ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка Web-платформи для пошуку розробників для спільної роботи над проектами з використанням ASP.NET Core MVC»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Артур ГАЛУЩАК
(підпис)

Виконав: здобувач вищої освіти групи ПД-42

Артур ГАЛУЩАК

Керівник: Оксана ЗОЛОТУХІНА
к.т.н. доцент

Рецензент: _____

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Галущаку Артуру Павловичу

1. Тема кваліфікаційної роботи: «Розробка Web-платформи для пошуку розробників для спільної роботи над проектами з використанням ASP.NET Core MVC»
керівник кваліфікаційної роботи к.т.н, доц., доцент кафедри ІПЗ Оксана ЗОЛОТУХІНА,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «27» лютого 2024 р. № 36.

2. Строк подання кваліфікаційної роботи «28» травня 2024 р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, документація бібліотек Asp.Net Core MVC, Entity Framework Core, C#.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Аналіз предметної
2. Огляд існуючих рішень для пошуку розробників для співпраці над проектом
3. Розробка платформи для пошуку розробників для співпраці над проектами.

5. Перелік графічного матеріалу: *презентація*

1. Мета, об’єкт та предмет дослідженння.
2. Задачі дипломної роботи.
3. Аналіз аналогів
4. Вимоги до програмного забезпечення.
5. Програмні засоби реалізації.
6. Діаграма варіантів використання.
7. Схема бази даних.
8. Мапа сайту веб-платформи.
9. Екранні форми.
10. Апробація результатів дослідження.

6. Дата видачі завдання «28» лютого 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	
1	Підбір та аналіз науково-технічної літератури	28.02.-06.03.2024	Виконано
2	Аналіз та дослідження існуючих аналогів	07.03-13.03.2024	Виконано
3	Дослідження програмних засобів	13.03-18.03.2024	Виконано
4	Моделювання об’єкту проектування	18.03-25.03.2024	Виконано
5	Програмна реалізація застосунку	26.03-22.04.2024	Виконано
6	Тестування застосунку	22.04-28.04.2024	Виконано
7	Оформлення роботи: вступ, висновки, реферат	29.04-05.05.2024	Виконано
8	Розробка демонстраційних матеріалів	06.05-12.05.2024	Виконано
9	Попередній захист роботи	13.05-31.05.2024	Виконано

Здобувач вищої освіти _____ Артур ГАЛУЩАК
(підпис)

Керівник
кваліфікаційної роботи _____ Оксана ЗОЛОТУХІНА
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 42 стор., 1 табл., 22 рис., 12 джерел.

Мета роботи – спростити пошук команди в проект для спільної розробки для розробників-початківців за рахунок веб-платформи створеної мовою C#..

Об'єкт дослідження – процес пошуку команди в проект для спільної розробки для розробників-початківців.

Предмет дослідження – веб-платформа для пошуку команди в проект для спільної розробки для розробників-початківців.

Короткий зміст роботи: В роботі проаналізовано існуючі рішення та інструменти для пошуку розробників, що дозволило визначити ключові функції та вимоги до системи. Розроблено архітектуру веб-платформи, яка включає основні компоненти та їх взаємодію. Програмно реалізовані ключові функціональні можливості платформи, зокрема: створення профілів користувачів, публікація проектів, використання фільтрів для пошуку проектів за назвою та технологіями, можливість залишати відгуки про співпрацю. В роботі використано ASP.NET Core MVC для серверної частини, Bootstrap для створення адаптивного інтерфейсу, Entity Framework для роботи з базою даних. Сферою використання застосунку є платформи для пошуку розробників та організації спільної роботи над проектами.

КЛЮЧОВІ СЛОВА: ПОШУК РОЗРОБНИКІВ, ВЕБ-ПЛАТФОРМА, ASP.NET CORE, C#, ENTITY FRAMEWORK.

ЗМІСТ

ВСТУП 9
1 ТЕОРЕТИЧНА ЧАСТИНА 11
1.1 Аналіз предметної галузі 11
1.2 Огляд існуючих рішень для пошуку розробників для співпраці над проектом 12
1.2.1 Teamfinding 13
1.2.2 DevPost 15
1.2.3 LinkedIn 16
2 ПРОЕКТУВАННЯ ТА РОЗРОБКА ВЕБ-ПЛАТФОРМИ 19
2.1 Моделювання вимог до програмного забеспечення 19
2.2 Опис архітектури MVC 20
2.2.1 Модель в архітектурі MVC 21
2.2.2 Вид в архітектурі MVC 23
2.2.3 Контролер в архітектурі MVC 25
2.2.4 Переваги архітектури MVC 26
2.2.5 Недоліки архітектури MVC 28
2.2.6 Розробка архітектури платформи 30
2.3 Вибір засобів програмної реалізації 31
2.3.1 Мова C# 31
2.3.2 ASP.NET Core MVC 32
2.3.3 Entity Framework Core 34
2.3.4 Microsoft SQL Server (MSSQL) 35
2.3.5 AutoMapper 36

