснювальної записки КвР згідно вимог | 01.05–25.05 2023 | |
| 6. Підготовка до захисту, написання доповіді для захисту | 26.05–31.05.2023 | |

Студент _____ Валігура О.Г.
Підпис Ініціали, прізвище

Керівник роботи _____ Яшина О.М.
Підпис Ініціали, прізвище

# АНОТАЦІЯ

Тема кваліфікаційної роботи: «Веб-застосунок для керування проєктами будівельної компанії»

Автор роботи: Валігура Олексій Григорович

Керівник роботи: к.т.н., доцент Яшина Оксана Миколаївна

Пояснювальна записка: 81 с., 27 рис., 2 додатки., 40 джерел.

Графічна частина: 3 діаграми.

ВЕБ-ЗАСТОСУНОК ДЛЯ КЕРУВАННЯ ПРОЄКТАМИ БУДІВЕЛЬНОЇ КОМПАНІЇ

Мета кваліфікаційної роботи: розробка веб-застосунку для керування проєктами будівельної компанії.

У кваліфікаційній роботі проведено аналіз предметної області та її інформаційного забезпечення, визначені функціональні вимоги до програмної системи, розроблена архітектура застосунку, спроєктована структура бази даних із використанням ER діаграми. Для розробки програмного продукту використано технології ASP.NET Core, Angular, Mediatr, NgRx Visual Studio та WebStorm. За допомогою цих засобів розроблено програмне забезпечення для керування проєктами будівельної компанії.

Впровадження розробленого програмного продукту дозволяє автоматизувати ведення проєктів компанії, ведення їх бюджету, матеріалів, найманих працівників та значно полегшити працю менеджменту компанії.

01.06.2023

Дата

Підпис

# ВІДОМІСТЬ ДОКУМЕНТІВ

| № рядка | Формат | Позначення документа | Найменування документа | К–сть аркушів | № екз. | Примітка |
|---|---|---|---|---|---|---|
| | | | | | | |
| | | | Текстові документи | | | |
| | | | | | | |
| 1 | А4 | КвРІПЗ.190126.19.02.ПЗ | Пояснювальна записка | 81 | | |
| 2 | А4 | | Завдання на | 1 | | |
| | | | кваліфікаційну роботу | | | |
| 3 | А4 | | Анотація | 1 | | |
| | | | | | | |
| | | | Графічні документи | | | |
| | | | | | | |
| 4 | А3 | КвРІПЗ.190126.19.02.Е8 | Діаграма відношень | 1 | | |
| | | | сутностей бази даних | | | |
| | | | | | | |
| 5 | А3 | КвРІПЗ.190126.19.02.Е8 | Діаграма класів запиту | 1 | | |
| | | | деталей проєкту | | | |
| | | | | | | |
| 6 | А3 | КвРІПЗ.190126.19.02.Е8 | Діаграма реактивної | 1 | | |
| | | | архітектури front-end | | | |
| | | | частини | | | |
| | | | | | | |
| | | | | | | |

| Змн. | Арк. | № докум. | Підпис | Дата | КвРІПЗ.190126.19.02.ПЗ | | |
|---|---|---|---|---|---|---|---|
| Виконав | | Валігура О.Г. | | 01.06 | *Веб–застосунок для керування проєктами будівельної компанії* | Літ. | Арк. 1 / Аркушів 81 |
| Керівник. | | Яшина О.М. | | 01.06 | | | |
| Нормоконтроль | | Праворська Н.І. | | 01.06 | | ХНУ, ІПЗ–19–1 | |
| Зав.кафедри | | Бедратюк Л.П. | | 5.06 | | | |

## ЗМІСТ

# ВСТУП

Веб-застосунки зробили революцію у роботі компаній і їх спілкуванню зі своїми клієнтами, постачальниками та співробітниками. І будівельна галузь тут не є винятком, адже менеджмент проєктів у ній є ключовим аспектом, який вимагає ефективної координації та комунікації між усіма зацікавленими сторонами. Використання веб-застосунків для управління проєктами в будівельному секторі може підвищити продуктивність, зменшити витрати та підвищити загальну ефективність компанії.

Управління проєктами в будівельній галузі передбачає нагляд, координацію та контроль проєкту від початку до кінця. Це включає в себе декілька етапів, таких як: проєктування, закупівлі, будівництво та експлуатацію. Кожен із цих етапів, в свою чергу, поділяється на кілька завдань, ресурсів і зацікавлених сторін, що робить управління проєктами в будівельному секторі досить складним процесом. Однак веб-застосунки можуть значно спростити даний процес, сприяючи спілкуванню та співпраці між різними зацікавленими сторонами, зменшуючи затримки та покращуючи результати проєкту.

Розробка веб-застосунку для управління проєктами в будівельному секторі вимагає глибокого розуміння галузевих технічних вимог і розуміння UX дизайну. Адже застосунок має бути зручним, ефективним та адаптованим до можливості ведення абсолютно різних проєктів та потреб усіх зацікавлених у цьому сторін. Застосунок також повинен розроблятись з урахуванням безпеки його користувачів, забезпечуючи захист комунікацій, зберігання та пошуку даних, з метою збереження конфіденційності даних.

Веб-застосунки для управління проєктами мають широкий спектр функцій, що можуть полегшити роботу на будівництві. Вони можуть як сприяти автоматизації та централізації задач, що допомагає відслідковувати прогрес проєкту, розподіляти ресурси та контролювати терміни виконання завдань. Так і сприяти обміну інформації в реальному часі, що забезпечує своєчасне усунення проблем та прийняття правильних рішень. Вони також забезпечують зручну

документацію та архівацію проєктів, що важливо для відстеження історії змін проєктів та забезпечення їх юридичної відповідносі.

Щоб забезпечити максимальну корисність та зручність використання, веб-застосунок повинен бути інтуїтивно зрозумілим та легким у використанні для всіх учасників процесу. Це включає в себе зручний інтерфейс, який забезпечує легкий доступ до всіх потрібних інструментів та даних, а також адаптацію до різних типів пристроїв, що дозволяє користувачам працювати з застосунком, незалежно від того, де вони перебувають. Велика увага повинна бути приділена також тестуванню та оптимізації застосунку щоб забезпечити його надійність та швидкодію при використанні.

Однією з головних переваг використання веб-застосунку для управління проєктами є покращена співпраця та спілкування між різними зацікавленими сторонами. За допомогою веб-застосунку керівники проєктів можуть з легкістю обмінюватися інформацією, оновленнями та звітами про хід роботи з усіма зацікавленими сторонами, включаючи підрядників, постачальників і клієнтів. Що сприяє підвищенню прозорості, швидшому прийняттю рішень і зменшенню помилок у спілкуванні.

Веб-застосунки для управління проєктами в будівельній галузі також можуть сприяти підвищенню продуктивності та ефективності. За допомогою веб-застосунку менеджери проєктів можуть автоматизувати повторювані завдання, відстежувати ресурси та контролювати прогрес у режимі реального часу. Це забезпечує своєчасну здачу проєктів, зменшує затримки та покращує якість.

Нарешті, веб-застосунки для управління проєктами в будівельному секторі можуть допомогти підвищити загальну якість і безпеку проєктів. За допомогою веб-застосунку керівники проєктів можуть контролювати дотримання стандартів безпеки, відстежувати заходи контролю якості та гарантувати, що проєкти відповідають нормативним вимогам.

При розробці сучасних веб-застосунків важливо використовувати надійні та передові технології, що забезпечують максимальну продуктивність та ефективність. Серед них – .Net та Angular, які є визнаними та широко

використовуваними в області розробки програмного забезпечення.

Мета кваліфікаційної роботи: розробка веб-застосунку для керування проєктами будівельної компанії.

Об’єкт дослідження: процес проєктування та розробки веб-застосунку для керування проєктами будівельної компанії.

Предмет дослідження: методи проєктування та розробки веб-застосунку для керування проєктами будівельної компанії.

Задачі кваліфікаційної роботи:

— розробити архітектуру та визначити функціональну структуру веб-застосунку для керування будівельними проєктами, враховуючи сучасні підходи до проєктування;

— спроєктувати користувацький інтерфейс веб-застосунку, орієнтований на зручність та ефективність роботи користувача;

— вибрати і обґрунтувати вибрані технології та методи розробки веб-застосунку, враховуючи їх стабільність, продуктивність, безпеку та зручність подальшого супроводу;

— реалізувати логіку веб-застосунку, забезпечити його інтеграційне та інтерактивне тестування, а також розробити керівництво користувача для зручності використання системи.

# 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

## 1.1 Змістовний аналіз предметної області

Ознайомлення з предметною областю є важливим кроком у процесі розробки програмного забезпечення. Вивчення предметної області допоможе ідентифікувати проблеми та невирішені питання з точки зору впровадження інформаційних технологій, автоматизації виробничих процесів, обробки та передачі інформації необхідної для даного веб-застосунку.

Галузь будівельних компаній є складною і динамічною, вона включає в себе велику кількість учасників, таких як архітектори, інженери, підрядники, будівельники та клієнти. Успішне управління будівельними проєктами вимагає великої координації, комунікації та співпраці між цими учасниками. Однак, в цій галузі є кілька проблем, які можуть призвести до затримок, перевищення витрат, низької якості та відсутності прозорості. Ці проблеми можуть бути зменшені за допомогою веб-застосунка, який полегшує комунікацію, співпрацю, управління ресурсами та забезпечує прозорість між учасниками.

Загалом можна виділити 7 основних пунктів ефективного менеджменту будівельної компанії:

— встановлення цілей. Керування будівельної компанією починається з встановлення чітких та досяжних цілей, які відповідають місії та баченню компанії. Це передбачає визначення конкретних проєктних цілей та результатів, а також встановлення ключових показників продуктивності (KPI), щоб виміряти успішність. Цілі можуть включати завершення проєкту у визначені терміни, дотримання бюджетних обмежень або надання продукту високої якості;

— створення плану. Після встановлення цілей створюється план, який визначає кроки, необхідні для їх досягнення. Він включає в себе визначення термінів проєкту, бюджетів та розподілу ресурсів. Також план проєкту може містити детальний розбір завдань, терміни, віхи та залежності від інших

компаній, а також стратегію управління ризиками для запобігання потенційних проблем або їх швидкому вирішенню;

— призначення відповідальності. Кожному члену команди призначають певні відповідальності на основі їхніх навичок та експертизи. Що забезпечує розуміння кожного члена команди, що від нього очікується і покращує ефективність людей у досягненні спільної мети. Зони відповідальності можуть включати в себе керування конкретними аспектами проєкту, такими як закупівлі, будівництво чи контроль якості, або контроль проєкту в цілому;

— моніторинг прогресу. Управління будівельною компанією передбачає постійний моніторинг прогресу, для забезпечення планового перебігу проєктів і своєчасного вирішення його проблем. Цей пункт може включати регулярні оновлення проєкту, звіти про стан виконання, зустрічі команди, генерація звітів і тому подібне. Менеджери проєктів можуть використовувати інструменти управління проєктами, такі як графіки Гантта, дошки Канбан або інформаційні панелі, щоб відстежувати прогрес та виявляти потенційні небезпечні проблеми;

— адаптація до змін. Будівельні проєкти є динамічними, під час їх виконання можуть виникати неочікувані проблеми. Ефективне управління будівельною компанією передбачає здатність адаптуватися до змін і робити необхідні корективи в плані за потреби. Це може включати перерозподіл ресурсів, перегляд термінів чи зміну обсягу проєкту. Ефективні менеджери проєктів можуть передбачати потенційні проблеми та мають запасні плани та стратегії дій для зменшення чи усунення ризику;

— забезпечення якості. Управління будівельною компанією передбачає забезпечення високої якості завершення проєктів. Це може включати регулярні перевірки якості та інспекції, а також застосування заходів контролю якості. Менеджери проєктів можуть працювати з професіоналами з контролю якості, щоб забезпечити, що продукти та послуги відповідають стандартам галузі та очікуванням клієнтів;

— забезпечення комунікації. Ефективна комунікація є ключовою для успішного управління будівельною компанією. Регулярна комунікація між членами команди, зацікавленими сторонами та клієнтами забезпечує, що всі інформовані, а будь-які проблеми вирішуються вчасно. Це може включати використовування веб-застосунків або інших інструментів комунікації, щоб сприяти співпраці та прозорості. Ефективна комунікація також передбачає активне слухання, співчуття та готовність вирішувати проблеми та виклики конструктивним та позитивним способом;

Однією ж з найбільших проблем, з якими стикаються будівельні компанії, є управління кількома проєктами одночасно. Кожен проєкт має свої вимоги, терміни виконання, бюджети та потребує постійної уваги менеджерів проєктів. В той же час, ручні методи управління проєктами, такі як електронні таблиці та ручна комунікація, часто призводять до помилок, неправильного розподілу ресурсів та поганої координації між членами проєкту. В результаті, проєкти можуть затримуватися, мати перевищення в бюджеті, а також бути недостатньо прозорими для їх вкладників.

Для вирішення цих проблем, будівельні компанії можуть використовувати процеси автоматизації. Одним із яких є веб-застосунок що проєктується. Він забезпечить централізовану платформу для управління проєктами, що дозволить головним менеджерам проєктів призначати завдання та відстежувати їх виконання у реальному часі. За допомогою цих інструментів, менеджери проєктів зможуть спостерігати за станом проєкту, розподіляти ресурси та прогнозувати потреби у бюджеті.

Крім того, веб-застосунок зможе забезпечити прозорість та комунікацію між членами команди, зацікавленими сторонами та клієнтами. Кожен з учасників проєкту може бачити статус завдань, терміни виконання та інформацію про бюджет, що дозволяє всім залишатися на одній хвилі та уникати непорозумінь. Також веб-застосунок надає можливість зберігати всю історію проєкту та

відстежувати зміни, що забезпечує прозорість та відповідальність. А також позбавляє компанію від величезної кількості необхідних документів.

Окрім безпосереднього управління проєктами, веб-застосунок може бути корисними для керування ресурсами та витратами. У будівництві, де витрати на матеріали, робочу силу та обладнання можуть значно впливати на успіх проєкту, ефективне управління ресурсами може допомогти знизити витрати та підвищити прибуток. Веб-застосунок дозволить менеджерам проєктів відстежувати витрати та ресурси в реальному часі, що дозволяє приймати правильні рішення щодо розподілу ресурсів та витрат.

## 1.2 Аналіз наявного програмно-технічного забезпечення предметної області

Для аналізу наявного програмного забезпечення в предметній області «веб-застосунок для управління проєктами будівельної компанії» необхідно ретельно оглянути програмні засоби, які в даний час використовуються в будівельній індустрії для менеджменту їх проєктів. Під час аналізу кожного програмного продукту важливо враховувати наступні фактори:

— мета програмного продукту. Чи є управління проєктами його основним фокусом, чи містить він інші функції, такі як відстеження часу та виставлення рахунків;

— розробник програмного продукту. Хто створив програмне забезпечення та який досвід у будівельній індустрії у розробника;

— основні вікна інтерфейсу. наскільки зручний інтерфейс для користувачів і якою є легкість його навігації;

— переваги та недоліки. Які переваги та недоліки має кожен програмний продукт у відношенні до функціоналу, користувацького досвіду та ціноутворення.

При аналізі існуючого програмного забезпечення для управління проєктами в будівельній індустрії необхідно виділити кілька популярних варіантів. Один з них – це Procore (Рисунок 1).

Procore – це програмне забезпечення для будівництва, яке надає інструменти для управління будівельними проєктами, ресурсами, фінансами, документами та іншими аспектами будівництва. Його функціональні можливості охоплюють всі етапи будівництва – від проєктування та планування до будівництва та експлуатації.


Рисунок 1 – Графічний інтерфейс застосунку procore - [https://bit.ly/procore-showoff](https://bit.ly/procore-showoff)

Переваги:

— комплексний функціонал для керування документами, включаючи можливість зберігання та організації документів в централізованому місці;

— сильні можливості планування, що дозволяють користувачам створювати та керувати графіками проєктів, термінами виконання та відстежувати прогрес;

— інструменти управління витратами, включаючи відстеження бюджету та звітів про витрати.

Недоліки:

— інтерфейс користувача може бути складним для навігації;

— програмне забезпечення має високий рівень складності для тих, хто не має досвіду в роботі у даній сфері;

— ціни на procore є досить високими, що може не підходити для невеликих або середніх будівельних компаній.

Інший приклад – CoConstruc (Рисунок 2). CoConstruct – це веб-застосунок для будівельних компаній, що надає інструменти для управління проєктами, витратами та комунікацією з клієнтами. Ця платформа дозволяє будівельним компаніям керувати проєктами будь-якої складності.

CoConstruct дозволяє менеджерам проєктів створювати та відстежувати завдання, назначати їх на конкретних робітників та відстежувати їх виконання в реальному часі. Крім того, на платформі доступний функціонал для відстежування витрат, управління ресурсами та планування бюджетами проєктів.

CoConstruct також надає можливість спілкуватися з клієнтами та зацікавленими сторонами, дозволяючи їм бачити статус проєкту, відстежувати зміни та вносити коментарі. Це допомагає зберігати прозорість та забезпечує ефективну комунікацію між учасниками проєкту.

Переваги:

— надає широкі можливості для співпраці команди, дозволяючи їм ефективно працювати разом;

— має потужні функції планування та відстеження бюджету;

— надає ряд опцій налаштуваннь, щоб відповідати потребам різних будівельних компаній.

Недоліки:

— coconstruct є досить дорогим, що не підходить для невеликих або середніх будівельних компаній;

— програма може повільно завантажуватися, що є великим недоліком для тих, хто працює над великими проєктами.

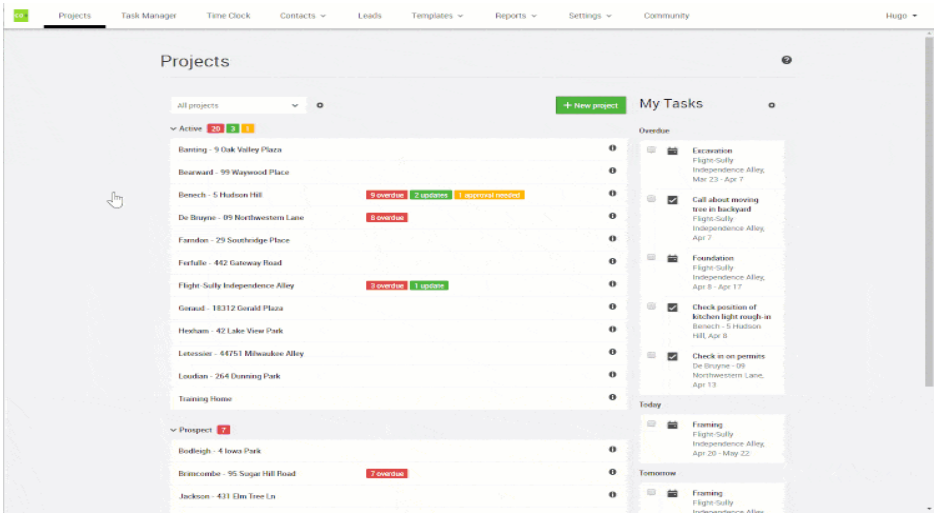



Рисунок 2 – Графічний інтерфейс платформи CoConstruct – https://bit.ly/coconstruct-showoff

Ще одним прикладом веб-застосунку для менеджменту проєктів будівельної компанії є Aconex (Рисунок 3). Це веб-застосунок для управління проєктами, що використовується переважно в будівельній, інфраструктурній та енергетичній галузях. Ця платформа дозволяє учасникам проєкту співпрацювати та управляти інформацією про проєкт, включаючи креслення, документи та кореспонденцію, з центрального місця.

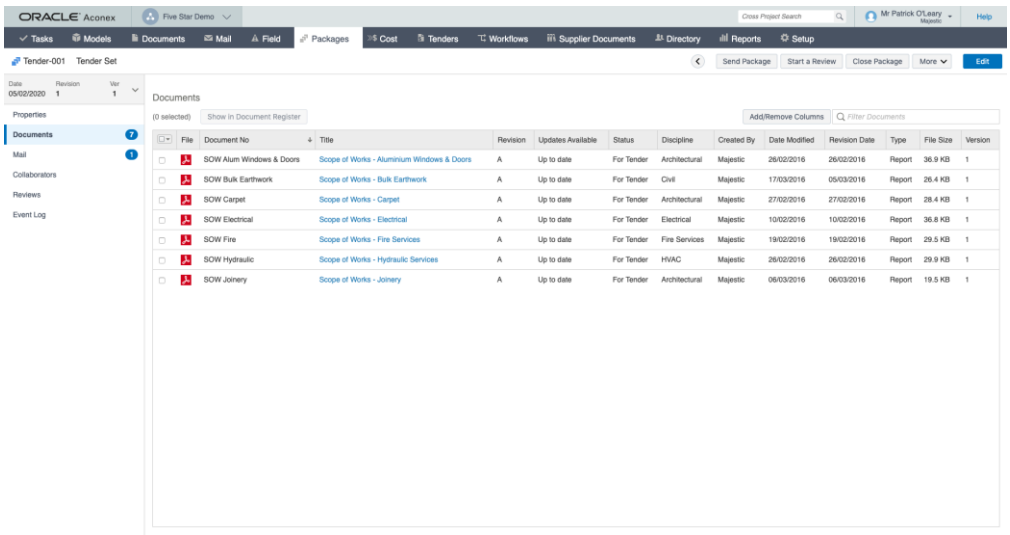



Рисунок 3 – Графічний інтерфейс Aconex - https://gdm-catalog-fmapi-prod.imgix.net/ProductScreenshot/ff6ae1d8-ade6-4530-a412-4bcf085c66da.png

Aconex надає широкий спектр функцій для управління та відстеження інформації про проєкт, включаючи управління документами, управління робочим процесом, управління змінами та інструменти для командної співпраці. Платформа також включає потужні інструменти аналітики та звітності, що дозволяють менеджерам проєктів відстежувати продуктивність проєкту, ідентифікувати області для покращення та приймати обґрунтовані рішення.

Переваги:

— централізоване управління проєктами. aconex дозволяє зберігати всю інформацію про проєкт в одному місці, що полегшує доступ до неї та покращує співпрацю між різними командами та стейкхолдерами;

— оптимізація процесу звітування. завдяки aconex можна швидко та ефективно формувати звіти про стан проєкту, що полегшує процес прийняття рішень та сприяє вчасному виявленню та усуненню проблем;

— зменшення часу та витрат. застосування aconex дозволяє зменшити час та витрати на збір, обробку та передачу інформації, що дозволяє економити ресурси та забезпечувати швидкий доступ до необхідної інформації.

— Недоліки:

— високі витрати на підтримку програмного забезпечення. aconex потребує великої кількості ресурсів для його підтримки та налаштування, що може бути дорогим для компаній, особливо для менших та середніх;

— складність використання. aconex має досить складний інтерфейс та може вимагати тренінгу для користувачів, що може забрати багато часу та ресурсів для компаній.

Останнім прикладом веб-застосунків такого роду є Buildertrend (Рисунок 4) – хмарне програмне забезпечення для управління проєктами, яке надає різноманітні інструменти для будівельних компаній, включаючи планування проєктів, відстеження бюджету та комунікацію з клієнтами. Крім того, воно має мобільний застосунок, який дозволяє користувачам отримувати доступ до даних проєкту в режимі реального часу.

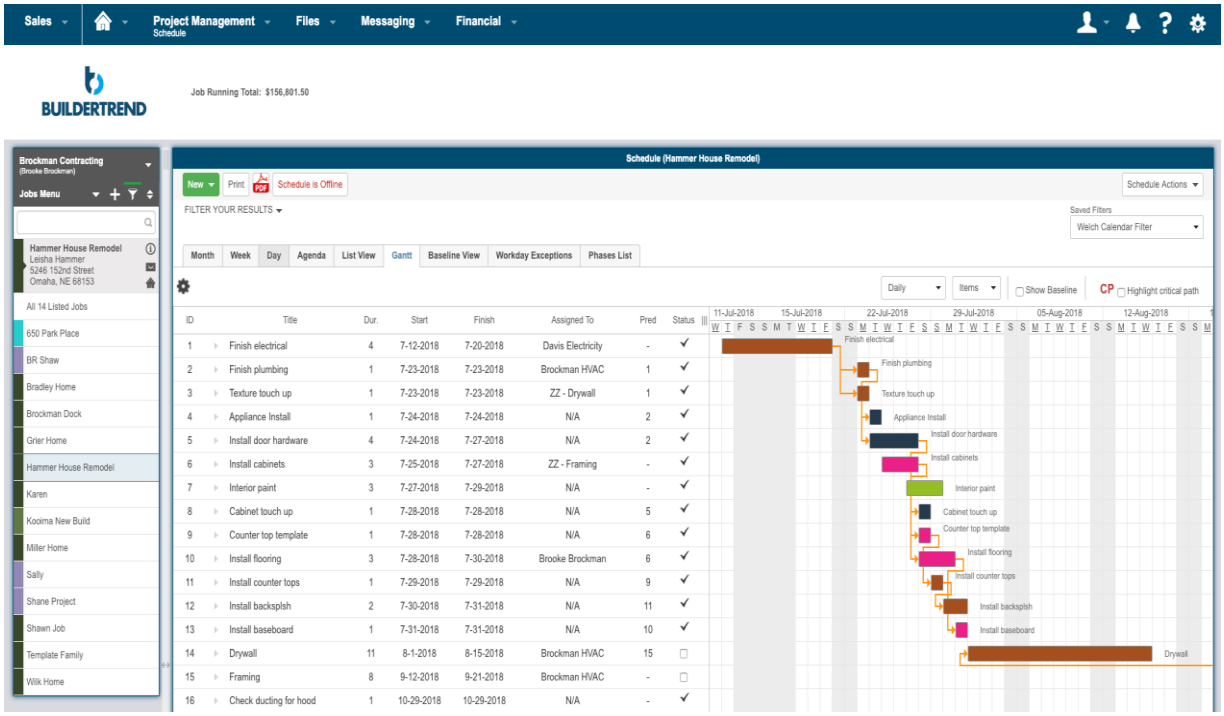



Рисунок 4 – Графічний інтерфейс Buildertrend - https://images.surferseo.art/8a15a354-021f-4012-806e-aaa2be79af9b.png

Переваги:

— користувацький інтерфейс з простою навігацією;

— надає широкий спектр інструментів для керування проєктами, включаючи планування, бюджетування та керування завданнями;

— надає потужні можливості звітів та аналітики.

