

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет кораблебудування імені адмірала Макарова
Навчально-науковий інститут комп'ютерних наук та управління проектами
Кафедра інформаційних управляючих систем та технологій

"Допущений до захисту"

Завідувач кафедри ІУСТ

к.т.н, доц. Михелєв І.Л.

" ____ " _____ 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти "магістр"

на тему:

Порівняльне дослідження ІТ-проектів на основі використання архітектурних
патернів

Виконав: студент групи 6141м

_____ Єфремов Г.І.

(підпис)

Керівник роботи:

(посада, науковий ступінь вчене звання)

_____ Казимиренко Ю.О.

(підпис)

м. Миколаїв – 2023 р

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет кораблебудування імені адмірала Макарова
Навчально-науковий інститут комп'ютерних наук та управління проектами
Кафедра інформаційних управляючих систем та технологій
Спеціальність 122 "Комп'ютерні науки"
Освітня програма "Комп'ютерні науки"

"ЗАТВЕРДЖУЮ"

Гарант освітньої
програми

к.т.н, доц. Ажищев В.Ф.

" ____ " _____ 2023

р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття ступеня вищої освіти "магістр"

Студенту Єфремову Георгію Івановичу

(прізвище, ім'я, по батькові)

1. Тема роботи: Порівняльне дослідження ІТ-проектів на основі використання архітектурних патернів

Керівник роботи д.т.н, професор Казимиренко Юлія Олексіївна

Затверджені наказом ректора № 1098-уч. від "18" вересня 2023 року.

2. Термін подання роботи: 15 грудня 2023 р.

3. Вихідні дані до проекту (роботи) ДСТУ щодо обробки інформації, літературні джерела, технічна документація на існуючі аналоги систем, матеріали наукового стажування

4. Перелік питань, що належать до розробки (найменування розділів)

Розділ 1 аналіз предметної області, пошук теоретичних відомостей

Розділ 2 огляд, аналіз та порівняння проектів

Розділ 3 огляд програмної реалізації проектів

Розділ 4 охорона праці

5. Перелік презентаційних матеріалів виконаний в програмі Power Point

6. Дата видачі завдання 05 вересня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів роботи	Примітка
1	Вивчення літературних джерел з предмету дослідження	протягом навчання на 5 курсі	виконано
2	Складання розгорнутого плану магістерської роботи та ознайомлення керівника з планом кваліфікаційної роботи	Вересень 2022	виконано
3	Написання розділу 1	Грудень 2022	виконано
4	Написання розділу 2	Грудень 2022	виконано
5	Написання розділу 3	Грудень 2022	виконано

6	Написання розділів з Охорони праці	Грудень 2022	виконано
7	Оформлення магістерської роботи	Грудень 2022	виконано
8	Передача магістерської роботи рецензенту для рецензування	Грудень 2022	
9	Передача магістерської роботи науковому керівникові для написання відгуку	Грудень 2022	
10	Захист магістерської роботи	21.12.2023	

Студент



(підпис)

Єфремов Г.І.

(ПІБ)

Керівник роботи



(підпис)

Казимиренко Ю.О.

(ПІБ)

АНОТАЦІЯ

Єфремов Г.І. "Порівняльне дослідження ІТ-проектів на основі використання архітектурних патернів". Магістерська робота за спеціальністю 122: Комп'ютерні науки, освітня програма «Комп'ютерні науки». Національний університет кораблебудування ім. адм. Макарова. Миколаїв, 2023. 92 сторінки.

У цій кваліфікаційній роботі було проведено глибокий аналіз та порівняння двох ІТ-проектів, що використовують різні архітектурні патерни. Дослідження зосереджено на вивченні та порівнянні Domain-Driven Design (DDD) та монолітних архітектур, а також на оцінці їх впливу на загальну ефективність та масштабованість проектів. Робота детально розглядає застосування CQRS та Mediator патернів у контексті DDD, оцінюючи їх роль у поліпшенні гнучкості, відмовостійкості та легкості підтримки складних систем. Надані результати підкреслюють важливість вибору адекватних архітектурних стратегій для різних типів ІТ-проектів.

Ключові слова: ІТ-проекти, архітектурні патерни, Domain-Driven Design, DDD, монолітна структура, CQRS, Mediator.

ABSTRACT

Yefremov H.I. "Comparative Study of IT Projects Based on the Use of Architectural Patterns". Master's thesis in specialty 122: Computer Science, educational program "Computer Science". Admiral Makarov National University of Shipbuilding. Mykolaiv, 2023. 92 pages

This qualification work conducted a deep analysis and comparison of two IT projects that use different architectural patterns. The study focuses on the study and comparison of Domain-Driven Design (DDD) and monolithic architectures, as well as assessing their impact on the overall efficiency and scalability of projects. The work thoroughly examines the application of CQRS and Mediator patterns in the context of DDD, assessing their role in improving flexibility, fault tolerance, and ease of maintenance of complex systems. The presented results emphasize the importance of choosing adequate architectural strategies for different types of IT projects.

Keywords: IT projects, architectural patterns, Domain-Driven Design, DDD, monolithic structure, CQRS, Mediator.

ЗМІСТ

ВТУП.....	9
РОЗДІЛ І АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ, ПОШУК ТЕОРЕТИЧНИХ ВІДОМОСТЕЙ	11
1.1 Аналіз архітектурних патернів, їх необхідність.	11
1.2 Антипатерни	13
1.3 Типи анипатернів	14
1.4 Аналіз стеку технологій, необхідних для дослідження	17
1.5 Аналіз проблеми	19
Висновки до розділу 1	20
РОЗДІЛ 2 ОГЛЯД, АНАЛІЗ ТА ПОРІВНЯННЯ ПРОЕКТІВ:	22
2.1 Опис Першого Застосунку: Телеграм-Бот "Holidays_helper"	22
2.2 Аналіз Використання Патернів у Телеграм-Боті "Holidays_helper" ...	23
2.3 Опис другого застосунку: Краудфаундингової платформи.	24
2.4 Аналіз Використання патернів у веб застосунку краудфаундингової платформи.	35
2.5 Порівняльний аналіз проектів та патернів застосованих при їх розробці.....	40
Висновки до розділу 2	41
РОЗДІЛ 3 ОГЛЯД ПРОГРАМНОЇ РЕАЛІЗАЦІЇ ПРОЕКТІВ.....	42
3.1 Програмна реалізація телеграм бота	42
3.2 Програмна реалізація Краудфаундингової платформи.	47
3.3 Порівняльний аналіз проектів на основі програмної реалізації	73
Висновки до розділу 3	75
РОЗДІЛ 4 ОХОРОНА ПРАЦІ.....	76
4.1 Нормативна база в галузі охорони праці при роботі з екранними пристроями	76
4.2. Загальні вимоги до роботодавців	76
4.3 Вимоги до приміщення з використанням екранних пристроїв	77
4.4 Вимоги до роботи з екранними пристроями	79
4.5. Вимоги щодо режиму відпочинку та праці на підприємствах з екранними пристроями	81

Висновки до розділу 4	82
РОЗДІЛ 5 ОХОРОНА НАВКОЛИШНЬОГО СЕРЕДОВИЩА	83
5.1 Вступ.....	83
5.2 Завдання охорони навколишнього середовища	83
5.3 Проблеми	84
5.4 Висновки до розділу 4	85
РОЗДІЛ 6 РОЗРАХУНОК І ОЦІНКА ЕКОНОМІЧНОГО ЕФЕКТУ ВІД РОЗРОБКИ.....	87
6.1 Розрахунок витрат на створення й експлуатацію програми	87
6.2 Розрахунок економічної ефективності	88
6.3 Розрахунок строку окупності програми	88
6.4 Висновок	88
ВИСНОВКИ	89
ВИКОРИСТАНІ ДЖЕРЕЛА ІНФОРМАЦІЇ.....	91

ВСТУП

Актуальність проблеми дослідження.

В сучасній динамічній технологічній епосі, де програмне забезпечення відіграє ключову роль у різних сферах, використання архітектурних патернів стає все більш важливим. Зростання обсягів програмного забезпечення та його складність призводять до необхідності вивчення та застосування патернів проектування для ефективної розробки та підтримки програмних продуктів.

Сьогоднішні програми стають все більшими і складнішими, і відправляють виклик розробникам у відповідності до принципів архітектурних патернів. Навіть найкращі технічні засоби можуть виявити свою неефективність без використання адекватних патернів. Професійні розробники та інженери програмного забезпечення повинні володіти вмінням використовувати патерни вірно, оскільки неправильне їх застосування може призвести до погіршення архітектури програмного продукту або утруднити його подальшу підтримку та розвиток.

Це дослідження має важливе значення для практичних застосувань в ІТ-сфері, оскільки з'ясування та порівняння архітектурних патернів може допомогти розробникам краще розуміти, які підходи найбільш відповідають вимогам конкретних проектів.

В рамках цього дослідження, важливим є огляд останніх тенденцій та публікацій в області архітектурних патернів. Актуальність цієї теми підкреслюється необхідністю розробки програмного забезпечення, яка зростає разом із швидким темпом технологічного розвитку. В умовах зростаючої конкуренції та вимог, що швидко змінюються, вміння використовувати ефективні архітектурні патерни може стати вирішальним для успіху проекту.

Метою мого дослідження є ґрунтовний аналіз, дослідження та порівняння ІТ-проектів на основі використання різних архітектурних патернів.

Предмет дослідження: механізми побудови архітектури програмних застосунків.

Об'єктом дослідження є процес розробки та підтримки ІТ-проектів.

Новизна дослідження. Це дослідження спрямоване на визначення ефективності застосування патернів проектування в реальних ІТ-проектах та виявлення найбільш оптимальних та продуктивних підходів.

Для досягнення мети дослідження було сформульовано наступні завдання:

- аналіз предметної області, пошук теоретичних відомостей;
- огляд та аналіз проектів з використанням різних архітектурних підходів, порівняння їх, характеризування впливу різних патернів на архітектуру, масштабованість та підтримку ІТ-продуктів;
- програмна реалізація застосунків та архітектурних патернів.

РОЗДІЛ І АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ, ПОШУК ТЕОРЕТИЧНИХ ВІДОМОСТЕЙ

1.1 Аналіз архітектурних патернів, їх необхідність.

Архітектурні патерни в програмному інжинірингу - це не просто шаблони проектування; вони є фундаментальними концепціями, які формують структуру та дизайн програмних систем. Ці патерни забезпечують перевірені рішення для стандартних архітектурних проблем, зустрічаються в різноманітних контекстах розробки та мають широкий спектр застосувань.

Актуальність архітектурних патернів полягає в їх здатності забезпечувати стабільність, масштабованість та високу продуктивність ІТ-проектів. Вони дозволяють розробникам уникати частих помилок, забезпечуючи тим самим більш ефективний процес розробки[1].

Архітектурний патерн - це універсальне рішення для загально поширеної проблеми в межах певного проекту. В основному, патерни діють як шаблони, які застосовуються для різних архітектурних конструкцій. Крім того, вони часто застосовуються в документації програмного забезпечення, допомагаючи зацікавленим особам спілкуватися та співпрацювати одна з одною.

Основна мета архітектурних патернів - вирішувати різні проблеми між компонентами програмного забезпечення. Також, вони забезпечують належні взаємовідносини між ними та забезпечують оптимальну продуктивність в межах одного середовища.

Сьогодні архітектори програмного забезпечення використовують кілька патернів, які значно поліпшують розробку проекту. Таким чином, архітектурні патерни можуть забезпечити переваги в життєвому циклі розробки, оптимізувати витрати проекту, покращити користувацький досвід, полегшити обслуговування програмного забезпечення і т.д.

Незважаючи на те, що архітектурний патерн - це приблизне зображення або концепція системи, він не є фактичною архітектурою. Замість цього, це

концепція, яка допомагає зрозуміти елементи архітектури програмного забезпечення. Може бути безліч архітектур, які реалізують той самий патерн. Тому патерни відомі як "строго описані та загально використовувані". Успіх системи залежить від вибору архітектури програмного забезпечення.

Відомі приклади архітектурних патернів - це *microservices*, *message bus*, *service requester/ consumer*, *MVC pattern*, *MVVM*, *microkernel*, *n-tier*, *domain-driven design components*, та *presentation-abstraction-control*.

Патерни архітектури програмного забезпечення мають велике значення, оскільки вони можуть вирішувати різні проблеми в різних областях. Наприклад, замість того, щоб покладатися на один сервер, складні запити користувачів можна легко розділити на менші частини та розподілити їх між кількома серверами. Ще один приклад – спрощення протоколів тестування за рахунок розділення різних сегментів програмного забезпечення замість тестування всього цілком в один раз.

Є декілька основних причин покликаних використовувати архітектурні патерни.

Визначення основних характеристик додатка. Знання особливостей, переваг і недоліків кожної архітектури стає важливим для вибору правильної архітектури, яка відповідає вашим бізнес-цілям. Було помічено, що патерни архітектури допомагають визначати основні характеристики та поведінку додатка. Наприклад, деякі патерни архітектури природно підходять для високомасштабованих додатків, тоді як інші можна використовувати для гнучких застосунків.

Забезпечення якості та ефективності. Існує велика ймовірність того, що будь-який створений додаток може стикнутися з проблемами якості. Згідно з властивостями якості розробки програмного забезпечення, вибір патерну архітектури може допомогти зменшити проблеми якості і забезпечити ефективність.

Забезпечення гнучкості. Для програмних застосунків є природнім пройти крізь численні зміни й ітерації під час розробки програмного забезпечення і навіть після випуску. Тому планування основної архітектури програми забезпечує гнучкість застосунку та спрощує майбутні модифікації.

Вирішення проблем. Попереднє планування та знання архітектури програмного забезпечення дають чітке уявлення про те, як функціонуватимуть додаток та його складові елементи. З наявною архітектурою команді розробників легше приймати кращі практики для вирішення складних процесів та виправлення помилок у майбутньому.

Підвищення продуктивності. Незалежно від навичок та знань, які є у кожного щодо мови програмування, фреймворку чи застосунку, мають бути певні стандартизовані принципи. З належним патерном архітектури застосунку компанія може швидко оцінити стан проекту. Крім того, рівень продуктивності підвищується, коли встановлений патерн архітектури для уточнення обсягу проекту.

В той же час, з огляду на широкий спектр доступних патернів, виникає важливе питання: як вибрати найбільш підходящий патерн для конкретного проекту? Це питання є ключовим для дослідження, оскільки неправильний вибір може призвести до зайвої складності системи, зниження продуктивності та інших проблем.

1.2 Антипатерни

Розробка програмного забезпечення - це хаотична діяльність. Отже, мудре застосування шаблонів проектування допоможе прискорити процес розробки, надаючи добре перевірені парадигми дизайну. Фактично, шаблони проектування надають повторно використовувані рішення для загальних проблем/питань. Крім того, якщо використовувати шаблони проектування у програмних рішеннях, вихідні коди стануть більш гнучкими, повторно

використовуваними та підтримуваними. Анти-патерни знаходяться на протилежному боці від шаблонів проектування та є небажаними.

Антипатерн - це літературна форма, яка описує повторюване рішення проблеми, що призводить до виражених негативних наслідків. Антипатерн може бути результатом того, що керівник або розробник не знає кращого рішення, не має достатніх знань або досвіду у вирішенні певного типу проблеми, або застосував абсолютно правильний шаблон у неправильному контексті[2].

Антипатерни надають реальний досвід у визнанні повторюваних проблем у сфері програмного забезпечення та надають детальний засіб для найбільш поширених ускладнень. Антипатерни висвітлюють найпоширеніші проблеми, з якими стикається галузь програмного забезпечення, та надають інструменти для визначення цих проблем та встановлення їх базових причин.

Крім того, антипатерни пропонують детальний план для усунення цих базових причин та впровадження продуктивних рішень. Антипатерни ефективно описують заходи, які можна прийняти на кількох рівнях для покращення розробки додатків, проектування програмних систем та ефективного управління проектами з розробки програмного забезпечення.

1.3 Типи антипатернів

Golden hammer - це антишаблон, який можна знайти не лише у розробці програмного забезпечення, а й у будь-якій сфері життя. Деякі люди схильні мати вже готове рішення, не розуміючи повністю проблему, подібно до лікаря, який призначає лікування, поки пацієнт ще розповідає про симптоми, або розробника, який намагається використовувати сучасні фреймворки користувацького інтерфейсу для створення простої форми зворотнього зв'язку із чотирма текстовими полями...

У розробці програмного забезпечення кожен шаблон проектування, кожен фреймворк або навіть кожна мова програмування підходить до типової

проблеми. Однак деякі люди захоплено використовують один і той самий інструмент для кожної проблеми, яку вони вирішують.

У результаті Золотий молот призводить до наступного:

- Низької продуктивності коду: обраний інструмент не є ефективним рішенням для проблеми.
- Надмірно складного коду: у вищезазначеному прикладі розробник замість використання простих рядків HTML-коду використав сучасний фреймворк, наприклад, Angular, що призвело до складності у підтримці і негативно впливає на продуктивність.
- Зайвого коду: обраний інструмент не лише не додає користі, але й приносить безліч непотрібних рядків коду.

Також є God Class. Типовий "Клас бога" це клас який буде контролювати багато інших класів, брати на себе багато відповідальностей, а також мати величезну кількість залежностей. Цей антипатерн прямо порушує принцип "Single responsibility".

"God" класи важко піддавати модульному тестуванню, налагодженню та документації. З часом вони будуть рости разом з додатком і можуть стати кошмаром підтримки, навіть недосяжними.

Застосунок може бути добре спроектованим спочатку, і тут взагалі не буде "God" класів. Однак вимоги та розмір розробницької команди будуть зростати з кожним днем. В результаті одні чітко визначені класи з часом перетворяться на "God" класи.

Далі наведено приклад "God" класу

```

public class GodClassStudent
{
    0 references
    public void GetStudentDetails(string studentId) { }
    0 references
    public void CreateNewStudent(StudentModel studentModel) { }
    0 references
    public void ValidateModel(StudentModel studentModel) { }
    0 references
    public void ExportStudentDetailsToCSV(StudentModel studentModel) { }
    0 references
    public void ImportStudentDetailsToDatabase(StudentModel studentModel) { }
}

4 references
public class StudentModel
{
    0 references
    public string StudentId
    {
        get;
        set;
    }
    0 references
    public string StudentName
    {
        get;
        set;
    }
    0 references
    public string StudentClass
    {
        get;
        set;
    }
    0 references
    public double StudentAddress
    {
        get;
        set;
    }
}

```

Рис.1.1 God class

Big ball of mud - дуже поширений антишаблон, який виникає, коли рішення/застосунок не має сприйнятливої, гнучкої, відповідної архітектури. Якщо ваш застосунок має такі симптоми, його можна вважати "великою купою бруду":

- Безладна та неакуратна структура без коментарів у коді.
- Містить багато "God" класів з більше ніж 6000 рядків коду.
- Статичні змінні/функції всюди.
- Перевантаження методів кілька разів.
- Відсутність стандартів оформлення коду.

- Дублювання коду всюди.

Поглинання помилок та ігнорування винятків

Ефективне управління помилками та винятками у кожному застосунку має велике значення. Це допомагає забезпечити приємний досвід для користувача, а також дозволяє розробникам легко відлагодити проблему та виправити її, коли виникають непередбачені збої.

У комп'ютерному програмуванні поглинання помилок (або ігнорування помилок) - це практика перехоплення помилки або винятка та подальше продовження роботи без реєстрації, обробки або повідомлення про помилку іншим частинам програмного забезпечення. Обробка помилок таким способом вважається поганою практикою та антишаблоном у комп'ютерному програмуванні. У мовах програмування з підтримкою обробки винятків, така практика називається поглинанням винятків.

Коли помилки поглинаються, розробник не матиме жодної інформації про кореневу причину проблеми. В результаті важко зрозуміти, що пішло не так або як це виправити.

1.4 Аналіз стеку технологій, необхідних для дослідження

У сучасному світі ІТ-розробки, вибір правильних технологій та інструментів є критично важливим для успішного виконання проектів. У рамках цього дослідження, було використано цілий набір технологій та бібліотек, кожна з яких була обрана з метою досягнення найвищої продуктивності, гнучкості та ефективності у розробці. Ось короткий огляд використаних інструментів та їхнього внеску в проект.

Технології:

.NET: Це відомий і широко використовуваний фреймворк від Microsoft, що дозволяє розробляти застосунки різної складності. Його вибір забезпечує потужні можливості для побудови надійних та високопродуктивних застосунків[4].