2.3.6 Visual Studio 37
3 ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 39
3.1 Розробка структури бази даних 39
3.2 Мапа сайту веб-платформи та порядок роботи з сайтом 40
4 ТЕСТУВАННЯ 45
4.1 Тестування кросбраузерності 45
4.2 Тестування адаптивності веб-сайту 47
ВИСНОВКИ 51
ПЕРЕЛІК ПОСИЛАНЬ 52
ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ 54
ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ 61

ВСТУП

Обґрунтування вибору теми та її актуальність: Сучасні тенденції в IT-сфері вимагають постійного вдосконалення навичок та співпраці між розробниками. Однією з найбільших проблем, з якою стикаються початківці в цій галузі, є складність у пошуку команди для спільної роботи над проектами. Це особливо важливо в умовах глобалізації та швидкого розвитку технологій, коли ефективна співпраця може суттєво вплинути на успіх проекту. Враховуючи важливість обміну досвідом та спільної роботи для професійного зростання, створення платформи, яка об’єднує розробників, є вкрай актуальним завданням. Така платформа допоможе розробникам знаходити один одного для співпраці, обміну знаннями та спільної роботи над проектами, що сприятиме їх професійному розвитку та підвищенню якості програмного забезпечення. Використання ASP.NET Core MVC для розробки такої платформи забезпечує зручне та ефективне створення веб-додатків, які відповідають сучасним вимогам до безпеки, продуктивності та масштабованості.

Об’єкт дослідження – процес пошуку команди в проект для спільної розробки для розробників-початківців.

Предмет дослідження – веб-платформа для пошуку команди в проект для спільної розробки для розробників-початківців.

Мета роботи – спростити пошук команди в проект для спільної розробки для розробників-початківців за рахунок веб-платформи створеної мовою C#.

Завдання дослідження:

1. Аналіз існуючих додатків та програмних засобів:
    - Проаналізувати наявні веб-платформи та інструменти, які пропонують схожий функціонал для пошуку розробників і команд.
    - Виявити їх переваги та недоліки для визначення найкращих практик та можливостей для вдосконалення.

2. Формування вимог до платформи:
    - Сформувати функціональні та нефункціональні вимоги до веб-платформи на основі аналізу аналогічних рішень
    - Визначити ключові потреби користувачів та вимоги до продуктивності, безпеки та зручності використання.
3. Проектування архітектури платформи:
    - Спроектувати архітектуру веб-платформи з урахуванням визначених вимог.
    - Визначити основні компоненти системи, їх взаємодію та використані технології.
4. Розробка програмного забезпечення для пошуку команди для співпраці
5. Тестування розробленої платформи..
6. Пройти апробацію тез на конференції.

Практична значущість дослідження полягає у наданні розробникам засобу для спрощення процесу пошуку команд для спільної роботи над проектами. Завдяки створенню веб-платформи, яка об'єднує розробників та дозволяє їм легко знаходити один одного для співпраці, значно спрощується процес формування команд, що підвищує ефективність та продуктивність проектів. Особливістю платформи є можливість залишати відгуки про співпрацю, що сприяє створенню надійних та продуктивних команд.

Галузь використання - IT-індустрія, включаючи стартапи, малі та середні підприємства, а також великі корпорації, які прагнуть підвищити ефективність формування команд для розробки програмного забезпечення.

Робота пройшла апробацію на міжнародній науково-практичної інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи» (20.05.2024, ВНТУ, м. Вінниця)[12].

1 ТЕОРЕТИЧНА ЧАСТИНА

1.1 Аналіз предметної галузі

Сучасна розробка програмних продуктів є дуже комплексна робота спеціалістів різних напрямків, враховуючи це, для здобуття досвіду розробки програмних засобів потрібна команда, де кожен член команди буде зосереджений на вирішені конкретних задач в проекті на яких він спеціалізується, що дозволить кожному розробнику не відволікатись на задачі які не пов’язані з його спеціалізацією.

Для пошуку команди у свій проект можна використовувати соціальні мережі, спільноти та форуми для програмістів. Популярні форуми такі як Stack Overflow або Reddit мають спеціальні розділи або теми, де можна шукати або пропонувати співпрацю над проектами. Використання соціальних мереж таких як Twitter, LinkedIn або Facebook дозволяє зв’язатись з іншими розробниками, що зацікавлені в спільній роботі над проектом. Також участь у хакатонах та змаганнях може бути відмінним способом знайти однодумців для спільної роботи над проектами.

Цільовою аудиторією можуть бути:

1. Студенти програмістських спеціальностей: Студенти, які вивчають програмування або пов'язані з ним технології, можуть шукати можливості для співпраці з однолітками над проектами або для підвищення своїх навичок шляхом співпраці в команді.
2. Початківці у програмуванні: Люди, які тільки починають свій шлях у програмуванні, можуть шукати можливостей для навчання та отримання досвіду шляхом участі в командних проектах. Професійні програмісти: Досвідчені програмісти також можуть бути зацікавлені у знаходженні нових можливостей для співпраці та розвитку своїх навичок шляхом роботи в команді над новими проектами.