Недоліки:

— програмне забезпечення може бути повільним у завантаженні, що може бути серйозним незручністю для тих, хто працює над великими проєктами;

## 1.3 Визначення вимог до програмного забезпечення

Основуючись на раніше описаних процесах менеджменту будівельних компаній та перевагах/недоліках існуючих рішень, можна виділити наступні вимоги до застосунку:

— широкі можливості для співпраці команди. Веб-застосунок повинен надавати зручний інтерфейс для комунікації між учасниками проєкту, забезпечуючи можливість обміну інформацією та документами, спільної роботи над завданнями та відстеження їх виконання;

— потужні функції планування та відстеження бюджету. Веб-застосунок повинен надавати інструменти для планування проєкту, включаючи можливість створювати графіки, визначення термінів виконання, розподіл завдань між учасниками та відстеження витрат, щоб забезпечити ефективне управління бюджетом проєкту;

— налаштування. Веб-застосунок повинен мати гнучкі налаштування задля відповідності потреб різних будівельних компаній, включаючи можливість додавати власні поля для введення додаткової інформації та інші налаштування.

— швидкість завантаження. Веб-застосунок повинен працювати швидко та надійно, надаючи користувачам зручний та ефективний інтерфейс для роботи з проєктом;

— інструменти звітності та аналітики. Веб-застосунок повинен мати функціонал надавання звітів та аналітики проєктів. Такий функціонал надасть менеджерам проєктів можливість відстежувати прогрес та продуктивність проєкту, ідентифікувати проблемні зони та приймати обґрунтовані рішення;

— комплексний функціонал для керування документами: Веб-застосунок повинен надавати можливість зберігання та організування документів в централізованому місці, зручний пошук, фільтрацію та відстеження змін документів;

— доступність та ціна. Веб-застосунок повинен бути доступним та відповідати бюджетним обмеженням будівельних компаній. Для цього, можуть бути введені різні тарифні плани з різним рівнем функціональності та ціною;

— безпека та конфіденційність даних: Веб-застосунок повинен забезпечувати надійний захист конфіденційної інформації, яка зберігається в ньому, та дотримання вимог законодавства у сфері захисту персональних даних;

— зручність та ергономіка інтерфейсу. Веб-застосунок повинен мати зручний та простий інтерфейс для користувача, що дозволить швидко та легко орієнтуватися в різних функціях та виконувати необхідні завдання;

— підтримка та навчання користувачів. Веб-застосунок повинен надавати підтримку та навчання користувачам, що допоможе їм ефективно використовувати застосунок та отримувати максимальну користь від його використання.

При урахуванні виставлених вимог до програмного забезпечення можна прийти до висновку що найкращою моделлю розробки у даному випадку буде – ітеративна модель:


Рисунок 5 – Ітеративна модель розробки

Ітеративна модель розробки – це підхід до створення програмного забезпечення, в якому проєкт розбивається на ітерації, кожна з яких включає в себе певні етапи життєвого циклу розробки. Під час кожної ітерації виконуються певні завдання, такі як планування, дизайн, розробка, тестування та випробування, після чого результати аналізуються та використовуються для планування наступної ітерації. Ітеративна модель розробки дозволяє

використовувати гнучкий підхід до розробки, де зміни можуть бути внесені на будь-якому етапі проєкту.

Першим плюсом ітеративної моделі розробки є в знижені ризику невдачі проєкту. Кожна ітерація є повноцінним циклом розробки, який включає в себе планування, дизайн, розробку, тестування та випробування. Кожна ітерація дозволяє перевірити, чи відповідає проєкт вимогам клієнта, і, якщо не відповідає, то внести відповідні зміни до наступної ітерації. Це дозволяє знизити ризик невдачі проєкту та забезпечити більш високу якість програмного забезпечення.

Другий плюс ітеративної моделі розробки полягає в тому, що вона дозволяє забезпечити більш гнучкий підхід до розробки. Кожна ітерація може бути розглянута як окремий цикл розробки, що дозволяє змінювати вимоги клієнта під час розробки проєкту. Це дозволяє розробнику або команді розробників більш ефективно працювати

## 1.4 Технічне завдання на розробку

1) Підстава для розробки:

Підставою для розробки є «Завдання на дипломний проєкт», затверджене завідувачем кафедри інженерії програмного забезпечення. Найменування проєкту: «Веб-застосунок для керування проєктами будівельної компанії».

2) Призначення розробки:

Функціональне призначення: Розробка веб-застосунку для керування проєктами будівельної компанії з можливістю планування завдань, відстеження прогресу, контролю бюджету, збереження історії проєктів та звітів.

Експлуатаційне призначення: Застосунок має застосовуватись в будівельних компаніях для оптимізації управління проєктами та підвищення ефективності роботи команди.

3) Вимоги до програми

3.1) Вимоги до функціональних характеристик:

— платформа повинна мати зручний інтерфейс користувача з легкою навігацією;

— платформа повинна дозволяти користувачам створювати, оновлювати та відстежувати проєкти;

— платформа повинна давати можливість призначати завдання та відстежувати їх виконання;

— платформа повинна мати можливість створювати звіти та аналітику по проєктам;

— платформа повинна забезпечувати можливість зберігання історії проєктів та відстежування змін.

3.2) Вимоги до надійності:

— платформа повинна забезпечувати захист персональних даних та інформації про проєкти.

— платформа повинна бути стійкою до відмов та непередбачуваних ситуацій;

— платформа повинна забезпечувати можливість відновлення даних у разі їх втрати.

3.3) Умови експлуатації та вимоги до технічних засобів:

— платформа повинна працювати на будь-яких пристроях з підтримкою браузерів;

— мінімальні вимоги до технічних характеристик пристроїв повинні бути низькими для максимальної доступності користувачів.

3.4) Вимоги до інформаційної та програмної сумісності:

— платформа повинна бути сумісною з різними браузерами та операційними системами;

— платформа повинна підтримувати інтеграцію з різними застосунками та сервісами, що використовуються в будівельній галузі.

4) Інструменти розробки:

— WebStorm для розробки фронтенду застосунку з використанням html, css та javascript;

— visual studio для розробки бекенду застосунку з використанням c# та .net framework;

— figma для розробки дизайну інтерфейсу користувача;

— github для зберігання та управління версіями коду.

5) Модель розробки:

— ітеративна модель розробки, що дозволить поетапно вдосконалювати функціонал та виправляти можливі помилки у процесі розробки.

6) План тестування:

— ручне тестування функціоналу застосунку на різних етапах розробки;

— автоматичне тестування з використанням єдиного фреймворку для забезпечення високої якості коду та функціоналу.

7) Умови реалізації проєкту:

— термін реалізації – 3 місяці;

— бюджет розробки не запланований.

## 1.5 Висновки. Постановка задачі

У першому розділі «Дослідження предметної області» було здійснено змістовний аналіз сфери керування проєктами в будівництві, що включало в себе дослідження сучасних методик і підходів до розробки ПЗ. Було проведено аналіз наявного програмно-технічного забезпечення, зокрема, з використанням сучасних веб та звичайних застосунків, що використовуються в будівельній сфері. На основі цих даних було визначено основні вимоги до майбутнього веб-застосунку для керування будівельними проєктами. В кінці розділу було сформульовано технічне завдання на розробку такого застосунку, враховуючи всі виявлені вимоги.

У процесі дослідження предметної області було виділено наступні задачі на кваліфікаційну роботу:

— розробити архітектуру та визначити функціональну структуру веб-застосунку для керування будівельними проєктами, враховуючи сучасні підходи до проєктування;

— спроєктувати користувацький інтерфейс веб-застосунку, орієнтований на зручність та ефективність роботи користувача;

— вибрати і обґрунтувати вибрані технології та методи розробки веб-застосунку, враховуючи їх стабільність, продуктивність, безпеку та зручність подальшого супроводу;

— реалізувати логіку веб-застосунку, забезпечити його інтеграційне та інтерактивне тестування, а також розробити керівництво користувача для зручності використання системи.

# 2 ПРОЄКТУВАННЯ

## 2.1 Архітектура та функціональна структура веб-застосунку

Для успішного проєктування застосунку необхідно вибрати його архітектуру. Так як даний застосунок має відповідати сучасним стандартам мережевих систем, основною архітектурою даного застосунку стане архітектура клієнт-сервер (Рисунок 6) – найпоширеніша модель розподілу ролей у мережевих застосунках, включаючи веб-застосунки. Вона полягає у розподілі обов'язків між двома основними компонентами: сервером (back-end) та клієнтом (front-end).

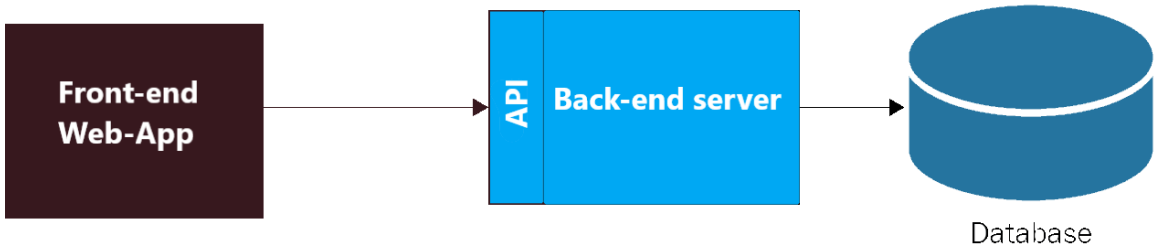



Рисунок 6 – Клієнт-серверна архітектура

Сервер (Back-end) – це компонент, який забезпечує обробку даних, виконання бізенс-логіки, взаємодію з базами даних і іншими зовнішніми сервісами. Він слухає і відповідає на запити від клієнтів. В межах веб-застосунків сервер часто створює API (Application Programming Interface), який використовує HTTP(S) для комунікації з клієнтами.

Клієнт (Front-end) – це компонент, з яким безпосередньо взаємодіє користувач. Він відображає дані, отримані від сервера, та надає інтерфейс для введення даних користувачем. Клієнт може бути будь-яким пристроєм з браузером або спеціалізованою програмою, наприклад, мобільним застосунком, який може надсилати запити серверу та інтерпретувати відповіді.

Розподіл між back-end та front-end дозволяє зосередитися на спеціалізованих обов'язках. Back-end розробники можуть зосередитися на створенні надійних, ефективних і безпечних серверних рішень, в той час як front-

end розробники можуть працювати над створенням інтуїтивно зрозумілих та привабливих інтерфейсів користувача.

За допомогою такої архітектури веб-застосунок має можливість вільного розширення в майбутньому. Прикладом такого розширення є додавання мобільної версії не змінюючи код серверної частини а всього лиш створивши додаткового споживача API.

Для реалізації бек-енд частини найкраще підійде архітектура «Clean Arhictecture» (Рисунок 7).

Clean Architecture(або чиста архітектура), – це підхід до проєктування програмного забезпечення, який пропагує SOLID і незалежність від зовнішніх рамок. Ця архітектура зосереджена на організації коду таким чином, що бізенс-логіка і прикладні правила програмування повинні бути незалежні один від одного та зовнішніх чинників.

Основні принципи Clean Architecture:

— незалежність від фреймворків. Архітектура не повинна залежати від певного фреймворку. Фреймворки і бібліотеки повинні бути інструментами, а не керівництвами для архітектури;

— тестова незалежність. Бізнес-правила повинні бути простими для тестування. Немає потреби в зовнішніх елементах, таких як база даних, сервери або веб-сервіси;

— незалежність від UI. Інтерфейс користувача повинен залежати від системи, а не навпаки. Це означає, що можна без проблем змінювати UI без зміни бізенс-логіки;

— незалежність від бази даних. Бізнес-правила не повинні залежати від конкретної бази даних. Необхідно мати можливість використовувати будь-яку базу даних без зміни бізнес-прави.

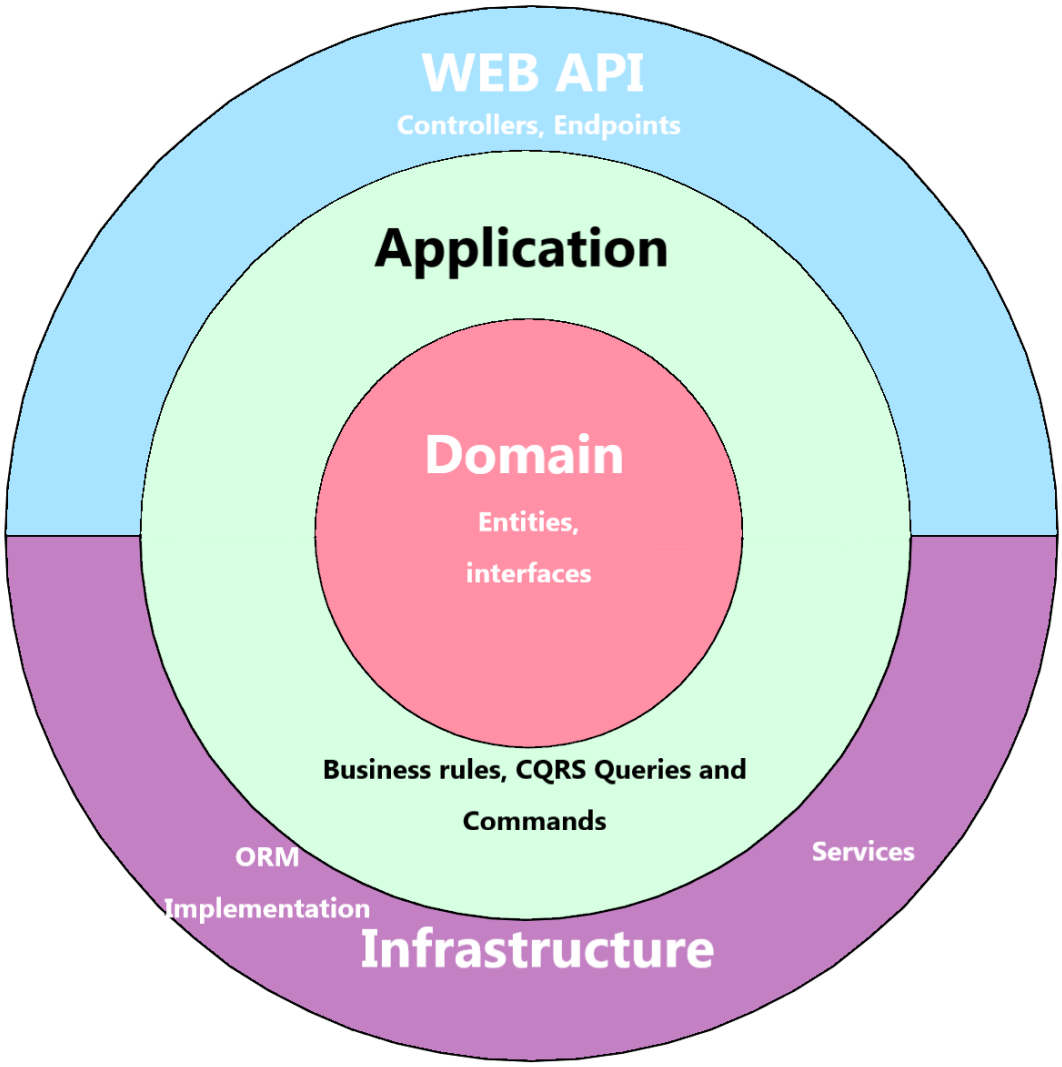



Рисунок 7 – Clean architecture

Clean Architecture центрується навколо доменного та застосункового шарів, відмовляючись від залежності від зовнішніх факторів, таких як база даних, UI шар або фреймворк:

— domain. Цей внутрішній шар є основою архітектури і містить основні бізнес-сутності та бізнес-правила. Ці об'єкти не залежать від інших аспектів програми, таких як зберігання даних, доступ до мережі тощо, що дозволяє зосередитися на моделюванні бізенс-логіки без втручання зовнішніх чинників;

— application. Шар Application визначає бізнес-правила на основі

сутностей та значеннєвих об'єктів домену. Він також містить команди та запити CQRS, а також інтерфейси, які реалізуються в проєкті Infrastructure.

— Infrastructure. Інфраструктура включає в себе реалізації інфраструктурних інтерфейсів, визначених в проєкті Application. Це розмежування дозволяє зручно змінювати будь-яку реалізацію інфраструктури в будь-який час;

— WEB UI. WEB UI містить контролери та представлення (views). Він залежить тільки від Infrastructure, щоб впровадити реалізації Infrastructure в проєкт Application під час запуску;

За рахунок центрування навколо доменного та застосункового шарів, Clean Architecture не обмежує бізнес-правила конкретною базою даних, такою як Oracle чи SQL Server. Замість цього, можна використовувати різноманітні бази даних, наприклад, MongoDB, BigTable, CouchDB та інші.

Однією з переваг цього підходу є те, що бізнес-правила можна тестувати в ізоляції, без залежності від зовнішніх елементів, таких як UI, веб-сервери або бази даних. Іншим плюсом є те, що деталі реалізації інфраструктури можна змінювати без впливу на ядро застосунку.

Також у даній архітектурі використовується інший шаблон а саме CQRS(Рисунок 8). Command Query Responsibility Segregation є архітектурним шаблоном, що забезпечує високу гнучкість, стабільність та продуктивність системи. Основний принцип CQRS полягає в розділенні моделей читання та запису, тобто в одному напрямку система використовує модель запису для виконання дій (команд), а в іншому – модель читання для відповіді на запити.

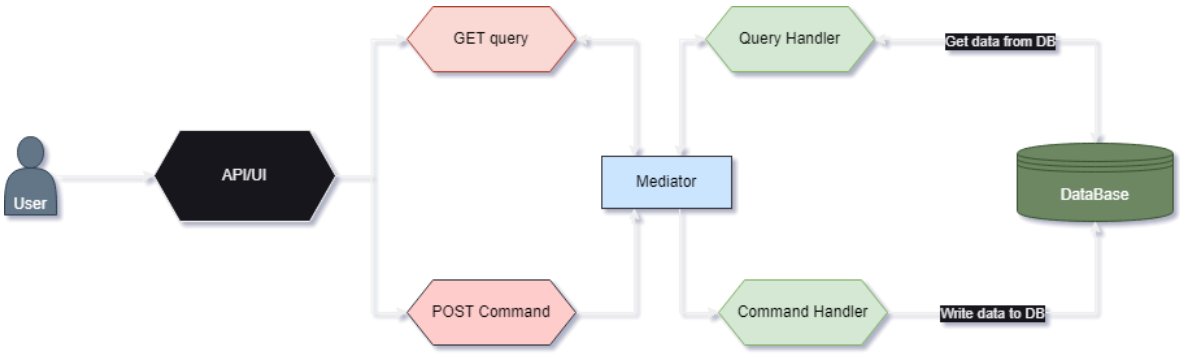



Рисунок 8 – Шаблон CQRS

Таке розділення дозволяє оптимізувати моделі читання та запису незалежно одна від одної. Таким чином, кожна з моделей може бути налаштована так, щоб максимально відповідати вимогам конкретних сценаріїв використання. Це може привести до збільшення продуктивності та масштабованості системи, оскільки операції читання та запису можуть бути виконані паралельно.

При використанні шаблону CQRS у сполученні з архітектурою Clean Architecture, CQRS забезпечує розподіл відповідальності між діями запису та читання. Clean Architecture, у свою чергу, стимулює створення системи з гнучкими, незалежними та взаємозамінними частинами. Це створює ідеальну платформу для використання CQRS, дозволяючи розробникам ізолювати та рефакторити читання та записи в моделях без впливу на загальну архітектуру програмного забезпечення.

Для взаємодії із базою даних найкраще підійде шаблон репозиторій. Шаблон «Репозиторій», в даному випадку, буде частиною шару Infrastructure у нашій архітектурі. Він розглядається як міст між доменними моделями або бізенс-логікою та кодом, що забезпечує зберігання та отримання даних. Цей шаблон приховує деталі впровадження технологій зберігання даних, надаючи інтефрейс для роботи з даними.

Репозиторій реалізує набір операцій, таких як створення, читання, оновлення та видалення (CRUD), для взаємодії з даними, які він керує. Основна мета репозиторію – ізолювати бізенс-логіку від деталей впровадження доступу до даних, що дозволяє змінювати спосіб зберігання даних без необхідності змінювати бізенс-логіку.

Створимо діаграму класів (Рисунок 9) бек-енд частини для запиту детальної інформації по проєкту GetProjectDetailed основуючись на архітектурі Clean Arhictecture, шаблонах CQRS та шаблоні репозиторій.

У процесі створення діаграми класів можна відзначити, що контролер не встановлює прямого зв'язку з основною масою класів. Така структура можлива завдяки шаблону CQRS, зокрема, через використання запиту GetProjectDetailedQuery, обробника GetProjectDetailedQueryHandler і медіатора.

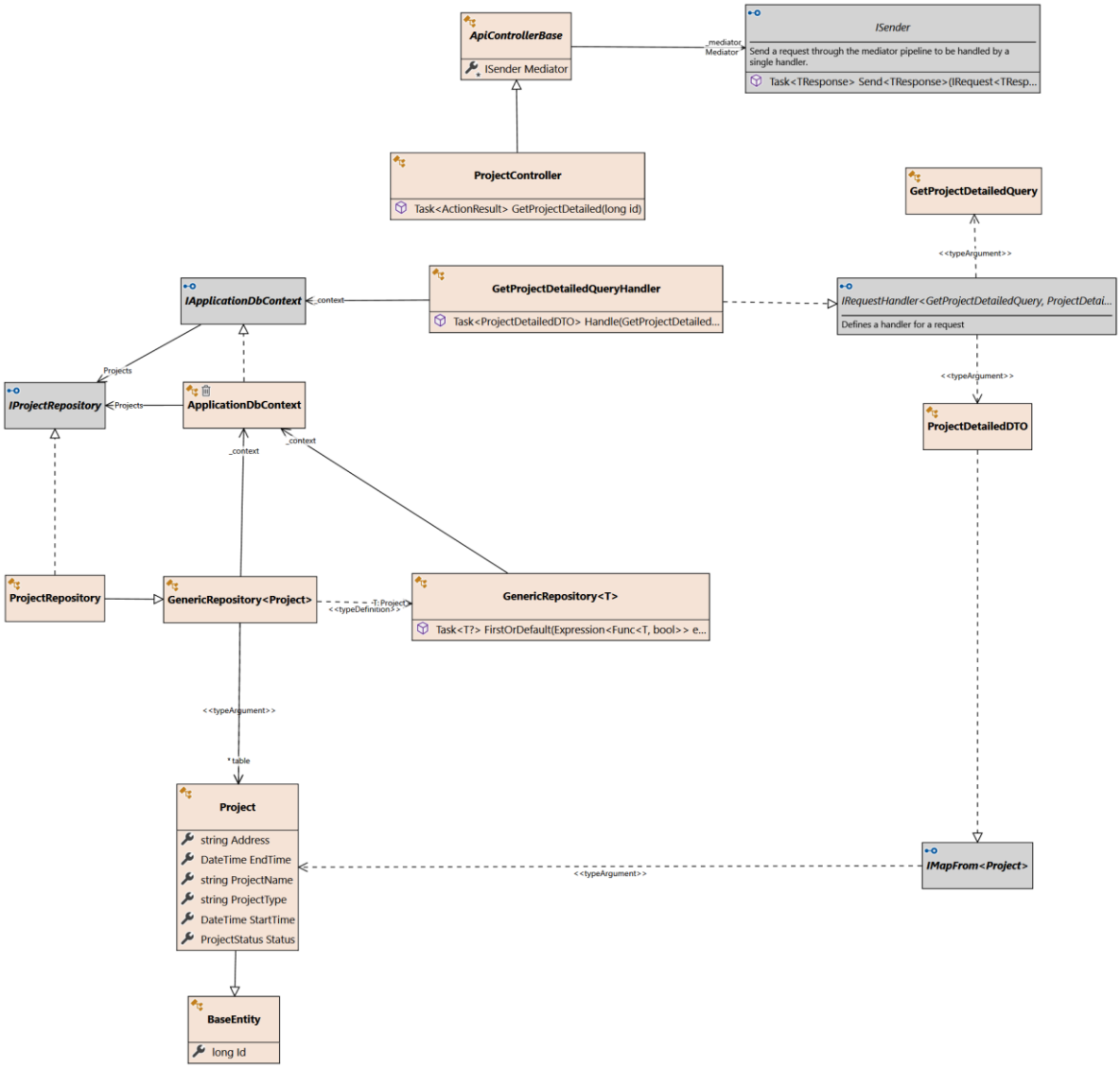



Рисунок 9 – Діаграма класів запиту GetProjectDetailed

Медіатор відіграє ключову роль у забезпеченні взаємодії між різними компонентами системи. Він приймає запити від контролера і передає їх відповідним обробникам, які виконують бізенс-логіку та викликають потрібні сервіси.

GetProjectDetailedQuery є конкретним запитом, що представляє бізнес-операцію отримання деталізованої інформації про проєкт. Він є частиною моделі CQRS і відповідає за опис того, яку інформацію необхідно отримати.

Обробник запиту, GetProjectDetailedQueryHandler, є компонентом, що безпосередньо виконує бізенс-логіку, пов'язану з обробкою даного запиту. Це забезпечує розділення відповідальності між контролером, який керує потоком

даних, та обробником, який виконує конкретні дії.

Задля реалізації патерну репозиторій, система використовує GenericRepository, що слугує загальним контрактом для всіх конкретних репозиторіїв, таких як ProjectRepository. Цей патерн спрощує взаємодію з даними, оскільки всі операції CRUD централізовані в одному місці.