C#: Об'єктно-орієнтована мова програмування, що відзначається своєю безпекою типів, ефективністю та гнучкістю. Вона ідеально підходить для розробки сучасних застосунків на .NET[4].

Angular: Популярний фреймворк для розробки веб-додатків. Angular використовується через свою модульність, ефективність та підтримку реактивного програмування, забезпечуючи високий рівень інтерактивності користувацького інтерфейсу[5].

TypeScript: Це надбудова для JavaScript, що додає строгу типізацію та об'єктно-орієнтовані особливості. Вона використовується для забезпечення більшої стабільності та читабельності коду, особливо в великих проектах[6].

PostgreSQL: Надійна та потужна система управління базами даних, що підтримує як SQL, так і NoSQL дані. Вона вибрана за свою продуктивність, гнучкість та сильні можливості щодо масштабування[8].

EF Core (Entity Framework Core): Це сучасна бібліотека для роботи з базами даних у .NET, що дозволяє розробникам здійснювати запити на мові LINQ, абстрагуючись від конкретного SQL-діалекту бази даних[7].

ASP.NET Core: Цей фреймворк використовується для побудови високопродуктивних веб-додатків та сервісів. Він надає широкі можливості для масштабування та високої продуктивності.

Docker: Платформа для контейнеризації, що дозволяє пакувати додатки та їх залежності у контейнери. Це забезпечує легкість розгортання, консистенцію середовищ та спрощує процеси CI/CD.

DBeaver: Універсальний інструмент для роботи з базами даних, що підтримує різноманітні СУБД, включаючи PostgreSQL. Його використання сприяє зручності управління даними та візуалізації схем баз даних.

Бібліотеки:

AutoMapper: Бібліотека для автоматичного відображення одного об'єкта в інший, зменшуючи кількість коду, потрібного для трансформації даних.

FluentValidation: Інструмент для легкого та інтуїтивно зрозумілого створення правил валідації в .NET.

ReinforcedTypings: Бібліотека для генерації TypeScript визначень з C# коду, що полегшує інтеграцію між бекендом та фронтендом.

MediatR: Бібліотека, яка дозволяє реалізувати шаблон "Mediator", сприяючи слабкому зв'язку між компонентами та спрощуючи масштабування програм.

Quartz: Розширений планувальник завдань для .NET, що дозволяє гнучко налаштовувати виконання фонових задач.

Microsoft Identity: Бібліотека для реалізації аутентифікації та авторизації, важлива для забезпечення безпеки додатків.

Telegram Bot API: Інтерфейс для створення ботів у месенджері Telegram, що дозволяє розширювати функціонал застосунків зовнішніми сервісами.

OpenAI API: Інтерфейс для інтеграції з можливостями штучного інтелекту, наданими OpenAI, що дозволяє розширити функціональність додатків за допомогою передових AI технологій.

Використання цих інструментів розробки у нашому проекті не тільки сприяло підвищенню ефективності та продуктивності роботи, але й дозволило глибше зануритися в суть архітектурних патернів та їх практичне застосування. Кожна з цих технологій та бібліотек зіграла свою унікальну роль у розробці, від інтеграції штучного інтелекту до управління базами даних, створюючи міцний фундамент для реалізації наших дослідницьких цілей. Таким чином, вони виявились не лише інструментами розробки, але й ключовими компонентами для розкриття теми архітектурних патернів у IT-проектах.

1.5 Аналіз проблеми

У сфері програмного інжинірингу існує чимало складних та невирішених проблем, однією з яких є оптимальне використання архітектурних патернів у різних контекстах проектів. Цей виклик полягає у виборі та адаптації патернів

таким чином, щоб максимально задовольнити специфічні вимоги проекту без надмірної складності або втрати гнучкості.

Сучасні ІТ-проекти часто вимагають високого рівня масштабованості, безпеки та продуктивності, що ставить перед розробниками завдання знайти ідеальне співвідношення між використанням стандартних архітектурних патернів та їх адаптацією під специфіку проекту. Надмірне або недостатнє використання патернів може привести до зайвої складності, зниження продуктивності або вразливостей у системі.

Ця проблема набуває особливої актуальності в контексті швидкого розвитку технологій та зростання складності програмних продуктів. Вона ставить питання про ефективність існуючих патернів та потребу в нових, більш гнучких та адаптивних підходах до архітектури.

Розв'язання цієї проблеми може полягати у розробці нових методів аналізу та оцінки впливу патернів на конкретний проект. Це включає дослідження їх впливу на масштабованість, продуктивність та гнучкість системи. Крім того, важливим аспектом є розвиток інструментів, які допоможуть розробникам ефективніше вибирати та адаптувати архітектурні патерни під конкретні потреби проекту.

Аналіз невирішеної проблеми використання архітектурних патернів має ключове значення для подальшого розвитку сфери програмного інжинірингу. Вирішення цього завдання відкриє нові можливості для розробки більш ефективних, гнучких та масштабованих ІТ-проектів, сприяючи технологічному прогресу у цій області.

Висновки до розділу 1

Актуальність архітектурних патернів у сучасних ІТ-проектах не може бути недооцінена. Вони є критично важливими для створення ефективних, масштабованих та гнучких програмних рішень. Однак, з підвищенням складності та розмаїттям вимог до програмного забезпечення, стає все більш

важливим розуміти, як правильно використовувати ці патерни, що є центральною темою даного дослідження.

РОЗДІЛ 2 ОГЛЯД, АНАЛІЗ ТА ПОРІВНЯННЯ ПРОЕКТІВ:

2.1 Опис Першого Застосунку: Телеграм-Бот "Holidays_helper"

Телеграм-бот "Holidays_helper" є унікальним інструментом, розробленим для спрощення вибору подарунків. Його основна функція полягає у наданні персоналізованих рекомендацій щодо подарунків на основі інформації про особу, для якої вони призначені. Застосунок взаємодіє з користувачем через простий і зрозумілий інтерфейс, що дозволяє зібрати необхідні дані: ім'я, вік, стать, захоплення та інші персональні інтереси.

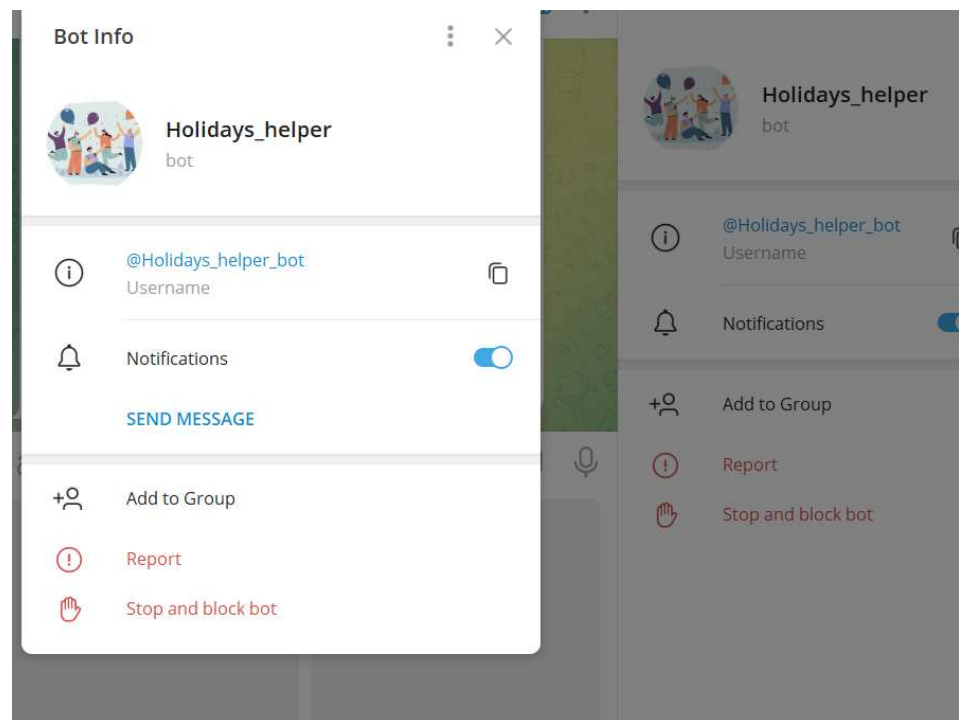


Рис 2.1 Телеграм-Бот "Holidays_helper"

Користувачеві необхідно відповісти на серію простих запитань, які бот використовує для збору даних. Після того, як інформація зібрана, використовуючи можливості ChatGPT, бот генерує список ідей для подарунків, кожна з яких ретельно підібрана, щоб відповідати інтересам та перевагам зазначеної особи.

Як видно з наданих скріншотів(рис 2.2-2.3), при взаємодії з користувачем, який шукає подарунок для свого друга-художника, що захоплюється автомобілями, бот ефективно зібрав відповідну інформацію. На основі цього,

бот запропонував ряд творчих і вдумливих подарунків, від набору олійних фарб високої якості до абонементу на автомобільний журнал, який підживлює його пристрасть до класичних автомобілів.

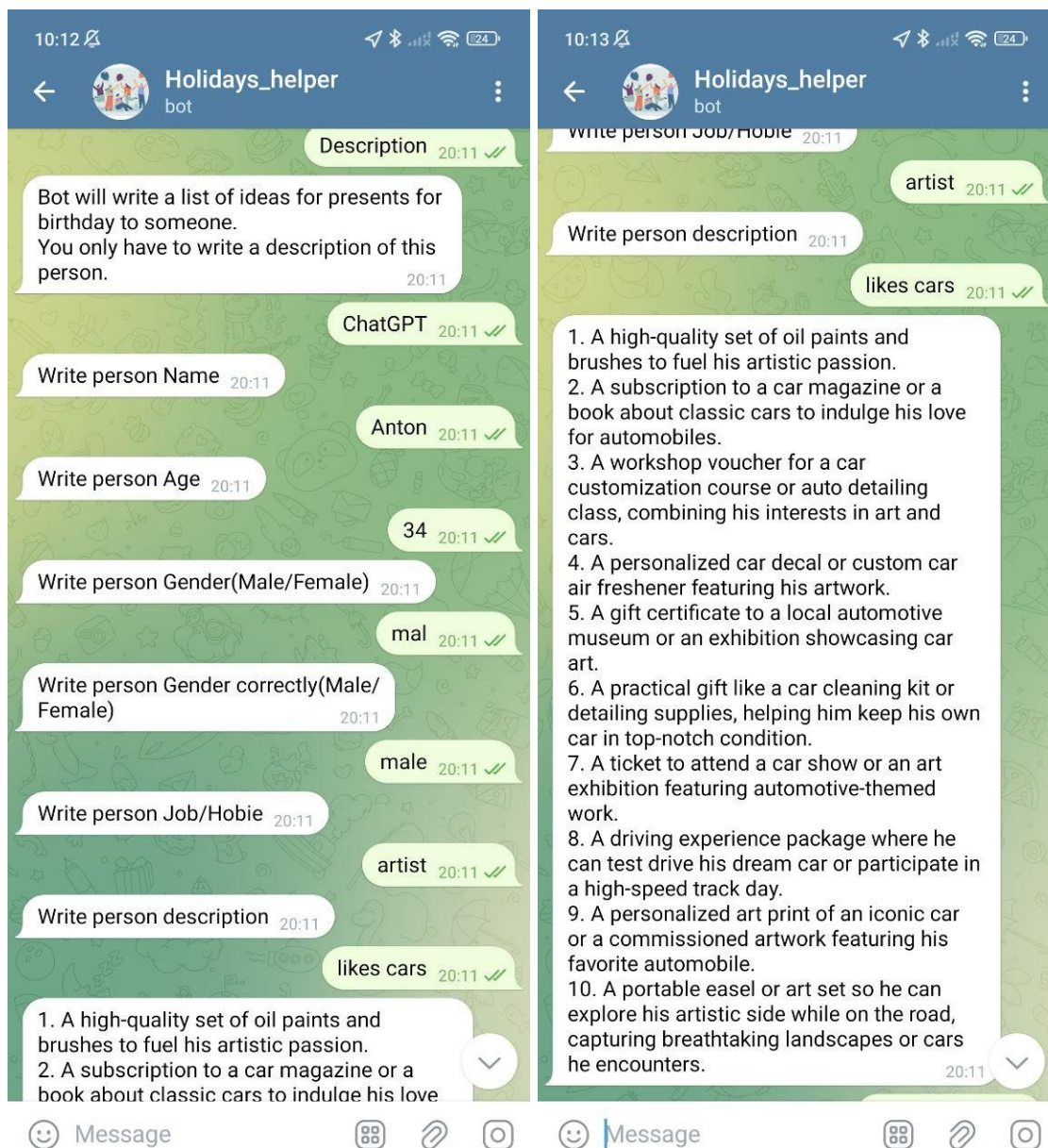


рис 2.2-2.3 Приклад Використання

2.2 Аналіз Використання Патернів у Телеграм-Боті "Holidays_helper"

Телеграм-бот "Holidays_helper" був розроблений як простий інструмент, орієнтований на виконання однієї конкретної функції — генерації ідей для подарунків. У таких проектах, де обсяг задач обмежений і вимоги до масштабованості відсутні, використання великої кількості архітектурних патернів може бути невиправданим. Інколи, додавання складних патернів може

навіть зменшити ефективність розробки та підтримки проекту, оскільки воно вносить зайву складність та може уповільнити роботу застосунку.

Незважаючи на відсутність складних архітектурних патернів, бот "Holidays_helper" все ж використовує патерн "Inverse of Control" (IoC) через механізм Dependency Injection (DI). DI дозволяє залежностям компонентів застосунку бути впровадженими ззовні, що забезпечує більшу гнучкість та спрощує тестування та розширення програми.

В контексті телеграм-бота, IoC може бути реалізований для впровадження сервісів, таких як модулі для спілкування з ChatGPT або інші сервіси обробки даних. Це важливо навіть для невеликих проектів, оскільки воно підвищує якість коду та готовність до можливих майбутніх змін.

У випадку "Holidays_helper", використання DI може сприяти легкому внесенню змін у логіку генерації подарунків, якщо, наприклад, виникне потреба інтегрувати нові джерела даних або змінити алгоритми обробки вхідної інформації. Такий підхід робить бота більш адаптивним і легко модифікованим, не вдаючись до радикального переписування існуючого коду.

Хоча телеграм-бот "Holidays_helper" і не використовує складні архітектурні патерни, він все ж втілює принципи IoC через DI, що свідчить про продуманий підхід до розробки навіть у невеликих проектах. Використання DI надає проекту гнучкості та спрощує можливість його розвитку в майбутньому.

2.3 Опис другого застосунку: Краудфаундингової платформи.

Другий проект представляє собою краудфандингову платформу, що є цифровим простором для збору коштів на реалізацію різноманітних ініціатив та ідей. В основі концепції платформи лежить залучення громадськості до підтримки творчих, інноваційних, соціальних або будь-яких інших проектів, які потребують фінансування. Платформа відкрита для широкого спектру проектів, від мистецтва та культури до технологій та розвитку громад. Ця система є втіленням сучасного підходу до фінансування проектів, де технологічні засоби

використовуються для з'єднання творчих особистостей та інноваторів з потенційними інвесторами та спонсорами. Платформа надає користувачам інтуїтивно зрозумілий інтерфейс, який дозволяє легко навігувати, взаємодіяти та вкладати кошти в обрані проекти.

Цей застосунок є демонстрацією розуміння того, як сучасні веб-технології можуть бути використані для сприяння соціальній взаємодії та підтримки. Він розроблений таким чином, що може безперервно адаптуватися та масштабуватися, реагуючи на змінні потреби його користувачів і динаміку ринку.

Архітектура платформи базується на сучасних принципах розробки, таких як модульність та розширюваність, дозволяючи легко вводити нововведення та функціональні оновлення. Це створює надійне та гнучке середовище, в якому можуть розвиватися та процвітати нові ідеї, стартапи та культурні проекти.

Краудфандингова платформа, яку ми розглядаємо як другий проект, є багатошаровим веб-застосунком, що включає множину сторінок, кожна з яких виконує унікальні функції для забезпечення повного циклу краудфандингу. Від користувацького інтерфейсу, який дозволяє відвідувачам оглядати опубліковані проекти, до деталізованих сторінок для створення та управління власними ініціативами — кожен аспект платформи спроектований для надання максимальної зручності та функціональності.

Система авторизації та аутентифікації є ключовим елементом платформи, забезпечуючи безпечний доступ до особистих облікових записів. Вона інтегрована з різними рівнями користувацьких привілеїв, що дозволяє налаштовувати дії, доступні для користувачів, відповідно до їх ролі на платформі — будь то автор проекту, спонсор, адміністратор чи простий відвідувач.

Система реєстрації користувачів використовує стандартні процедури верифікації, включаючи валідацію електронної пошти, що є важливою для

запобігання шахрайства та забезпечення довіри між користувачами. Для зберігання даних профілів використовується захищена база даних, що гарантує конфіденційність особистої інформації.

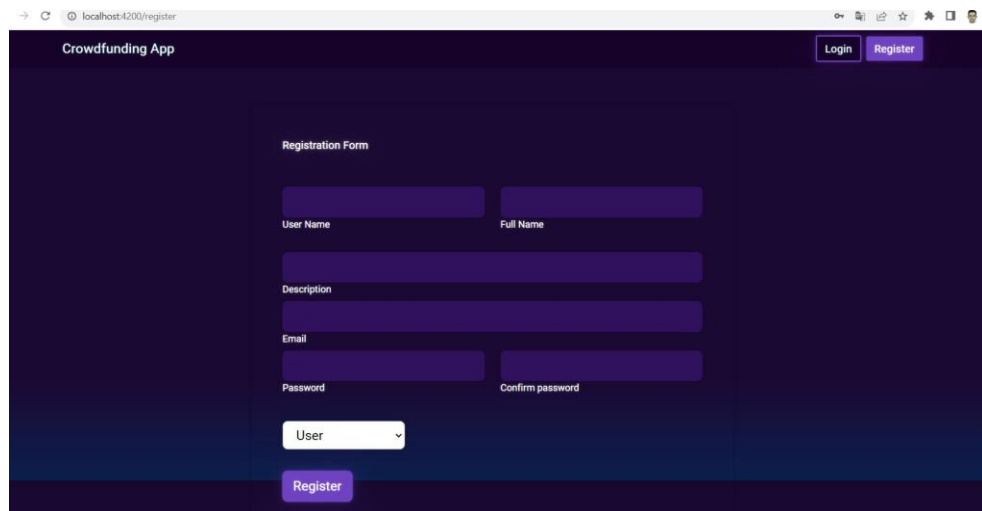


рис 2.4 сторінка реєстрації

Коли користувачі відвідують сторінку авторизації, вони зустрічаються з чітко структурованою формою(рис 2.5), яка вимагає введення наступних даних:

- Ім'я користувача: Унікальне ідентифікаторне ім'я, яке користувач обрав під час реєстрації.
- Пароль: Секретний пароль, створений користувачем, що є ключем до його облікового запису.

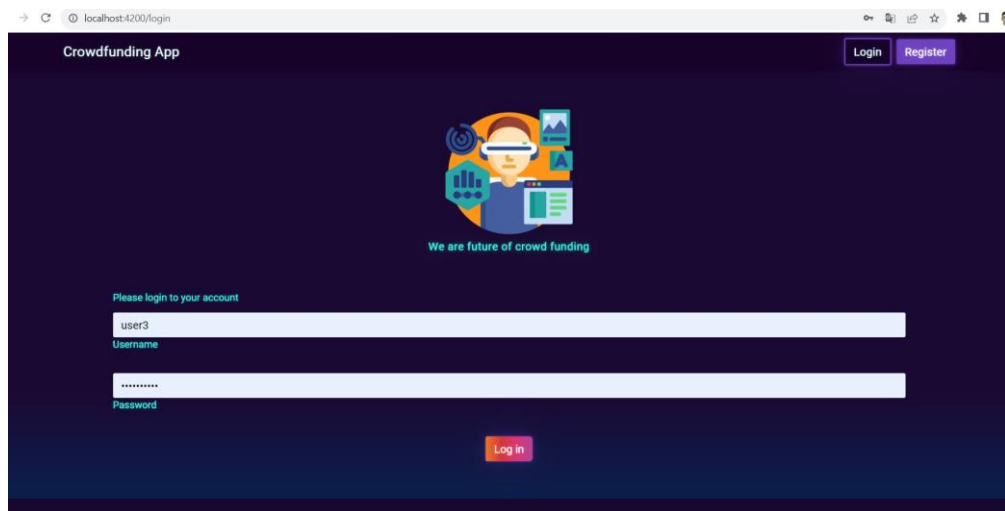


рис 2.5 сторінка авторизації

Сторінка авторизації реалізована з урахуванням сучасних стандартів безпеки, включаючи шифрування даних форми і захист від несанкціонованого доступу. Сервер обробляє дані входу, використовуючи безпечні методи аутентифікації, щоб переконатися в тому, що доступ надається лише легітимним користувачам.

Після успішного входу користувачі отримують доступ до свого персоналізованого дашборду, де вони можуть переглядати свої проекти, внески, а також управляти налаштуваннями облікового запису. Сторінка авторизації є воротами до взаємодії з платформою, де користувачі можуть реалізувати свої творчі та інвестиційні ініціативи.

Головна сторінка краудфандингової платформи відіграє роль вітрини для представлення проектів, які шукають фінансування. Вона демонструє відвідувачам зразки проектів із залученими зображеннями та короткими описами, створюючи візуально привабливий і інформативний досвід.

В залежності від статусу користувача, головна сторінка адаптується, пропонуючи різні рівні інтерактивності:

Для гостей (неавторизованих користувачів): Сторінка відображає проекти в загальнодоступному режимі, дозволяючи переглядати доступні ініціативи та ознайомлюватися з ними. Опції «Login» та «Register» чітко виділені, надаючи можливість зареєструватися або увійти в систему для отримання додаткових можливостей(рис 2.6).

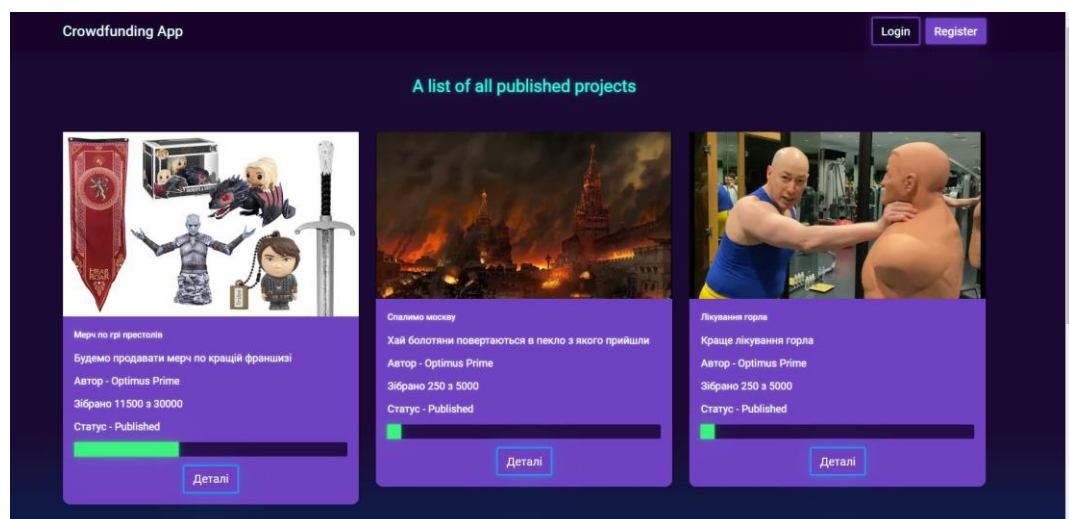


рис 2.6 Головна сторінка для неавторизованих користувачів.

Для авторизованих користувачів: Коли користувач входить в систему, головна сторінка збагачується додатковими функціями, такими як створення та управління власними проектами, перегляд статусів і прогресу вже підтриманих ініціатив, та швидкий доступ до особистого профілю(рис 2.7).

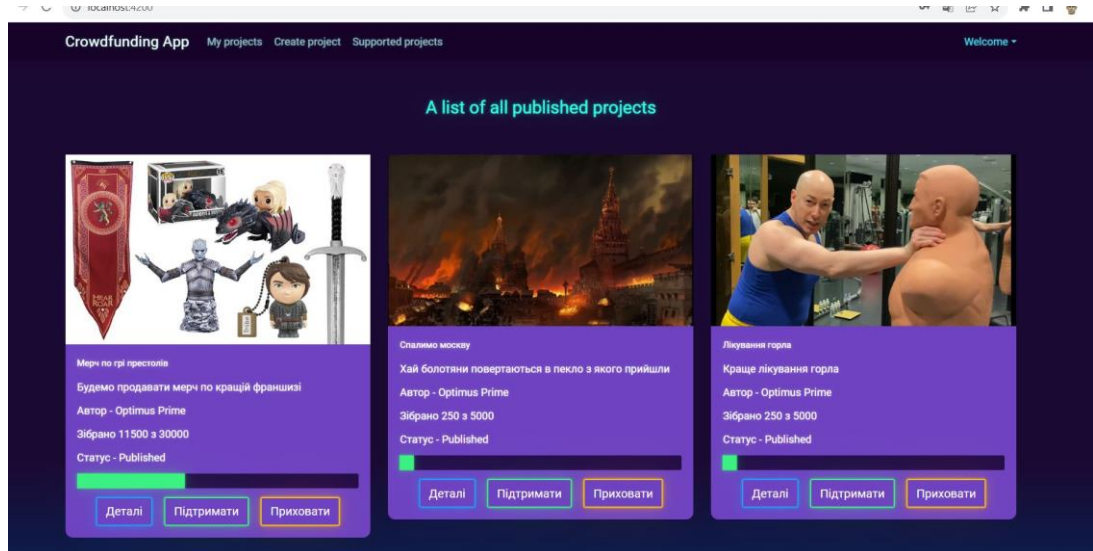


рис 2.6 Головна сторінка для авторизованих користувачів.

На головній сторінці відображаються:

- Список проектів: Проекти розміщені у формі тайлів або карточок, що надають інформацію про кожен проект, включаючи назву, короткий опис, автора проекту та суму, яка вже зібрана.
- Візуальні елементи: Кожен проект супроводжується зображенням, яке ілюструє або символізує проект, привертаючи увагу та сприяючи емоційному зв'язку з потенційними спонсорами.
- Динамічні кнопки: Кнопки «Деталі», «Підтримати» та інші, що змінюються в залежності від контексту, забезпечують легке навігування та взаємодію з проектами.

Сторінка з деталями проекту(рис 2.7) на краудфандинговій платформі відіграє критичну роль у представленні інформації, яка допомагає потенційним

спонсорам зрозуміти суть проекту та його потреби. Це місце, де автор проекту може детально розповісти про свою ідею, включаючи мотивацію, цілі, план виконання, а також специфіку використання зібраних коштів.

Основними компонентами на цій сторінці є:

- Заголовок проекту: Виділений та легко запам'ятовуваний назва проекту.
- Медіа-матеріали: Зображення, відео та інші медіа, які візуально демонструють проект або його елементи.
- Повний опис проекту: Докладна інформація про проект, його мету, важливість та те, як він буде реалізований.
- Фінансові цілі: Сума коштів, яку потрібно зібрати, та що буде зроблено на кожному етапі фінансування.
- Прогрес-бар збору коштів: Візуальне відображення прогресу збору коштів, що показує, скільки вже зібрано та скільки ще потрібно.
- Рівні спонсорства: Різні опції для внесків, кожна з яких може включати певні вигоди або відзнаки для спонсорів.
- Кнопка підтримки: Дозволяє користувачам швидко зробити внесок у проект.

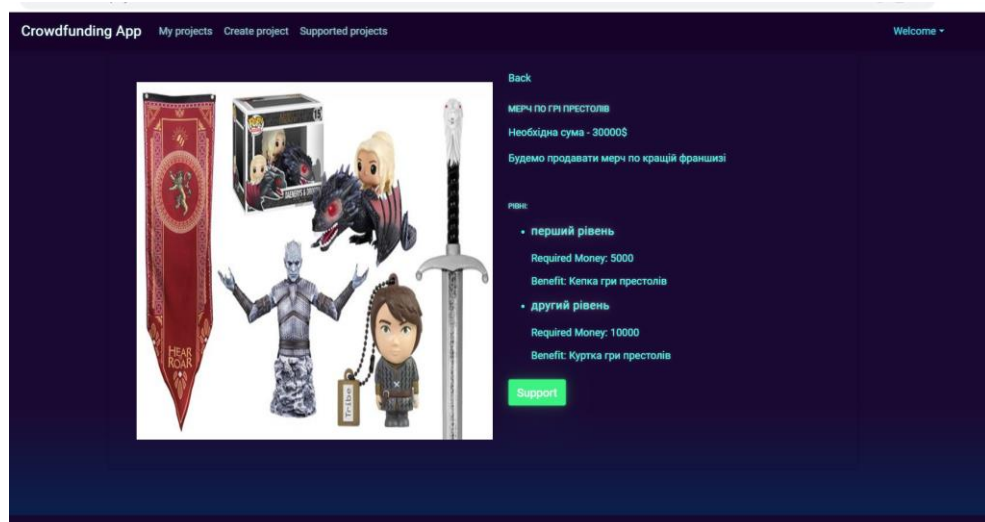


рис 2.6 Сторінка детальної інформації проекту.

Далі розглянемо сторінку підтримки проекту(рис 2.7) на Краудфандинговій Платформі. Сторінка підтримки проекту є ключовим інструментом для мобілізації ресурсів та залучення фінансових внесків від спільноти. Це місце, де інтерес користувачів перетворюється на конкретну дію — фінансову підтримку, яка є життєво необхідною для реалізації проекту.

- Інструкції для підтримки: Короткий опис процесу внесення коштів, який допомагає користувачам зрозуміти, як здійснити пожертву.
- Вибір суми внеску: Шкала або введення вільної суми для визначення розміру внеску.
- Динамічна інформація про збір коштів: Поточний статус зібраних коштів та як вони співвідносяться з загальною ціллю проекту.
- Кнопка підтримки: Центральний елемент, який ініціює процес пожертвування.

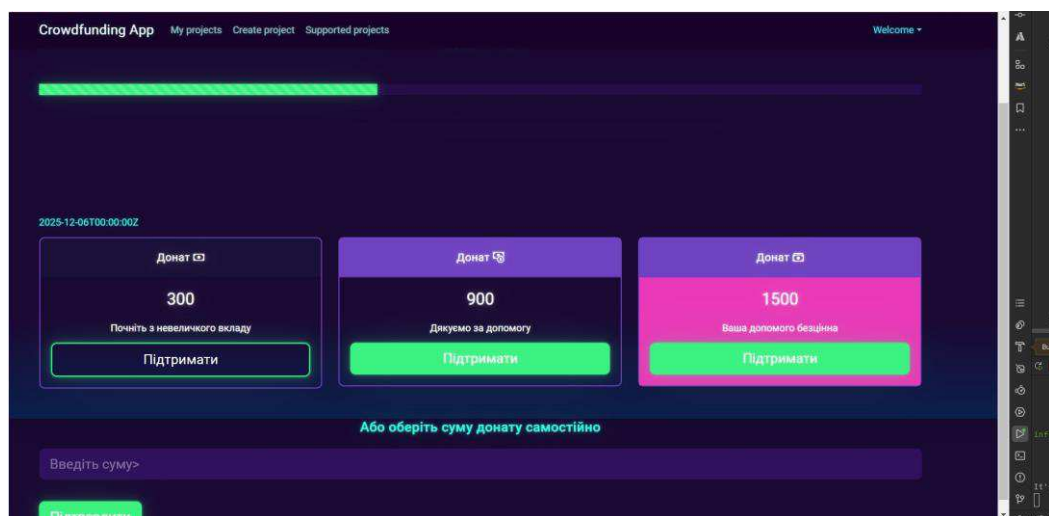


рис 2.7 Сторінка підтримки проекту.

Сторінка "Мої Проекти" на краудфандинговій платформі служить особистим кабінетом для авторів проектів, де вони можуть керувати своїми ініціативами. Ця сторінка дозволяє авторам переглядати різні стадії своїх проектів, від ідеї до реалізації, та відслідковувати їх статус.

На сторінці "Мої Проекти"(рис 2.8-2.9) проекти класифікуються за кількома типами для зручності керування та організації:

- Створені: Проекти, які знаходяться в процесі підготовки і ще не були опубліковані.
- Опубліковані: Проекти, які були відправлені на платформу і доступні для перегляду та фінансування громадськістю.
- Закінчення Терміну: Проекти, для яких вийшов крайній термін збору коштів, але не було досягнуто фінансової мети.
- Завершені: Проекти, які успішно завершили збір коштів та перебувають у стадії виконання або вже були реалізовані.

Функціонал Сторінки:

- Фільтрація: Автори можуть фільтрувати свої проекти за статусом, що спрощує управління та доступ до необхідної інформації.
- Додавання Рівнів: Автори мають можливість додавати рівні фінансування з конкретними вигодами для спонсорів, що стимулює внески.
- Редагування Проектів: Для проектів, що ще не опубліковані, автори можуть редагувати інформацію, вносячи зміни до опису, мети та іншого контенту.
- Управління Статусом: Кнопки, такі як "Деталі", "Опублікувати" та іконка смітника для видалення, дозволяють легко управляти кожним проектом.

Сторінка розроблена з урахуванням зручності користувачів, забезпечуючи легку навігацію та інтуїтивно зрозумілі елементи управління. Використання кольорових кодів та прогрес-барів сприяє візуальній привабливості та організації інформації

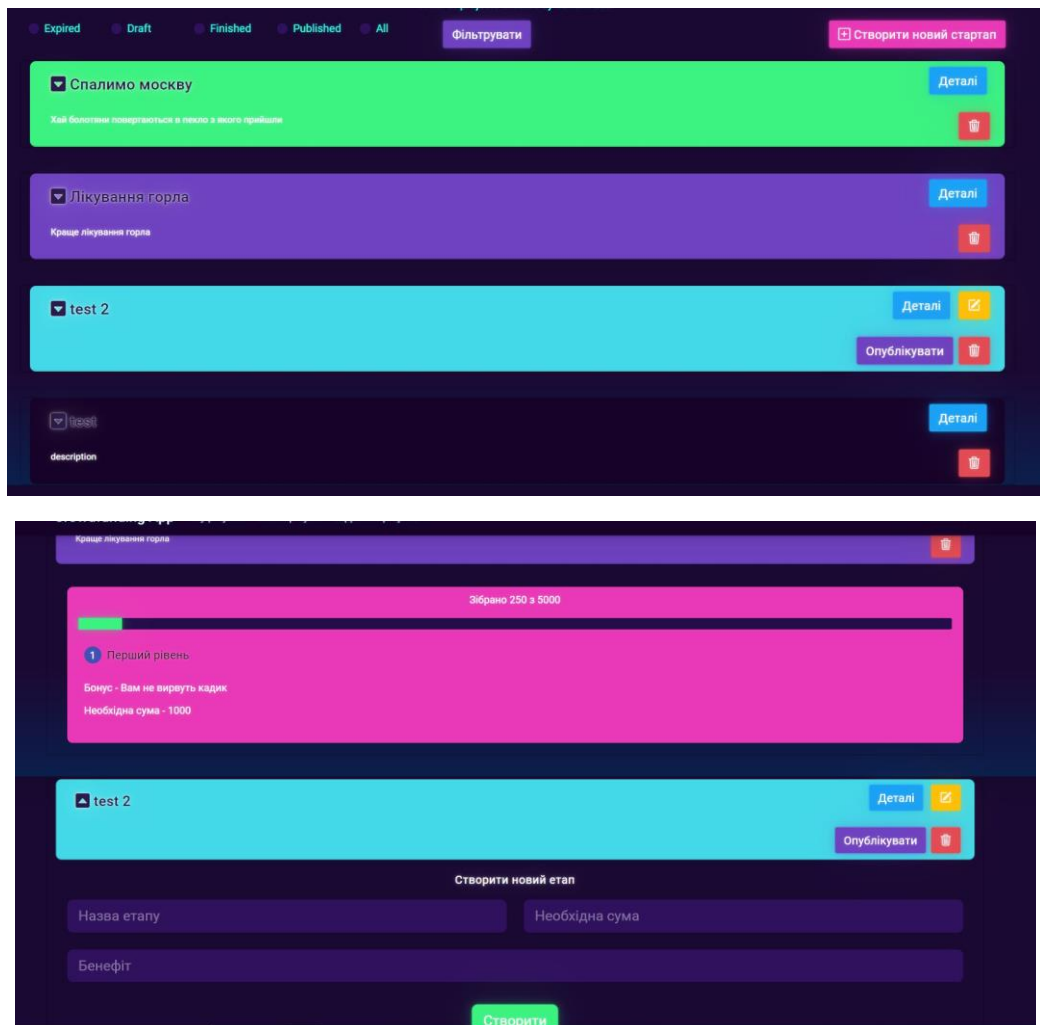


Рис 2.8-2.9 Сторінка Мої Проекти

Сторінка редагування проекту(рис 2.10) — це інструмент, який надає авторам можливість оновити або змінити інформацію про їхні проекти. Тут вони можуть внести корективи в назву проекту, опис, визначити кінцевий термін збору коштів, встановити фінансову мету та завантажити медіа-матеріали.

Основні Елементи для Редагування:

Назва Проекту: Поле для введення або зміни назви проекту, яка має бути короткою та влучною.

Кінцевий Термін: Вибір дати, до якої повинен завершитись збір коштів. Це допомагає створити терміновість та може стимулювати швидші внески.

Фінансова Мета: Встановлення або зміна суми коштів, яка необхідна для реалізації проекту.

Опис Проекту: Багатий текстовий редактор, де можна додати детальний опис проекту, його цілей та значення.

Вибір Файлу: Функція для завантаження або зміни зображень або інших медіа-файлів, що представлятимуть проект.

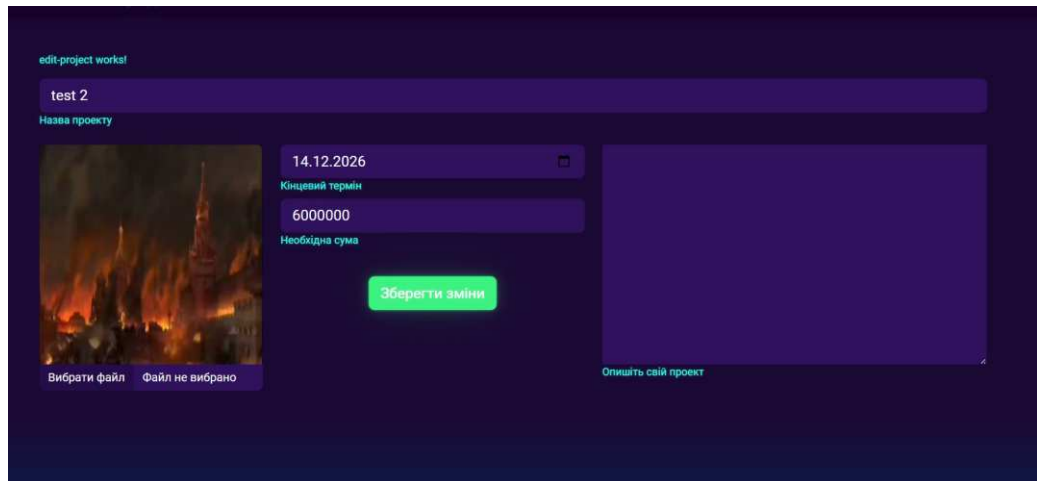


Рис 2.10 Сторінка редагування проекту

Сторінка створення проекту(2.11) — це інтерактивний інтерфейс, який дозволяє користувачам ініціювати нові краудфандингові кампанії. Вона забезпечує платформу, де автори можуть відобразити свою візію, мету та план дій для залучення фінансування.

Основні Елементи:

- **Форма Введення:** Центральний елемент сторінки, куди автор вносить всю ключову інформацію про свій проект, включно з назвою, детальним описом, фінансовими цілями та термінами. Завантаження
- **Медіа:** Функціонал для завантаження зображень та відео, який допомагає візуалізувати концепцію проекту.
- **Встановлення Фінансових Цілей:** Поля для введення суми, необхідної для втілення проекту, та можливості вказати різні рівні винагород для донорів.

Створення проекту проходить крок за кроком:

- Введення Інформації: Автор заповнює основні дані проекту, включаючи його назву та опис.
- Фінанси та Терміни: Встановлення необхідної суми та кінцевих термінів збору коштів.
- Медіа-контент: Додавання ілюстративних матеріалів, що робить проект привабливішим.
- Перегляд та Підтвердження: Перед завершенням, автори мають можливість переглянути всю введену інформацію та внести останні корективи.