Інтерфейс IApplicationDbContext представляє контекст застосунка, який забезпечує доступ до бази даних через репозиторії. Він використовується в обробнику GetProjectDetailedQueryHandler для отримання доступу до даних про проєкти.

Сутність Project, що унаслідується від BaseEntity, використовується для моделювання основних бізнес-об'єктів у домені. Вона містить ключові характеристики проєкту, які потрібні для бізенс-логіки.

Клас ProjectDetailedDTO служить моделлю передачі даних (Data Transfer Object), що використовується для передачі деталізованої інформації про проєкт між різними шарами системи. Він включає додаткові поля, що необхідні для відображення деталей проєкту.

Метод ImapFrom, що використовується між Project і ProjectDetailedDto, служить для встановлення відповідності між полями цих двох моделей. Він допомагає конвертувати дані з однієї моделі в іншу, спрощуючи передачу інформації між різними частинами системи.

Фінальним кроком створення архітектури застосунку є створення діаграми моделі даних проєкту(Рисунок 10). Так як ця модель даних розроблена для управління проєктами в галузі будівництва. Вона складається з численних таблиць, які відображають різні аспекти проєкту, включаючи інформацію про проєкт, матеріали, компанії, користувачів, фази проєкту та ін.

Таблиця «Project» є центральною таблицею і містить інформацію про кожен проєкт, таку як ім'я проєкту, тип, бюджет, опис, статус, час початку і закінчення, а також ідентифікатори компанії і користувача, що відповідають за проєкт.

Таблиця «User» відображає інформацію про користувачів, включаючи їх

ім'я, електронну пошту, пароль, роль, ідентифікатор компанії, до якої вони належать, та контактний номер.

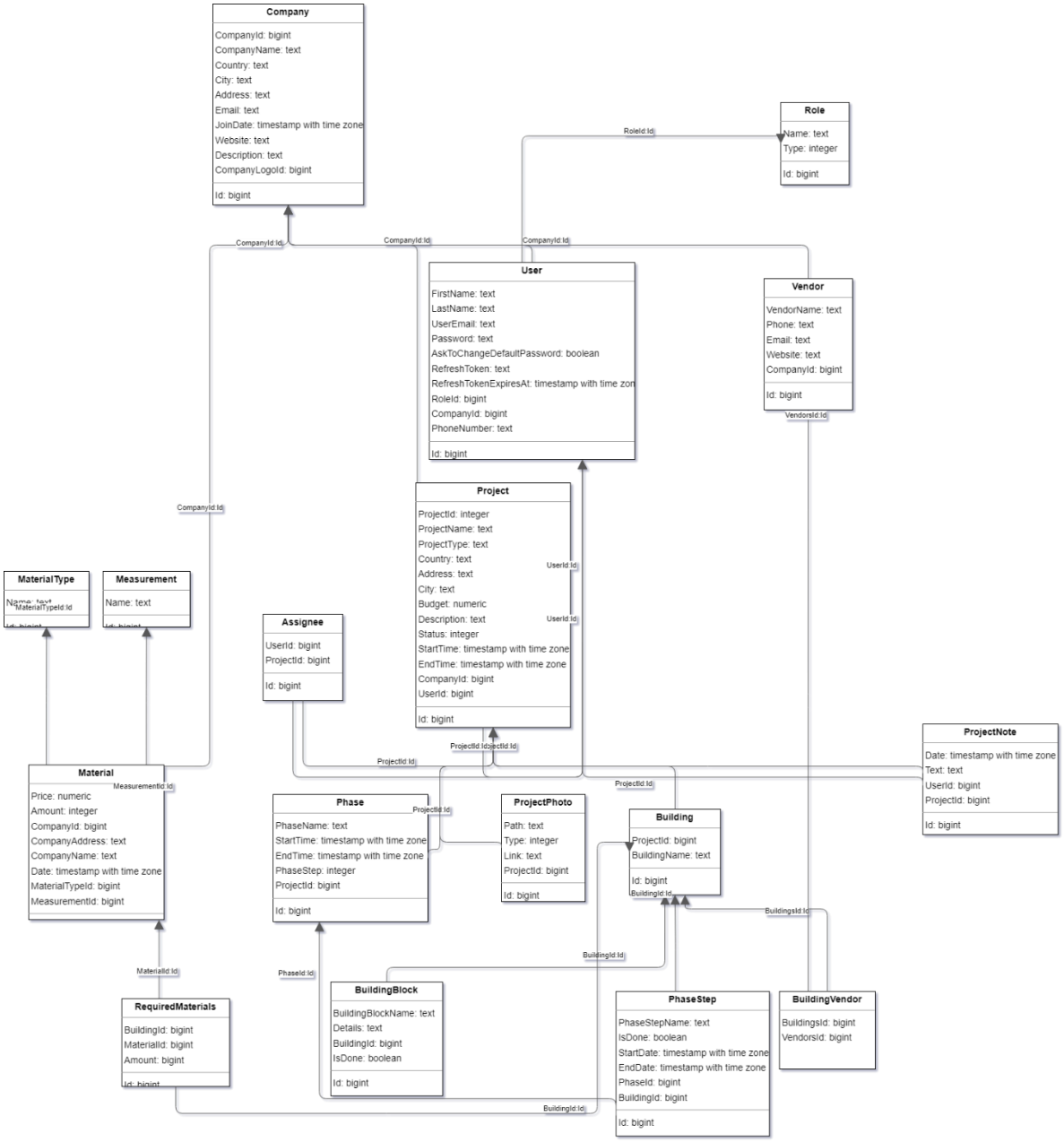



Рисунок 10 – ER Діаграма моделі даних

Таблиця «Company» містить деталі про компанії, які беруть участь у проєктах, включаючи назву, країну, місто, адресу, електронну пошту, дату приєднання, веб-сайт, опис і ідентифікатор логотипу компанії.

Таблиці «Building», «BuildingBlock», «Material», «Phase», «PhaseStep» відображають детальну інформацію про будівництво, використовувані матеріали

і етапи проєкту відповідно.

Таблиці «ProjectNote», «ProjectPhoto» містять додаткові ресурси, пов'язані з проєктом, такі як примітки, документи та фотографії.

Таблиця «Assignee» використовується для прив'язки користувачів до проєктів, на які вони призначені, вказуючи ідентифікатори користувача і проєкту.

Накінець, таблиці «Vendor», «VendorType» представляють інформацію про постачальників та їх типи, які можуть бути використані в рамках проєкту.

## 2.2 Проєктування інтерфейсу користувача

Для проєктування інтерфейсу використовується веб-застосунок Figma. Цей інструмент ідеально підходить як для командної, так і для індивідуальної роботи. Він має багато переваг для одного дизайнера. По-перше, Figma — це хмарне рішення, тому з ним можна працювати з будь-якого комп'ютера з підключенням до Інтернету. По-друге, Figma включає в себе потужні інструменти для створення прототипів, які можна легко перетворити дизайнерські концепції в інтерактивні прототипи, що дає можливість протестувати і відточувати інтерфейс перед його реалізацією. По-третє, Figma має вбудовані інструменти для роботи з компонентами та стилями, що спрощує створення та управління дизайн-системами. Також важливо зазначити, що Figma дозволяє експортувати дизайни в різні формати, що забезпечує максимальну сумісність з іншими інструментами розробки.

Першим етапом в процесі розробки інтерфейсу є створення головного екрану вибору проєктів, як показано на рисунку 11. Цей екран стане центральним місцем для взаємодії користувача з проєктами. Він дозволить переглянути список всіх існуючих проєктів, включаючи дати їх створення та назви.

Кожен проєкт у списку буде представлений карточкою, що включатиме всю ключову інформацію про проєкт, а також кнопку для переходу до деталізованої інформації про проєкт. На цьому екрані також буде кнопка «Додати новий проєкт». При натисканні на кнопку відкриється форма створення нового

проєкту, де користувач зможе ввести всю необхідну інформацію та зберегти новий проєкт в системі.

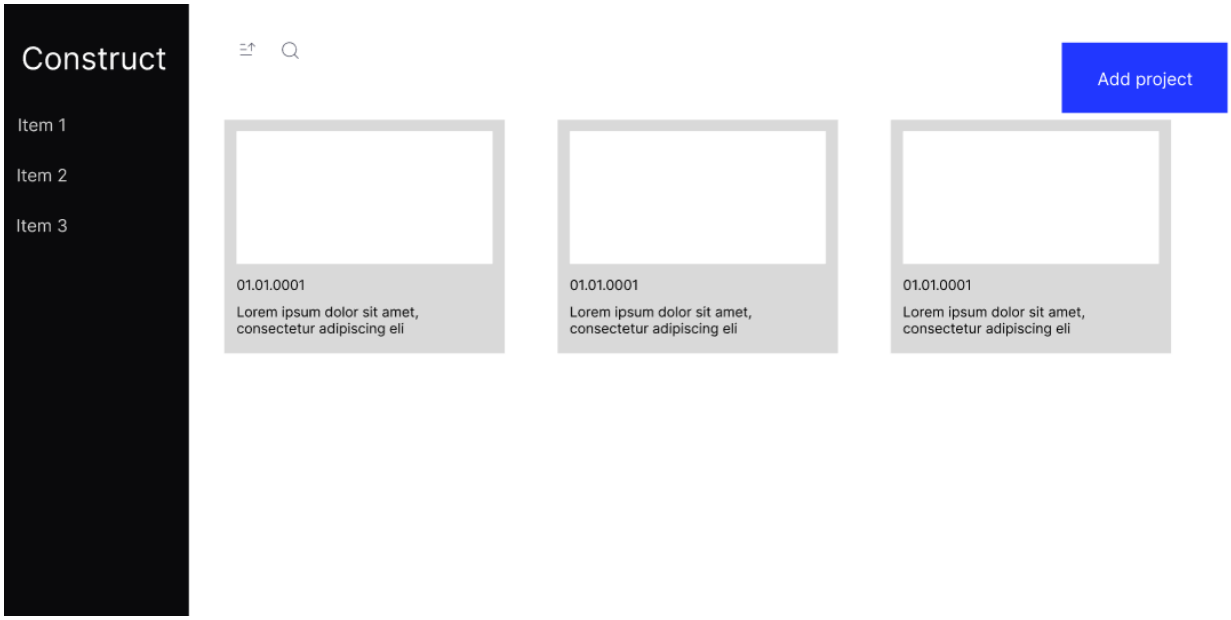



Рисунок 11 – Макет головного екрану вибору проєктів

Додатково, на головному екрані вибору проєктів буде реалізована функція пошуку. Користувач зможе ввести назву проєкту в поле пошуку, і система відобразить всі проєкти, що відповідають критеріям пошуку.

Функція сортування проєктів дозволить користувачам впорядкувати список проєктів за різними параметрами, наприклад, за датою створення, назвою або статусом.

Нарешті, на головному екрані вибору проєктів будуть розташовані посилання на інші пункти меню, які дозволять користувачам переходити до інших частин веб-застосунку, таких як налаштування, допомога, профіль користувача тощо.

Наступним кроком у розробці веб-застосунку є створення вікна перегляду/додавання проєкту, представленого на рисунку 12. Цей екран створений для детального перегляду і редагування інформації про проєкт.

На цьому екрані користувач може переглянути та редагувати основну інформацію про проєкт, таку як назва, дата початку, дата завершення, опис, і так далі. Крім того, можна переглянути список будівель, що належать до проєкту, з

можливістю додавання нових будівель.

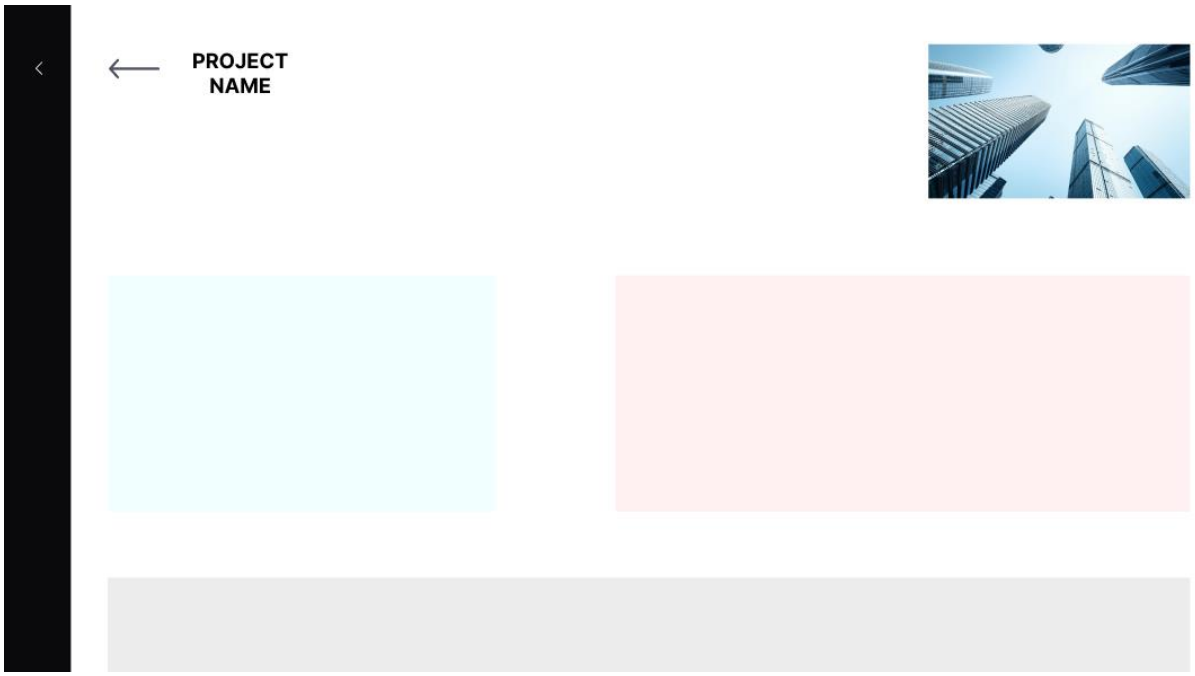



Рисунок 12 – Вікно перегляду/додавання проєкту

Для кожної будівлі буде доступна галерея фотографій, яка дозволить користувачам переглянути вигляд будівлі з різних ракурсів. Користувач також зможе додати нові фотографії до галереї.

Екран також включає в себе перегляд стадій проєкту. Користувач може переглянути поточну стадію проєкту, а також переглянути список усіх стадій, що були пройдені, і стадій, які ще попереду. Для кожної стадії можна переглянути детальну інформацію, включаючи опис, дату початку та завершення.

Нарешті, на цьому екрані користувач може переглянути команду, призначену на проєкт. Для кожного члена команди доступний перегляд інформації про контакт, роль в проєкті та інші важливі деталі. Користувач також може додати нових членів команди або змінити ролі існуючих членів команди.

Однією з найбільших переваг даного екрану є його гнучкість. Незалежно від стадії проєкту, користувач може легко навігувати та редагувати інформацію. Таким чином, вікно перегляду/додавання проєкту не лише забезпечує доступ до актуальної інформації про проєкт, але й підтримує її актуальність.

У контексті команди проєкту, наявність доступу до деталей про кожного

члена команди є дуже важливою. Відображення контактної інформації, ролей та відповідальностей усіх учасників проєкту сприяє ефективній комунікації і співпраці команди.

Більше того, цей екран дозволяє користувачам прослідковувати прогрес проєкту на різних стадіях. Кожна стадія відображається з датою початку та завершення, що дозволяє команді простежити за тим, як проєкт розвивається у часі, та планувати наступні кроки.

Також цей екран забезпечує візуальний зв'язок з будівельними об'єктами проєкту за допомогою фотографій. Це особливо важливо для віддалених команд, які не мають можливості фізично відвідувати місце будівництва. Візуалізація проєкту допомагає команді краще розуміти контекст та обсяг робіт, що виконуються. Всі ці функції спрощують управління проєктом і сприяють більшій продуктивності та ефективності всієї команди.

Закінчимо проєктування інтерфейсу шаблоном екрану входу в систему:

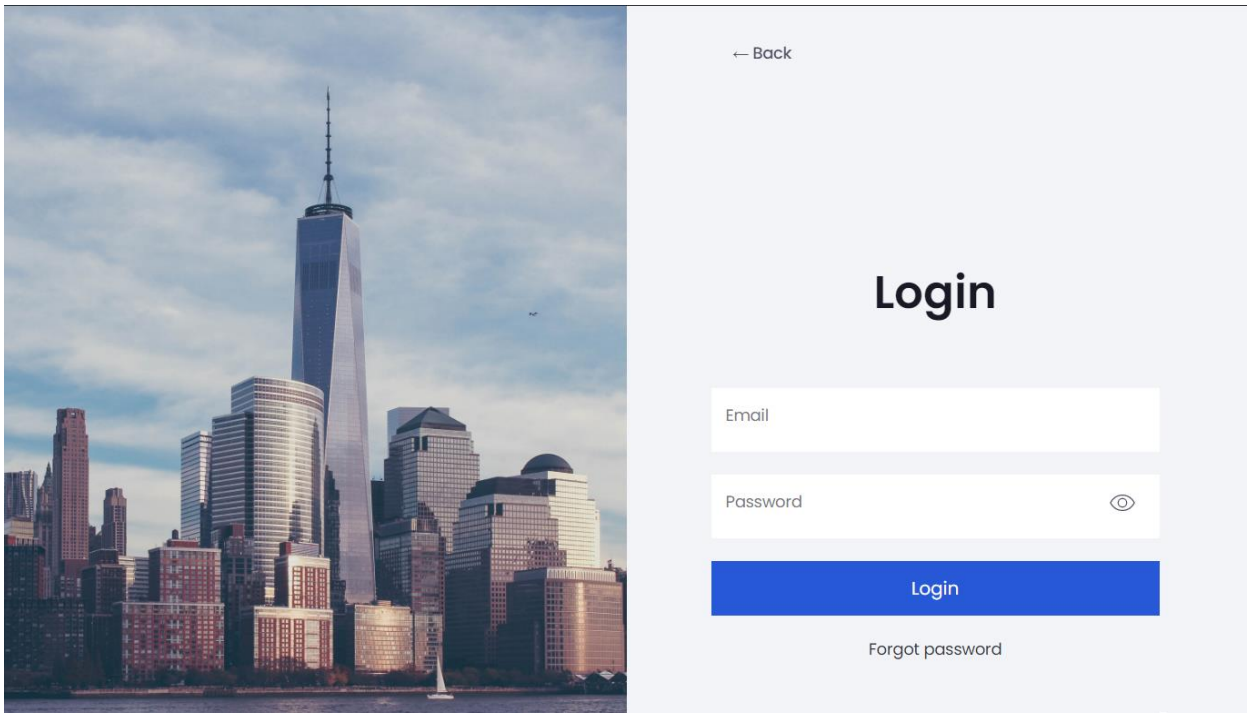



Рисунок 13 – Шаблон вікна входу в систему

Вікно логіну, розташоване в правій частині екрану, представляє собою простий і інтуїтивно зрозумілий інтерфейс для доступу до системи. Воно містить

два основних поля: email та password, де користувачі вводять свої персональні дані для входу. Кнопка Login розташована під цими полями і служить для підтвердження введених даних та входу в систему. Додатково, нижче кнопки Login розміщена посилання «Forgot password», що дозволяє користувачам відновити свій пароль у випадку, якщо вони його забули.

Ліва частина вікна логіну прикрашена ілюстрацією одного з проєктів компанії. Це фото не лише додає візуального привабливості сторінці входу, але й демонструє реальні результати роботи команди. Отже, це не просто естетичний елемент, а й засіб продемонструвати потенційним користувачам досвід та професіоналізм компанії.

## 2.3 Аналіз та вибір технологій і методів розробки застосунку

Основною технологією для розробки застосунку є ASP.NET Core – це крос-платформне, високопродуктивне середовище з відкритим вихідним кодом для створення сучасних веб-сервісів, яке має наступні переваги:

— висока продуктивність та швидкодія;

— безпека за замовчуванням з підтримкою новітніх стандартів;

— підтримка крос-платформної розробки.

За допомогою цієї технології необхідно створити back-end(серверну) частину веб-застосунку, тобто частину застосунку яка відповідає за бізенс-логіку та збереження даних. Основним механізмом роботи цієї технології є контролери та моделі. Моделі – класи які репрезентують дані застосунку та відповідають за їх тип. Контролери в свою чергу відповідають за логіку роботи із цими даними, тобто бізнес логіку застосунку. Back-end частина надає користувачу, тобто Front-end частині.

На допомогу Clean Architecture та CQRS описаним у пункті 2.1 приходить бібліотека MediatR – це проста бібліотека, яка впроваджує шаблон «Медіатор» в .NET. Цей шаблон дозволяє компонентам (в цьому випадку класам у коді)

спілкуватися один з одним без прямого знання про існування інших. Це досягається за допомогою «медіатора», який передає повідомлення між компонентами. В контексті MediatR, повідомленнями можуть бути команди (дії, які мають змінити стан системи) та запити (дії, які повертають дані).

MediatR допомагає у реалізації Clean Architecture на декількох рівнях:

— розділення відповідальностей. Clean Architecture передбачає чітке розділення відповідальностей між компонентами системи. За допомогою MediatR, можна визначити окремі класи для кожної команди та запиту, забезпечуючи тим самим їхню незалежність та спрощуючи розуміння їхньої ролі;

— відокремленість. MediatR дозволяє розробляти команди та запити незалежно один від одного, знижуючи зв'язність між різними частинами вашої системи. Це означає, що зміни в одній команді або запиті не повинні впливати на інші;

— тестування. MediatR полегшує юніт-тестування, оскільки він дозволяє легко мокувати залежності та тестувати команди та запити в ізоляції;

— подієва модель. MediatR підтримує події, які дозволяють реагувати на певні дії в системі без необхідності зв'язувати код, який виконує ці дії, з кодом, який реагує на них.

Для розробки front-end частини найкраще підходить Angular.

Angular – це платформа для розробки веб-застосунків, що включає в себе масштабовані можливості, різні пакети та покращену швидкість розробки. Angular має багато переваг, таких як:

— компонентна архітектура дозволяє розбивати застосунок на перевикористовувані частини, що полегшує розробку та тестування;

— типізація TypeScript сприяє розробці надійного коду, допомагає виявити помилки на стадії розробки;

— вбудована підтримка реактивного програмування через RxJS;

— angular CLI для автоматизації рутинних задач розробки.

— розбивка застосунку на компоненти, що покращує його

структуризацію та полегшує роботу над ним;

— використання строго-типізованої мови TypeScript а не стандартної JavaScript, вона допомагає передбачувати синтаксичні помилки в коді та покращує авто-доповнення коду у вибраній IDE;

— ін’єкція залежностей (dependency injection), так само які і у ASP.NET Core, дозволяє вставляти сервіси, тобто спеціальні класи які реалізують певну логіку застосунку, у потрібний компонент просто вказавши цей сервіс у конструкторі компонента, без необхідності створювати новий об’єкт цього сервісу;

— навігація по сайту за допомогою Router, тобто завантаження потрібних компонентів сайту в залежності від URL.

Також у реалізації front-end angular частини допоможе бібліотека NgRx (рисунок 14). NgRx – це бібліотека для Angular, що впроваджує шаблон управління станом на основі Redux із використанням спостерігачів RxJS. Це дозволяє підтримувати стан застосунку у певному порядку і забезпечує потужний і прогнозований спосіб управління потоком даних.

Хоча NgRx в основному використовується на клієнтській частині (front-end) у Angular застосунках і не має прямого відношення до .NET (який зазвичай використовується на серверній частині), її використання може вплинути на загальну архітектуру застосунку і співпрацю між front-end та back-end.

Із плюсів NgRx можна виділити:

— Однорідність. При використанні шаблону Redux на обох сторонах (наприклад, за допомогою бібліотеки, подібної до Redux на .NET), можна досягти більшої однорідності в архітектурі застосунку. Це може полегшити розуміння потоку даних в застосунку.

— Прогнозованість. NgRx робить стан застосунку прогнозованим, що може спростити взаємодію між клієнтом і сервером. Коли сервер отримує запит, він може бути впевнений, що клієнт знаходиться у відповідному стані.

— Відлагодження. NgRx дозволяє відслідковувати зміни стану в часі,

що може допомогти при відлагодженні проблем, що виникають при взаємодії між клієнтом та сервером.

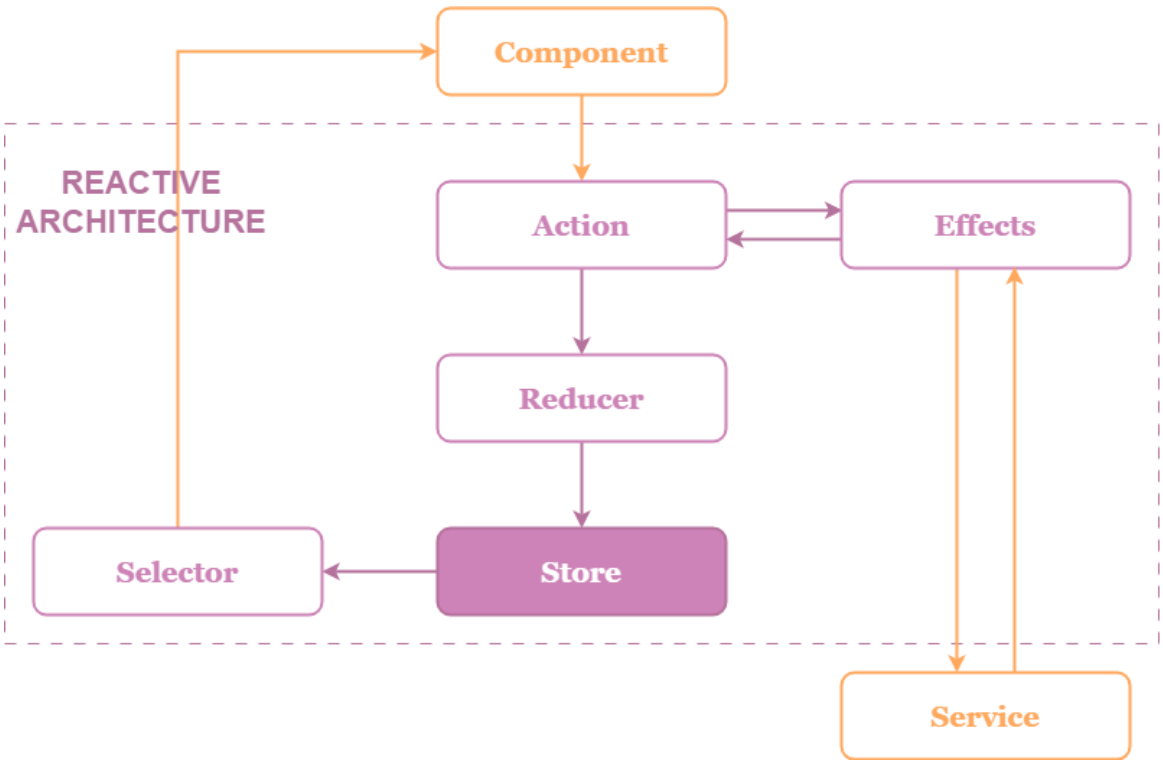



Рисунок 14 – архітектура ngRx

Окрім технологій ASP.NET Core та Angular, у проєкті використовується платформа Azure(Рисунок 15) від Microsoft.