Рис 2.11 Сторінка створення проекту

Сторінка зі списком підтриманих проектів є персональним портфоліо користувача на краудфандинговій платформі, де відображаються всі проекти, які він фінансово підтримав. Ця сторінка дозволяє користувачам відслідковувати свої інвестиції та бачити вплив своїх внесків на розвиток різних ініціатив.

Функції Сторінки:

- Перелік Проектів: Список всіх проектів, які користувач підтримав, з основною інформацією про кожен з них.

-Статус Проекту: Індикатори, які показують поточний статус кожного проекту, наприклад, чи він активний, чи вже фінансово завершений, чи в процесі виконання.

-Фінансові Внески: Детальна інформація про суму, яку користувач вніс у кожен проект, та специфікації винагород, якщо такі передбачені.

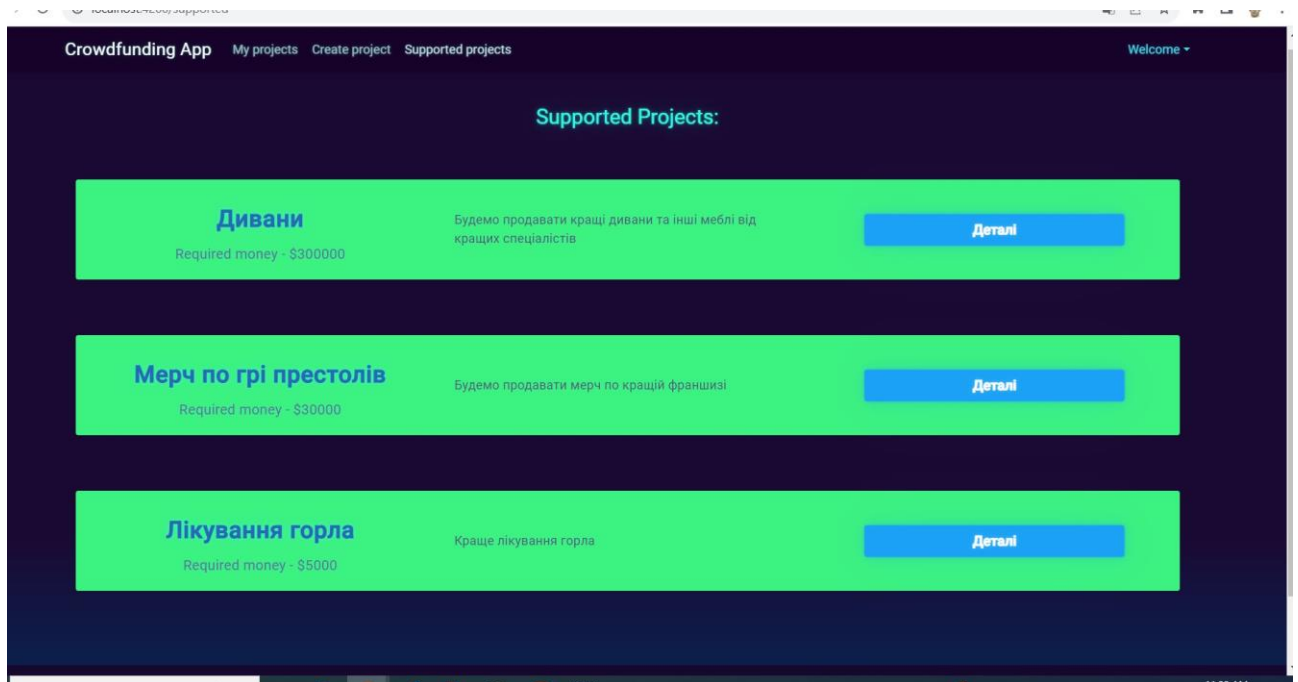


рис 2.12 Сторінка підтриманих проектів

2.4 Аналіз Використання патернів у веб застосунку краудфаундингової платформи.

У другому застосунку, який характеризується значно більшою складністю та розширеним функціоналом порівняно з простим телеграм-ботом, застосування архітектурних патернів стає не просто бажаним, але й необхідним. Ці патерни відіграють вирішальну роль у структуруванні коду, підвищенні його модульності, забезпеченні масштабованості та спрощенні подальшої підтримки.

Одним із патернів застосованих в другому застосунку є Domain Driven Design або DDD. Доменно-орієнтоване проектування (DDD) — це підхід до розробки програмного забезпечення, який зосереджується на моделюванні

програми навколо домену бізнесу, його логіки та поведінки. Це означає, що розробка програмного забезпечення ведеться з акцентом на складність самого бізнесу, а не на технології.

Традиційна трьохшарова архітектура, яка часто орієнтується на базу даних як центральний компонент, включає шари представлення, бізнес-логіки та доступу до даних. Хоча ця модель є перевіреною та ефективною для багатьох додатків, вона може обмежувати у випадках, коли бізнес-логіка є особливо складною або коли система повинна швидко адаптуватися до змінних бізнес-вимог. Тут DDD пропонує рішення, орієнтуючи проектування системи навколо бізнес-домену та його логіки.

Переваги DDD:

- Моделювання Бізнес-Процесів: DDD фокусується на створенні багатого доменного моделювання, що точно відображає реальні бізнес-процеси та правила. Це дозволяє розробникам та бізнес-аналітикам мати спільне розуміння системи.
- Гнучкість та Розширюваність: DDD спрощує процес внесення змін у бізнес-логіку, не впливаючи на інші частини системи. Архітектура стає більш гнучкою та легко адаптується під нові вимоги.
- Підтримка й Рефакторинг: Оскільки доменна логіка є відділеною від інфраструктурного коду, рефакторинг та підтримка системи стають менш ризикованими та більш контрольованими.
- Відділення Від БД: На відміну від трьохшарової архітектури, де зміни в базі даних можуть спричинити ланцюгові зміни у бізнес-логіці та на шарі представлення, DDD дозволяє ізолювати доменну модель від змін у схемі бази даних.
- Bounded Contexts: DDD використовує концепцію "Bounded Contexts" для організації доменної логіки, що забезпечує чітке розділення відповідальності та сприяє модульності системи.

Командно-Запитна Відповідальність Розподілу (CQRS) — це шаблон архітектури, який розділяє читання даних (запити) від їхньої модифікації (команди). Цей підхід відрізняється від традиційної архітектури, де один і той же модельний об'єкт використовується для обох операцій. Впровадження CQRS у проєкті сприяє створенню добре структурованої, ефективної та легко підтримуваної системи. Цей підхід підкреслює ідею, що різні аспекти обробки даних можуть вимагати різних архітектурних рішень, відповідно до їх специфіки та вимог до продуктивності. У вашому випадку, CQRS є ключовим елементом для реалізації високофункціональної та адаптивної системи.

Переваги CQRS:

- Простота та Чистота Коду: Розділення команд та запитів сприяє легкості розуміння та підтримки коду. Команди мають одну відповідальність — змінювати стан, в той час як запити лише читають дані, що робить код менш схильним до помилок.
- Гнучкість Масштабування: Читання та запис можуть бути масштабовані незалежно, що особливо корисно, коли система має велику кількість читань у порівнянні з кількістю записів, або навпаки.
- Оптимізація Продуктивності: Можливість використання різних моделей зберігання та оптимізації для читання та запису дозволяє оптимізувати кожну операцію за потребою.
- Вдосконалення Безпеки: Оскільки команди, що змінюють стан, відокремлені від запитів на читання, це може знизити ризик небажаних змін даних через помилки або зловмисні дії.

MediatR — це бібліотека для реалізації шаблону "медіатор" у програмному забезпеченні. Вона дозволяє об'єктам спілкуватися між собою через медіатора замість прямого виклику методів один одного. Це вводить рівень абстракції, який сприяє слабкому зв'язку та вищій модульності системи.

У контексті проєкту, MediatR використовується для обробки команд і запитів, що стосуються бізнес-операцій. Замість безпосереднього виклику

сервісів або репозиторіїв, компоненти вашого застосунку генерують "запити" або "команди", які передаються до MediatR. MediatR, у свою чергу, визначає, який обробник використовувати для даного запиту чи команди та делегує виконання відповідному обробнику.

Переваги MediatR:

- Слабкий Зв'язок (Loose Coupling): Компоненти системи стають менш залежними один від одного, що спрощує зміни та розширення функціоналу.
- Організація Коду: Завдяки централізації логіки обробки команд і запитів, код стає чистішим і легшим для розуміння.
- Тестування: Окремі обробники легше тестувати, оскільки вони не залежать від інших частин системи.
- Гнучкість: MediatR дозволяє легко додавати або змінювати обробники без зміни коду, який генерує запити або команди.

У даному проєкті MediatR використовується для спрощення архітектури CQRS, де розподіл команд та запитів відбувається ще зручніше. Кожен запит або команда має власний обробник, який може бути легко модифікований або замінений без впливу на іншу частину системи.

Включення MediatR у проєкт значно підвищує якість та ефективність розробки. Це надає великі переваги у підтримці чистоти коду та архітектурної гнучкості, роблячи застосунок більш масштабованим та легко адаптованим до мінливих бізнес-вимог і технологічних удосконалень.

Code First — це підхід до роботи з базами даних у програмуванні, який є частиною методології Entity Framework. Він дозволяє розробникам створювати об'єктні моделі в коді, а потім згенерувати з цих моделей схему бази даних. Тобто розробка починається із коду, а не із структури бази даних.

Code First можна вважати аспектом архітектурного підходу, оскільки він визначає спосіб взаємодії між додатком та базою даних, а також спосіб проєктування та впровадження моделей даних. Хоча він не охоплює всю

архітектуру програмного забезпечення, Code First має значний вплив на архітектурні рішення, пов'язані з даними.

Переваги Code First:

- Гнучкість Моделювання: Code First дозволяє легко модифікувати та розширювати доменні моделі у проекті, що сприяє адаптивності до змін у бізнес-вимогах.
- Ітеративна Розробка: можна швидко ітерувати моделі даних та відображати ці зміни у базі даних, не переймаючись про ручне створення таблиць та зв'язків.
- Контроль Версій: Code First дозволяє використовувати міграції для контролю версій схеми бази даних, що полегшує відстеження змін та управління ними.

Repository патерн — це широко відомий шаблон проектування, який використовується для абстрагування та капсулювання логіки доступу до даних. Він діє як проміжний шар між доменним шаром бізнес-логіки та шаром доступу до даних, дозволяючи останньому бути заміненим без впливу на бізнес-логіку.

Entity Framework Core (EF Core) вбудовує принципи Repository патерну на системному рівні. За допомогою класу DbContext та відповідних класів DbSet, EF Core надає готовий до використання рівень абстракції, який дозволяє взаємодіяти з базою даних через високорівневі операції. Це означає, що можна робити запити, вставки, оновлення та видалення без необхідності писати конкретний SQL код або вручну керувати підключенням до бази даних.

У цьому проекті рішення не використовувати явний Repository патерн зумовлене вже існуючою реалізацією в EF Core, яка виконує необхідні функції. Це відображає сучасний підхід до використання ORM (Object-Relational Mapping) інструментів, де додатковий шар репозиторію може стати надлишковим та додавати зайву складність.

Переваги Використання EF Core Замість Явного Repository Патерну:

- Простота Архітектури: Відсутність окремого репозиторію означає менше коду для підтримки, легшу архітектуру та менші затрати на розробку.
- Ефективність Розробки: Використання готових рішень від EF Core дозволяє швидше розробляти та впроваджувати функціонал.
- Управління Станом Об'єктів: EF Core ефективно керує станом об'єктів, їх життєвим циклом та відносинами, що спрощує роботу з даними.
- Міграції та Версіонування: Вбудовані інструменти для міграцій дозволяють легко вносити зміни в структуру бази даних.

2.5 Порівняльний аналіз проектів та патернів застосованих при їх розробці

Обидва проекти репрезентують різні кінці спектру програмного забезпечення в плані складності та бізнес-вимог.

Перший проект, телеграм-бот, є прикладом простого застосунку, де основна задача полягає в обробці вхідних запитів та наданні відповідей без складного бізнес-логічного шару або глибокої інтеграції з різноманітними даними. Завдяки своїй простоті, використання архітектурних патернів у цьому боті може бути зайвим і не виправданим, оскільки воно може призвести до надмірної складності та збільшити трудомісткість розробки та підтримки. Телеграм-бот не потребує великої кількості архітектурних шаблонів, оскільки його домен — це простий набір взаємодій, і вся його цінність полягає в автоматизації конкретних запитів користувачів. Тут використання складних архітектурних патернів могло б ускладнити систему без значного збільшення її функціональності або продуктивності.

Навпаки, краудфандингова система має складну бізнес-модель, яка включає різноманітні види взаємодій користувачів, фінансові транзакції, управління проектами та інші. Для такої системи використання архітектурних патернів не лише доречне, але й критично необхідне для забезпечення її надійності, масштабованості та підтримки.

В контексті краудфандингової системи, DDD, CQRS, та MediatR виконують ключові ролі:

- DDD допомагає структурувати комплексний домен у вигляді моделі, яка відображає бізнес-правила та обмеження, сприяючи розробці функцій, що точно відповідають бізнес-логіці.
- CQRS дозволяє оптимізувати читання та запис в різних частинах системи, таким чином підвищуючи продуктивність та дозволяючи легше масштабувати кожен компонент незалежно.
- MediatR забезпечує ефективну комунікацію між компонентами системи, дозволяючи легко внести зміни в бізнес-операції без потреби змінювати кожен компонент, який взаємодіє з цими операціями.

В телеграм-боті відсутність архітектурних патернів є прийнятною через низький рівень бізнес-складності та обмежений обсяг функціональності. EF Core виконує роль ORM і дає достатньо можливостей для реалізації необхідного рівня взаємодії з базою даних, тому додатковий шар у вигляді Repository патерну не потрібен.

Порівняння цих двох проектів виявляє, що архітектурні рішення мають бути обрані на основі конкретних потреб та вимог кожного окремого проекту. Де для одного проекту складні патерни є зайвими, для іншого — вони невід'ємні для забезпечення необхідної функціональності, продуктивності та можливості розвитку в майбутньому.

Висновки до розділу 2

Вибір архітектурного підходу та патернів повинен ґрунтуватися на конкретних потребах та складності проекту. У випадку простого бота, додавання складних патернів було б зайвим, тоді як у складній краудфандинговій системі вони забезпечують необхідну організацію, масштабованість та гнучкість для підтримки багат шарового бізнес-функціоналу.

РОЗДІЛ 3 ОГЛЯД ПРОГРАМНОЇ РЕАЛІЗАЦІЇ ПРОЕКТІВ

3.1 Програмна реалізація телеграм бота

Огляд Структури Проекту Телеграм-Бота:

Папка "Abstract"(рис 3.1) зберігає в собі інтерфейси та базові класи для сервісів, що використовуються в застосунку:

- IChatGptService.cs: Інтерфейс, що описує сервіс для взаємодії з ChatGPT, можливо, для генерації відповідей бота.
- IReceiverService.cs: Інтерфейс для сервісу, який отримує оновлення від Телеграму.
- PollingServiceBase.cs: Базовий клас для сервісу, який може періодично опитувати Телеграм на предмет нових повідомлень.
- ReceiverServiceBase.cs: Базовий клас для сервісу, що отримує дані від Телеграму.

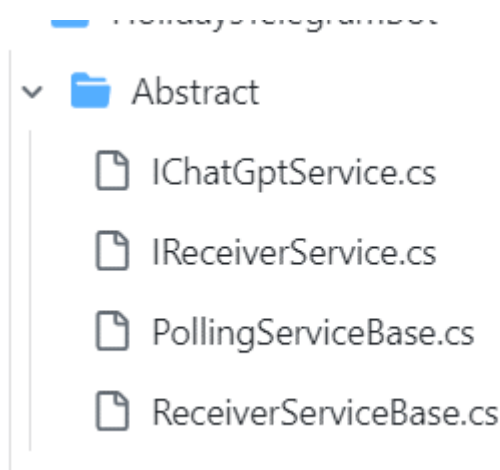


Рис 3.1 папка Abstract

Папка "Domain"(рис 3.2):

- Gender.cs: Enum, що містить визначення статей користувачів.
- States.cs: Enum, що зберігає стани користувачів.
- UserContext.cs: Клас, який зберігає контекст користувача, його дані.

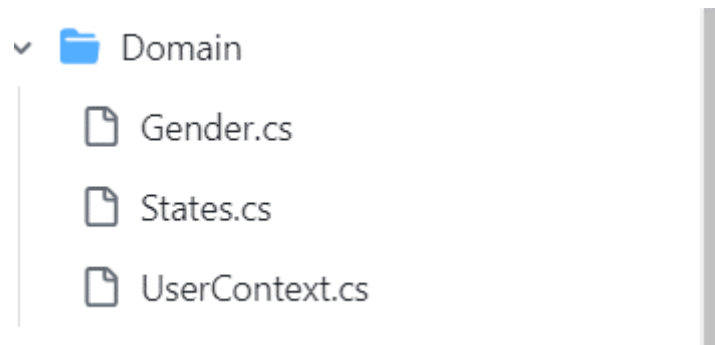


Рис 3.2 папка Domain

Папка "Migrations" – папка де зберігаються усі міграції а також файл, який містить знімок поточної моделі бази даних для EF Core, використовується для міграцій.

Папка "Options" має в собі лише один клас - BotOptions.cs(рис 3.3). Клас BotOptions являє собою конфігураційний клас для параметрів Телеграм-бота. Він містить властивість Token, яка ініціалізується як null. Ця властивість, використовується для зберігання токена аутентифікації бота, який необхідний для взаємодії з Телеграм API.

Використання таких класів опцій є типовим у .NET додатках для управління конфігураційними налаштуваннями та дозволяє зробити код більш чистим та легко змінюваним.

```
1      namespace HolidaysTelegramBot.Options;
2
3      public class BotOptions
4      {
5          public string Token { get; init; } = null!;
6      }
```

Рис 3.3 клас BotOptions

Папка "Services" має в собі основні необхідні сервіси:

- ChatGptService(рис 3.4) використовується для спілкування з OpenAI API і обробки запитів від користувачів, трансформуючи їх у відповіді від чат-бота.

```

public async Task<string> Ask(UserContext context)
{
    var key = Environment.GetEnvironmentVariable("OpenAiKey");
    var openAi = new OpenAIAPI(key);
    var chat = openAi.Chat.CreateConversation();
    var queryBuilder = new StringBuilder();
    queryBuilder.Append("Name - " + context.Name);
    queryBuilder.Append(" Age - " + context.Age);
    queryBuilder.Append(" Gender - " + context.Gender);
    queryBuilder.Append(" Job/Hobie - " + context.Hobie);
    queryBuilder.Append(" Description - " + context.Description);
    chat.AppendSystemMessage("You get a description of a person and have to write some ideas for present" +
        " to this person. Write only propositions for present to this person.");
    chat.AppendUserInput(queryBuilder.ToString());
    return await chat.GetResponseFromChatbotAsync();
}
}

```

Рис 3.3 клас ChatGptService

- ReceiverService обробляє вхідні оновлення від телеграм-бота і передає їх до обробника оновлень.

```

7 // Compose Receiver and UpdateHandler implementation
8 public class ReceiverService : ReceiverServiceBase<UpdateHandler>
9 {
10     public ReceiverService(
11         ITelegramBotClient botClient,
12         UpdateHandler updateHandler,
13         ILogger<ReceiverServiceBase<UpdateHandler>> logger)
14         : base(botClient, updateHandler, logger)
15     {
16     }
17 }

```

Рис 3.4 клас ReceiverService

- PollingService відповідає за періодичну перевірку оновлень з телеграм-сервера. Він успадковує функціональність від базового класу, який надає спільну логіку для політінгу.
- UpdateHandler визначає логіку реакції на різні типи оновлень, які може отримати бот — нові повідомлення або команди від користувачів.

Кожен з цих сервісів виконує визначену роль у загальній архітектурі бота, забезпечуючи модульність і чітке розділення відповідальності. Таке розділення дозволяє легко модифікувати або оновлювати окремі частини системи без впливу на інші, що є важливим аспектом при розробці програмного забезпечення.

Решта Файлів та Папок:

- ApplicationDbContext.cs: Клас, який репрезентує контекст бази даних в EF Core.
- Program.cs: Вхідна точка виконання програми.
- Файл docker-compose.yml(рис 3.5), який є частиною налаштувань Docker для проекту. Docker дозволяє створювати, розгортати та управляти додатками в ізольованих середовищах, званих контейнерами. У цьому налаштовано сервіс postgres, який використовує офіційний образ PostgreSQL з Docker Hub. Сервіс налаштований на рестарт у разі збою і зберігає дані в папці db-data/postgres на хост-машині, що забезпечує постійність даних навіть після перезапуску контейнера. Конфігурація сервісу включає в себе встановлення користувача бази даних, пароля та назви бази даних, що забезпечує зручне та швидке розгортання бази даних для розробки та тестування.

```
1    version: '3'
2
3    services:
4      postgres:
5        image: 'postgres:14.2'
6        ports:
7          - "5432:5432"
8        restart: always
9        environment:
10         POSTGRES_USER: postgres
11         POSTGRES_PASSWORD: password
12         POSTGRES_DB: bot
13        volumes:
14          - ./db-data/postgres:/var/lib/postgresql/data/
```

Рис 3.5 файл docker-compose.yml

Аналізуючи структуру та файли проекту, можна зробити висновок, що його архітектура є прямолінійною та модульною, що робить його простим для розуміння. Ось декілька ключових моментів:

- Простота структури: Проект організований у логічні блоки, такі як Services, Options, та інші, що робить структуру чистою та легкою для навігації.
- Використання абстракцій: Інтерфейси, такі як IChatGptService, сприяють розосередженню відповідальності та можливості заміни компонентів без впливу на інші частини системи.
- Зосередження на функціональності: Сервіси як ChatGptService та UpdateHandler зосереджені на конкретних задачах, що свідчить про чітке розмежування функціональності.
- Використання сторонніх бібліотек: Включення таких залежностей, як бібліотека Telegram Bot API та OpenAI, дозволяє використовувати готові рішення замість розробки власних, що спрощує розробку.
- Конфігурація через змінні середовища: Налаштування, такі як токени, зберігаються в змінних середовища, що є кращою практикою для управління конфіденційною інформацією.
- Використання Docker для контейнеризації полегшує розгортання та управління проектом.
- Мінімалізм у залежностях: Проект використовує мінімально необхідні залежності, що робить його легким для розуміння та підтримки.

Проект реалізований таким чином, що він зосереджується на виконанні конкретних задач з мінімумом зайвої складності. Це робить його досить простим для розуміння і легким для масштабування чи модифікації. Відсутність складних архітектурних патернів не впливає на ефективність проекту, оскільки він не передбачає розгорнутих архітектурних вимог, які можуть бути необхідними в більш великих і комплексних системах. Цей підхід дозволяє швидко розробляти та впроваджувати нові функції, в той час як контейнеризація через Docker гарантує легке та консистентне розгортання.

3.2 Програмна реалізація Краудфандингової платформи.

Другий проект - це краудфандингова платформа, яка складається з бекенду та фронтенду. Бекенд поділений на кілька основних частин:

- API (Application Programming Interface): Це інтерфейс, який виступає в ролі містка між фронтендом та бекендом. Контролери API обробляють запити від клієнта та відповідають на них, використовуючи бізнес-логіку та дані, отримані з доменного та шару персистентності.
- Domain: Цей шар містить бізнес-логіку та бізнес-моделі. Він визначає бізнес-правила та операції, які стосуються об'єктів домену, таких як проекти краудфандингу, користувачі, внески тощо.
- Core: Де знаходяться усі необхідні Dto.
- Persistence: Шар персистентності відповідає за зберігання та отримання даних з бази даних. Він містить контексти бази даних, класи доступу до даних та конфігурації сутностей.

На фронтенді це Web. Це частина, з якою взаємодіють користувачі. Вона побудована на фреймворку Angular, і відповідає за представлення інтерфейсу користувача, взаємодію з API та представлення даних.

Кожен з цих шарів виконує свою роль у загальній архітектурі додатку та допомагає в організації коду таким чином, щоб він був модульним, легко розширюваним та легко підтримуваним.

У шарі API є директорія Controllers, яка містить контролери, такі як ImagesController, ProjectsController, TiersController і UserController. Контролери служать входом для HTTP запитів, які надходять на ваш сервер, і зазвичай містять логіку для обробки різних API запитів, направляючи їх до відповідних сервісів чи обробників.

Контролер ImagesController має методи для завантаження і отримання зображень, ProjectsController управляє проектами (отримання, створення, підтримка, публікація, видалення), TiersController відповідає за управління

рівнями підтримки проєктів, і UserController виконує дії, пов'язані з користувачами (отримання, реєстрація, вхід, видалення).

Використання атрибутів TsFunction і TsClass з бібліотеки Reinforced.Typings дозволяє автоматично генерувати TypeScript визначення з C# коду, що спрощує інтеграцію між фронтом та бекендом.

Для забезпечення безпеки, використовуються атрибути Authorize, що дозволяє рестрикувати доступ до певних методів тільки для авторизованих користувачів або користувачів з певними ролями. Атрибут EnableCors використовується для встановлення політик CORS, що дозволяє здійснювати міждоменні запити.

Ось короткий огляд того, як працюють ці контролери:

ImagesController:

- UploadPhoto - завантаження зображення.
- GetPhoto - отримання зображення за ID.

ProjectsController:

- GetAllPublished - отримання всіх опублікованих проєктів.
- GetAll - отримання всіх проєктів (тільки для адміністраторів).
- GetAllByUser - отримання проєктів, створених користувачем, відфільтрованих за статусом.
- GetById - отримання деталей проєкту за ID.
- Create - створення нового проєкту.
- SupportProject - підтримка проєкту.
- PublishProject - публікація проєкту.
- Delete - видалення проєкту (тільки для адміністраторів).

TiersController:

- Create - створення нового рівня підтримки.
- GetAll - отримання всіх рівнів підтримки.

- GetById - отримання рівня підтримки за ID.
- Delete - видалення рівня підтримки.
- Update - оновлення рівня підтримки.

UserController:

- GetAll - отримання списку всіх користувачів.
- GetById - отримання детальної інформації користувача за ID.
- Login - вхід користувача в систему.
- Register - реєстрація нового користувача.
- Delete - видалення користувача.

Папка Typings в проекті містить класи, які налаштовують генерацію TypeScript типів і сервісів для Angular з використанням бібліотеки Reinforced.Typings. Вона дозволяє автоматично генерувати TypeScript відповідності для C# моделей, DTOs (Data Transfer Objects), і контролерів, забезпечуючи синхронізацію типів між серверною та клієнтською частинами програми.

AngularActionCallGenerator(рис 3.6) та AngularControllerGenerator є кастомними генераторами, які модифікують стандартну поведінку генерації коду в Reinforced.Typings. AngularActionCallGenerator перетворює C# методи контролерів на TypeScript функції, які повертають Observable з бібліотеки RxJS, і налаштовує HttpClient запити з урахуванням шляху до ендпоінту і HTTP методу. AngularControllerGenerator створює класи Angular сервісів з методами, що відповідають ендпоінтам API.

```

using System.Globalization;
using System.Reflection;
using System.Text;
using System.Text.RegularExpressions;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Routing;
using Reinforced.Typings;
using Reinforced.Typings.Ast;
using Reinforced.Typings.Ast.TypeNames;
using Reinforced.Typings.Generators;

namespace CrowdFundingAPI.Typings;

public class AngularActionCallGenerator : MethodCodeGenerator
{
    public override RtFunction? GenerateNode(MethodInfo element, RtFunction result, TypeResolver resolver)
    {
        private static string GetPath(MethodInfo element, HttpMethodAttribute? methodAttribute)
        {
            var routeTemplate = element.DeclaringType?.GetCustomAttributes<RouteAttribute>().FirstOrDefault()?.Template;
            var actionRoute = methodAttribute?.Template;

            string GetRouteValue(MethodInfo m, string val) => m.GetParameters().Any(p => p.Name == val) ? $"${{{val}}}" : "0";
            string RouteParam(string p) => @"\" + p + @":?\w*\??\";
            string? AdjustedRoute(MethodInfo method) => actionRoute != null ?
                Regex.Replace(actionRoute, RouteParam(@"(\w+)"), m => GetRouteValue(method, m.Groups[1].Value)) : null;

            return $"/{routeTemplate}/{AdjustedRoute(element)}";
        }
    }
}

```

Рис 3.6 клас AngularActionCallGenerator

Конфігураційний клас ReinforcedTypingsConfiguration визначає, які типи повинні бути експортовані як TypeScript інтерфейси і налаштовує загальні параметри для всіх типів, такі як використання CamelCase і модулів.

Клас Visitor(рис 3.7) дозволяє вам модифікувати поведінку відвідувача при виводі TypeScript коду, якщо вам потрібно внести якісь специфічні зміни, які не покриваються стандартною функціональністю Reinforced.Typings.

```

1    using Reinforced.Typings;
2    using Reinforced.Typings.Ast;
3    using Reinforced.Typings.Visitors.TypeScript;
4
5    namespace CrowdFundingAPI.Typings;
6
7    public class Visitor : TypeScriptExportVisitor
8    {
9        public Visitor(TextWriter writer, ExportContext exportContext)
10           : base(writer, exportContext)
11        {
12        }
13
14        public override void Visit(RtField node)
15        {
16            base.Visit(node);
17        }
18    }

```

Рис 3.7 клас Visitor

Використання цих класів значно спрощує процес інтеграції фронкенду та бекенду, автоматизуючи створення клієнтського API сервісу, який точно відповідає контролерам сервера. Це також зменшує можливість помилок, пов'язаних із несинхронізацією між фронкендом і бекендом, і підвищує ефективність роботи розробників.

Для того, щоб використовувати бібліотеку Reinforced.Typings для генерації TypeScript типів і сервісів, потрібно сконфігурувати файл ReinforcedTypings.settings.xml, який є частиною процесу збірки MSBuild. Цей файл визначає, куди генерувати вихідні файли TypeScript, які класи і методи експортувати, і які особливі параметри використовувати під час генерації.

Ось ключові параметри конфігурації:

- RtTargetFile: Це шлях до файлу, де буде згенеровано TypeScript код. Якщо RtDivideTypesAmongFiles встановлено в true, цей параметр не використовується.
- RtConfigurationMethod: Вказує на метод конфігурації, який використовується для налаштування експортування типів. У вашому випадку це метод CrowdFundingAPI.Typings.ReinforcedTypingsConfiguration.Configure.
- RtDivideTypesAmongFiles: Коли встановлено в true, генерує файли типів TypeScript у вигляді окремих файлів, подібно до структури класів у C# або Java.
- RtTargetDirectory: Якщо використовується RtDivideTypesAmongFiles, це шлях до каталогу, де будуть розміщені всі згенеровані файли.

Крім того, є параметри для тимчасового вимкнення генерації типів або TypeScript компіляції (RtDisable, RtBypassTypeScriptCompilation), а також параметри для вказівки додаткових збірок, з яких потрібно експортувати типи (RtAdditionalAssembly).

Файл `ReinforcedTypings.settings.xml` дозволяє детально контролювати процес генерації і є централізованим місцем для управління типами, які використовуються між фронтендом і бекендом. Завдяки цьому, можна ефективно синхронізувати типи даних і API виклики, забезпечуючи консистентність і безпеку типів у додатку.

Валідатори — це важливий компонент у програмній інженерії, який забезпечує перевірку та забезпечення відповідності даних, що вводяться, встановленим правилам та обмеженням. Вони допомагають запобігти некоректній або шкідливій інформації в системі, що може призвести до помилок або вразливостей. У вашому проекті використовуються валідатори для забезпечення правильного введення даних у різних сценаріях, таких як створення проекту, реєстрація користувачів тощо.

В даному проекті використовується `FluentValidation`, бібліотека для .NET, яка дозволяє легко побудувати потужні правила валідації для моделей домену. Наприклад, `CreateProjectValidator` (рис 3.8) забезпечує, що заголовок проекту не є порожнім і не перевищує 50 символів у довжину, а опис проекту не перевищує 2000 символів. Це допомагає уникнути помилок з боку користувачів і забезпечує, що дані відповідають очікуванням системи.

```

using Domain.Features.ProjectFeatures.Commands;
using FluentValidation;

namespace CrowdfundingAPI.Validators.ProjectValidators
{
    ✓ public class CreateProjectValidator : AbstractValidator<CreateProjectCommand>
    {
        public CreateProjectValidator()
        {
            RuleFor(x => x.Title)
                .NotEmpty().WithMessage("Title is required!")
                .Length(0, 50).WithMessage("Title Must be less than 50 symbols");
            RuleFor(x => x.Description)
                .Length(0, 2000)
                .WithMessage("Description must be less than 2000 symbols");
            //RuleFor(x => x.LastDay)
            //    .NotEmpty();
            RuleFor(x => x.RequiredMoney)
                .NotNull()
                .GreaterThan(0).WithMessage("Required amount must be greater than 0");
        }
    }
}

```

Рис 3.8 Валідатор CreateProjectValidator

Використання валідаторів в автоматизованій манері дозволяє підтримувати високий рівень якості даних і спрощує процес перевірки на стороні сервера. Це також полегшує розробку та підтримку за рахунок чіткого визначення правил валідації в одному місці, що робить код більш читабельним та легким для розуміння.

- Валідатори в проекті розділені на три основні категорії: ProjectValidators, TierValidators, та UserValidators. Кожна з цих категорій має власні класи валідаторів, які відповідають за перевірку вхідних даних конкретних доменів або функцій системи.
- ProjectValidators: Ця категорія містить валідатори для проектів, які перевіряють вхідні дані при створенні та оновленні проектів. Наприклад, CreateProjectValidator перевіряє, що назва проекту не є порожньою та не перевищує встановлену довжину символів, а опис проекту відповідає вимогам до максимальної довжини.

- TierValidators: Тут розміщені валідатори для рівнів підтримки проектів, забезпечуючи, що дані для створення та оновлення рівнів є коректними та відповідають визначеним правилам.
- UserValidators: Ця категорія включає валідатори для користувачів, які застосовуються при реєстрації нових користувачів та при вході в систему. Вони перевіряють коректність таких даних як ім'я користувача, електронна адреса, пароль, забезпечуючи, що вони відповідають необхідним критеріям безпеки та форматування.

Використання цієї структурованої системи валідації дозволяє легко управляти перевіркою даних для різних аспектів системи, уникати дублювання коду та забезпечити однаковий підхід до валідації по всьому проекту. Кожен валідатор розширює базовий клас `AbstractValidator` від `FluentValidation`, що дозволяє використовувати потужну і гнучку систему правил для перевірки моделей.

Файл `Program.cs` в проекті `.NET` є точкою входу для запуску веб-додатку та конфігурування сервісів і міدلварів, які будуть використовуватися в додатку. Нижче наведений огляд ключових компонентів, які налаштовуються у цьому файлі:

- Створення `WebApplication Builder`: Використовується для налаштування сервісів та конфігурації, які будуть використовуватися додатком.
- Конфігурація Серіалізації `JSON`: Встановлює налаштування серіалізації для обробки циклічних посилань в `JSON` об'єктах.
- `Swagger`: Додає інструментарій для документації `API`, що дозволяє легко тестувати ендпоінти.
- Доступ до Контексту `HTTP`: Надає сервіси доступу до `HTTP` контексту всередині додатку.

- Валідація: Підключає автоматичну валідацію та валідатори з окремого зібрання (assembly), в цьому випадку, валідатори для рівнів підтримки проектів.
- MediatR: Включає MediatR для імплементації шаблону "Mediator", що дозволяє зменшити пряму залежність між компонентами системи.
- Entity Framework та Identity: Налаштовує контекст бази даних та систему ідентифікації користувачів.
- CORS: Встановлює політику CORS для контролю того, які запити з інших доменів можуть взаємодіяти з додатком.
- Quartz.NET: Інтегрує планувальник задач для запуску фонових завдань на певні проміжки часу.
- Аутентифікація та JWT: Конфігурує аутентифікацію з використанням JWT токенів для безпечної перевірки користувачів.
- AutoMapper: Налаштовує AutoMapper для автоматизації процесу відображення даних між об'єктами.
- Запуск додатку: Налаштовує пайплайн HTTP запитів, включаючи редирект на HTTPS, використання CORS, аутентифікації та авторизації, і визначає маршрути контролерів.
- ErrorHandler Middleware: Додає проміжне програмне забезпечення (middleware) для обробки помилок, що виникають під час запитів.

Цей файл відображає складність та широту налаштувань, які можуть бути виконані при створенні сучасного веб-додатку на .NET, забезпечуючи гнучке управління залежностями та конфігурацією.

Наступним можна розібрати Core шар. Цей шар містить Data Transfer Objects (DTOs), які є ключовою частиною шаблону проектування, відомого як "Layered Architecture" або "N-Tier Architecture". DTOs слугують для обміну даними між різними рівнями аплікації, наприклад, між бізнес-логікою та презентаційним шаром.

У Core леєрі можуть знаходитися такі категорії DTOs:

Project:

- ProjectDto.cs: Містить базову інформацію про проект.
- ProjectWithTiersDto.cs: Може містити деталі проекту разом з інформацією про його рівні фінансування або "tiers".
- PublishedProjectDto.cs: Специфічний DTO для опублікованих проектів, який можливо містить поля, що необхідні для відображення проекту у публічному доступі.

Tier:

- TierDto.cs: Містить інформацію про конкретні рівні або пакети фінансування в рамках проектів.

User:

- LoginDto.cs: Використовується для передачі даних авторизації користувача.
- UserDto.cs: Містить інформацію про користувачів системи.

Кожен файл DTO визначає структуру даних, яку можна легко передати мережею, оскільки вони зазвичай не містять складних бізнес-правил або логіку обробки. Це спрощує обмін даними між клієнтами та сервером, оскільки DTO визначають чіткий контракт інтерфейсу.

Користування такими DTO дозволяє зменшити кількість залежностей між різними рівнями додатку, оскільки кожен рівень може використовувати лише ту інформацію, яка йому потрібна, без необхідності знання внутрішньої структури інших частин системи. Це також сприяє кращому розділенню відповідальності та легшому тестуванню і підтримці коду.

Чиста архітектура (Clean Architecture) та доменно-орієнтоване проектування (Domain-Driven Design, DDD) є методологіями, які спрямовані на створення програмного забезпечення з високим рівнем абстракції та модульності. Ось як вони реалізовані в Domain леєрі.

Цей леєр має наступну структуру:

- **Abstract:** Ця папка, містить абстракції та інтерфейси, які визначають контракти для класів та сервісів, використовуваних у проекті. У DDD, абстракції визначають інтерфейси та базові класи, які можуть бути реалізовані в домені або інфраструктурних шарах. Це відокремлює доменний шар від зовнішніх залежностей та дозволяє зосередитися на бізнес-логіці.
- **DomainModels:** Тут знаходяться класи доменної моделі, які представляють концепції домену бізнесу та містять бізнес-логіку. У DDD, доменний модель — це серце системи, де визначається бізнес-логіка та правила.
- **Features:** Папка, яка організована за компонентами проекту, містить підпапки для різних доменних об'єктів та функцій.
- **Mapper:** Містить класи та конфігурації для відображення одного об'єкта на інший, часто використовується з AutoMapper.
- **Services:** Папка для сервісів, які виконують бізнес-логіку та операції на рівні домену.
- **ErrorHandler:** Містить середній рівень обробки помилок для обробки винятків на рівні додатку.

Ці компоненти та структура проекту дозволяють зосередитися на бізнес-правилах та логіці, а не на деталях імплементації. Вони також допомагають уникнути жорсткої залежності від бази даних або інших зовнішніх сервісів, що робить систему більш гнучкою та легкою для тестування і підтримки.

Папка Features у проекті організована відповідно до принципів CQRS (Command Query Responsibility Segregation), де логіка розподіляється між командами, які змінюють стан системи (запис), та запитам, які отримують стан (читання).

Далі розглянемо приклад коду реалізації шаблону CQRS, Команда CreateProjectCommand (рис 3.9) інкапсулює всю інформацію, яка необхідна для

створення нового проекту, включаючи заголовок, опис, URL зображення, кінцевий термін і необхідну суму коштів.

```
public class CreateProjectCommandHandler : IRequestHandler<CreateProjectCommand>
{
    private readonly IApplicationDbContext _context;
    private readonly IUserService _userService;
    private readonly IMapper _mapper;

    public CreateProjectCommandHandler(IApplicationDbContext context, IUserService userService, IMapper mapper)
    {
        _context = context;
        _userService = userService;
        _mapper = mapper;
    }

    public async Task Handle(CreateProjectCommand request, CancellationToken cancellationToken)
    {
        var project = new Project()
        {
            Status = Status.Draft,
            CreatorId = _userService.GetUserId(),
            InvestedMoney = 0
        };

        project = _mapper.Map(request, project);

        var result = _context.Projects.Add(project);

        if (result is null)
        {
            throw new Exception("Can't create project");
        }

        await _context.SaveChangesAsync();
    }
}
```

Рис 3.9 Команда CreateProjectCommand

Обробник команди CreateProjectCommandHandler відповідає за прийняття команди та її виконання. Він використовує інтерфейс IApplicationDbContext для взаємодії з базою даних та зберігання нового проекту, IUserService для отримання ідентифікатора користувача, який створює проект, та IMapper для мапінгу даних з команди на екземпляр сутності Project.