Azure — це комплекс хмарних сервісів, що допомагає у розвитку, управлінні та масштабуванні застосунків в найрізноманітніших сценаріях, як запевняє Microsoft. Інструменти, що надає Azure, включають широкий спектр сервісів, які зазвичай розробникам доводиться створювати самостійно, але це часто є недоцільним або сильно трудомістким. Це включає такі можливості як обчислення, аналітика, автентифікація, бази даних, конфігурація, зберігання файлів, push-повідомлення, машинне навчання, і т.д. Всі ці служби розміщуються в хмарі та масштабуються без будь-яких зусиль з боку розробника, що робить Azure потужним інструментом для розробки сучасних застосунків.

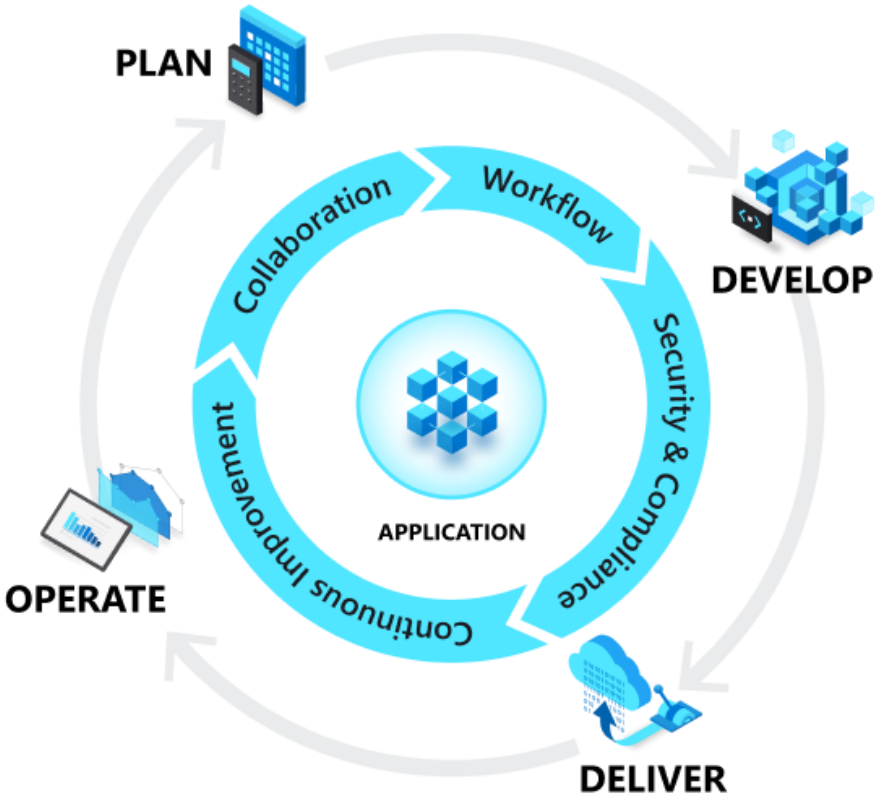



Рисунок 15 – Процес використання Azure https://learn.microsoft.com/en-us/devops/_img/devops-lifecycle.png

Основним засобом розробки любого додатку або інтернет-сервісу є середовище розробки. Саме в ньому програміст пише основний код, зберігає зміни у github чи компілює повну програму в один .exe файл. З урахуванням вибраних технологій потрібні два середовища для розробки – Visual Studio та WebStorm. Обидва IDE представляють повний спектр можливостей, тобто редагування коду, компіляція коду, дебагінг та інше. Проте вони відрізняються за призначенням.

Visual Studio (Рисунок 16) найкраще підходить для створення ASP.NET Core частини веб-додатку. Адже воно, як і ASP.NET Core, підтримується Microsoft і відповідно має тісну інтеграцію із іншими їх технологіями. Visual Studio - це потужне середовище розробки (IDE) від Microsoft, яке надає набір інструментів для проектування, написання і відлагодження програм. Це надає розробникам багатий набір функцій для підвищення продуктивності, включаючи автозаповнення коду, навігацію по коду, розуміння коду та багато іншого

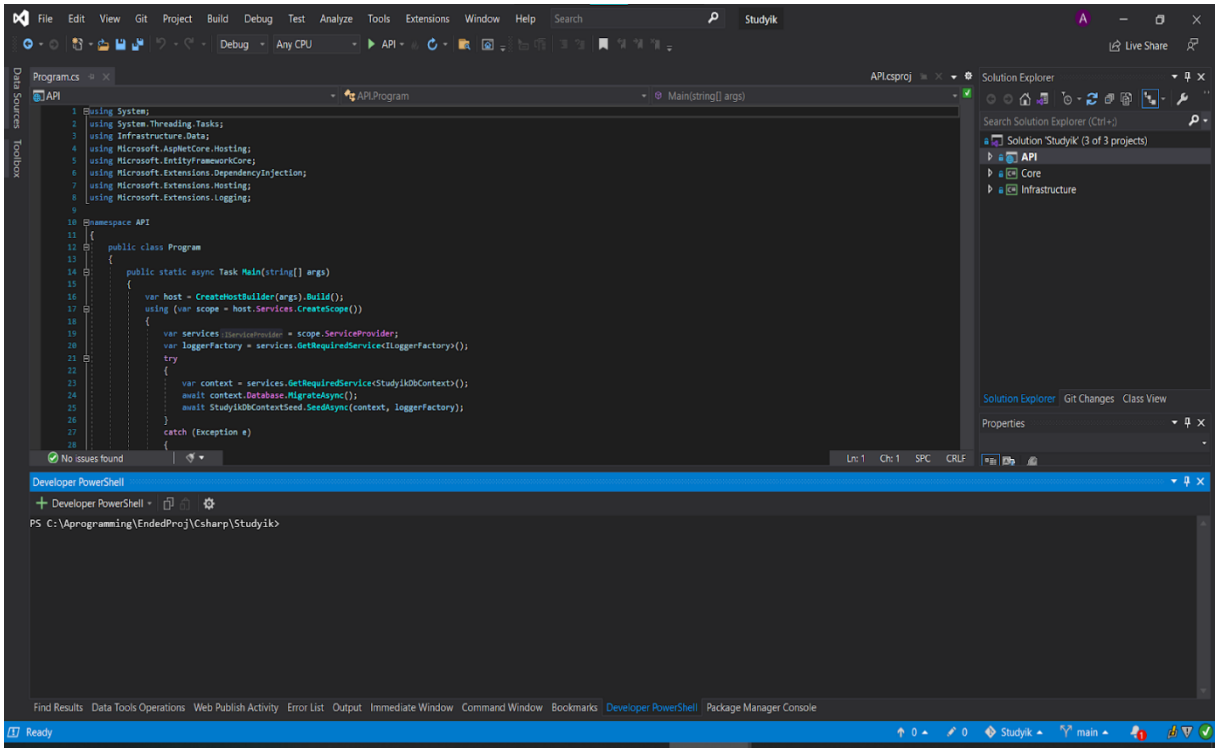



Рисунок 16 – Зовнішній вигляд Visual Studio

WebStorm(Рисунок 17) є інтегрованим середовище розробки (IDE) від компанії JetBrains. Воно спеціалізується на розробці технологій, пов'язаних з JavaScript, таких як веб-застосунки, веб-сайти та Node.js. WebStorm надає розширену підтримку для розробки на основі JavaScript, включаючи інструменти для автодоповнення коду, налагоджувач, валідатор коду, керування пакетами та багато іншого.

WebStorm, від JetBrains, виступає не як звичайний редактор коду, а як потужне середовище розробки, що забезпечує розширені можливості для роботи з кодом. Цей інструмент виконує десятки автоматичних перевірок коду на льоту, що значно спрощує процес виявлення та виправлення потенційних проблем.

Це не просто включає в себе синтаксичний аналіз, але й включає перевірку стилю коду, виявлення виключень та потенційних помилок в логіці програми. Додатково, WebStorm надає підказки відносно оптимізації коду, допомагаючи розробникам писати більш ефективний та зрозумілий код.

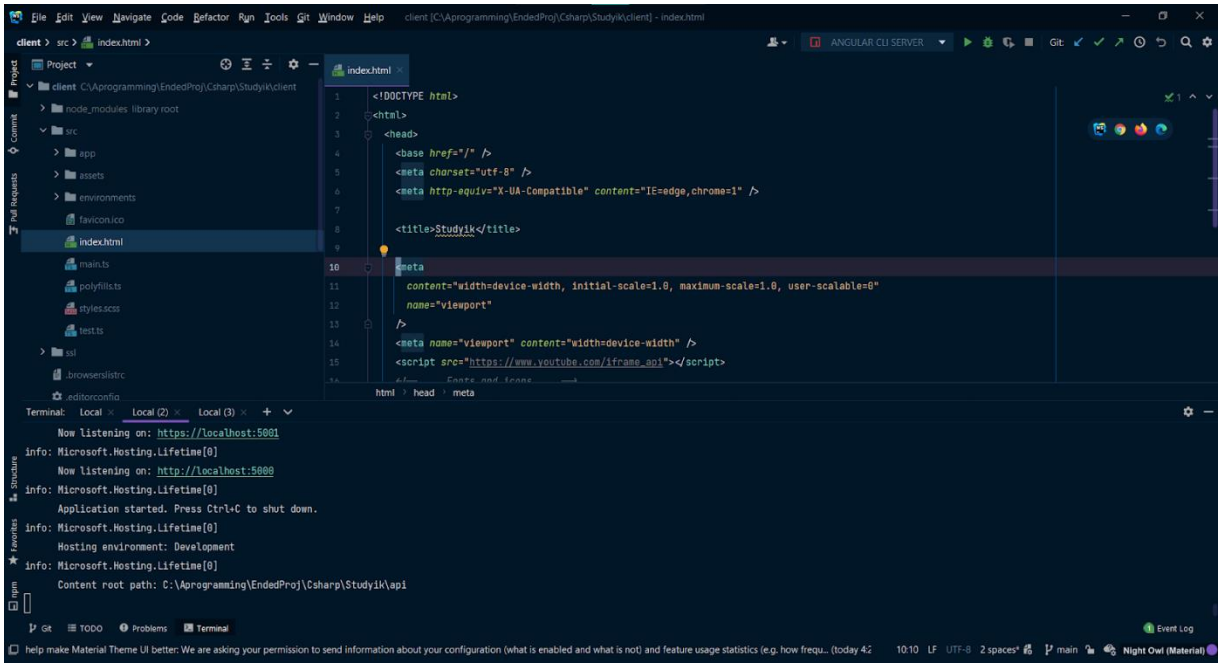



Рисунок 17 – Зовнішній вигляд WebStorm

Крім того, WebStorm включає в себе можливості для потужного рефакторингу коду, що дозволяє з легкістю реорганізувати кодову базу за кілька клацань миші. Це включає в себе можливість перейменування змінних, функцій, класів та інших елементів коду, перенесення та виділення частин коду, а також автоматичну адаптацію змін до всіх відповідних частин проєкту.

Під час впровадження великих структурних змін, WebStorm впевнено зберігає контроль над процесом, забезпечуючи те, що жодна частина коду не буде випущена або забута. Все це робить WebStorm високо цінним інструментом для розробників, який їм допомагає зосередитися на тому, що важливо – розробці високоякісних програм.

## 2.4 Висновки до розділу 2

У розділі «Проєктування» було створено детальну модель майбутнього веб-застосунку для керування будівельними проєктами. Архітектура та функціональна структура застосунку були розроблені з урахуванням найновіших тенденцій в сфері розробки програмного забезпечення. Розділ також містить

проєктування користувацького інтерфейсу, який ставить акцент на зручність та ефективність користувача. При виборі технологій та методів розробки були враховані такі чинники, як стабільність, продуктивність, безпека та зручність подальшого супроводу веб-застосунку.

# 3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

## 3.1 Реалізація логіки додатку

Для реалізації логіки додатку необхідно перш за все реалізувати модель даних додатку. А отже необхідно створити всі необхідні сутності. Розглянемо наступний код що показує головну сутність – Project:

```
public class Project : BaseEntity
{
    public int ProjectId { get; set; }
    public string ProjectName { get; set; }
    public string ProjectType { get; set; }
    public string Country { get; set; }
    public string Address { get; set; }
    public string City { get; set; }
    public decimal Budget { get; set; }
    public string? Description { get; set; }
    public ProjectStatus Status { get; set; }
    public DateTime StartTime { get; set; }
    public DateTime EndTime { get; set; }

    public virtual ICollection<Building> Building { get; set; }
    public virtual ICollection<Phase> Phases { get; set; }
    public virtual ICollection<ProjectDocument> ProjectDocuments { get; set; }
    public virtual ICollection<ProjectNote> ProjectNote { get; set; }
    public virtual ICollection<ProjectPhoto> ProjectPhotos { get; set; }
    public ICollection<Assignee> Assignees { get; set; }

    public long CompanyId { get; set; }
    public Company Company { get; set; }

    public long? UserId { get; set; }
    public User? User { get; set; }

}
```

Клас Project у цьому коді представляє собою сутність «Проєкт» у доменній моделі аплікації, і це є частиною шару домену в архітектурі Clean Architecture.

Він включає в себе основні властивості, які відображають інформацію про проєкт, включаючи ID проєкту (ProjectId), його назву (ProjectName), тип (ProjectType), країну (Country), адресу (Address), місто (City), бюджет (Budget), опис (Description), статус (Status), час початку (StartTime) та час закінчення (EndTime).

У класі Project також є віртуальні колекції для зв'язаних об'єктів, таких як будівлі (Building), фази (Phases), документи проєкту (ProjectDocuments), примітки проєкту (ProjectNote), фотографії проєкту (ProjectPhotos) та відповідальні особи (Assignees). Ці колекції представляють собою зв'язки «один до багатьох» між проєктом і цими сутностями, що дозволяє кожному проєкту мати кілька об'єктів кожного з цих типів.

Окрім того, в класі Project є властивості для CompanyId та Company, які представляють зв'язок «один до одного» між проєктом і компанією, яка його володіє, і UserId та User, які представляють зв'язок «один до одного» між проєктом і користувачем, що його створив.

Такий підхід є частиною архітектури Clean Architecture, де вся бізенс-логіка розміщена в доменному шарі, і цей шар не залежить від інших шарів (наприклад, шару інфраструктури або шару презентації). Це дозволяє забезпечити високу гнучкість та тестованість застосунка.

Після того, як структура бази даних була визначена за допомогою сутностей в програмному коді, використаємо потужну ORM EntityFramework.

Міграції, створені EntityFramework, є описом змін, які потрібно внести до структури бази даних, щоб вона відповідала моделі даних програми.

Щоб застосувати ці зміни до бази даних, використовується команда database-update. Ця команда автоматично визначає, які міграції ще не були застосовані до цільової бази даних, і виконує їх в правильному порядку. Це дозволяє безпечно і статично вносити зміни до бази даних без необхідності ручного управління скриптами міграції.

PostgreSQL – це потужна відкрита система керування базами даних, що підтримує широкий спектр типів даних і має розширені можливості для запитів. EntityFramework підтримує PostgreSQL, тому конфліктів при використанні database-update не буде.

Далі розглянемо клас GenericRepository що реалізує паттер репозиторій та надає додаткову обгортку доступу до даних БД:

```
public class GenericRepository<T> : IGenericRepository<T> where T : BaseEntity
{
    protected ApplicationDbContext _context;
    protected DbSet<T> table;
    private Lazy<IMapper> _mapper;

    public GenericRepository(ApplicationDbContext context, Lazy<IMapper> mapper)
    {
        this._context = context;
        table = _context.Set<T>();
        _mapper = mapper;
    }

    public Task<List<TResult>> Get<TResult>(Expression<Func<T, bool>> filter)
    {
        var query = GetInternal(filter);
```

```
        return query.ProjectTo<TResult>(_mapper.Value.ConfigurationProvider).ToListAsync();
    }

    public Task<List<T>> Get(Expression<Func<T, bool>> filter)
    {
        var query = GetInternal(filter);
        return query.ToListAsync();
    }

    public Task<List<T>> GetAll()
    {
        return table.ToListAsync();
    }
    public Task<List<TResult>> GetAll<TResult>()
    {
        return table.ProjectTo<TResult>(_mapper.Value.ConfigurationProvider).ToListAsync();
    }
    public async Task<T> GetById(long id, CancellationToken ct)
    {
        return await FirstOrDefault(e => e.Id == id, ct)
            ?? throw new NotFoundException();
    }

    public async Task<T?> FirstOrDefault(Expression<Func<T, bool>> expression, CancellationToken cancellationToken = default)
    {
        var query = table;

        return await FirstOrDefaultInternal(query, expression, cancellationToken);
    }

    public async Task<TResult?> FirstOrDefault<TResult>(Expression<Func<TResult, bool>> expression, CancellationToken
cancellationToken = default)
    {
        var query = table.ProjectTo<TResult>(_mapper.Value.ConfigurationProvider);

        return await FirstOrDefaultInternal<TResult>(query, expression, cancellationToken);
    }

}
```

Клас GenericRepository<T> є реалізацією шаблону репозиторія, який використовується для отримання та зберігання даних в базі даних. Він розміщений у шарі інфраструктури Clear Architecture.

Цей клас містить ряд методів, які реалізують загальні операції з базою даних, такі як отримання всіх записів (GetAll), отримання запису за ID (GetById), отримання записів за визначеним фільтром (Get) та отримання першого запису, який відповідає заданому виразу (FirstOrDefault).

У цьому класі T є типом сутності, над якою виконується робота, і це має бути тип, який успадковує BaseEntity.

Конструктор класу приймає ApplicationDbContext (контекст бази даних) та IMapper (інструмент для відображення даних між об'єктами різних типів), які використовуються у всіх методах класу.

Оператор ProjectTo використовується для перетворення об'єктів одного типу на об'єкти іншого типу. Так, наприклад, можна перетворити сутність бази даних на модель DTO.

Важливо зазначити, що всі методи виконуються асинхронно, що означає, що вони не блокують потік виконання програми під час очікування відповіді від бази даних.

Далі реалізуємо основну бізенс-логіку застосунку на прикладі запиту для раніше створеної діаграми GetProjectDetailedQuery:

```
public record GetProjectDetailedQuery(long id) : IRequest<ProjectDetailedDTO>;


public class GetProjectDetailedQueryHandler : IrequestHandler<GetProjectDetailedQuery, ProjectDetailedDTO>
{

   private readonly IapplicationDbContext _context;

   public GetProjectDetailedQueryHandler(IapplicationDbContext context)
   {
      _context = context;
   }

   public async Task<ProjectDetailedDTO> Handle(GetProjectDetailedQuery request, CancellationToken cancellationToken)
   {
      return await _context.Projects.FirstOrDefault<ProjectDetailedDTO>(p => p.Id == request.id, cancellationToken);
   }
}
```

Та як цей запит взаємодії із контроллером ProjectController:

```
[Authorize]
[Route(«api/projects»)]
public class ProjectController : ApiControllerBase
{
   [HttpGet]
   public async Task<ActionResult> GetProjectsWithParams([FromQuery] ProjectPaginationQueryDTO queryModel)
   {
      return Ok(await Mediator.Send(new GetProjectsOverviewQuery(queryModel)));
   }

   [HttpGet(«{id:long}»)]
   public async Task<ActionResult> GetProjectDetailed(long id)
   {
      return Ok(await Mediator.Send(new GetProjectDetailedQuery(id)));
   }

   [HttpPut]
   public async Task<ActionResult> PutProject([FromBody] ProjectDetailedRequestDto project)
   {
      return Ok(await Mediator.Send(new PutProjectDetailedCommand(project)));
   }

   [HttpPost(«team»)]
   public async Task<IactionResult> SetTeam([FromBody] SetProjectTeamCommand command)
   {
      return Ok(await Mediator.Send(command));
   }
}
```

GetProjectDetailedQuery та GetProjectDetailedQueryHandler є частиною шару Application в Clean Architecture. GetProjectDetailedQuery – це клас-команда, що визначає дані, необхідні для отримання деталізованої інформації про проєкт. GetProjectDetailedQueryHandler є обробником цієї команди, який відповідає за виконання відповідної бізенс-логіки.

У цьому випадку, обробник використовує контекст застосунку (IapplicationDbContext) для отримання деталей проєкту за його ID, перетворюючи результат в ProjectDetailedDTO. Це означає, що коли Handle метод цього обробника викликається, він знаходить проєкт у базі даних з вказаним ID і повертає його деталі у вигляді ProjectDetailedDTO.

ProjectController є контролером у шарі WebUI. Він обробляє вхідні HTTP-запити, створює відповідні команди, а потім використовує MediatR для відправки цих команд. Метод GetProjectDetailed приймає параметр id, створює новий екземпляр GetProjectDetailedQuery із цим id та відправляє його через MediatR. Це дозволяє асинхронно викликати відповідний обробник команди (в даному випадку, GetProjectDetailedQueryHandler), що забезпечує отримання деталей проєкту. Результат обробки команди потім повертається клієнту через HTTP-відповідь.

Отже, ці два класи взаємодіють за допомогою MediatR, що дозволяє ефективно роз'єднати відповідальності, забезпечуючи гнучкість та можливість легкої модифікації коду в майбутньому.

Далі створимо front-end частину проєкту. Проаналізуємо html вкладки «Проєкт» даного застосунку:

```
<div class="container">
  <div class="projects-page">
    <div class="head">
      <div class="controll">
        <mat-tab-group (selectedTabChange)="changeStatus($event)">
          <mat-tab label="Current projects"></mat-tab>
          <mat-tab label="Not started"></mat-tab>
          <mat-tab label="Completed"></mat-tab>
          <mat-tab label="Suspended"></mat-tab>
        </mat-tab-group>
        <div class="flex">
          <div mat-icon-button class="sortMenu" [matMenuTriggerFor]="menu" aria-label="Example icon-button with a menu">
            <div class="sort-svg">
              <mat-icon svgIcon="sort"></mat-icon>
            </div>
          </div>
          <mat-menu #menu="matMenu">
            <button mat-menu-item (click)="sortProjectsByName()">
              <span>By name</span>
            </button>
            <button mat-menu-item (click)="sortProjectsByStartTimeDesc()">
              <span>New</span>
            </button>
            <button mat-menu-item (click)="sortProjectsByStartTime()">
              <span>Old</span>
            </button>
          </mat-menu>
          <app-filter-input (onSearchQuery)="onSearchQuery($event)"></app-filter-input>
        </div>
      </div>
    </div>
    <div class="create-project">
```

```
      <!— create project button —>
      <app–button (click)=“add()”>add project</app–button>
      <!—<app–datepicker  class=“endDate”  label=“End time”></app–datepicker>—>
     </div>
    <div class=“projects” *ngIf=“projects$ | async as projects”>
     <app–card
      *ngFor=“let project of projects”
      [cardInformation]=“getCardInformation(project)”
      (click)=“redirectToProjectPage(project.id)”
     >
     </app–card>
    </div>
   </div>
</div>
```

На даному куску верстки ми бачимо групу вкладок <mat-tab-group>, яка дозволяє переключатись між різними станами проєкту: «Current projects», «Not started», «Completed» та «Suspended». Коли вибрана вкладка змінюється, викликається метод changeStatus($event).

Далі є випадаюче меню, яке містить кнопки для сортування проєктів за різними критеріями: за ім'ям, за часом старту (нові або старі).

Компонент <app-filter-input> використовується для фільтрації проєктів за допомогою пошукового запиту. Коли користувач вводить пошуковий запит, викликається метод onSearchQuery($event).

Кнопка «add project» дозволяє створювати новий проєкт, викликаючи метод add() при натиску.

Нарешті, для кожного проєкту в списку projects$, генерується компонент <app-card>, який відображає деталі проєкту. Компоненту передається інформація про проєкт через властивість [cardInformation] та перенаправлення на сторінку проєкту при натиску на картку за допомогою методу redirectToProjectPage(project.id).

У даному прикладі коду використовує Angular Material для UI компонентів і RxJS (асинхронні projects$) для управління даними.