Паттерн Mediator тут реалізується за допомогою бібліотеки MediatR, яка дозволяє об'єктам взаємодіяти без прямого посилання один на одного, замість цього використовуючи посередника - медіатора. В даному випадку MediatR

відокремлює виклик команди від її виконання, дозволяючи обробнику команди зосередитись виключно на логіці обробки команди. Коли `CreateProjectCommand` створюється і відправляється через `MediatR`, бібліотека автоматично знаходить і викликає відповідний обробник команд, в даному випадку `CreateProjectCommandHandler`. Це дозволяє легко розширювати та тестувати ваші команди, оскільки кожна команда має один обробник, який відповідальний за її виконання.

Також у наведеному коді `CreateProjectCommandHandler` використовує `AutoMapper` та контекст бази даних `IApplicationDbContext` для створення нового запису проекту. `AutoMapper` - це бібліотека, що спрощує процес перетворення одного об'єкта в інший. У цьому випадку, `AutoMapper` використовується для відображення даних з `CreateProjectCommand` на новий екземпляр `Project`. Це знижує кількість коду, необхідного для ручного присвоєння значень кожного поля, і дозволяє централізовано управляти правилами перетворення.

`IApplicationDbContext` - це абстракція, яка представляє сесію з базою даних і може використовуватися для запиту та зберігання екземплярів сутностей. У цьому випадку, `IApplicationDbContext` використовується для додавання нового проекту в базу даних і збереження цієї зміни.

Метод `Add` на `DbSet<Image>` в контексті `IApplicationDbContext` додає новий об'єкт `Image` в контекст, який пізніше буде вставлений у базу даних при виклику `SaveChanges`. Метод `SaveChanges` асинхронно зберігає всі зміни, зроблені в контексті бази даних, тобто вставляє новий проект в базу даних.

Цей підхід відділяє логіку доступу до даних від логіки бізнес-операцій, спрощуючи тестування та обслуговування коду, а також забезпечуючи чистоту архітектури, оскільки залежності між різними частинами системи мінімізовані.

У папці `Extensions` (рис 3.11), яка містить методи розширення, знаходиться файл `MapperExtensions.cs`. В цьому файлі реалізовано статичний клас `MapperExtensions`, який включає метод `ConfigureMapper(3.10)`. Цей метод розширення використовується для додавання конфігурації мапінгу до

контейнера залежностей, забезпечуючи централізоване місце для управління мапінгами AutoMapper.

```
public static class MapperExtensions
{
    public static void ConfigureMapper(this IServiceCollection services)
    {
        var mapperConfig = new MapperConfiguration(mc =>
        {
            mc.AddProfile(new TierProfile());
            mc.AddProfile(new CreateTierProfile());
            mc.AddProfile(new UpdateTierProfile());
            mc.AddProfile(new UserProfile());
            mc.AddProfile(new RegisterUserProfile());
            mc.AddProfile(new CreateProjectProfile());
            mc.AddProfile(new UpdateProjectProfile());
            mc.AddProfile(new PublishedProjectProfile());
            mc.AddProfile(new GetProjectByIdProfile());
            mc.AddProfile(new ProjectProfile());
        });
        var mapper = mapperConfig.CreateMapper();
        services.AddSingleton(mapper);
    }
}
```

▼ Extensions

MapperExtensions.cs

Рис 3.10-3.11 метод розширення ConfigureMapper, папка Extensions

Конфігурація мапінгу в ConfigureMapper включає ряд профілів мапінгу, кожен з яких містить визначення того, як об'єкти одного типу мають бути перетворені в об'єкти іншого типу. Наприклад, може бути профіль для мапінгу даних DTO (Data Transfer Object) в сутності бази даних або навпаки.

Ось ключові моменти щодо цього методу розширення:

- **MapperConfiguration:** Це клас AutoMapper, який дозволяє визначити конфігурацію мапінгу, яка потім може бути використана для створення мапера. У цьому методі мапінг конфігурується за допомогою виклику AddProfile, де кожен профіль відповідає за мапінг певного типу об'єктів.
- **Profiles:** Використовуються для організації та інкапсуляції конкретних правил мапінгу для різних типів об'єктів. Вони

визначають, як властивості одного об'єкта мають відображатися на властивості іншого. Це дозволяє розділити та управляти конфігурацією мапінгу на високому рівні.

- `IServiceCollection`: `services` є змінною, що представляє колекцію служб, які реєструються в контейнері залежностей. Метод `ConfigureMapper` додає сконфігурований мапер як одиничний екземпляр (`singleton`), що означає, що один і той же екземпляр мапера буде використовуватися в усьому застосунку.
- `Singleton`: Реєстрація мапера як одиничного екземпляра гарантує, що буде створено тільки один екземпляр `IMapper` на все життя застосунку, що допомагає зберегти ресурси та забезпечити консистентність використання мапінгу.

Цей підхід є досить розповсюдженим у застосунках, що використовують `AutoMapper`, оскільки він спрощує управління мапінгами та інтеграцію з `Dependency Injection` системою, дозволяючи легко впроваджувати мапер у будь-який компонент застосунку.

Папка `Mapper` у проекті містить конфігурації для мапінгу, які визначають як дані з одних об'єктів переносяться в інші. У прикладі, наданому у класі `CreateProjectProfile.cs`, створюється мапінг між командою `CreateProjectCommand` та сутністю `Project`.

Цей профіль мапінгу визначає, що коли команда `CreateProjectCommand` виконується, дані з її властивостей `Title`, `Description`, `ImageUrl`, `LastDay`, та `RequiredMoney` будуть автоматично перенесені в відповідні властивості сутності `Project`. Завдяки цьому, не потрібно вручну присвоювати кожне поле при створенні нового проекту, що значно спрощує процес кодування та зменшує можливість помилок.

Такий підхід є дуже ефективним при роботі з шарами, де об'єкти домену (бізнес-логіки) відрізняються від моделей передачі даних (DTOs), що використовуються у комунікації з зовнішніми агентами, такими як користувачі

через API. Він дозволяє чітко відокремити бізнес-логіку від логіки транспортування даних, що є ключовим аспектом чистої архітектури і DDD.

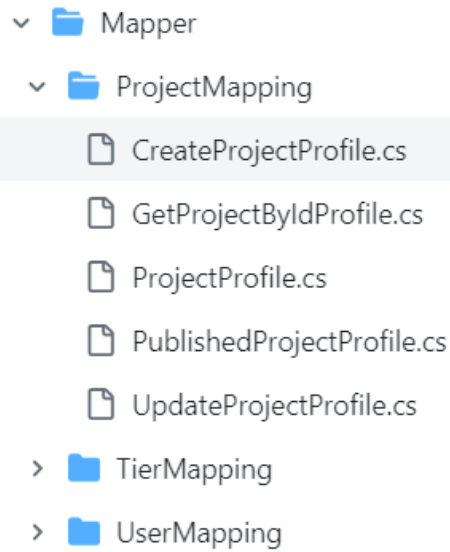


Рис 3.12 Папка Mapper

```
public class CreateProjectProfile : Profile
{
    public CreateProjectProfile()
    {
        CreateMap<CreateProjectCommand, Project>()
            .ForMember(dest =>
                dest.Title,
                opt => opt.MapFrom(src => src.Title))
            .ForMember(dest =>
                dest.Description,
                opt => opt.MapFrom(src => src.Description))
            .ForMember(dest =>
                dest.TitleImage,
                opt => opt.MapFrom(src => src.ImageUrl))
            .ForMember(dest =>
                dest.LastDay,
                opt => opt.MapFrom(src => src.LastDay))
            .ForMember(dest =>
                dest.RequiredMoney,
                opt => opt.MapFrom(src => src.RequiredMoney));
    }
}
```

Рис 3.13 класі CreateProjectProfile.cs

Папка Services(рис 3.14) у проекті призначена для розміщення сервісів, які виконують певні бізнес-функції або допоміжні операції. В цій папці розташовані наступні сервіси: ErrorHandler, JwtTokenService, UserService.

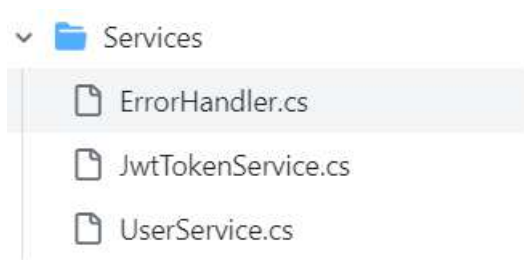


Рис 3.14 Папка Services

ErrorHandler є middleware, який перехоплює винятки, що виникають під час обробки HTTP-запитів. Цей сервіс забезпечує централізоване управління помилками, дозволяючи логічно та ефективно реагувати на різні типи помилок.

Коли виникає виняток у коді, обробник помилок визначає тип винятку і на основі цього встановлює відповідний HTTP статус-код у відповіді. Наприклад, якщо виникає ApplicationException, повертається статус-код 400 (BadRequest), що вказує на помилку в запиті, надісланому користувачем. У разі NotFoundException, повертається статус-код 404 (NotFound), що означає, що запитаний ресурс не знайдено. У випадку UnauthorizedAccessException, повертається статус-код 401 (Unauthorized), що сигналізує про відсутність прав доступу користувача. І нарешті, для всіх інших винятків за замовчуванням встановлюється статус-код 500 (InternalServerError), що вказує на непередбачену помилку на сервері.

JSON-відповідь з помилкою формується і відправляється клієнту, включаючи повідомлення про помилку. Це дозволяє клієнтам API легко розпізнавати та обробляти помилки на своїй стороні.

JwtTokenService є сервісом, який відповідає за генерацію JSON Web Token (JWT), який використовується для аутентифікації та авторизації користувачів у веб-додатку. Цей сервіс реалізує інтерфейс IJwtTokenService, що

забезпечує абстракцію та можливість легкого внесення змін у майбутньому без необхідності змінювати велику кількість коду.

Метод `GetToken` приймає список клеймів (`claims`), які представляють різні атрибути користувача (наприклад, його ID, ролі тощо) та використовуються при генерації токена. Ці клейми включаються до токена як його вміст, що дозволяє серверу перевіряти та використовувати їх при подальших запитах від користувача.

Сервіс використовує параметри з конфігурації (`_configuration`), такі як `"JWT:Secret"` для визначення ключа, який використовується для підпису токена, а також `"JWT:ValidIssuer"` та `"JWT:ValidAudience"` для встановлення відповідного видавця та аудиторії токена. Це важливо для забезпечення того, що токен може бути перевірений та визнаний дійсним лише призначеним сервером чи сервісами.

Токен підписується за допомогою алгоритму `HmacSha256`, що забезпечує безпеку його вмісту, і встановлюється час його дії (у цьому випадку на 6 годин з моменту створення). Після того як токен створено, він повертається викликаючому коду, який може використовувати його для надсилання в HTTP-відповіді користувачу, який успішно пройшов процес аутентифікації. Користувач, в свою чергу, повинен використовувати цей токен при відправленні запитів до сервера, щоб довести свою аутентичність.

`UserService` є сервісом, який надає інформацію про користувача у поточному HTTP-контексті. Цей сервіс реалізує інтерфейс `IUserService`, що дозволяє отримувати ідентифікатор користувача з клеймів, що знаходяться в токені, який користувач надіслав із запитом.

Сервіс використовує `IHttpContextAccessor` для доступу до поточного `HttpContext`, який представляє HTTP-запит. Це дозволяє сервісу читати клейми, що були встановлені під час аутентифікації користувача. Зокрема, метод `GetUserId` шукає клейм із типом `"UserId"`.

Якщо контекст не null і відповідний клейм знайдено, метод повертає значення цього клейму як рядок. Це значення потім може бути використано для ідентифікації користувача в рамках додатку, наприклад, для визначення прав користувача, доступу до його даних, чи виконання дій від його імені.

Використання сервісу, який абстрагує логіку отримання інформації про користувача, робить код чистішим та легшим для тестування, оскільки його можна легко мокувати або замінювати під час юніт-тестування.

Persistence layer(рис 3.15) є складовою в архітектурі програмного забезпечення, яка відповідає за зберігання та відновлення даних. У даному проєкті вона поділена на дві частини: Context та Migrations.

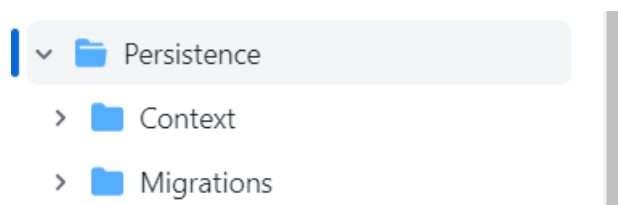


Рис 3.15 Persistence layer

У папці Context вашого проєкту реалізовано контекст бази даних та визначено конфігурації для різних сутностей.

Клас ApplicationDbContext(рис 3.16), який походить від IdentityDbContext, служить для взаємодії з базою даних через Entity Framework. Він включає в себе DbSet для кожного типу сутності, які дозволяють виконувати CRUD-операції на відповідних таблицях бази даних. Також цей клас визначає метод SaveChanges, який асинхронно зберігає всі зміни, внесені у контексті.

У методі OnModelCreating класу ApplicationDbContext викликаються методи ApplyConfiguration для кожної конфігурації, що дозволяє Entity Framework знати про визначення таблиць та їх відносини. Також тут відбувається встановлення перелічуваного типу за допомогою NpgsqlConnection.GlobalTypeMapper.MapEnum<Status>(), що дозволяє використовувати перелічувані типи в базі даних PostgreSQL.

```

public class ApplicationDbContext : IdentityDbContext<ApplicationUser>, IApplicationDbContext
{
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) : base(options)
    {
        Database.EnsureCreated();
    }

    static ApplicationDbContext()
    {
        NpgsqlConnection.GlobalTypeMapper.MapEnum<Status>();
    }

    protected override void OnModelCreating(ModelBuilder builder)
    {
        builder.ApplyConfiguration(new ApplicationUserConfiguration());
        builder.ApplyConfiguration(new ProjectConfiguration());
        builder.ApplyConfiguration(new TierConfiguration());
        builder.ApplyConfiguration(new UserProjectConfiguration());
        builder.HasPostgresEnum<Status>();
        base.OnModelCreating(builder);
    }

    public async Task<int> SaveChanges()
    {
        return await base.SaveChangesAsync();
    }

    public DbSet<Project> Projects { get; set; }
    public DbSet<Tier> Tiers { get; set; }
    public DbSet<ProjectUser> ProjectUsers { get; set; }
    public DbSet<Image> Images { get; set; }
}

```

Рис 3.16 клас ApplicationDbContext

Конфігураційні класи в Entity Framework Core дозволяють вказати, як моделі сутностей мають бути відображені в базі даних. Детальніше розглянемо приклад конфігурації UserProjectConfiguration, яка стосується відносин між користувачами та проектами.

Клас UserProjectConfiguration (рис 3.17) реалізує інтерфейс IEntityConfiguration з загальним типом ProjectUser, що представляє зв'язок багато-до-багатьох між користувачами та проектами.

Метод Configure визначає, як сутність ProjectUser повинна бути відображена в базі даних:

- builder.ToTable("ProjectUsers") вказує, що для цієї сутності має бути створена таблиця з назвою ProjectUsers.

- `builder.HasKey(up => new { up.UserId, up.ProjectId })` встановлює композитний первинний ключ, який складається з `UserId` та `ProjectId`, що унікально ідентифікує кожен запис в таблиці.
- `builder.HasOne(up => up.User).WithMany(u => u.SupportedProjects)` визначає відношення один-до-багатьох між `ProjectUser` і `User`, де один користувач може підтримати багато проектів.
- `HasForeignKey(up => up.UserId)` вказує, що стовпець `UserId` в таблиці `ProjectUsers` є зовнішнім ключем, який зв'язується з первинним ключем сутності `User`.
- `.OnDelete(DeleteBehavior.NoAction)` означає, що при видаленні користувача або проекту, записи в таблиці `ProjectUsers` не будуть автоматично видалятися. Це запобігає каскадному видаленню.

Аналогічно, `builder.HasOne(up => up.Project).WithMany(p => p.Supporters)` визначає відношення один-до-багатьох між `ProjectUser` і `Project`, де один проект може мати багато підтримуючих користувачів.

Ці конфігурації дозволяють Entity Framework Core розуміти структуру бази даних і як сутності пов'язані одна з одною, що є ключовим для правильного відображення об'єктно-орієнтованої моделі у додатку на структуру реляційної бази даних.

```

internal class UserProjectConfiguration : IEntityTypeConfiguration<ProjectUser>
{
    public void Configure(EntityTypeBuilder<ProjectUser> builder)
    {
        builder.ToTable("ProjectUsers");

        builder.HasKey(up => new { up.UserId, up.ProjectId });

        builder.HasOne(up => up.User)
            .WithMany(u => u.SupportedProjects)
            .HasForeignKey(up => up.UserId)
            .OnDelete(DeleteBehavior.NoAction);

        builder.HasOne(up => up.Project)
            .WithMany(p => p.Supporters)
            .HasForeignKey(up => up.ProjectId)
            .OnDelete(DeleteBehavior.NoAction);
    }
}

```

Рис 3.17 Клас UserProjectConfiguration

В цьому проекті використовується PostgreSQL як систему управління базами даних. Зображення(Рис 3.18) показує схему бази даних, яка представляє структуру таблиць і зв'язки між ними в проекті

Ось кілька ключових елементів, які можна виділити з цієї схеми:

Таблиці:

- Tiers: Здається, ця таблиця зберігає різні рівні пожертвувань або "винагороди" для проектів, з полями, такими як назва рівня, необхідна сума грошей, переваги, і чи була досягнута мета.
- Projects: Основна таблиця для проектів, яка зберігає інформацію про кожен проект, включаючи назву, опис, зображення, статус (наприклад, Чернетка, Опублікований), дати початку і завершення, необхідні і зібрані кошти.
- AspNetUsers та AspNetRoles: Стандартні таблиці для ідентифікації користувачів та ролей в системі, що зазвичай використовуються в проектах ASP.NET Core Identity.

- Image: Таблиця, яка зберігає зображення, пов'язані з проектами.

Зв'язки:

- Між таблицями Projects і Tiers існує зв'язок один-до-багатьох, де один проект може мати багато рівнів.
- Таблиця Image здається пов'язаною з Projects через зовнішній ключ, що дозволяє кожному проекту мати пов'язане зображення.
- Таблиця ProjectUsers (не показана на схемі, але ймовірно існує) створює зв'язок багато-до-багатьох між AspNetUsers і Projects, дозволяючи користувачам підтримувати різні проекти.

Індекси і первинні ключі:

- Кожна таблиця має первинний ключ (PK), зазвичай з ідентифікатором Id, який унікально ідентифікує записи в таблиці.
- Зовнішні ключі (FK) використовуються для встановлення зв'язків між таблицями.