Розберемо на прикладі методу сортування sortProjectsByName як саме ngRx використовується у коді даного застосунку. Спочатку у project.component, в конструкторі компонента ініціалізується NgRx Store. Зокрема, він диспетчеризує дві дії openMenu() та revealMenu(), а також вибирає дані про проєкти зі стану за допомогою this.projects$ = this.store.pipe(select(selectProjects)):

```
this.store.dispatch(openMenu());
```

```
this.store.dispatch(revealMenu());
this.projects$ = this.store.pipe(select(selectProjects));
```

Коли користувач клацне, щоб відсортувати проєкти за іменем, метод sortProjectsByName() в project.component діспетчеризує дію changeParams, передаючи параметри сортування:

```
this.store.dispatch(changeParams({
  params: {
    sort: “ProjectName”,
    order: Order.ASC
  }
}));
```

Дія changeParams визначена в project.actions і призначена для зміни параметрів проєктів:

```
export const changeParams = createAction(
  '[Project List Component] Change Projects\' Params',
  props<{ params: Partial<Params> }>()
)
```

В project.effects, NgRx ефект changeParams$ реагує на дію changeParams, автоматично викликаючи дію getProjectsWithParams():

```
ofType(ProjectActions.changeParams),
map(() => ProjectActions.getProjectsWithParams())
```

Згодом, в project.reducer, дія changeParams обробляється в методі reducer, який змінює стан params у відповідності до даних вантажу (payload) дії changeParams:

```
on(ProjectActions.changeParams, (state, action) => {
  return { ...state, params: { ...state.params, ...action.params } }
}),
```

Таким чином, дія, що була відправлена користувачем через UI (клацання на кнопці для сортування проєктів), проходить через ряд NgRx об'єктів (дії, ефекти, редуктори), змінюючи стан додатку відповідно.

## 3.2 Керівництво користувача

При переході на сайт, перше, що ви побачите – це вікно входу у особистий кабінет користувача. Переконайтеся, що адміністратор вашої компанії створив для вас аккаунт для управління проєктами.

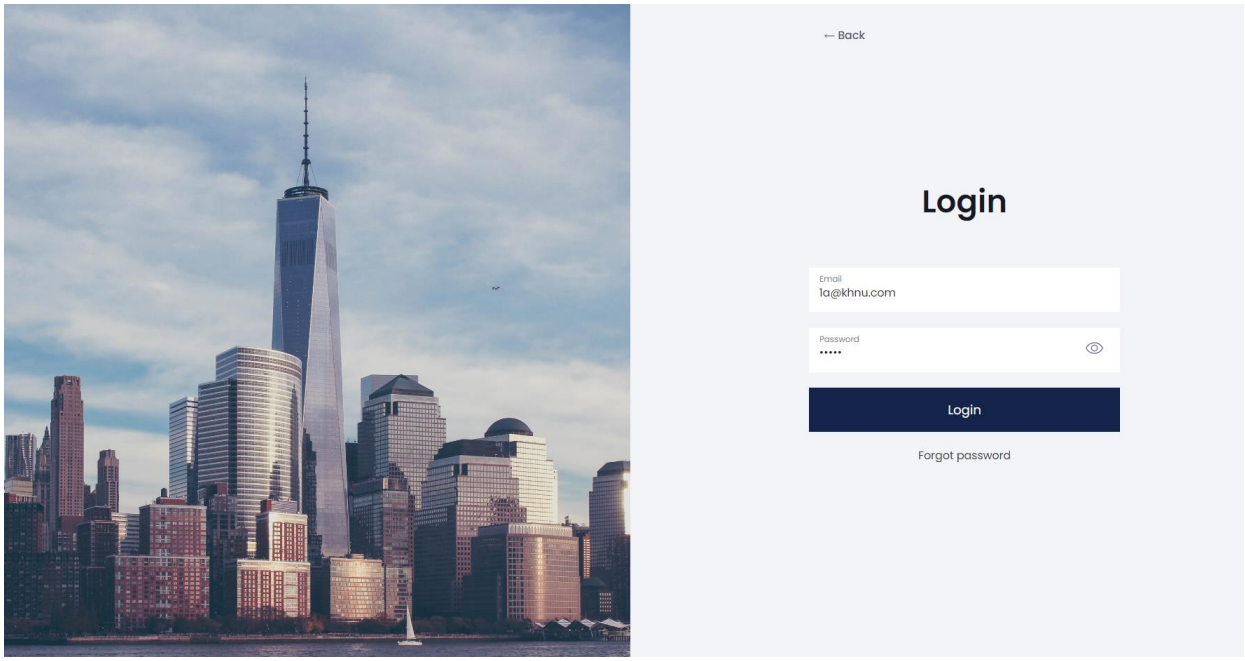



Рисунок 18 – Вікно входу в особистий кабінет користувача

Вам потрібно ввести дані цього аккаунта у відповідні поля. Якщо ви хочете змінити пароль, просто натисніть кнопку «Forgot password».

Будьте уважні при введенні даних, оскільки доступ до вашого кабінету відкриває широкі можливості для управління проєктами компанії.

Після успішного входу в аккаунт, система автоматично перенаправить вас на екран перегляду діючих проєктів вашої компанії(Рисунок 19). Перше, на що варто звернути увагу – це сповіщення про успішний вхід у кабінет, яке розташоване в правому верхньому куті екрана.

Майте на увазі, що зелені сповіщення означають успішні дії, тоді як червоні сповіщення повідомляють про виникнення помилок. Завдяки інтуїтивно зрозумілому інтерфейсу, ви можете легко перемикатися між проєктами різного статусу: «Діючі», «Не початі», «Завершені» та «Відмінені», використовуючи верхню частину екрану.

Додати новий проєкт дуже просто – просто натисніть кнопку «Add project». Для переходу до детального перегляду проєкту, клацніть на карточку відповідного проєкту.

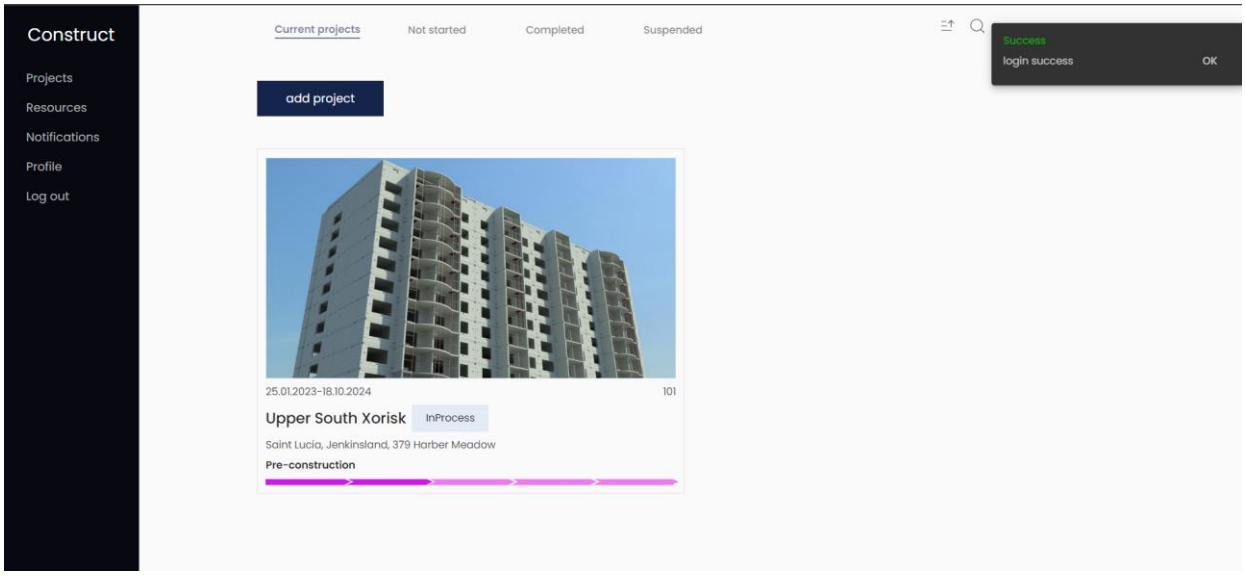



Рисунок 19 – Екран перегляду діючих проєктів

Варто також зазначити, що у лівій частині екрану розташоване меню навігації, яке дозволяє переходити до інших вкладок застосунку, зокрема «Проєкти», «Ресурси», «Нотифікації», «Профіль» та «Вийти».

Клацнемо на карточку проєкту, що нас цікавить. Після цього ми автоматично перейдемо на головне вікно перегляду та редагування проєкту, як це показано на Рисунку 20.

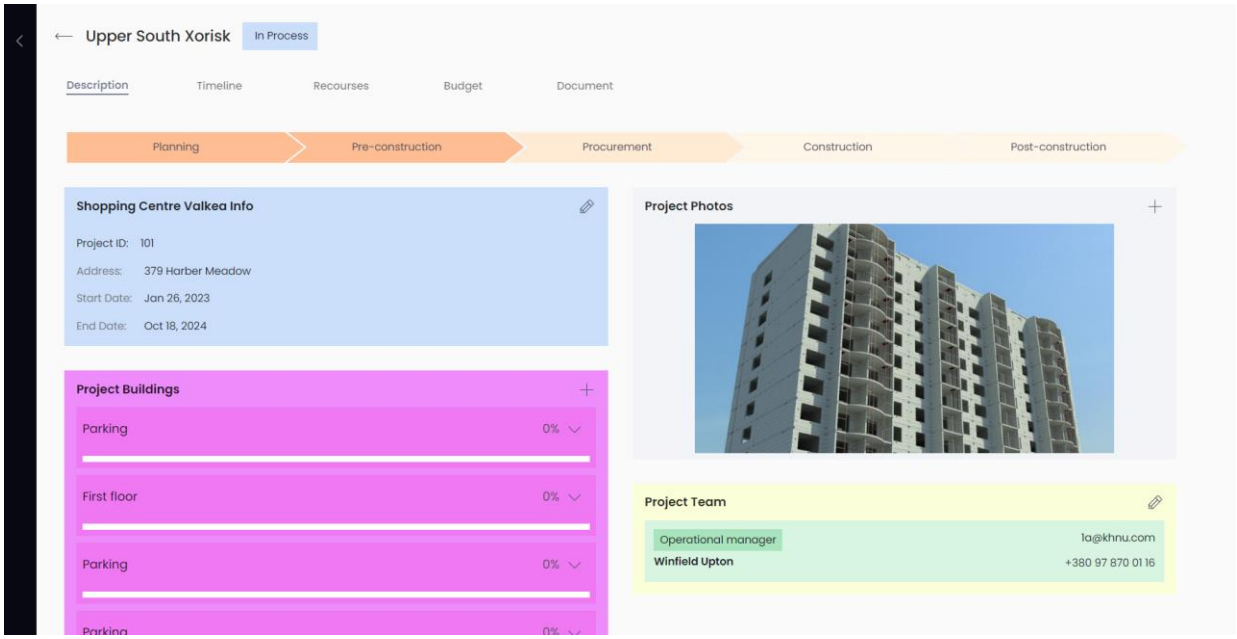



Рисунок 20 – Вікно детального перегляду проєктів

Погляньте вверх екрану. Тут ви зможете побачити поточну стадію проєкту, яка виділена темно-оранжевим кольором для більшої зручності. Побіч знаходяться фотографії проєкту, які ви можете видаляти або переглядати, використовуючи відповідні іконки.

У верхній лівій частині екрану ви знайдете основну інформацію про проєкт. Це включає в себе ідентифікатор проєкту, його адресу, а також дати початку та завершення проєкту. Ця інформація важлива для керування процесом виконання проєкту.

Зверніть увагу на нижню ліву частину екрану. Тут ви побачите список будівель, пов'язаних з проєктом. Для кожної з цих будівель ви можете виділити необхідні матеріали, додати документи та вказати потрібні сервіси.

А тепер давайте перейдемо до нижньої правої частини екрану. Тут розміщена вкладка «Команда». Завдяки їй ви можете легко додати нових членів команди до проєкту або, навпаки, видалити тих, хто вже більше не бере участь у роботі над проєктом.

Щоб оцінити ресурси, що спрямовані на проєкт, давайте перейдемо на вкладку «Ресурси» (Рисунок 21). Як можна побачити, це вікно містить деталізований список всіх ресурсів, пов'язаних з проєктом.


Рисунок 21 – Вікно ресурсів проєкту

Тут можна знайти не тільки назву кожного ресурсу, але і де він зберігається. Додатково, представлена інформація про вартість кожного ресурсу за одиницю та загальну вартість ресурсів, що використовуються в проєкті.

Тепер давайте звернемося до кнопки «Репорт», яка розміщена на цьому екрані. Після клацання на неї, можна завантажити pdf-копію даної сторінки. Ця функція може бути дуже корисною для зберігання або друку звіту про ресурси вашого проєкту.

Наступний етап нашого огляду веде нас до головного меню, а саме до вкладки «Ресурси». Зверніть увагу на перелік ресурсів, які доступні для вашої компанії. Тут ви можете бачити детальну інформацію про кожен ресурс, включаючи їх кількість і одиниці виміру.

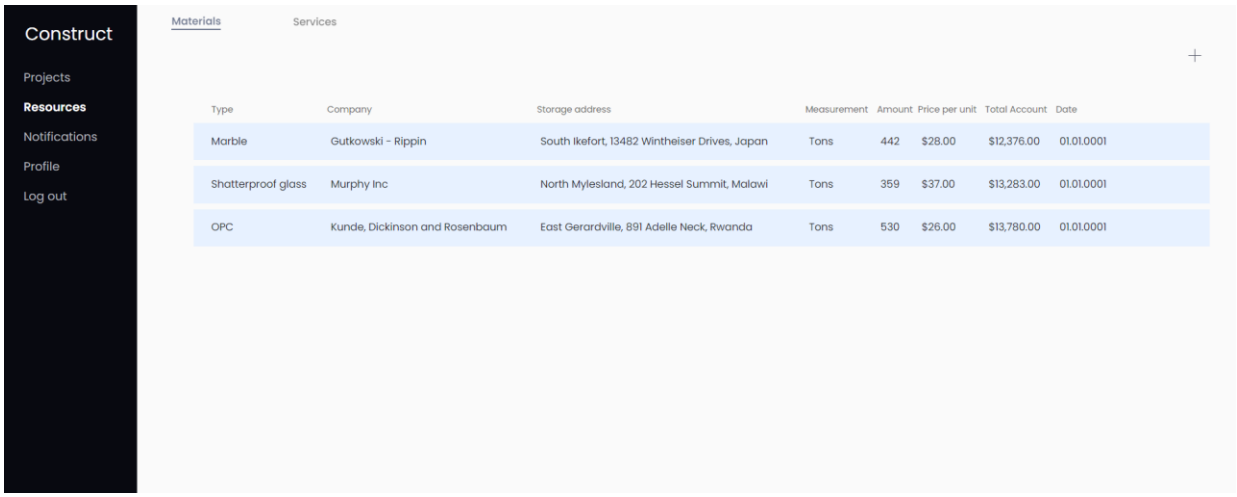



Рисунок 22 – Вікно ресурсів компанії

Іншим корисним елементом на цьому екрані є відображення дати додавання кожного ресурсу. Це допомагає тримати під контролем ресурси та їх використання в різний час.

Але що, якщо ви хочете додати новий ресурс? Немає проблем, просто використовуйте функцію додавання нових ресурсів прямо тут, на цій вкладці.

Також важливо зазначити, що ресурси поділяються на два основних типи: матеріали і сервіси. Це означає, що ви можете легко визначити і відстежувати потреби вашого проєкту в обох цих категоріях.

І останнім кроком даної інструкції буде перехід по вкладці меню «Профіль компанії». Дане вікно відкриває перед нами велику кількість інформації про компанію та її проєкти.

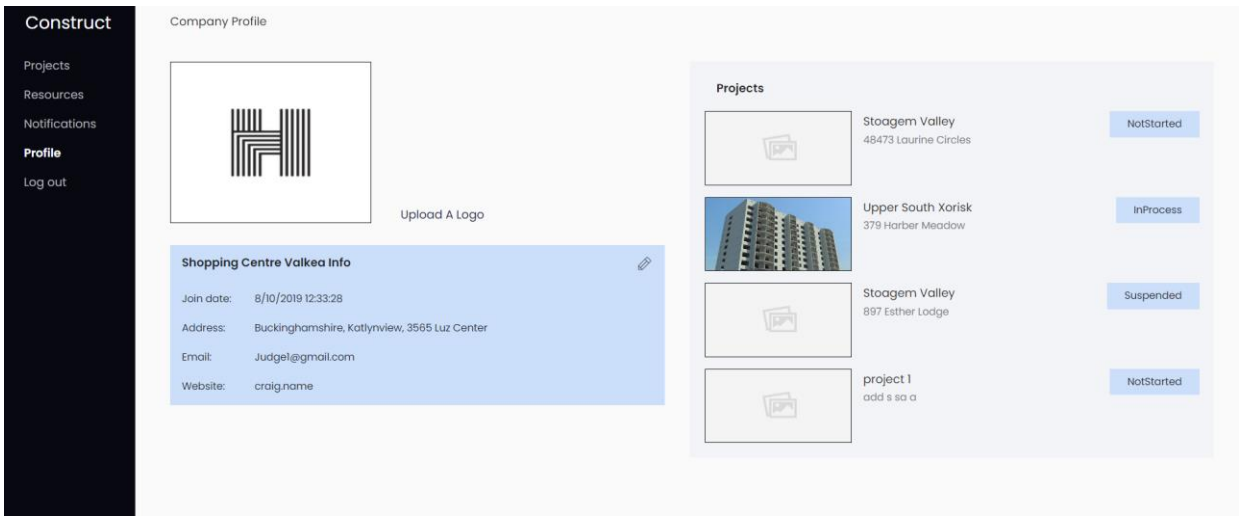



Рисунок 23 – Вікно профілю компанії

Спочатку звернімо увагу на верхній лівий кут екрану, де розташоване лого компанії. Це не просто статичне зображення. Насправді, це інтерактивний елемент, який дає вам можливість внести зміни. Ви можете видалити поточне лого, змінити його або завантажити нове зображення. Таким чином, ви зможете легко оновлювати візуальне представлення вашої компанії в будь-який час.

Тепер давайте перемістимо свій погляд до нижньої лівої частини вікна. Тут ви знайдете корпоративну інформацію про вашу компанію. Серед інших речей, ви можете знайти дату приєднання вашої компанії до платформи, її фізичну адресу, електронну пошту та веб-сайт. Це місце слугує як головний джерело інформації про вашу компанію для всіх користувачів платформи.

## 3.3 Інтеграційне тестування застосунку

Інтеграційні тести – це тип тестування, де окремі модулі об'єднуються і тестуються як група. Ці тести використовуються для виявлення проблем з взаємодією та інтеграцією між різними модулями системи:

```
public class GetTeamTests : ProjectControllerFixture
{
    [Fact]
    public async Task GetsTeamSuccessfully()
    {
        // Arrange
        var projectId = 1;
        var companyId = 1;

        Authorize(2, UserRoles.OperationalManager, companyId);

        // Act
        var response = await Client.GetAsync($"{ProjectControllerApi}/team?projectId={projectId}");

        // Assert
        response.EnsureSuccessStatusCode();

        var dto = await response.Content.ReadFromJsonAsync<GetProjectTeamDto>();

        Assert.NotNull(dto);
        Assert.Equal(1, dto!.Users.Count);
    }
}
```

У наведеному вище коді, інтеграційний тест GetsTeamSuccessfully перевіряє, чи успішно виконується запит на отримання команди проєкту в проєктному контролері.

Він включає такі основні кроки:

— Arrange (Підготовка): У цьому кроці створюються та ініціалізуються вхідні дані для тесту. Тут вказуються projectId та companyId. Також викликається метод Authorize для авторизації користувача з відповідними ролями та companyId.

— Act (Дія): Цей крок включає в себе виклик функціоналу, який потрібно протестувати. У цьому випадку, це HTTP GET запит до контролера проєктів для отримання інформації про команду проєкту.

— Assert (Перевірка): Нарешті, виконуються перевірки для впевненості, що результат відповідає очікуванням. У цьому тесті перевіряється, що відповідь успішна, а також що результат можна успішно десеріалізувати в GetProjectTeamDto. Крім того, перевіряється, що отримана команда проєкту не є null і містить правильну кількість користувачів.

Таким чином, цей інтеграційний тест перевіряє, що функціонал отримання команди проєкту в контролері проєктів працює правильно у випадку, коли користувач має належні ролі та права.

Попередній клас тесту наслідує наступний:

```
public abstract class ProjectControllerFixture : IClassFixture<TestingWebAppFactory<Program>>, IDisposable
{
```

```
    protected const string ProjectControllerApi = "/api/projects";

    protected TestingWebAppFactory<Program> Factory { get; }

    protected HttpClient Client { get; }

    protected ProjectControllerFixture()
    {
        Factory = new TestingWebAppFactory<Program>();
        Client = Factory.CreateClient();
    }

    public void Dispose()
    {
        Client.Dispose();
        Factory.Dispose();
    }

    protected void Authorize(long userId, UserRoles role, long companyId)
    {
        var tokenProvider = Factory.Services.GetRequiredService<ITokenProvider>();
        var token = tokenProvider.GenerateAccessToken(2, UserRoles.OperationalManager, companyId);

        Client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", token.Value);
    }
}
```

Даний клас є основою для створення тестового оточення для тестування контролера проєктів в ASP.NET Core проєкті. Він є частиною фреймворку XUnit і використовується для створення спільних ресурсів, які можуть бути використані в кількох тестах, замість того, щоб створювати їх для кожного окремого тесту:

— ProjectControllerApi – це константа, яка містить базовий шлях до контролера проєктів;

— Factory та Client – це властивості, що зберігають екземпляр TestingWebAppFactory<Program> та HttpClient відповідно. TestingWebAppFactory<Program> використовується для створення тестового оточення, а HttpClient використовується для відправлення HTTP запитів до контролера.

— метод Dispose використовується для звільнення ресурсів після завершення використання класу ProjectControllerFixture;

— метод Authorize використовується для авторизації користувача перед відправленням запиту до контролера. Він генерує токен доступу з вказаними ролями користувача та companyId та встановлює його як заголовок авторизації для майбутніх HTTP запитів.

Загалом, цей клас використовується для налаштування тестового оточення для контролера проєктів, авторизації користувачів та надання спільних ресурсів,

які можуть бути використані в різних тестах.

Далі створимо ще один тест, для перевірки успішного завантаження фото проєкту PostProjectDocument:

```
public class PostProjectDocument : ProjectDocumentControllerFixture
{
    private readonly Testing _testing;

    public PostProjectDocument(Testing testing)
    {
      _testing = testing;
    }

    [Fact]
    public async Task Delete_ProjectDocument()
    {
      // Arrange
      var building = new Building { BuildingName = “name”, ProjectId = 100 };

      await _testing.AddAsync(building);

      await using FileStream file = File.OpenRead(“TestPhotos/dummy.png”);
      using StreamContent content = new(file);

      var postCommand = new PostProjectDocumentCommand(
        new[] { (await content.ReadAsByteArrayAsync(), file.Name) },
        building.Id,
        new[] { “https://icatcare.org/app/uploads/2018/07/Thinking–of–getting–a–cat.png” });

      // Act
      var postResult = await _testing.SendAsync(postCommand);

      // Assert
      postResult.Should().NotBeNullOrEmpty();
      postResult.Should().HaveCount(2);
    }

  }
```

В цьому тесті відбувається наступна послідовність дій:

— у розділі «Arrange» (Підготовка) створюється новий об'єкт Building, який додається до тестового контексту за допомогою методу _testing.AddAsync(building);

— наступним кроком є відкриття файлу зображення за допомогою класу FileStream. Вміст цього файлу перетворюється у StreamContent.

— після цього створюється команда PostProjectDocumentCommand, що має байтовий масив зображення, назву файлу, ідентифікатор будівлі та URL зображення;

— в розділі «Act» (Виконання) виконується команда PostProjectDocumentCommand за допомогою методу _testing.SendAsync(postCommand). Цей крок виконує дію додавання документу до проєкту;

— у розділі «Assert» (Перевірка) виконується кілька перевірок. Спочатку перевіряється, чи результат не є нульовим або порожнім, використовуючи метод Should().NotBeNullOrEmpty(). Наступною перевіркою є переконання, що кількість елементів у результаті дорівнює двом, використовуючи метод Should().HaveCount(2).

У результаті, цей тестовий випадок перевіряє, факт успішного додавання фото до проєкту, і їх наявність у системі після додавання.

## 3.4 Інтерактивне тестування застосунку

Перш за все перевіримо справність проєкту за допомогою ручного тестування, зайдемо у аккаунт користувача та перейдемо по сторінці вікна перегляду проєктів. Тут можемо побачити проєкти що виконуються у нинішній час, функцію сортування, пошуку та швидкий огляд проєктів:

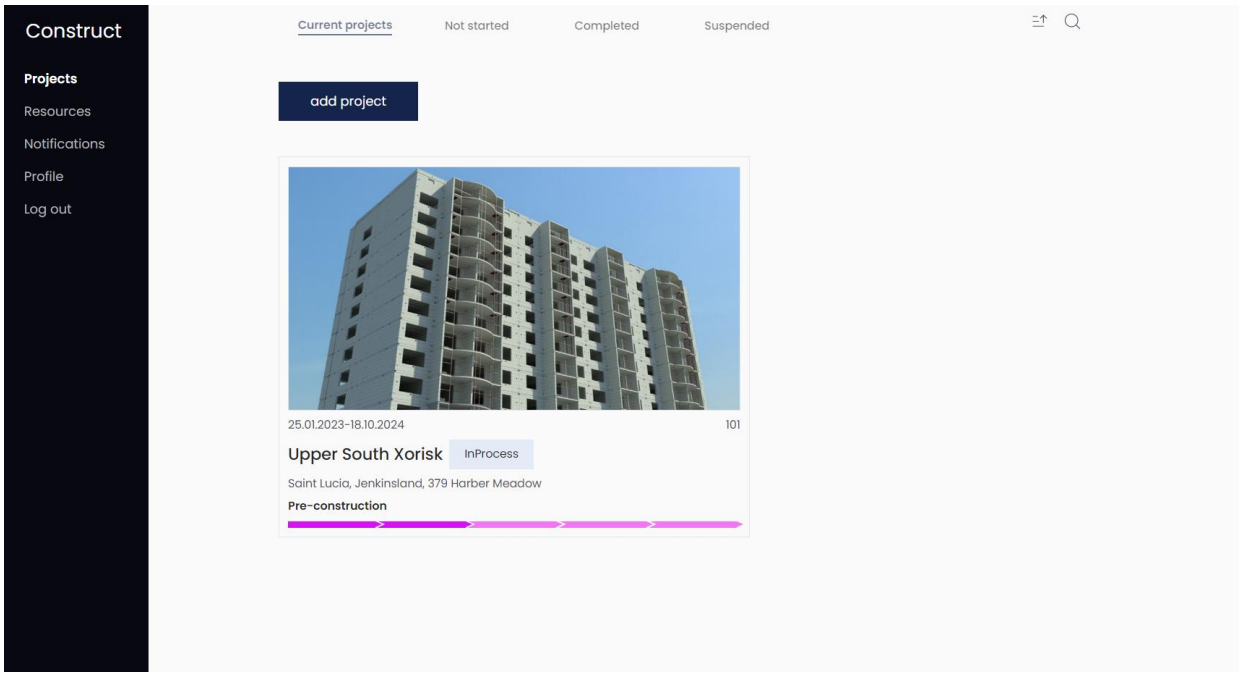



Рисунок 24 – Вікно перегляду проєктів

Далі клацнемо по карточці проєкту та перейдемо на головне вікно перегляду/редагування (Рисунок 25). На екрані можна помітити фотографії

проєкту. Вони надають візуальне представлення проєкту та його розвитку. Ці зображення можна листати, а також видалити за потреби, що дає гнучкість у відображенні та управлінні візуальною інформацією про проєкт.