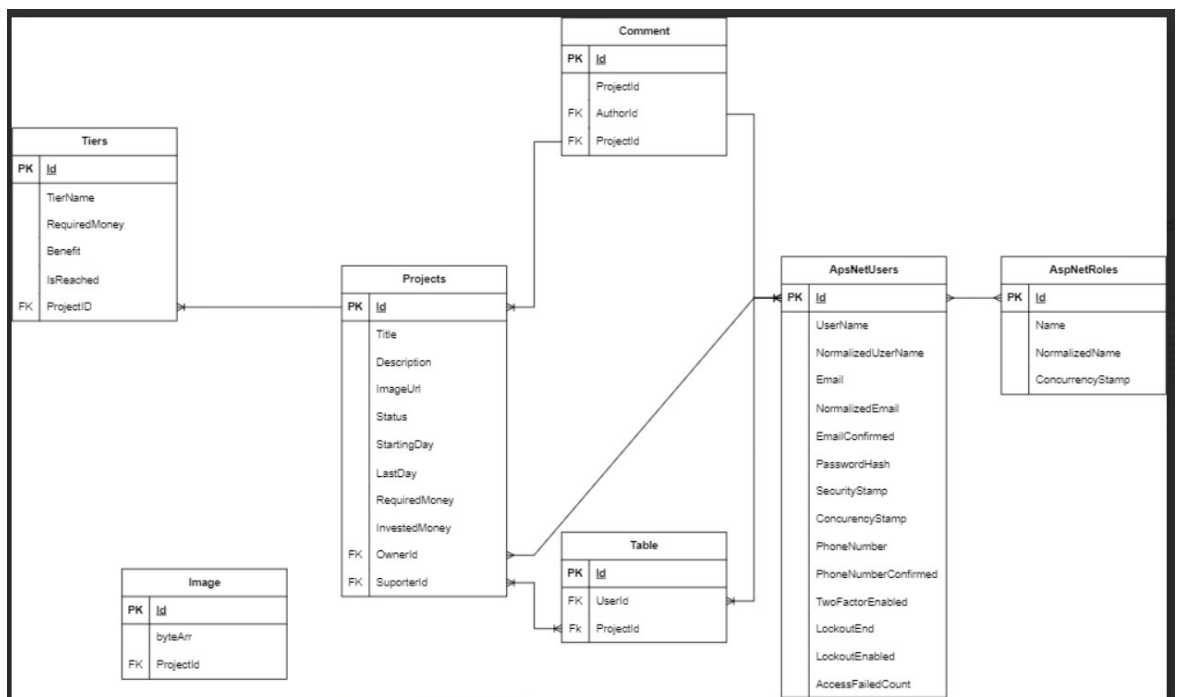
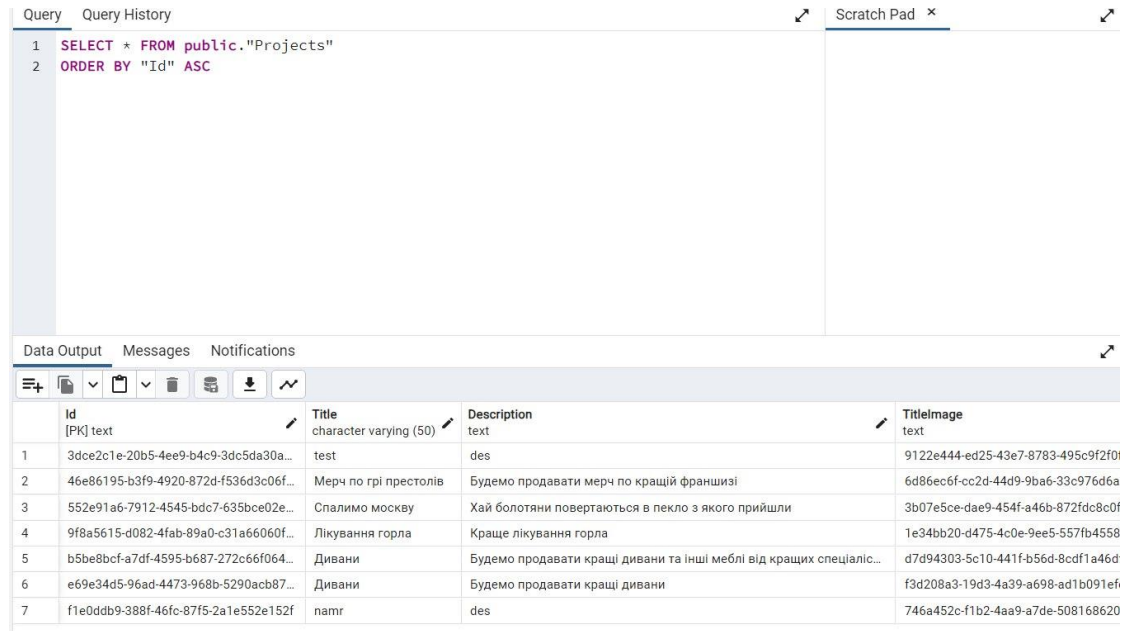


Рис 3.18 схема бази даних

Далі наведено приклад запиту до бази даних PostgreSQL за допомогою інструменту pgAdmin 4(рис 3.19). За допомогою цього інструменту було

виконано SQL-запит для вибірки даних з таблиці "Projects", яка знаходиться в публічній схемі даної бази даних. В результаті запиту отримано відсортований список проектів, організованих за ідентифікатором у порядку зростання, що ілюструє структуру та дані, які зберігаються в базі даних.



The screenshot shows a database query interface. At the top, there's a 'Query' tab with a SQL query: `1 SELECT * FROM public."Projects"` and `2 ORDER BY "Id" ASC`. Below the query, there's a 'Data Output' tab showing the results of the query. The results are displayed in a table with four columns: 'Id', 'Title', 'Description', and 'TitleImage'. The table contains 7 rows of data.

	Id [PK] text	Title character varying (50)	Description text	TitleImage text
1	3dce2c1e-20b5-4ee9-b4c9-3dc5da30a...	test	des	9122e444-ed25-43e7-8783-495c9f2f0...
2	46e86195-b3f9-4920-872d-f536d3c06f...	Мерч по гри престолів	Будемо продавати мерч по кращій франшизі	6d86ec6f-cc2d-44d9-9ba6-33c976d6a...
3	552e91a6-7912-4545-bdc7-635bce02e...	Спалимо москву	Хай болотяни повертаються в пекло з якого прийшли	3b07e5ce-dae9-454f-a46b-872fdc8c0f...
4	9f8a5615-d082-4fab-89a0-c31a66060f...	Лікування горла	Краще лікування горла	1e34bb20-d475-4c0e-9ee5-557fb4558...
5	b5be8bcf-a7df-4595-b687-272c66f064...	Дивани	Будемо продавати кращі дивани та інші меблі від кращих спеціаліс...	d7d94303-5c10-441f-b56d-8cdf1a46d...
6	e69e34d5-96ad-4473-968b-5290acb87...	Дивани	Будемо продавати кращі дивани	f3d208a3-19d3-4a39-a698-ad1b091ef...
7	f1e0ddb9-388f-46fc-87f5-2a1e552e152f	namr	des	746a452c-f1b2-4aa9-a7de-508168620...

Рис 3.19 запит до бази даних

Запит виконується для таблиці Projects, яка знаходиться у схемі public. Команда `SELECT *` забирає всі колонки для кожного рядка в таблиці. `ORDER BY "Id" ASC` означає, що результати будуть впорядковані за зростанням згідно зі значеннями в колонці Id.

Далі представлено структуру каталогу фронтенд-частини проекту(рис 3.20), яка використовує Angular як фреймворк. Angular проекти мають властиву структуру, що допомагає організувати код та ресурси.

- Каталог `node_modules` містить бібліотеки JavaScript, які встановлюються за допомогою `npm` (Node Package Manager).
- Каталог `.angular` зберігає конфігураційні файли для Angular CLI.
- Каталог `src` є основним каталогом для коду додатка, стилів та активів.

- Всередині `src`, каталог `app` містить компоненти, сервіси та інші класи Angular.
- Каталог `assets` використовується для зберігання статичних ресурсів, таких як зображення, файли JSON та інші активи.
- Папка `environments` містить конфігураційні файли, які можуть змінюватися в залежності від середовища розгортання (розробка, тестування, виробництво).
- Файл `index.html` є основною HTML-сторінкою, яка завантажується в браузері.
- Файл `main.ts` є точкою входу для Angular додатка, де ініціалізується основний модуль.
- Файл `styles.css` містить глобальні стилі для всього додатка.
- Конфігураційні файли, такі як `.editorconfig`, `.gitignore`, `angular.json`, `package.json`, і `tsconfig.json` файли, використовуються для налаштування редактора, ігнорування певних файлів в git, конфігурування Angular CLI та TypeScript відповідно.

Ця структура забезпечує чітке розділення відповідальностей і допомагає розробникам легко орієнтуватися в проекті, створюючи масштабоване та легко підтримуване додаток.

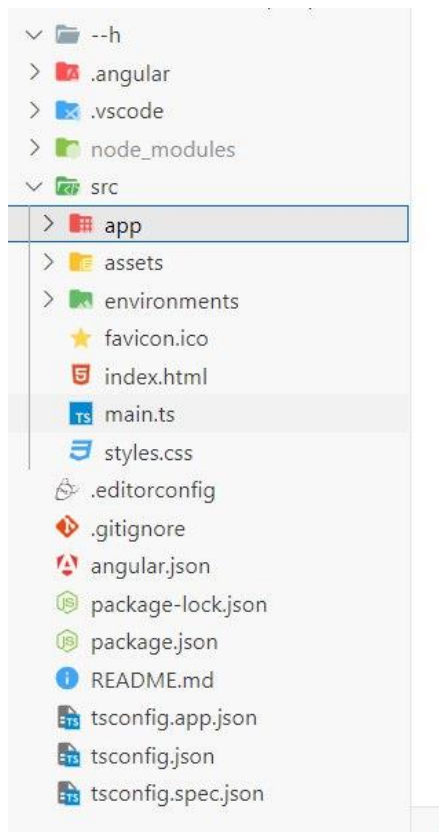


Рис 3.20 стандартна angular архітектура

Проект на Angular має структурований підхід, що сприяє організації коду та спрощує розуміння його функцій:

- `_errors`: Ця папка містить компоненти для обробки помилок, такі як сторінки `not-found` та `server-error`. Це дозволяє централізовано управляти помилками у фронтенд-частині застосунку.
- `_models`: Тут розташовані TypeScript інтерфейси, які визначають типи даних, з якими працює фронтенд, що дозволяє мати сильно типізовані моделі для більшої надійності та легшої інтеграції з бекендом.
- `_projects`: Включає всі сторінки та компоненти, пов'язані з проектами, і являє собою основну частину фронтенду застосунку.
- `_services`: В цій папці розташовані сервіси, які забезпечують зв'язок з бекендом та обробляють логіку аутентифікації (`auth-interceptor`), управління токенами (`token-interceptor`, `token-storage`) та взаємодію з даними проектів (`get-projects.service`).

- `_state-management`: Це реалізація стейт менеджменту, який управляє станом застосунку через NgRx в екосистемі Angular.
- `timer`: Компонент, який відповідає за відлік часу використовується для візуалізації часу до завершення проектів.
- `scripts`: Містить автоматично генеровані TypeScript інтерфейси та методи за допомогою бібліотеки `Reinforced.Typings`, що забезпечує синхронізацію між бекенд-моделями та клієнтськими типами даних.
- `nav`: Навігаційний компонент, який відображається на всіх сторінках та дозволяє користувачам переходити між різними частинами застосунку.

3.3 Порівняльний аналіз проектів на основі програмної реалізації

З огляду на надану інформацію про обидва проекти, можна визначити декілька ключових відмінностей у реалізації патернів, архітектурі та потенційних перспективах обох:

Архітектура

Telegram Bot Project:

- Використовує підход `.NET Core` для створення бота, який, імовірно, орієнтований на події і повідомлення.
- Може бути зосередженим на монолітній архітектурі, де всі компоненти бота тісно інтегровані.

Crowdfunding Platform:

- Використовує чисту архітектуру і DDD (Domain-Driven Design), яка підтримує високий рівень абстракції і модульності, що полегшує масштабування і підтримку.
- Бекенд і фронтенд чітко розділені, використовуючи API-фірст підхід.

Реалізація патернів

Telegram Bot Project:

Явні шаблони проектування відсутні.

Crowdfunding Platform:

- Явно використовує CQRS (Command Query Responsibility Segregation) та Mediator патерни для розділення команд і запитів, а також для зменшення залежностей між компонентами.
- Використовує AutoMapper для мапінгу даних між об'єктами, що є прикладом патерну Adapter.

Перспективи та Масштабування

Telegram Bot Project:

- Простіше масштабувати для розширення функціональності бота через додавання нових команд і обробників.
- Може виникнути потреба в рефакторингу архітектури при значному розширенні проекту для підтримки нових сервісів чи інтеграції.

Crowdfunding Platform:

- Архітектура готова до масштабування і підтримки завдяки розділенню логіки на домени і служби.
- Впровадження стейт-менеджменту на фронтенді підтримує складні користувацькі інтерфейси і динамічні взаємодії.
- Явно використовує CQRS (Command Query Responsibility Segregation) та Mediator патерни для розділення команд і запитів, а також для зменшення залежностей між компонентами.
- Використовує AutoMapper для мапінгу даних між об'єктами, що є прикладом патерну Adapter.

Підтримка та Розвиток

Telegram Bot Project:

- Може бути більш простим у підтримці при невеликому розмірі проекту.
- Зміни в одній частині системи можуть вплинути на інші через високу ступінь зв'язаності.

Crowdfunding Platform:

- Висока модульність і розділення відповідальності спрощують внесення змін і додають нову функціональність.
- Застосування DDD сприяє більш чіткому визначенню бізнес-логіки і правил домену, що спрощує розвиток і впровадження змін.

Висновки до розділу 3

Обидва проекти використовують сучасні підходи до розробки ПЗ, але Crowdfunding Platform є більш гнучким і масштабованим за рахунок чіткого розділення концернів і впровадження сучасних архітектурних патернів. Telegram Bot Project може бути швидшим до розробки та впровадження, проте може виявитися менш гнучким при необхідності значного розширення.

РОЗДІЛ 4 ОХОРОНА ПРАЦІ

4.1 Нормативна база в галузі охорони праці при роботі з екранними пристроями

Екранний пристрій – електронний засіб для відтворення будь-якої графічної або буквено-цифрової інформації (на основі електронно-променевих трубок, рідких кристалів, плазми, проєкцій, органічних світлодіодних дисплеїв та інших нових розробок у сфері інформаційних технологій) [12].

Мінімальні вимоги до безпеки та охорони здоров'я при використанні екранних пристроїв усіх типів і моделей установлюють вимоги безпеки та охорони здоров'я працівників при використанні екранних пристроїв, затверджені наказом Мінсоцполітики від 14.02.2018 р. № 207. всім суб'єктам господарювання, незалежно від форм власності, організаційно-правової форми та діяльності, та встановлюють мінімальні вимоги безпеки та охорони праці до робіт, пов'язаних із використанням екранних пристроїв, незалежно від їх типу та моделі.

4.2. Загальні вимоги до роботодавців

Нижче наведено частину вимог з документу про затвердження Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями:

Зокрема, роботодавець зобов'язаний:

- організовувати роботу таким чином, щоб забезпечити безпечні та гігієнічні умови праці,
- забезпечити дотримання положень та правил захисту праці та безпеки на робочому місці, видавати інструкції щодо усунення недоліків у цьому плані та відстеження виконання цих інструкцій,
- реагувати на потреби забезпечення охорони здоров'я та безпеки праці та адаптації заходів, що вживаються для підвищення існуючого рівня охорони здоров'я та життя працівників, враховуючи зміни умов роботи,

- забезпечити розробку послідовної політики щодо запобігання нещасних випадків у виробничих та професійних захворюваннях, з урахуванням технічних питань, організації праці, умов праці, соціальних відносин та впливу факторів робочого середовища,
- враховувати захист здоров'я підлітків, вагітних або годувальних працівників та інвалідів у рамках профілактичних заходів,
- забезпечити виконання замовлень, заяви, рішень та наказів керівних органів,
- забезпечити виконання рекомендацій інспектора соціальної праці.

4.3 Вимоги до приміщення з використанням екранних пристроїв

Робочі місця працівників, які використовують екранні пристрої, повинні відповідати ергономічним, антропологічним, психофізіологічним вимогам, характеру виконуваної роботи. Вони повинні бути сконструйовані та розраховані таким чином, щоб працівники мали можливість змінювати своє робоче положення та рухи. Освітлення робочого місця повинно створювати відповідний контраст між екраном і навколишнім середовищем, враховуючи вид роботи та відповідаючи ДСанПіН 3.3.2.007-98. Мікроклімат промислових об'єктів та робочих місць працівників із екранним обладнанням необхідно підтримувати на постійному рівні відповідно до ДСН 3.3.6.042-99 «Гігієнічні норми мікроклімату на промислових об'єктах», затверджених МОЗ та Головним державним санітарним лікарем України. від 01.12.1999 № 42 . Робочі місця з екранними пристроями не повинні обмежувати пересування працівників [13].

Робочі місця з використанням екранних пристроїв повинні відповідати ергономічним, антропологічним, психофізіологічним вимогам і характеру виконуваної роботи. Вони повинні бути розроблені та розраховані таким чином, щоб працівники мали можливість змінювати свої робочі положення та рухи. Освітлення робочого місця повинно створювати відповідний контраст

між екраном і навколишнім середовищем, враховуючи вид роботи та відповідаючи ДСанПіН 3.3.2.007-98. Відповідно до ДСН 3.3.6.042-99 «Стандарт мікроклімату гігієни промислових об'єктів», затвердженого МОЗ та державного головного санітарного лікаря, мікроклімат промислових об'єктів і робочих місць працівників має підтримуватися на постійному рівні України. Від 01.12.1999 № 42. Робочі місця з екранними пристроями не повинні обмежувати пересування працівників.

Крім цього для зменшення впливу негативних факторів на працівника, виробниче приміщення повинно відповідати нормам щодо параметрів мікроклімату, освітлення, шуму та вібрації, рівні електромагнітного та іонізуючого випромінювання.

По-перше, на усіх робочих місцях із електронно-обчислювальними машинами обов'язково має бути забезпечено оптимальні значення параметрів температури відносної вологості й рухливості повітря (ГОСТ 12.1.005-88, СН 4088-86) (табл. 4.1).

Таблиця 4.1 – Нормовані величини температури, відносної вологості та швидкості руху в робочій зоні виробничих приміщень з ВДТ

Пора року	Категорія робіт	Оптимальна температура повітря, °С, не більше	Оптимальна відносна вологість повітря, %	Оптимальна швидкість руху повітря, м/с
Холодна	Легка – Іа	22-24	40-60	0,1
	Легка – Іб	21-23	40-60	0,1
Тепла	Легка – Іа	23-25	40-60	0,1
	Легка – Іб	22-24	40-60	0,2

До категорії робіт Іа належать, що виконується робота сидячі і не потребує фізичного напруження людини, до категорії робіт Іб належать, що робота виконується сидячі, стоячи або пов'язані з ходінням та потребує деякого фізичного напруження.

По-друге, показники рівню позитивних та негативних іонів в повітрі приміщень з ВДТ мають відповідати санітарно – гігієнічним нормам №2152-80 (табл. 4.2).

Таблиця 4.2 – Рівні іонізації повітря приміщень при роботі на ВДТ

Рівні	Кількість іонів в 1 см ³ повітря	
	n+	n-
Мінімально необхідний	400	600
Оптимальний	1500-3000	3000-5000
Максимально допустимий	50000	50000

Вимірювання кількості іонів та його полярності порядку поточного нагляду проводиться 1 разів у квартал. Вимірювання проводяться також у випадках:

- встановлення нових або відремонтованих іонізаторів,
- організації нових робочих місць,
- впровадження нових технологічних процесів, що потенційно

можуть. змінити іонний режим у зоні дихання персоналу [14].

Все необхідне обладнання, необхідне для роботи на комп'ютері, повинно бути в межах досяжності працівника, не змушуючи його змінити місце роботи.

4.4 Вимоги до роботи з екранними пристроями

На сьогодні існує ряд вимог, які повинні використовувати працівники на підприємствах, основним місцем роботи яких є місце за персональним комп'ютером. Ці вимоги можна поділити за наступними критеріями:

Ергономіка положення:

- робоче місце повинно бути достатньо великим, щоб вільно використовувати всі елементи з ручним управлінням у межах світла;
- відстань між моніторами повинна бути 60 см, а між працівником та задньою частиною іншого сусіднього монітора - 80 см;
- відстань між працівником та монітором повинна бути від 40 см до 75 см;
- для кожного працівника має бути 2 м² вільної площі та 13 м² обсягу приміщення, без обмежень на меблі, обладнання тощо;
- екран монітора повинен відповідати певним вимогам: написи на екрані повинні бути чіткими та розбірливими, зображення повинно бути стабільним, екран повинен бути охоплений просвітлюючим шаром або оснащеним відповідним фільтром тощо;
- клавіатура комп'ютера повинна бути не менше 10 см від краю таблиці і повинна бути окремим елементом основного обладнання робочого місця, його поверхня повинна бути матовою, а символи контрастні та вибіркові.