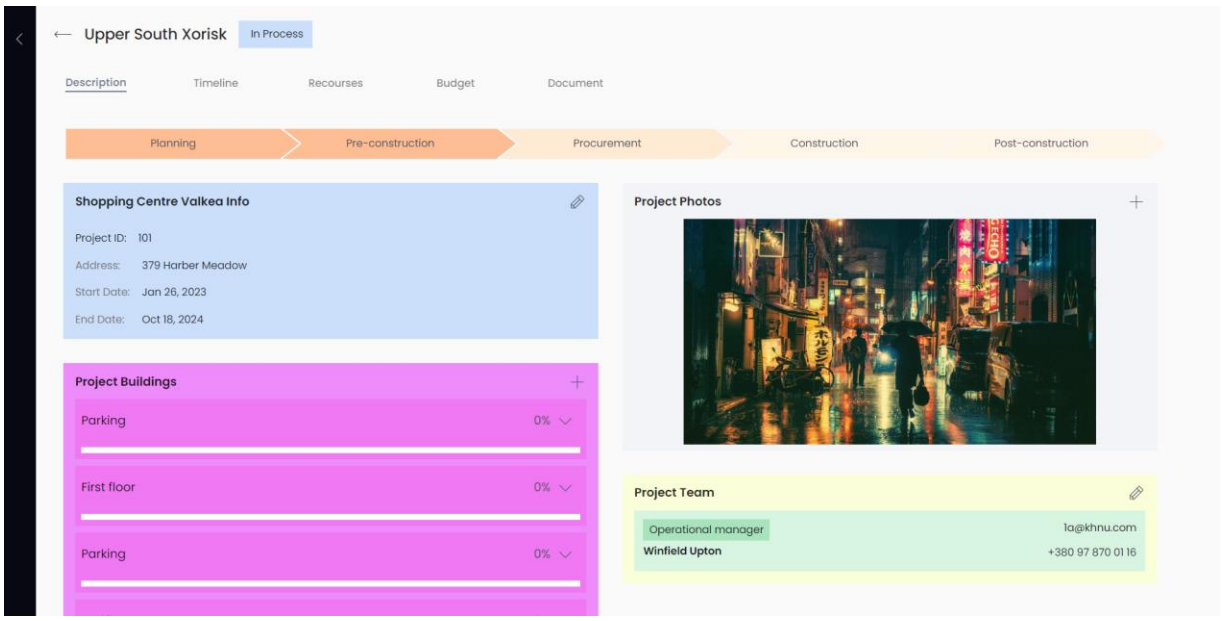



Рисунок 25 – Головне вікно перегляду проєктів

Потім, на екрані відображається основна інформація про проєкт, яка включає деталі, такі як його назва, місце, часові рамки та інше. Ця інформація допомагає керівникам проєкту та іншим учасникам команди бути в курсі основних параметрів проєкту.

Далі, ми бачимо список будівель, що входять до проєкту. Вони детально показують структуру проєкту та різні елементи, з якими працюють команди.

Наостанок, на екрані також присутні різні вкладки, що включають додаткову інформацію та функціонал, необхідний для ефективного менеджменту проєктів. Це можуть бути деталі ресурсів, планування, комунікацій, документів, тощо.

У цілому, цей екран надає повне та гнучке оточення для ефективного управління проєктами.

Далі переглянемо вкладку timeline даного проєкту(Рисунок 26). Вкладка «Timeline» надає детальну візуалізацію прогресу проєкту. Вона використовує

діаграму Ганта, яка є потужним інструментом для представлення часових рамок та етапів реалізації проєкту. За допомогою цієї діаграми користувачі можуть легко відслідковувати поточний етап проєкту і зрозуміти, які завдання вже виконані, а які ще чекають свого виконання.

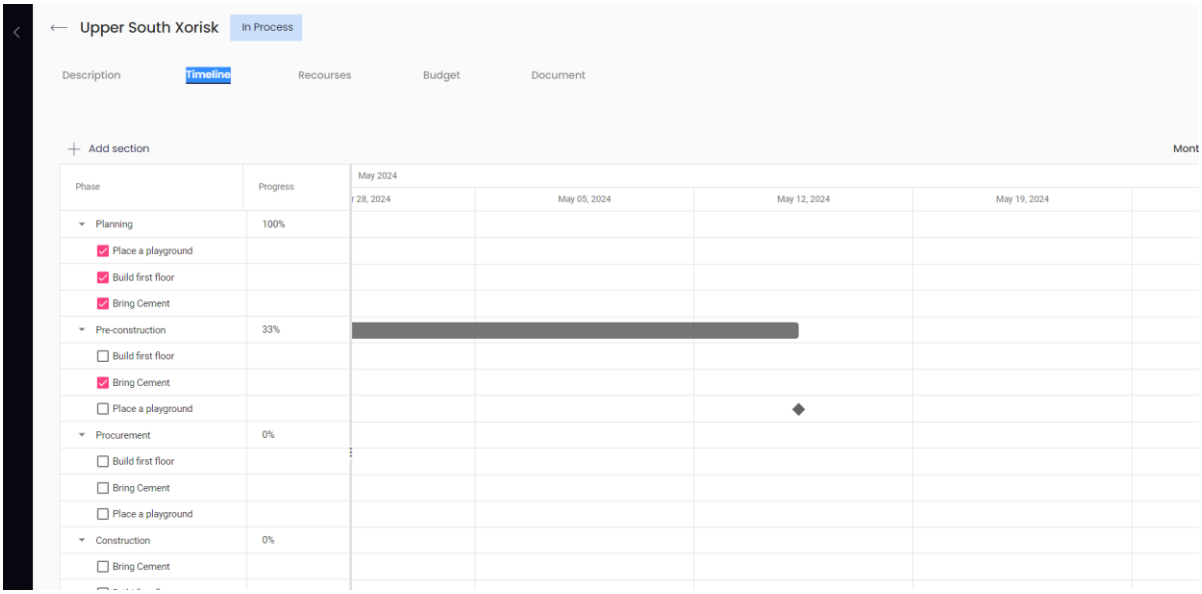



Рисунок 26 – Вкладка timeline

Біля кожної смуги може наведено додаткову інформацію про фазу або підфазу, наприклад, конкретні завдання, які були зроблені в рамках цієї фази. Користувачі також можуть взаємодіяти з діаграмою, натисканням на певну фазу або підфазу, що відкриває додаткові деталі та дозволяє оновити статус завдань.

Таким чином, вкладка «Timeline» не просто відображає поточний стан проєкту, але й служить активним інструментом для планування та контролю реалізації проєкту.

Вікно профілю «Profile» (Рисунок 27) відображає детальну інформацію про компанію та її проєкти. У верхній лівій частині розташоване лого компанії. Це не просто зображення, а інтерактивний елемент. Користувач має можливість видалити поточне лого, внести зміни в нього або завантажити нове, що дозволяє з легкістю підтримувати актуальність візуального представлення компанії.

У нижній лівій частині вікна розміщена корпоративна інформація про компанію. Тут можна знайти дату приєднання до платформи, адресу, електронну

пошту та веб-сайт компанії. Це поле є джерелом основної інформації про компанію для користувачів.

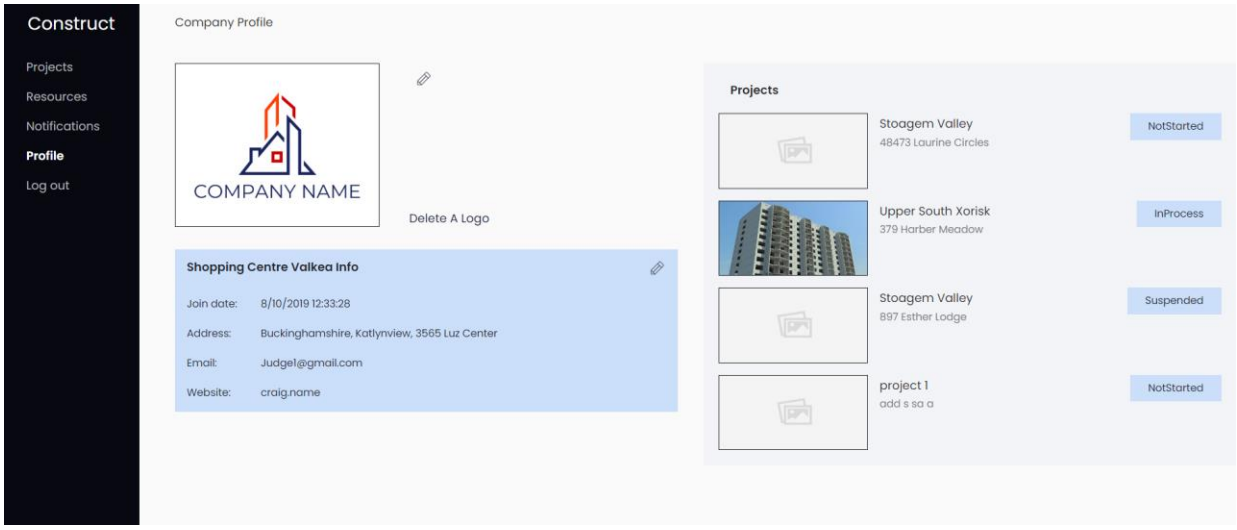



Рисунок 27 – Вікно профілю компанії

Праворуч від центру екрану розташовано список проєктів, які знаходяться в розробці компанією. Кожний проєкт супроводжується коротким описом, що дозволяє користувачам швидко ознайомитися з основними деталями. Вони можуть вибрати будь-який проєкт для отримання більш детальної інформації або управління ним.

## 3.5 Висновки до розділу 3

У третьому розділі «Програмна реалізація та тестування» було зосереджено увагу на розробці та перевірці функціональності веб-застосунку. Розділ включає деталізоване описання процесу програмування логіки застосунку, інтеграції різних компонентів системи, а також проведення інтеграційного та інтерактивного тестування для перевірки правильності роботи системи. На останок, було створено керівництво користувача яке дозволить новим користувачам легко освоїти використання веб-застосунку.

# ВИСНОВКИ

Кваліфікаційна робота мала за мету розробити веб-застосунок для керування проєктами будівельної компанії. Розробка його архітектури, написання коду та дизайн його інтерфейсу В ході розробки було виконано комплекс досліджень, проєктування, програмування та тестування, що дозволило успішно досягнути поставленої мети. Об'єктом дослідження є процес проєктування та розробки веб-застосунку для керування проєктами будівельної компанії. Предметом дослідження – методи проєктування та розробки веб-застосунку для керування проєктами будівельної компанії. Завданням – проектування та розробка веб-застосунку для керування проєктами будівельної компанії

У першому розділі дослідження предметної області було проведено глибокий аналіз сучасних підходів до керування проєктами в будівельній сфері. Це дослідження включало аналіз наявного програмно-технічного забезпечення, що використовується в цій області, а також визначення основних вимог до розроблюваного веб-застосунку. В результаті цього аналізу було сформульовано технічне завдання на розробку веб-застосунку.

У другому розділі, що відповідає за проєктування системи, було розроблено детальну модель веб-застосунку. Перш за все, було визначено архітектуру та функціональну структуру веб-застосунку, враховуючи сучасні підходи та методики розробки. Було також спроєктовано інтерфейс користувача, що орієнтовано на зручність та ефективність роботи. Під час вибору технологій і методів розробки було зосереджено увагу на стабільності, продуктивності, безпеці та зручності подальшого супроводу.

Третій розділ роботи зосереджений на програмній реалізації та тестуванні розробленого застосунку. Цей розділ описує деталі програмування логіки додатку, процес інтеграції різних компонентів системи та подальшого їх тестування. Розділ також містить керівництво користувача, що допоможе новим користувачам швидко зорієнтуватися у системі.

Усе це дозволило розробити веб-застосунок, що надає користувачам можливість ефективно керувати проєктами будівельної компанії, використовуючи сучасні веб-технології. Завдяки інтуїтивно зрозумілому інтерфейсу, а також набору потужних інструментів для планування, моніторингу та контролю проєктів, користувачі можуть значно підвищити ефективність своєї роботи.

У результаті, кваліфікаційна робота дозволила успішно спроєктувати та реалізувати розробку веб-застосунку для керування проєктами будівельної компанії. Мета роботи була досягнута, а основні задачі виконані. Подальший розвиток цього проєкту може включати додаткову адаптацію застосунку до специфічних потреб будівельної компанії та додавання нового функціоналу.

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ


1. Методичні вказівки до виконання дипломного проекту для студентів спеціальності ІПЗ 121 «Інженерія програмного забезпечення». URL: https://msn.khmnu.edu.ua/pluginfile.php/626097/mod_resource/content/1/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4%D0%B8%D1%87%D0%BA%D0%B0%20%D0%BF%D0%BE%20%D0%94%D0%9F.pdf (дата звернення: 20.03.2023).
2. Clean Architecture with ASP.NET Core. URL: https://ardalis.com/clean-architecture-asp-net-core/ (дата звернення: 26.02.2023).
3. Visual Studio. URL: https://visualstudio.microsoft.com/ (дата звернення: 20.02.2023).
4. Офіційна документація Microsoft .Net documentation. URL: https://docs.microsoft.com/en-us/dotnet/ (дата звернення: 10.04.2022).
5. Офіційна документація Angular Docs. URL: https://angular.io/docs (дата звернення: 13.04.2022).
6. What is building management company. URL: https://www.allpropertymanagement.com/blog/post/what-is-a-building-management-company/ (дата звернення: 23.02.2023).
7. Офіційна документація NgRx. URL: https://ngrx.io/guide/store (дата звернення: 05.04.2023).
8. Angular NgRx Entity – Complete Practical Guide. URL: https://blog.angular-university.io/ngrx-entity/ (дата звернення: 06.05.2023).
9. Using The CQRS Pattern In C#. URL: https://www.c-sharpcorner.com/article/using-the-cqrs-pattern-in-c-sharp/ (дата звернення: 03.03.2023).
10. Angular NgRx Entity – Complete Practical Guide. URL: https://blog.angular-university.io/ngrx-entity/ (дата звернення: 06.05.2023).

11. Using The CQRS Pattern In C#. URL: https://www.c-sharpcorner.com/article/using-the-cqrs-pattern-in-c-sharp/. (дата звернення: 03.03.2023.)

12. Ford N., Richards M., Sadalage P. Software Architecture: The Hard Parts: Modern Trade-Off Analyses for Distributed Architectures. Sebastopol : O'Reilly. С. 75.

13. Bellemare A. Building Event-Driven Microservices: O'Reilly. С. 86.

14. Skeet J. C# in Depth. Shelter Island : Manning Publications, 2019. С. 64.

15. Box D., Sells C. Essential .NET, Volume I: The Common Language Runtime. Boston : Addison-Wesley Professional, 2003. С. 86.

16. Fowler M. Patterns of Enterprise Application Architecture. Boston : Addison-Wesley Professional, 2002. 98 с.

17. Hohpe G., Woolf B. Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Boston : Addison-Wesley Professional, 2003. С. 93.

18. Seshadri S. Angular: Up and Running. Sebastopol : O'Reilly Media, 2018. С. 67.

19. Fain Y., Moiseev A. Angular Development with Typescript. Shelter Island : Manning Publications, 2018. С. 88.

20. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Boston : Addison-Wesley Professional, 1994. С. 70.

21. Freeman E., Bates B., Sierra K., Robson E. Head First Design Patterns. Sebastopol : O'Reilly Media, 2004. С. 92.

22. Krause L. Microservices: Patterns and Applications: Designing fine-grained services by applying patterns. Victoria, BC : Leanpub, 2016. С. 85.

23. Metzgar D. .NET Core in Action. Shelter Island : Manning Publications, 2018. С. 94.

24. Reynders F. Modern API Design with ASP.NET Core 2: Building Cross-Platform Backend Systems. Berkeley, CA : Apress, 2018. С. 68.

25. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software. Boston : Addison-Wesley Professional, 2003. С. 99.

26. Vernon V. Implementing Domain-Driven Design. Boston : Addison-Wesley Professional, 2013.

27. Newman S. Building Microservices: Designing Fine-Grained Systems. Sebastopol : O'Reilly Media, 2015. С. 92.

28. Richardson C. Microservices Patterns: With examples in Java. Shelter Island : Manning Publications, 2018. С. 63.

29. Lock A. ASP.NET Core in Action. Shelter Island : Manning Publications, 2018. С. 98.

30. Uluca D. Angular for Enterprise-Ready Web Applications. Birmingham : Packt Publishing, 2019. С. 77.

31. Kunz G. Mastering Angular Components. Birmingham : Packt Publishing, 2018. С. 89.

32. Terrell R. Functional Concurrency in .NET: With examples in C# and F#. Shelter Island : Manning Publications, 2018. С. 95.

33. Martin R.C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Boston : Prentice Hall, 2017. С. 78.

34. Cleary S. Concurrency in C# Cookbook: Asynchronous, Parallel, and Multithreaded Programming. Sebastopol : O'Reilly Media, 2014. С. 92.

35. Noring C., Kraus M. Learn NgRx: Harness the power of reactive programming with Angular and ngrx. Birmingham : Packt Publishing, 2020. С. 81.

36. Koutnik R. Build Reactive Websites with RxJS: Master Observables and Wrangle Events. Raleigh, NC : Pragmatic Bookshelf, 2019.

37. Daniels P. P., Atencio L. RxJS in Action. Shelter Island : Manning, 2017.

38. Noring C. Architecting Angular Applications with Redux, RxJS, and NgRx: Learn to build Redux style high-performing applications with Angular 6. Birmingham : Packt Publishing, 2018.

39. Oliveira E. de S. Mastering Reactive JavaScript: Building asynchronous and high performing web apps with RxJS. Birmingham : Packt Publishing, 2017.

40. Kalb I. Object-Oriented Python: Master OOP by Building Games and GUIs. San Francisco : No Starch Press, 2022.

.

**ДОДАТОК А**
(обов’язковий)

**КОД КЛАСІВ**

```
public class Project : BaseEntity
{
    public int ProjectId { get; set; }
    public string ProjectName { get; set; }
    public string ProjectType { get; set; }
    public string Country { get; set; }
    public string Address { get; set; }
    public string City { get; set; }
    public decimal Budget { get; set; }
    public string? Description { get; set; }
    public ProjectStatus Status { get; set; }
    public DateTime StartTime { get; set; }
    public DateTime EndTime { get; set; }

    public long CompanyId { get; set; }
    public Company Company { get; set; }

    public long? UserId { get; set; }
    public User? User { get; set; }

      }



[Authorize]
[Route("api/projects")]
public class ProjectController : ApiControllerBase
{
    [HttpGet]
    public async Task<ActionResult> GetProjectsWithParams([FromQuery]
ProjectPaginationQueryDTO queryModel)
    {
        return Ok(await Mediator.Send(new GetProjectsOverviewQuery(queryModel)));
    }

    [HttpGet("{id:long}")]
    public async Task<ActionResult> GetProjectDetailed(long id)
    {
        return Ok(await Mediator.Send(new GetProjectDetailedQuery(id)));
    }

    [HttpPut]
    public async Task<ActionResult> PutProject([FromBody] ProjectDetailedRequestDto
project)
    {
        return Ok(await Mediator.Send(new PutProjectDetailedCommand(project)));
    }

    [HttpPost("")]
    public async Task<IActionResult> CreateProjects([FromBody] CreateProjectDTO
project)
    {
        return Ok(await Mediator.Send(new CreateProjectCommand(project)));
    }

    [HttpGet("company/{id}")]
    public async Task<ActionResult<List<ProjectOverviewDto>>>
GetProjectsByCompanyId([FromRoute] long id)
    {
        return await Mediator.Send(new GetProjectsByCompanyIdQuery(id));
    }

    [HttpGet("{id:long}/photos")]
```

```
    public async Task<ActionResult<List<GetProjectPhotoByProjectIdResponseDto>>>
GetProjectPhotos(
        long id)
    {
        return await Mediator.Send(
            new GetProjectPhotosByProjectIdQuery(id));
    }

    [HttpPut("{id:long}/photos")]
    public async Task<ActionResult<List<PutProjectPhotoResponseDto>>> PutProjectPhotos(
        [FromForm] IFormCollection data, long id)
    {
        return await Mediator.Send(
            new PutProjectPhotosCommand(await Task.WhenAll(data.Files.Select(file =>
file.GetBytes())), id));
    }

    [HttpDelete("{id:long}/photos/{photoId:long}")]
    public async Task<ActionResult<DeleteProjectPhotoByIdResponseDto>>
DeleteProjectPhotoById(long id, long photoId)
    {
        return await Mediator.Send(new DeleteProjectPhotoByIdCommand(id, photoId));
    }

    [HttpPut("change-status")]
    public async Task<IActionResult> ChangeStatus([FromBody] ChangeStatusCommand
command)
    {
        return Ok(await Mediator.Send(command));
    }

    [HttpGet("team")]
    public async Task<IActionResult> GetTeam([FromQuery] GetProjectTeamQuery query)
    {
        return Ok(await Mediator.Send(query));
    }

    [HttpPost("team")]
    public async Task<IActionResult> SetTeam([FromBody] SetProjectTeamCommand command)
    {
        return Ok(await Mediator.Send(command));
    }
      }



public record GetProjectsOverviewQuery(ProjectPaginationQueryDTO QueryModel) :
IRequest<PaginationModel<ProjectOverviewDto>>;

public class GetProjectsOverviewQueryHandler :
IRequestHandler<GetProjectsOverviewQuery, PaginationModel<ProjectOverviewDto>>
{
    private readonly IApplicationDbContext _context;
    private readonly ICurrentUserService _currentUserService;

    public GetProjectsOverviewQueryHandler(IApplicationDbContext context,
ICurrentUserService currentUserService)
    {
        _context = context;
        _currentUserService = currentUserService;
    }

    public async Task<PaginationModel<ProjectOverviewDto>>
Handle(GetProjectsOverviewQuery request, CancellationToken cancellationToken)
```

```
    {
        var query = request.QueryModel.Query;

        Expression<Func<Domain.Entities.Project, bool>> filterPredicate = project
            => (string.IsNullOrWhiteSpace(query) ||
project.ProjectName.ToLower().Contains(query.ToLower()))
                && (project.UserId == _currentUserService.Id!.Value ||
project.Assignees.Any(a => a.User.Id == _currentUserService.Id!.Value))
                && project.Status == request.QueryModel.Status;

        var sort = request.QueryModel.Sort;
        var order = request.QueryModel.Order;
        var page = request.QueryModel.Page;
        var count = request.QueryModel.Count;

        var (projects, total) = await _context.Projects
            .GetFilteredWithTotalSum<ProjectOverviewDto>(filterPredicate, page, count,
sort, order);

        return new() { List = projects, Total = total };
    }
      }


public record GetProjectDetailedQuery(long id) : IRequest<ProjectDetailedDTO>;

public class GetProjectDetailedQueryHandler : IRequestHandler<GetProjectDetailedQuery,
ProjectDetailedDTO>
{

    private readonly IApplicationDbContext _context;

    public GetProjectDetailedQueryHandler(IApplicationDbContext context)
    {
        _context = context;
    }

    public async Task<ProjectDetailedDTO> Handle(GetProjectDetailedQuery request,
CancellationToken cancellationToken)
    {
        return await _context.Projects.FirstOrDefault<ProjectDetailedDTO>(p => p.Id ==
request.id, cancellationToken);
    }
}


public class GetProjectsDetailedQueryValidator :
AbstractValidator<GetProjectDetailedQuery>
{
    public GetProjectsDetailedQueryValidator()
    {
        RuleFor(p => p.id).GreaterThan(0);
    }
}


public class ProjectDetailedDTO : ProjectOverviewDto, IMapFrom<Project>
{

    public override void Mapping(Profile profile)
    {
        profile.CreateMap<Project, ProjectDetailedDTO>()
            .ForMember(dest => dest.StartTime, opt => opt.MapFrom(src =>
src.StartTime.ToString()))
```

```
            .ForMember(dest => dest.EndTime, opt => opt.MapFrom(src =>
src.EndTime.ToString()))
            .ForMember(dest => dest.Status, opt => opt.MapFrom(src =>
src.Status.ToString()))
            .ForMember(dest => dest.Phases, opt => opt.MapFrom(src =>
src.Phases.OrderBy(p => p.PhaseStep)));
    }
}


public record GetProjectsOverviewQuery(ProjectPaginationQueryDTO QueryModel) :
IRequest<PaginationModel<ProjectOverviewDto>>;

public class GetProjectsOverviewQueryHandler :
IRequestHandler<GetProjectsOverviewQuery, PaginationModel<ProjectOverviewDto>>
{
    private readonly IApplicationDbContext _context;
    private readonly ICurrentUserService _currentUserService;

    public GetProjectsOverviewQueryHandler(IApplicationDbContext context,
ICurrentUserService currentUserService)
    {
        _context = context;
        _currentUserService = currentUserService;
    }

    public async Task<PaginationModel<ProjectOverviewDto>>
Handle(GetProjectsOverviewQuery request, CancellationToken cancellationToken)
    {
        var query = request.QueryModel.Query;

        Expression<Func<Domain.Entities.Project, bool>> filterPredicate = project
            => (string.IsNullOrWhiteSpace(query) ||
project.ProjectName.ToLower().Contains(query.ToLower()))
                && (project.UserId == _currentUserService.Id!.Value ||
project.Assignees.Any(a => a.User.Id == _currentUserService.Id!.Value))
                && project.Status == request.QueryModel.Status;

        var sort = request.QueryModel.Sort;
        var order = request.QueryModel.Order;
        var page = request.QueryModel.Page;
        var count = request.QueryModel.Count;

        var (projects, total) = await _context.Projects
            .GetFilteredWithTotalSum<ProjectOverviewDto>(filterPredicate, page, count,
sort, order);

        return new() { List = projects, Total = total };
    }
}
public class ProjectOverviewDto : IMapFrom<Domain.Entities.Project>
{
    public long Id { get; set; }
    public string ProjectName { get; set; }
    public string ProjectType { get; set; }
    public string Country { get; set; }
    public string Address { get; set; }
    public string City { get; set; }
    public string StartTime { get; set; }
    public string EndTime { get; set; }
    public string Status { get; set; }
    public List<PhaseOverviewDto> Phases { get; set; }
    public string? ImageUrl { get; set; }
```

```
    public virtual void Mapping(Profile profile)
    {
        profile.CreateMap<Project, ProjectOverviewDto>()
            .ForMember(dest => dest.ImageUrl, opt => opt.MapFrom(src =>
src.ProjectPhotos.FirstOrDefault()!.Link))
            .ForMember(dest => dest.StartTime, opt => opt.MapFrom(src =>
src.StartTime.ToString("dd.MM.yyyy")))
            .ForMember(dest => dest.EndTime, opt => opt.MapFrom(src =>
src.EndTime.ToString("dd.MM.yyyy")))
            .ForMember(dest => dest.Status, opt => opt.MapFrom(src =>
src.Status.ToString()))
            .ForMember(dest => dest.Phases, opt => opt.MapFrom(src =>
src.Phases.OrderBy(p => p.PhaseStep)));
    }
}



public class GetProjectPhotosByProjectIdQueryHandler :
IRequestHandler<GetProjectPhotosByProjectIdQuery,
List<GetProjectPhotoByProjectIdResponseDto>>
{
    private readonly IApplicationDbContext _context;
    private readonly IMapper _mapper;

    public GetProjectPhotosByProjectIdQueryHandler(IApplicationDbContext context,
IMapper mapper)
    {
        _context = context;
        _mapper = mapper;
    }

    public async Task<List<GetProjectPhotoByProjectIdResponseDto>>
Handle(GetProjectPhotosByProjectIdQuery request,
        CancellationToken cancellationToken)
    {
        var projectPhotos = await _context.ProjectsPhotos.Get(f => f.ProjectId ==
request.ProjectId);
        if (!projectPhotos.Any()) throw new NotFoundException("No photos for project
was found");

        var response =
_mapper.Map<List<GetProjectPhotoByProjectIdResponseDto>>(projectPhotos);
        return response;
    }
}
@Component({
  selector: 'app-project-description[projectId]',
  templateUrl: './project-description.component.html',
  styleUrls: ['./project-description.component.scss'],
})
export class ProjectDescriptionComponent implements OnInit {

  @Input() projectId: number = 0;

  project$?: Observable<IProjectDetailed>
  projectInformationForm$?:
Observable<FormGroupState<fromProjectInformationForm.ProjectInformationFormValue>>;

  isEditEnabled = false

  teamMembers$ = this.store.select(fromProjectSelectors.selectCurrentProjectTeam);

  hasTeamMembers$ = this.teamMembers$.pipe(
```

```
    map(members => members && members.length > 0)
  );

  constructor(
    private store: Store<AppState>,
    iconRegistry: MatIconRegistry,
    sanitizer: DomSanitizer,
  ) {
    iconRegistry.addSvgIcon(
      'pencil',
      sanitizer.bypassSecurityTrustResourceUrl('assets/icons/pencil.svg')
    );
    iconRegistry.addSvgIcon(
      'plus',
      sanitizer.bypassSecurityTrustResourceUrl('assets/icons/plus.svg')
    );
    iconRegistry.addSvgIcon(
      'copy',
      sanitizer.bypassSecurityTrustResourceUrl('assets/icons/copy.svg')
    );
  }

  addPhotosModal(id: number) {
    this.store.dispatch(
      openModalDialog({
        component: AddProjectPhotosComponent, config: {
          data: id
        }
      }))
    this.projectInformationForm$ = this.store.pipe(
      select(selectProjectInformationForm)
    );
  }

  changeEdit(event: Event) {
    event.preventDefault()
    this.isEditEnabled = !this.isEditEnabled
  }


  onFormEdit() {
    this.isEditEnabled = true;
    this.store.dispatch((editProjectInformationForm()));
  }

  onCancel() {
    this.isEditEnabled = false;
    this.store.dispatch(cancelEditProjectInformationForm());
  }

  ngOnInit(): void {
    this.project$ = this.store.pipe(select(selectProjectInformation))
    this.projectInformationForm$                                                    =
this.store.pipe(select(selectProjectInformationForm))

    this.store.dispatch(fromProjectActions.getProjectTeam({  projectId:  this.projectId
}));
  }

  onSave(id: number, form: fromProjectInformationForm.ProjectInformationFormValue) {
    this.isEditEnabled = false;

    this.store.dispatch(
      submitProjectInformationForm({
```

```
        id: id,
        address: form.address,
        startTime: form.startTime,
        endTime: form.endTime
      } as IProjectUpdate)
    );
  }

  addTeamModal() {
    this.store.dispatch(
      openModalDialog({
        component: AddProjectTeamComponent, config: {
          data: {
            projectId: this.projectId,
          },
          autoFocus: false,
        },
      })
    );
  }

  mapPhases(phases: IPhaseOverviewDTO[]): { name: string, isFinished: boolean }[] {
    return phases
      .map(phase => {
        return {
          name: phase.phaseName,
          isFinished: phase.isFinished,
        }
      });
  }

}
```

**ДОДАТОК Б**
(обов’язковий)

**ПРЕЗЕНТАЦІЙНИЙ МАТЕРІАЛ**


Рисунок Б.1 – Слайд №1

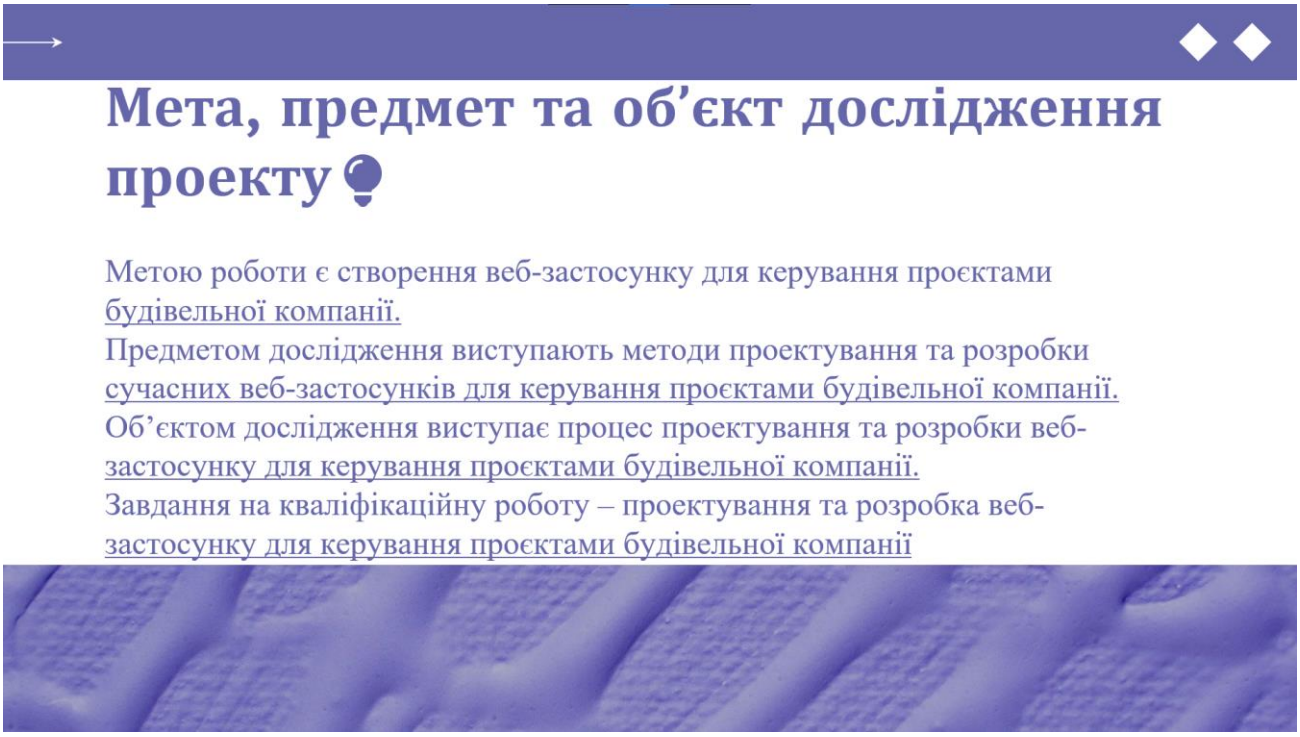



Рисунок Б.2 – Слайд №2


Рисунок Б.3 – Слайд №3

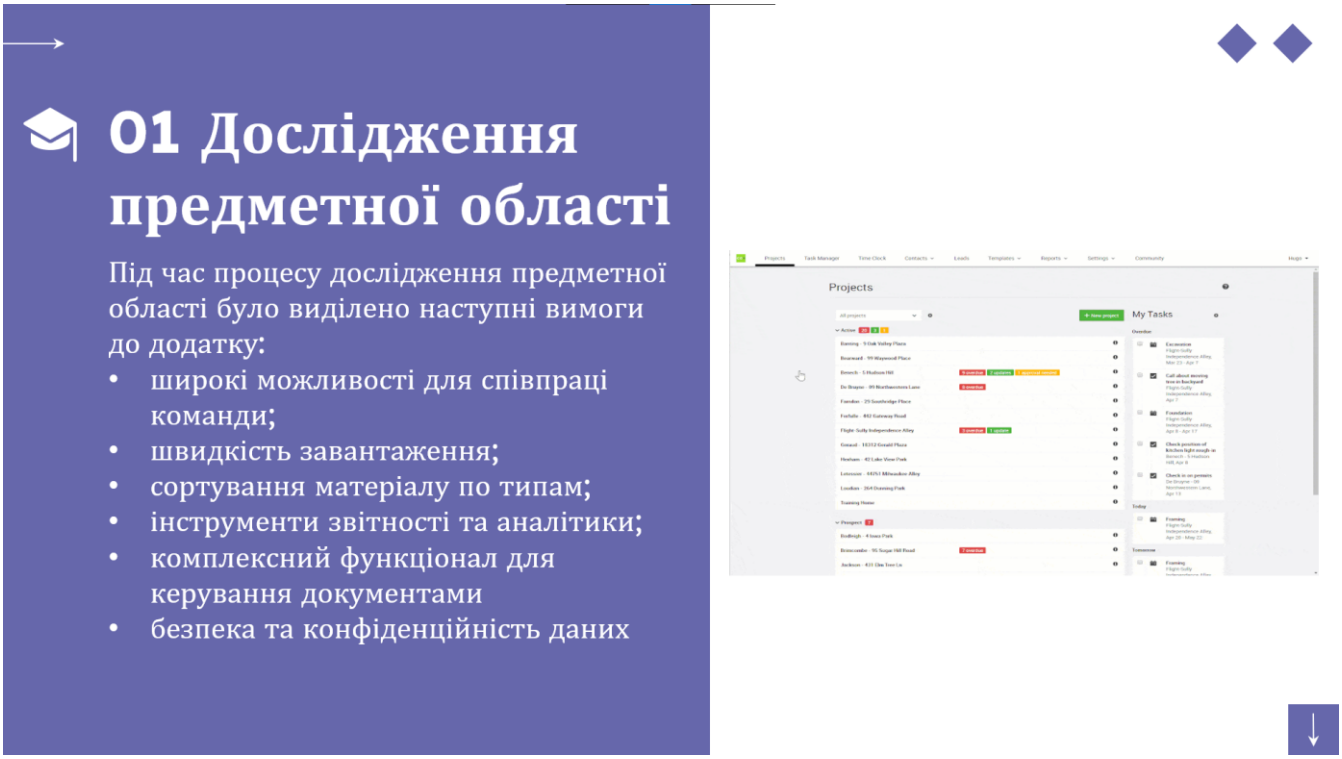



Рисунок Б.4 – Слайд №4

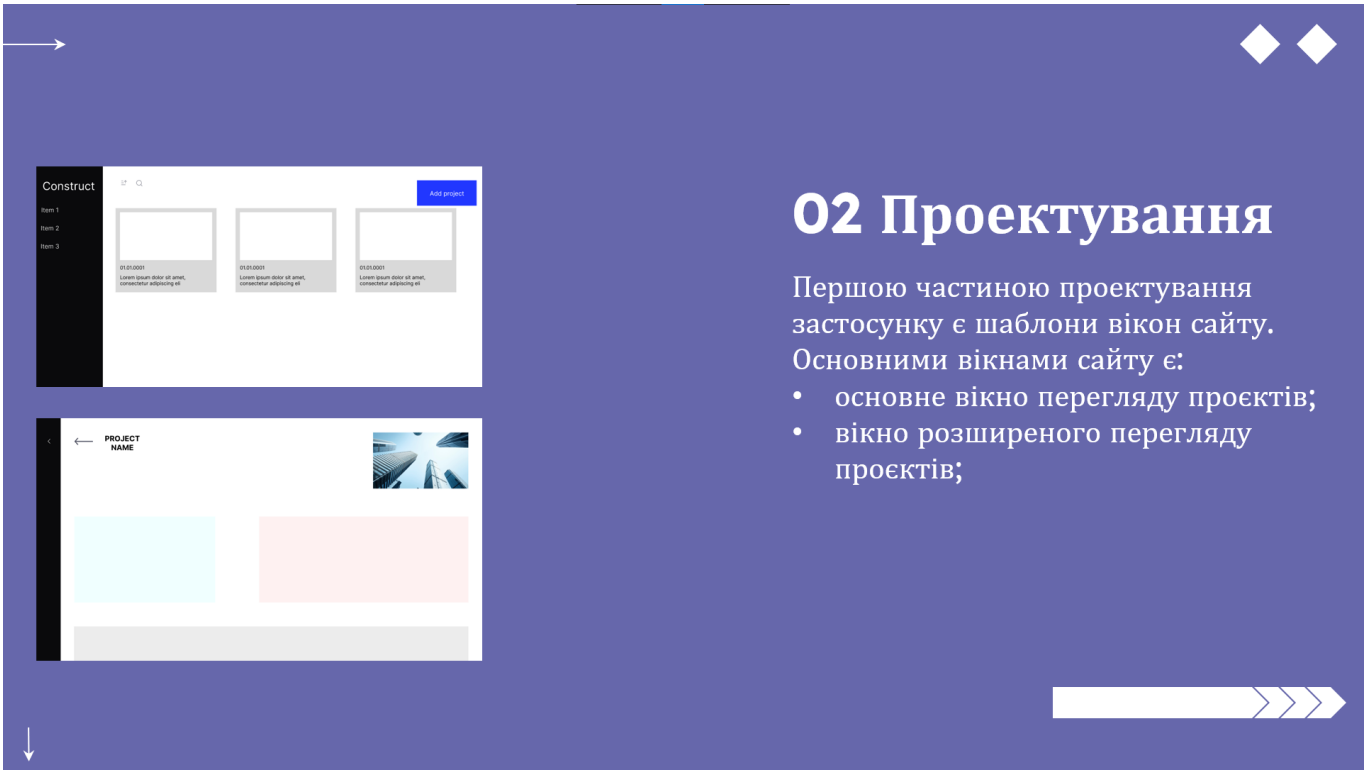



Рисунок Б.5 – Слайд №5

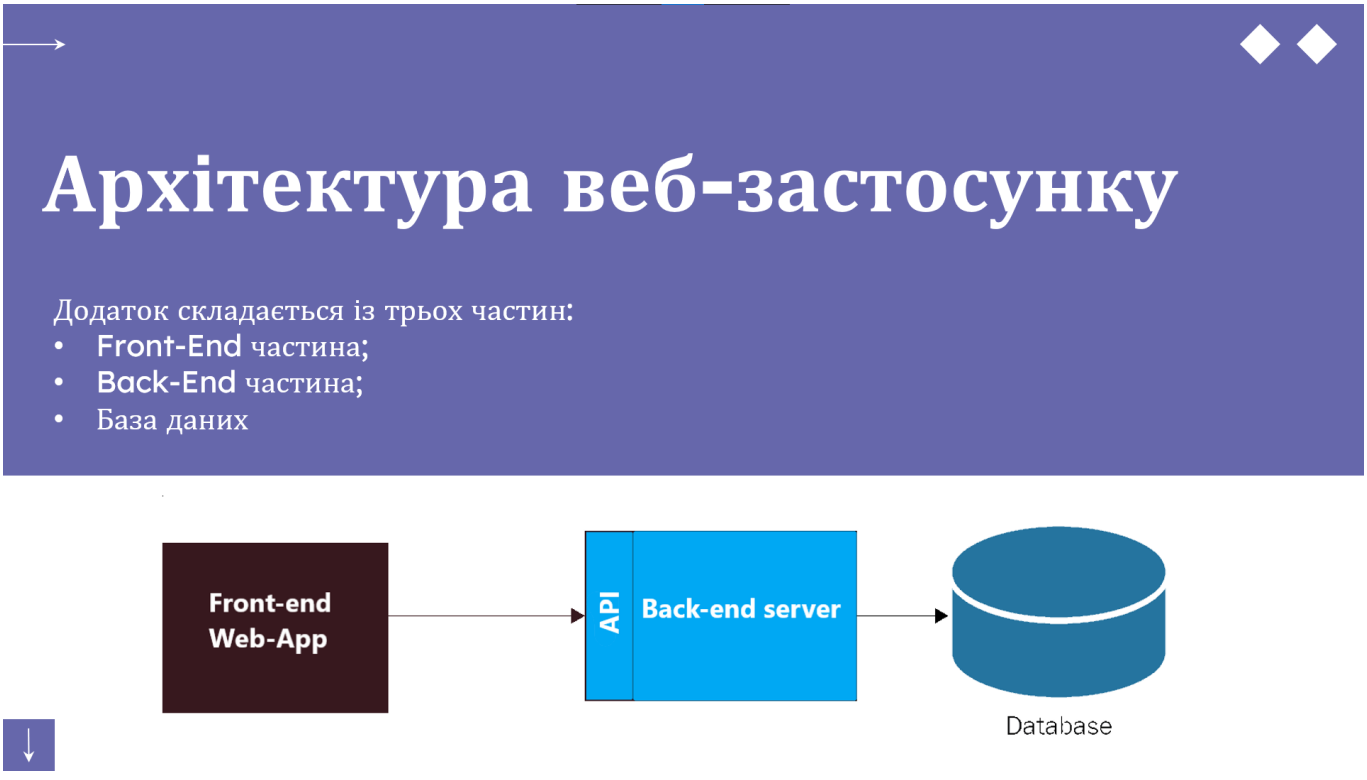



Рисунок Б.6 – Слайд №6

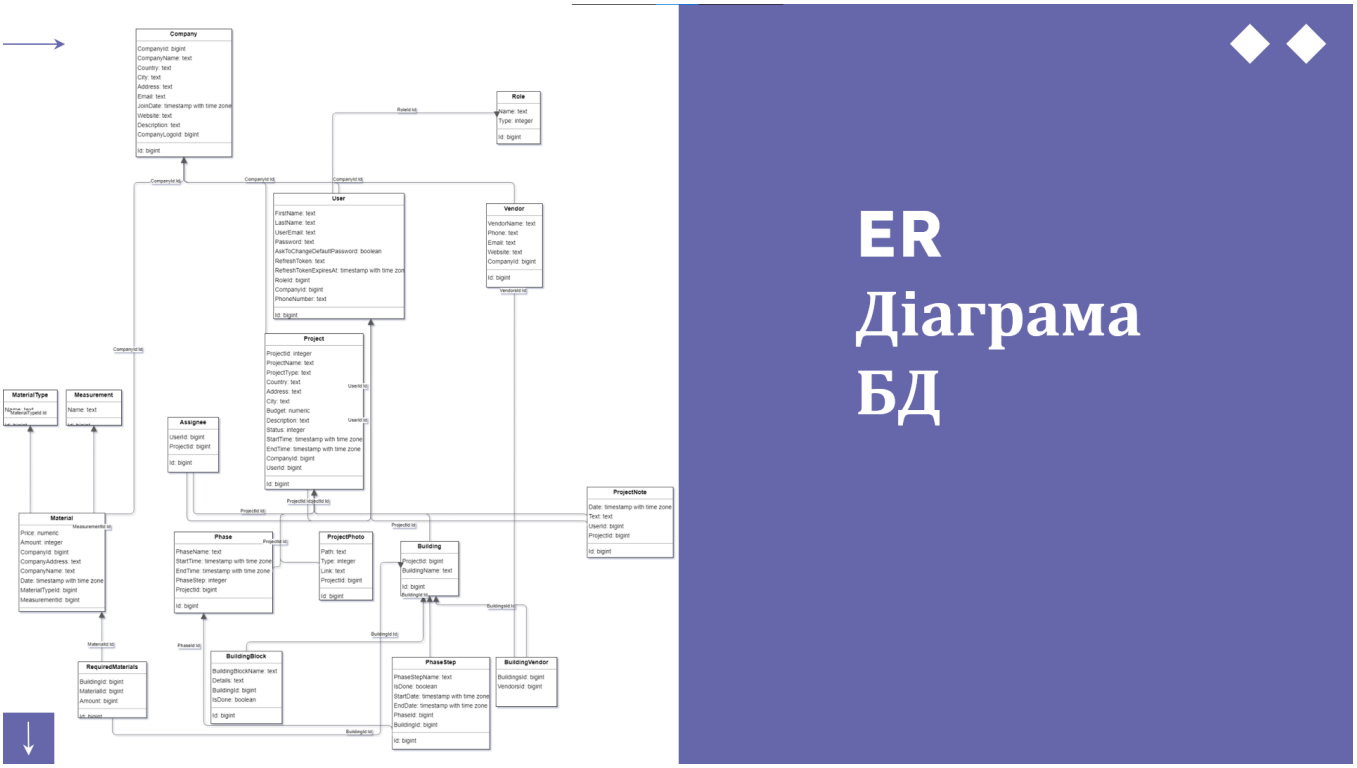



Рисунок Б.7 – Слайд №7

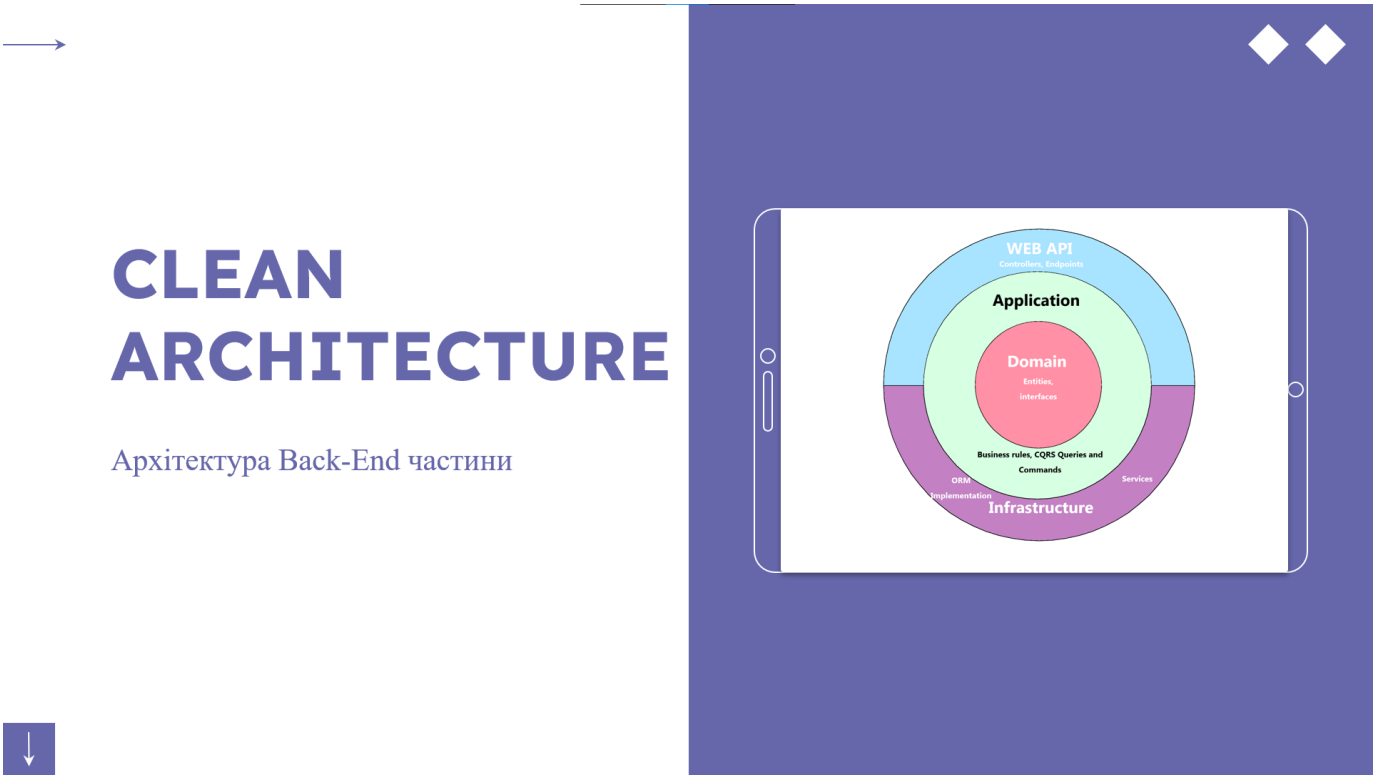



Рисунок Б.8 – Слайд №8

# REACTIVE ARCHITECTURE

Архітектура Front-End частини

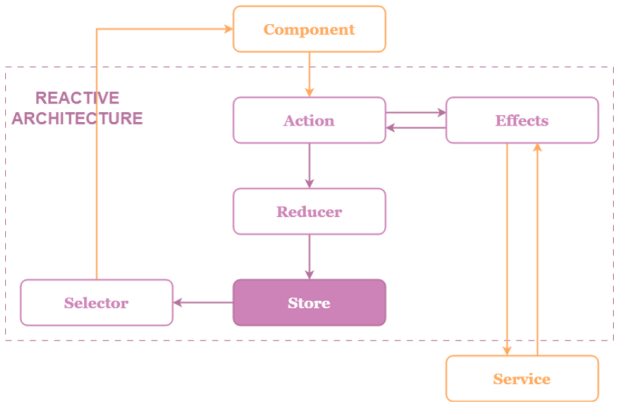



Рисунок Б.9 – Слайд №9

# 03 ТЕХНОЛОГІЇ РОЗРОБКИ

**ASP.NET Core**

Серверна (Back-End) сторона додатку

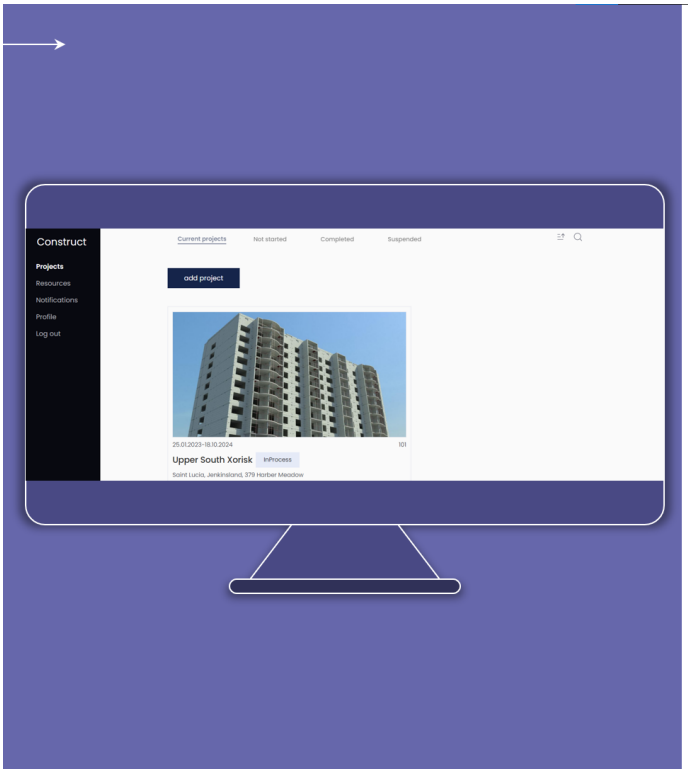
**Angular**

Інтерфейс (Front-End) сторона додатку

**Azure**

Система хмарного розгортання додатку

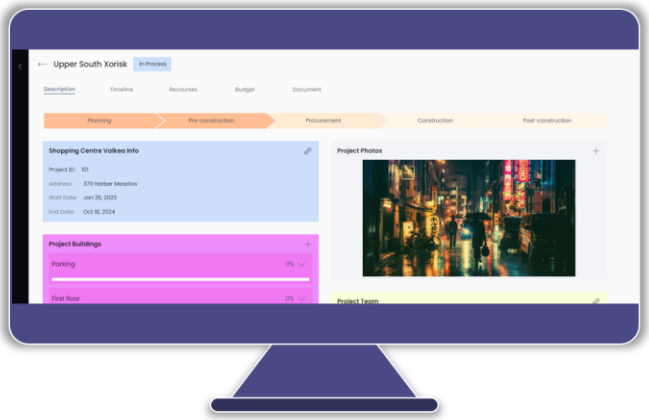
Рисунок Б.10 – Слайд №10




# ГОЛОВНЕ ВІКНО

За допомогою головного вікна додатку можна виконувати перегляд, створення та пошук по проєктам.

Рисунок Б.11 – Слайд №11

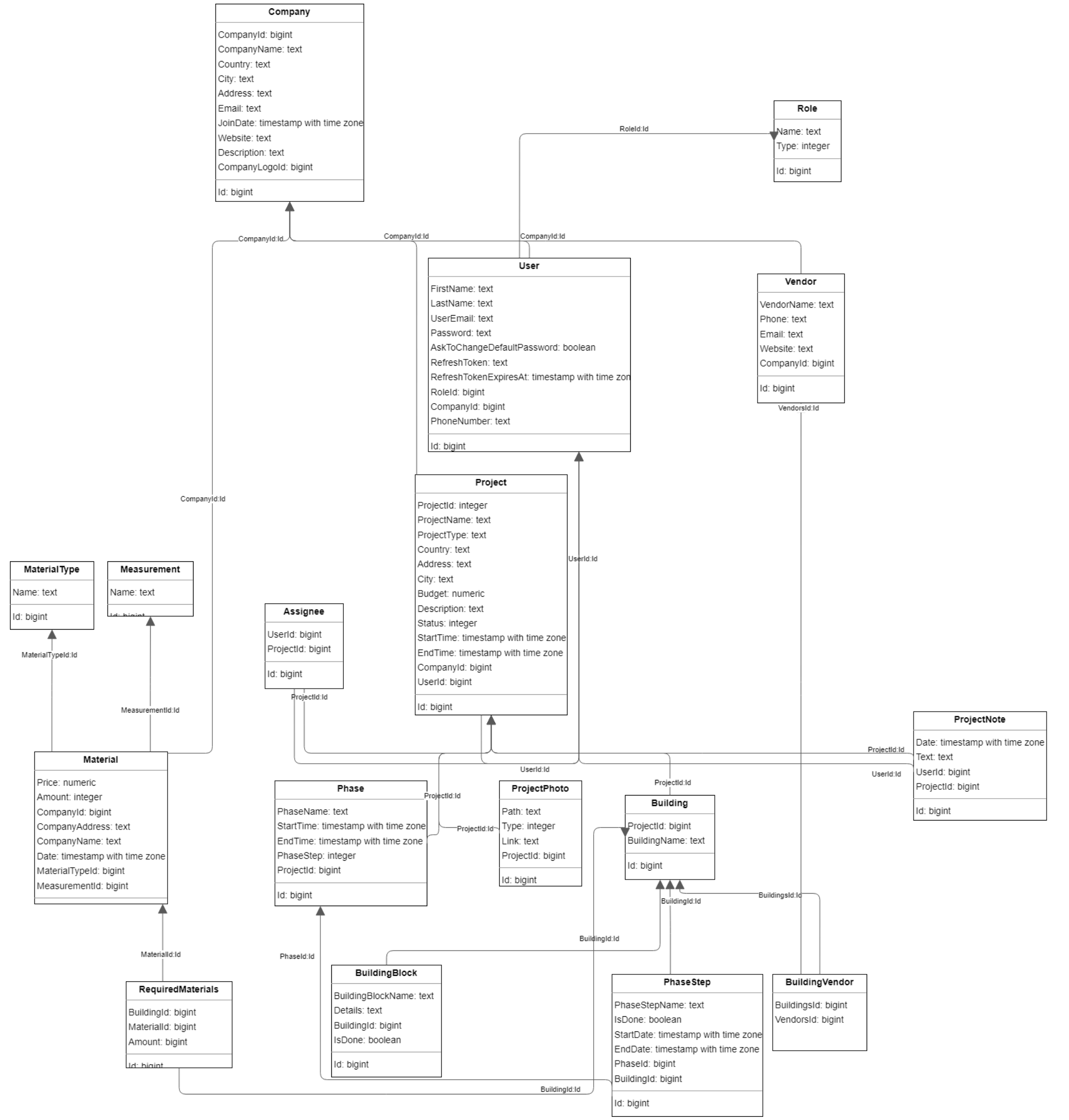



# ВІКНО ПЕРЕГЛЯДУ ПРОЄКТУ

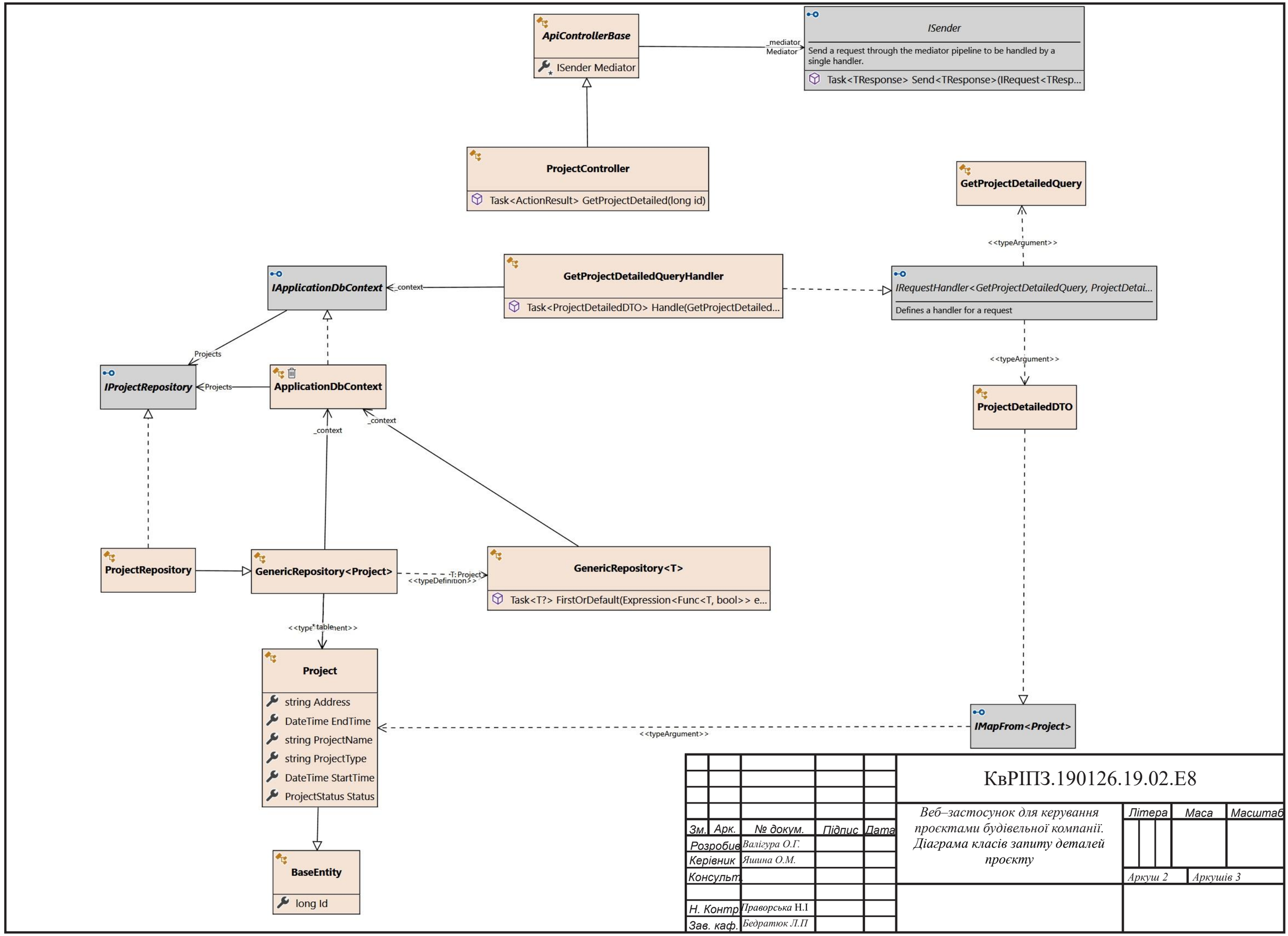
Рисунок Б.12 – Слайд №12


Рисунок Б.13 – Слайд №13

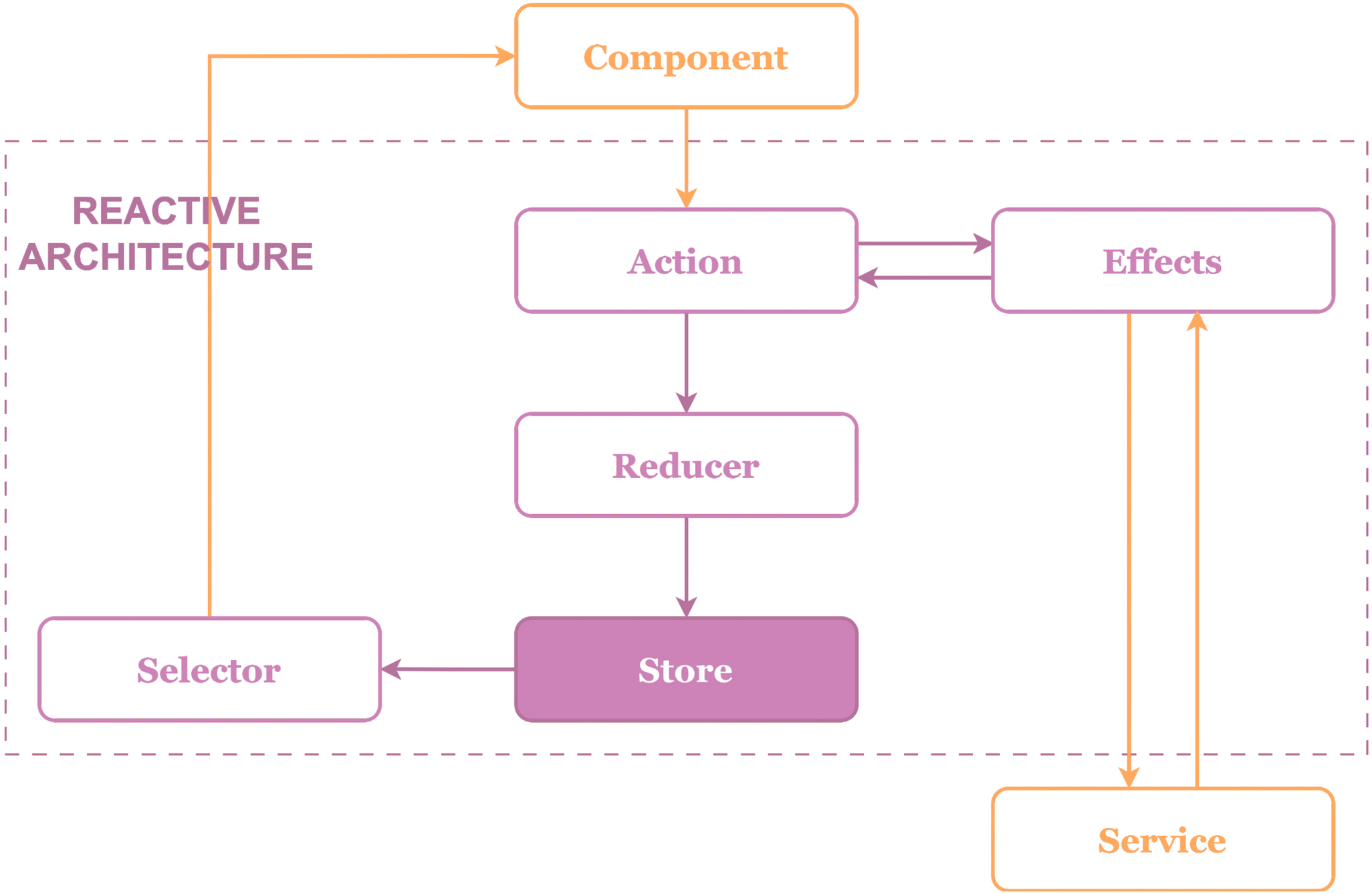
# ГРАФІЧНА ЧАСТИНА




| | | | | | КвРІПЗ.190126.19.02.Е8 | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | *Веб–застосунок для керування проєктами будівельної компанії. Діаграма відношень сутностей бази даних* | *Літера* | *Маса* | *Масштаб* |
| *Зм.* | *Арк.* | *№ докум.* | *Підпис* | *Дата* | | | | |
| *Розробив* | | *Валігура О.Г.* | | | | | | |
| *Керівник* | | *Яшина О.М.* | | | | *Аркуш* 1 | *Аркушів 3* | |
| *Консульт.* | | | | | | | | |
| *Н. Контр.* | | *Праворська* Н.І | | | | | | |
| *Зав. каф.* | | *Бедратюк Л.П.* | | | | | | |




| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | | КвРІПЗ.190126.19.02.Е8 | |
| Зм. | Арк. | № докум. | Підпис | Дата | *Веб–застосунок для керування проєктами будівельної компанії. Діаграма класів запиту деталей проєкту* | Літера / Маса / Масштаб |
| Розробив | | Валігура О.Г. | | | | |
| Керівник | | Яшина О.М. | | | | Аркуш 2 / Аркушів 3 |
| Консульт. | | | | | | |
| Н. Контр. | | Праворська Н.І | | | | |
| Зав. каф. | | Бедратюк Л.П | | | | |




| | | | | | КвРІПЗ.190126.19.02.Е8 | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | *Веб–застосунок для керування проєктами будівельної компанії. Діаграма реактивної архітектури* | Літера | Маса | Масштаб |
| Зм. | Арк. | № докум. | Підпис | Дата | | | | |
| Розробив | | *Валігура О.Г.* | | | | | | |
| Керівник | | *Яшина О.М.* | | | | *Аркуш 3* | *Аркушів 3* | |
| Консульт. | | | | | | | | |
| Н. Контр. | | *Праворська* Н.І | | | | | | |
| Зав. каф. | | *Бедратюк Л.П.* | | | | | | |

# СУПРОВІДНІ ДОКУМЕНТИ

Завідувачу кафедри інженерії програмного забезпечення проф. Бедратюку Л. П.

здобувача вищої освіти
Валігура О.Г.
Прізвище, ініціали

факультет ІТ, 4 курс, група ІПЗ-19-1

**ЗАЯВА**

З правилами чинного Положення «Про систему забезпечення академічної доброчесності в Хмельницькому національному університеті» від 01.07.2022, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів дисциплінарної та академічної відповідальності, ознайомлений. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений. та надаю свою згоду на обробку й збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та/або Anti-Plagiarism) і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

01.06.2023
дата

підпис

UNICHECK

Ім'я користувача:
**Кафедра ІПЗ**

ID перевірки:
**1015392819**

Дата перевірки:
**02.06.2023 13:25:22 EEST**

Тип перевірки:
**Doc vs Internet + Library**

Дата звіту:
**02.06.2023 13:27:14 EEST**

ID користувача:
**100005589**

---

Назва документа: **КвР_Валігура_антиплагіат**

Кількість сторінок: **67** Кількість слів: **11414** Кількість символів: **92638** Розмір файлу: **6.38 MB** ID файлу: **1015057231**

---

# 5.31%
# Схожість

Найбільша схожість: **1.52%** з джерелом з Бібліотеки (ID файлу: **1015045630**)

| | | |
|---|---|---|
| 3.54% Джерела з Інтернету | 489 | Сторінка 69 |
| 2.85% Джерела з Бібліотеки | 105 | Сторінка 71 |

# 0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

# 0%
# Вилучень

Немає вилучених джерел

# Anti-Plagiarism v-15.257

**Максимальне співпадіння з одним документом 6.0%**

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 17%**

| | Документ | | Сумарний збіг по Базі Даних | |
|---|---|---|---|---|
| ID: 114573<br>Назва: БКР Веб-застосунок для керування проєктами будівельної компанії<br>Додано в БД: 2023-06-02<br>Автора: Валігура О.Г.<br>Керівники: Яшина О.М. к.т.н. доц.<br>Консультанти:<br>Опоненти: | Символи | Лексеми | Символи | Лексеми |
| | 78426 | 707 | 6970 (9%) | 80 (11%) |

Джерело плагіату

| ID | Опис | Наявність плагіату в документі | |
|---|---|---|---|
| | | Символи | Лексеми |

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

**РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ**
освітнього ступеня «Бакалавр»

Дипломник Валігура Олексій Григорович

Тема Веб-застосунок для керування проєктами будівельної компанії

Спеціальність 121 – Інженерія програмного забезпечення

**Обсяг кваліфікаційної роботи:**

Кількість листів креслень 3; кількість сторінок записки 81