Крісло як елемент робочого місця біля екрану монітора:

- має бути стійким з як мінімум 5 опорною базою, на колесах з можливістю обертання на 360° навколо вертикальної осі;
- розміри спинки та сидіння повинні забезпечувати зручне положення тіла та свободу рухів;
- з можливістю регулювання висоти сидіння в діапазоні 40-50 см, рахуючи від підлоги;
- з можливістю регулювання висоти спинки та регулювання нахилу спинки в діапазоні 5° вперед і 30° назад;
- сидіння та спинка мають бути сконструйовані таким чином, щоб відповідати природному вигину хребта та стегна;
- повинні бути обладнані підлокітниками;

- за бажанням працівника, а також у випадках, коли висота крісла унеможливилює розміщення працівника ступнями на підлозі, робоче місце має бути обладнане підставкою для ніг.

Розміри та налаштування стола:

- висота столу повинна забезпечувати вільне положення працівника, зберігаючи прямий кут між плечем і передпліччям під час роботи за комп'ютером;
- висота столу, на якому розташований екран монітора, повинна забезпечувати відповідний кут огляду екрана монітора в діапазоні 20-50° вниз (враховуючи від горизонтальної лінії на рівні очей співробітника до лінії, проведеної від очима до центру екрана), верхнім краєм екран монітора не повинен бути над очима працівника;
- коліна працівника, що сидить за столом, не повинні торкатися стільниці, простір під столом має бути вільним;
- поверхня стільниці повинна бути матовою, бажано світлого кольору.

4.5. Вимоги щодо режиму відпочинку та праці на підприємствах з екранними пристроями

Обов'язок роботодавця – організувати роботу так, щоб кожен працівник після кожної години безперервної роботи за комп'ютером мав можливість змінити вид роботи на такий, який не обтяжуватиме зір або буде виконуватися в зміненому положенні тіла. Якщо роботодавець не може забезпечити зміну роботи, він повинен забезпечити перерву не менше 5 хвилин на кожну годину роботи перед екраном. Така перерва зараховується до робочого часу працівника і не зменшує оплату праці.

При виконанні протягом дня робіт, що належать до різних видів трудової діяльності, за основну роботу з ВДТ ЕОМ і ПЕОМ слід вважати таку, що займає не менше 50% часу впродовж робочої зміни мають передбачатися:

- перерви для відпочинку і вживання їжі (обідні перерви);
- перерви для відпочинку і особистих потреб (згідно з трудовими нормами);
- додаткові перерви, що вводяться для окремих професій з урахуванням особливостей трудової діяльності;
- Встановлюються такі внутрішньо змінні режими праці та відпочинку при роботі з ЕОМ при 8-годинній денній робочій зміні в залежності від характеру праці;
- для розробників програм із застосуванням ЕОМ, слід призначати регламентовану перерву для відпочинку тривалістю 15 хвилин через кожну годину роботи за ВДТ;
- для операторів із застосування ЕОМ, слід призначати регламентовані перерви для відпочинку тривалістю 15 хвилин через кожні дві години;

Висновки до розділу 4

У ході виконання теми охорони праці проведено розслідування щодо організації праці працівників на підприємствах і в установах, а також щодо правил роботи з електронно-обчислювальними приладами. ІТ-професія з кожним роком стає все більш важливою: програмісти, дизайнери, тестувальники, розробники баз даних, менеджери проектів — лише деякі з тих професіоналів, які потребують бізнесу як ніколи. Зростання попиту на таких спеціалістів було очевидним і до глобальної пандемії коронавірусу, але реальне життя фактично призвело до дефіциту людей, які можуть виконувати таку роботу на високому рівні. Люди такого виду діяльності проводять багато часу з екранними пристроями у сидячому положенні і тому охорона праці також важлива в ІТ.

За допомогою проведення цієї роботи були освоєні права та обов'язки робітників та роботодавців. Також було детально розглянуто умови середовища, у яких працівники мають виконувати свої професійні обов'язки.

РОЗДІЛ 5 ОХОРОНА НАВКОЛИШНЬОГО СЕРЕДОВИЩА

5.1 Вступ

В останні роки питання охорони навколишнього середовища набуло особливої актуальності. Зростання промисловості та інтенсивне використання природних ресурсів призводить до збільшення навантаження на екологію, що вимагає відповідних заходів та інвестицій для забезпечення сталого розвитку та охорони довкілля. Протягом наступних 5-10 років перед Україною постає завдання не тільки підтримання, але й поліпшення екологічного стану країни в умовах обмежених фінансових ресурсів.

Важливо визначити ключові напрями та завдання, що спрямовані на розробку реалістичних, ефективних та економічно вигідних рішень, які б враховували поточний екологічний стан та потреби країни. До основних критеріїв та факторів, що впливають на екологічну політику, слід віднести:

- Забруднення середовища та його вплив на здоров'я населення, що вимагає вжиття термінових заходів для зниження рівня забруднення повітря, води та ґрунтів.
- Економічні втрати, пов'язані зі зниженням якості природних ресурсів та впливом на фізичний капітал, включаючи погіршення умов для сільськогосподарської та індустріальної діяльності.
- Загрози біологічному різноманіттю, що вимагають зусиль зі збереження природних ландшафтів та видів, а також розробки та впровадження екологічно стійких методів господарювання.

Таким чином, охорона навколишнього середовища в Україні вимагає комплексного підходу, що поєднує економічну ефективність, екологічну безпеку та соціальну відповідальність.

5.2 Завдання охорони навколишнього середовища

Важливо визначити конкретні завдання та дії, що спрямовані на захист та збереження природи, а також на розвиток сталих екологічних практик у різних секторах економіки та соціального життя. Основні завдання, які необхідно реалізувати в рамках охорони навколишнього середовища, включають наступні пункти:

Мінімізація Рівня Радіаційного Забруднення:

Здійснення заходів щодо зменшення впливу радіаційного забруднення на навколишнє середовище та здоров'я людей.

Охорона Атмосфери:

Впровадження ефективних заходів для запобігання та зменшення забруднення повітря, особливо в містах і промислових районах.

Захист Земельних Ресурсів:

Збереження та раціональне використання земель, запобігання їх забрудненню та виснаженню.

Збереження Природних Ландшафтів:

Розширення територій з природним ландшафтом та підвищення рівня природоохоронної діяльності.

Охорона Лісів:

Підвищення стійкості лісів та їх екологічних функцій.

Управління Відходами:

Ефективне знешкодження, утилізація та захоронення промислових та побутових відходів.

Охорона Водних Ресурсів:

Запобігання забрудненню водойм, охорона підземних вод.

Збереження Малих Річок:

Розвиток системи управління водними ресурсами, збереження малих річок.

Моніторинг Довкілля:

Створення та завершення державної системи моніторингу навколишнього середовища.

Прогнозування та Реагування на Надзвичайні Ситуації:

Розробка системи прогнозування та оперативних дій у разі надзвичайних екологічних ситуацій.

Екологічний Супровід Конверсії:

Забезпечення екологічного супроводу процесу конверсії військово-промислового комплексу.

Екологічний Контроль Збройних Сил:

Здійснення заходів щодо екологічного контролю за діяльністю Збройних Сил.

Розробка Схем Природокористування:

Впровадження механізмів реалізації схем природокористування.

Економічний Вплив на Систему Природокористування:

Впровадження економічних інструментів для оптимізації використання природних ресурсів.

Екологічна Освіта:

Розвиток системи екологічної освіти та інформування громадськості.

Ці завдання та заходи спрямовані на створення балансу між економічним розвитком та охороною навколишнього середовища, забезпечуючи стале та здорове майбутнє для наступних поколінь.

5.3 Проблеми

У розділі, присвяченому проблемам охорони навколишнього середовища, слід виділити основні виклики, з якими стикаються ІТ-проекти, а також загальні екологічні проблеми, які впливають на розвиток технологічної галузі. Основні проблеми можуть включати:

Енергоспоживання та Вуглецевий Відбиток:

- Високе енергоспоживання інформаційних технологій та великий вуглецевий відбиток, пов'язаний з цим.
- Необхідність зменшення енергоспоживання та перехід на відновлювані джерела енергії.

Виробництво та Утилізація Електронного Обладнання:

- Проблеми, пов'язані з виробництвом електронного обладнання, включаючи використання токсичних матеріалів.
- Необхідність ефективної утилізації та переробки електронних відходів.

Вплив на Біорізноманіття та Екосистеми:

- Негативний вплив технологій на природні екосистеми та біорізноманіття.
- Забруднення навколишнього середовища хімічними речовинами та відходами.

Проблеми Водного Ресурсу:

- Велике споживання води при виробництві технологічної продукції.
- Поліпшення систем управління водними ресурсами та їх раціональне використання.

Зміни Клімату та Їх Вплив:

- Вплив змін клімату на стабільність та ефективність ІТ-інфраструктури.
- Розробка адаптивних стратегій для мінімізації негативного впливу.

Екологічна Освіта та Свідомість:

- Розвиток екологічної освіти та підвищення свідомості працівників ІТ-сфери.
- Промоція сталого використання та екологічно відповідального підходу у розробці та використанні технологій.

Ці проблеми потребують комплексного підходу та розробки ефективних стратегій для їх вирішення. Залучення громадськості, влади, наукових та промислових організацій є ключовим для досягнення екологічно стійкого розвитку в ІТ-галузі.

5.4 Висновки до розділу 4

У сфері охорони довкілля важливою є реалізація комплексних стратегій та системних заходів на всіх рівнях промисловості, щоб забезпечити стале та збалансоване використання природних ресурсів. Розв'язання екологічних проблем в Україні вимагає інтеграції екологічних норм у різні галузі промисловості, включаючи металургію, хімічну та нафтохімічну промисловість, будівництво та машинобудування.

Запропоновані заходи для ефективного розв'язання екологічних проблем включають проведення екологічного аудиту, впровадження новітніх технологій, ресурсозбереження, розробку та впровадження альтернативних матеріалів та джерел енергії. Важливими напрямками є також заходи зі зменшення викидів і скидань, забезпечення екологічної безпеки та збереження біорізноманіття.

Окрема увага має бути приділена галузям, які найбільше впливають на довкілля, таким як цементна промисловість. Реалізація програми для зменшення їх впливу передбачає мінімізацію викидів, впровадження енергоефективних технологій та розробку екологічно чистих методів виробництва.

Загалом, вирішення екологічних проблем в Україні потребує комплексного та цілеспрямованого підходу, що включає забезпечення екологічної безпеки, збереження природних ресурсів та розвиток сталої промисловості. Важливо координувати зусилля на різних рівнях – міждержавному, державному та місцевому – для реалізації визначених екологічних цілей та досягнення сталого розвитку.

РОЗДІЛ 6 РОЗРАХУНОК І ОЦІНКА ЕКОНОМІЧНОГО ЕФЕКТУ ВІД РОЗРОБКИ

Під час виконання дипломної роботи необхідно було розробити два застосунки. Розрахунок економічної ефективності дозволить обґрунтувати доцільність розробки.

6.1 Розрахунок витрат на створення й експлуатацію програми

Для більш детального розрахунку економічного ефекту від розробки двох проектів, Telegram-бота та системи краудфандингу, розглянемо наступні елементи витрат і потенційного доходу:

Проект Telegram-бот

Витрати на розробку:

1. Заробітна плата розробників: $20,000 \text{ грн/міс} * 2 \text{ міс} = 40,000 \text{ грн}$.
2. Витрати на обладнання та ПЗ: Одноразові витрати - 10,000 грн.

Експлуатаційні витрати:

1. Сервер та хостинг: $2,000 \text{ грн/міс} * 12 \text{ міс} = 24,000 \text{ грн/рік}$.
2. Технічна підтримка та оновлення: $5,000 \text{ грн/міс} * 12 \text{ міс} = 60,000 \text{ грн/рік}$.

Потенційний дохід:

1. Підписка або рекламні доходи: $5,000 \text{ грн/міс} * 12 \text{ міс} = 60,000 \text{ грн/рік}$.

Проект системи краудфандингу

Витрати на розробку:

2. Заробітна плата розробників: $25,000 \text{ грн/міс} * 6 \text{ міс} = 150,000 \text{ грн}$.
3. Витрати на обладнання та ПЗ: Одноразові витрати - 25,000 грн.

Експлуатаційні витрати:

1. Сервер та хостинг: $4,000 \text{ грн/міс} * 12 \text{ міс} = 48,000 \text{ грн/рік}$.
2. Технічна підтримка та оновлення: $10,000 \text{ грн/міс} * 12 \text{ міс} = 120,000 \text{ грн/рік}$.

Потенційний дохід:

1. Комісія з зібраних коштів: Припустимо, середній місячний дохід - 20,000 грн, що в рік складе 240,000 грн.

Загальний розрахунок витрат та доходів

Telegram-бот:

- Загальні витрати: $40,000 \text{ (зарплата)} + 10,000 \text{ (обладнання)} + 24,000 \text{ (сервер)} + 60,000 \text{ (підтримка)} = 134,000 \text{ грн}$.
- Потенційний дохід: 60,000 грн/рік.

Система краудфандингу:

- Загальні витрати: $150,000 \text{ (зарплата)} + 25,000 \text{ (обладнання)} + 48,000 \text{ (сервер)} + 120,000 \text{ (підтримка)} = 343,000 \text{ грн}$.
- Потенційний дохід: 240,000 грн/рік.

Ці розрахунки дозволяють зробити приблизну оцінку економічної ефективності обох проектів. Важливо враховувати, що потенційний дохід може

змінюватися в залежності від ринкових умов, стратегій монетизації та залучення користувачів.

6.2 Розрахунок економічної ефективності

Визначимо пряму економічну ефективність системи краудфіндингу ґрунтуючись на тім, що впровадження програмного продукту вивільняє 0,7 працівника, з урахуванням затрат на супроводження програми.

Заробітна платня 0,7 працівника в рік складає:

$$(12000 * 12) * 0,7 = 100800,00 \text{ грн.};$$

Пряма економічна ефективність впровадження розробки складе:

$$E_{\text{к}} = 100800,00 - 10280,96 - 0,6 * 84812,88 = 39631,31 \text{ грн.};$$

6.3 Розрахунок строку окупності програми

Розрахуємо за яку кількість місяців програмний продукт окупиться:

$$T_{\text{ОК}} = \Delta K / \Delta C = (39631,31) / 90519,04 = 0,43 \text{ р.}$$

Таким чином витрати на створення програмного продукту окупляться приблизно за 2 місяці експлуатації.

6.4 Висновок

Виходячи з розрахунків програма окупить себе за 4 місяці. Це вказує на доцільність розробки.

ВИСНОВКИ

Під час виконання дослідження в рамках дипломної роботи було здійснено порівняльний аналіз двох ІТ-проектів, вивчено особливості їхньої архітектури та використання архітектурних патернів. Було детально розглянуто принципи організації та структури проектів, а також технологічні підходи, які вони використовують. Зокрема, було проаналізовано впровадження патернів CQRS та Mediator в проекті краудфандингової платформи, а також оцінено вплив цих патернів на розширюваність, відмовостійкість та легкість підтримки системи. Результатом аналізу стала розробка веб-застосунка, який демонструє переваги використання сучасних патернів проектування для ефективного вирішення задач студентських стартап-проектів.

В процесі аналізу різних ІТ-проектів, заснованих на використанні архітектурних патернів, було виявлено ключові переваги та недоліки кожного підходу. Встановлено, що застосування патернів дозволяє досягти більшої гнучкості та масштабованості систем, однак може збільшити складність розробки та вимагає глибшого розуміння цих патернів. У випадку проектів з високим рівнем вимог до відмовостійкості та легкості супроводу, застосування таких патернів як CQRS та Mediator є виправданим та ефективним. Натомість, у проектах з меншими вимогами до масштабованості та управління станом, може бути застосована більш проста архітектура без розподілення на команди та квері, що спрощує розробку та підтримку. У дослідженні також було враховано специфіку краудфандингу, яка вимагає високого рівня взаємодії з користувачем та прозорості процесів. Враховуючи це, було приділено особливу увагу проектуванню користувацького інтерфейсу та взаємодії зі стейкхолдерами в обох проектах.

Дослідження показало, що вибір архітектурних патернів має значний вплив на гнучкість, масштабованість та відмовостійкість систем. В той час як монолітна архітектура, використана у проекті Telegram бота, забезпечує швидкість розробки та простоту впровадження, мікросервісна архітектура в

проекті Crowdfunding платформи пропонує кращу підтримку розширення та змін.

Використання патернів CQRS та Mediator у Crowdfunding проекті сприяє чіткій сегрегації команд і запитів, полегшуючи розробку та підтримку. Натомість у Telegram боті, простіші патерни використовувалися для конкретних задач, що забезпечує простоту та ефективність у менших масштабах.

Для проектів, що потребують швидкого розгортання та простої структури, рекомендовано використовувати підхід, схожий на Telegram бота. Для більш складних систем, де потрібно масштабування та гнучке управління змінами, підходить мікросервісна архітектура, як у Crowdfunding проекті.

Існує перспектива подальшого дослідження змішаних архітектурних підходів, які можуть поєднувати переваги обох моделей. Також актуальним залишається вивчення впливу нових технологій, таких як хмарні обчислення та штучний інтелект, на розвиток архітектурних патернів у IT-проектах.

У цьому дослідженні була досягнута поставлена мета – оцінити та порівняти вплив різних архітектурних підходів на розробку та ефективність IT-проектів. Результати дослідження надають цінні вказівки для вибору підходів до проектування в залежності від конкретних вимог та цілей розробки.

ВИКОРИСТАНІ ДЖЕРЕЛА ІНФОРМАЦІЇ

1. Modern Software Architecture Patterns: The Main Things to Know. Веб сайт: <https://ideasoft.io/blog/modern-software-architecture-patterns/> (дата звернення: 4.11.2023)
2. Anti-Patterns in Software Development That You Should Avoid. <https://medium.com/geekculture/anti-patterns-in-software-development-that-you-should-avoid-780841ce4f1d> (дата звернення: 5.11.2023)
3. Don't Use Design Patterns! <https://medium.com/the-coding-matrix/https-medium-com-the-coding-matrix-dont-use-design-patterns-35bcff59dbb5> (дата звернення: 6.11.2023)
4. C# programming guide Веб сайт: <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/> (дата звернення 7.11.2023)
5. Angular docs Офіційний сайт із документацією по ангуляру. Веб сайи: <https://angular.io/docs> (дата звернення: 8.11.2023)
6. Typescriptlang docs Офіційний сайт із документацією по тайпскрипту. Веб сайи: <https://www.typescriptlang.org/> (дата звернення: 9.11.2023)
7. Entity Framework Core Веб сайт: <https://docs.microsoft.com/en-us/ef/core/> (дата звернення 10.11.2023)
8. PostgreSQL documentation Офіційний сайт із документацією по PostgreSQL Веб сайт: <https://www.postgresql.org/docs/> (дата звернення 12.11.2023)
9. MediatR — Beyond the basics. Веб сайт: https://medium.com/@cristian_lopes/mediatr-beyond-the-basics-8ab90841a732 (дата звернення 13.11.2023)
10. Microsoft technical documentation. Офіційний сайт із документацією Майкрософт. Веб сайт: <https://docs.microsoft.com/en-us/> (дата звернення: 15.11.2023)

11. Охорона праці на підприємстві: що потрібно знати? Веб сайт: <https://te.dsp.gov.ua/ohorona-pratsi-na-pidpryyemstvi-shho-potribno-znaty/> (дата звернення 27.11.2023)
12. Охорона праці в ІТ компанії. Веб сайт: <https://legalitgroup.com/service/ohorona-praci-v-it-kompanii/> (дата звернення 27.11.2023)
13. Про затвердження Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями. Наказ: <https://zakon.rada.gov.ua/laws/show/z0508-18#Text> (дата звернення 27.11.2023)
14. Санітарно-гігієнічні норми допустимих рівнів іонізації повітря виробничих та громадських приміщень. Веб сайт: <https://dnaop.com/html/2296/doc> (дата звернення 27.11.2023)
15. Санітарні норми ультрафіолетового випромінювання у виробничих приміщеннях. Веб сайт: https://dnaop.com/html/2299/doc%D0%A1%D0%9D_4557-88 (дата звернення 27.11.2023)
16. Природне та штучне освітлення. Веб сайт: https://dnaop.com/html/45036/doc-%D0%A1%D0%9D%D0%B8%D0%9F_II-4-79_ (дата звернення 27.11.2023)
17. Санітарні норми мікроклімату виробничих приміщень. Веб сайт: https://dnaop.com/html/34094/doc%D0%94%D0%A1%D0%9D_3.3.6.042-99 (дата звернення 27.11.2023)
18. Dependency injection Веб сайт: <https://habr.com/ru/post/350068/> (дата звернення 01.12.2023)