1.Короткий зміст пояснювальної записки та прийнятих рішень У кваліфікаційній роботі було вивчено та проаналізовано сферу керування проєктами будівельної компанії, визначено вимоги до веб-застосунку. Було виконано аналіз сучасних підходів до керування проєктами в будівельній сфері, а також оцінено існуюче програмне забезпечення, визначено його переваги і недоліки. Такий аналіз довів актуальність створення нового веб-застосунку Було визначено відповідні інструменти та технології для реалізації спроектованих рішень, що дозволило створити новий веб-застосунок. Процес розробки включав проєктування архітектури, написання коду, дизайн інтерфейсу, а також детальне тестування. Результати тестування підтвердили коректність роботи розробленого програмного забезпечення.

2. Висновок про відповідність роботи поставленому завданню Кваліфікаційна робота виконана відповідно до поставленого завдання та з дотриманням всіх вимог

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки та передових методів роботи У вступі кваліфікаційної роботи було доведено актуальність теми розробки веб-застосунку для керування проєктами будівельної компанії. Визначено мету та поставлено конкретні завдання проекту. У першому розділі було проведено глибокий аналіз предметної області, вивчено існуючі рішення, їх переваги та недоліки, і на основі цього визначено вимоги до розроблюваного веб-застосунку. У другому розділі було проведено детальний аналіз сучасних архітектур, їх переваг та недоліків. На основі цього аналізу, вибрана чиста архітектура серверної та реактивна архітектура клієнтської частин веб-застосунку. У третьому розділі, після підготовки всіх залежностей для написання коду, було виконано практичну розробку програмних модулів і описано їх особливості. Таким чином, було створено програмний продукт. Далі було проведено інтеграційне тестування системи відповідно до визначених раніше вимог. Результати тестування підтвердили коректну роботу розробленого веб-застосунку.

4. Позитивні сторони роботи Тематика кваліфікаційної роботи є вкрай актуальною у сучасних умовах. З огляду на постійний розвиток та динаміку будівельного сектору в Україні, потреба в автоматизації та цифровізації процесів керування проектами в цій сфері постійно зростає. На даний момент, існуючі рішення не завжди можуть задовольнити всі потреби користувачів, що підкреслює актуальність цієї роботи.

5. Негативні сторони роботи У даній кваліфікаційній роботі було приділено недостатньо уваги науковій частині предметної області, зокрема аналізу наявних наукових робіт у цій сфері.

6. Оцінка графічного оформлення та пояснювальної записки Графічне оформлення виконано відповідно до теми кваліфікаційної роботи та подано у вигляді діаграм і рисунків. Пояснювальна записка оформлена згідно вимог чинних стандартів

7. Відгук про кваліфікаційну роботу В цілому кваліфікаційна робота заслуговує високої оцінки. Матеріал пояснювальної записки ретельно структурований, послідовний, ясний та легко сприйманий, що дозволяє глибше зрозуміти тематику проектування. Графічна частина роботи надає можливість візуально ознайомитись з деталями проектованої системи. Все це свідчить про якісну підготовку автора та відповідність роботи сучасним вимогам до проектування веб-застосунків для керування проєктами будівельних компаній

8. Інші зауваження

9. Оцінка кваліфікаційної роботи Через виконання кваліфікаційної роботи у повному обсязі та відповідності поставленій темі, робота заслуговує оцінку Добре

РЕЦЕНЗЕНТ (прізвище, ім'я, по-батькові, посада, місце роботи) Лисенко Сергій Миколайович, доктор технічних наук, професор кафедри комп'ютерної інженерії та інформаційних систем (КІІС) ХНУ

" 1 " февраля 2023 р. _____
(підпис)

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ
КАФЕДРИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатами звіту/звітів подібності щодо роботи, продукованими програмно-технічним засобом (ами) перевірки текстів на плагіат:

Назва: «Веб-застосунок для керування проєктами будівельної компанії»

Автор: Валігура Олексій Григорович
Спеціальність: 121 – Інженерія програмного забезпечення
Освітня програма: Освітньо-професійна програма «Інженерія програмного забезпечення»
Науковий керівник: Яшина Оксана Миколаївна кандидат технічних наук, доцент
Після аналізу звіту подібності зроблено такий висновок:

| № | Висновок | Позначка про відповідність |
|---|---|---|
| 1 | Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту. | **відповідає** |
| 2 | Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої й електронної версії роботи | |
| 3 | Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того, як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат. | |
| 4 | Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту. | |
| 5 | Інше: | |

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) у тексті кваліфікаційної роботи системами перевірки на плагіат виявлено схожість з деякими документами в частині загальновживаних обов'язкових словосполучень у стандартних бланках (титулка, відомість документів), у структурі змісту, назвах розділів/підрозділів тощо, у назвах публікацій у переліку джерел посилання;

2) в якості запозичень системою було зафіксовано деякі послідовності вихідного коду і посиланання на бібліотеки, які є стандартними мовними конструкціями програмування та не можуть розглядатися як об'єкт авторських прав і, відповідно, їх порушення;

3) усі запозичення є фрагментарними або мають належним чином оформленні посилання;

4) виявлені модифікації тексту не впливають на відсоток схожості.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів ідентичності/ схожості, складає 5.31% і адресується до 594 джерел, що, з урахуванням наведених обґрунтувань, відповідає характеру теми і свідчить на користь кваліфікаційної роботи.

Дата 01.06.2023

Завідувач кафедри _____ Леонід БЕДРАТЮК

Гарант освітньої програми _____ Леонід БЕДРАТЮК

Керівник кваліфікаційної роботи _____ Оксана ЯШИНА