3. Фрілансери та замовники: Люди, які шукають фріланс-розробників для виконання своїх проектів, можуть також зацікавитись у знаходженні команди для співпраці над конкретними завданнями або проектами.
4. Професійні програмісти: Досвідчені програмісти також можуть бути зацікавлені у знаходженні нових можливостей для співпраці та розвитку своїх навичок шляхом роботи в команді над новими проектами.

Вимоги цільової аудиторії можуть бути різноманітними й залежать від конкретних потреб та очікувань користувачів.

Однією з вимог є простота використання, користувачам потрібна легкий та зрозуміла в використані інструмент. Користувач повинен мати змогу легко знайти цікаві йому проекти, з потрібними для нього технологіями або створити самому такий проект.

Ще однією вимогою може бути можливість зворотного зв’язку, користувачі можуть мати потребу залишити відгук на користувача з яким вони працювали, що допомагає покращити якість співпраці на платформі.

Для вирішення проблеми пошуку досвіду розробки в команді буде розроблене рішення, яке допоможе вирішити ці задачі. Користувач зможе створити свій проект, вказати технології які будуть використовуватись та описати ідею проекту. Користувач матиме змогу знайти цікаві йому проекти, та приєднатись до них, а також залишити відгук на користувачів з якими він працював.

1.2 Огляд існуючих рішень для пошуку розробників для співпраці над проектом

Аналіз існуючих рішень для пошуку розробників для співпраці над проектом є важливим етапом у розробці нової веб-платформи. Це дає змогу зрозуміти, які функції і можливості вже реалізовані в інших сервісах, а також виявити їхні недоліки та проблеми, щоб уникнути їх у майбутньому. Найбільш

відомими сервісами для пошуку розробників і проектів є Teamfinding, DevPost, LinkedIn.

1.2.1 Teamfinding

«Teamfinding» є платформою для пошуку розробників для спільної роботи над проектами. Вона дозволяє користувачам створювати профілі, публікувати проекти, і знаходити команди для спільної роботи. Користувач моє змогу створювати проекти, описувати ідеї проектів, коментуваті проекти, вести профіль користувача та шукати інших користувачів.

Переваги:

- Широкий функціонал: Можливість створення профілів, публікації проектів, пошуку за технологіями та залишення відгуків про співпрацю.
- Гнучкість пошуку: Користувачі можуть шукати проекти за різними критеріями, такими як технології та назва проекту.

Недоліки:

- Інтерфейс може бути трохи застарілим.
- Відсутня можливість залишити відгук на сторінці користувача с яким велась співпраця.

Приклад графічного інтерфейсу головної сторінки, сторінки створення проекту та сторінки користувача платформи зображені на рисунках 1.1, 1.2, 1.3.

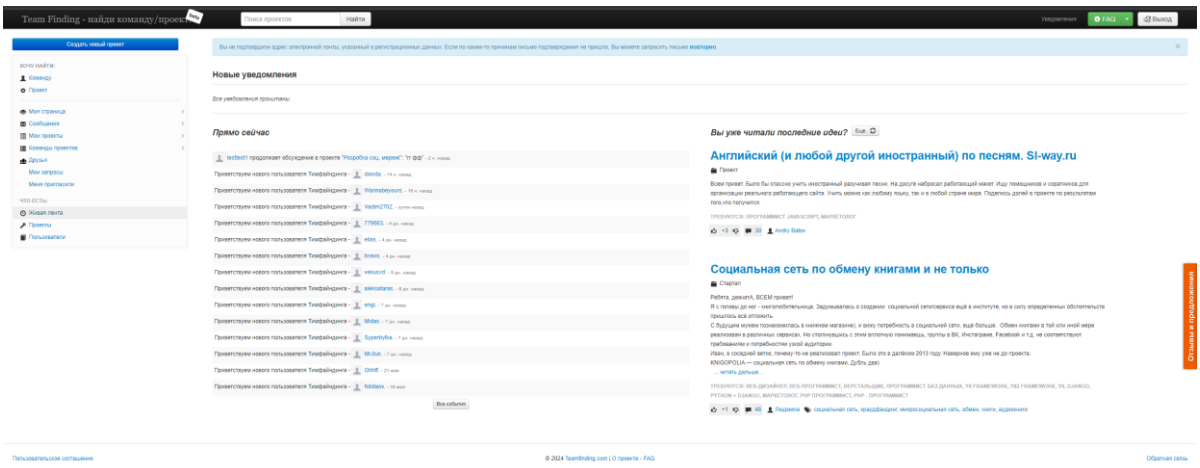


Рис. 1.1 Приклад головної сторінки Teamfinding

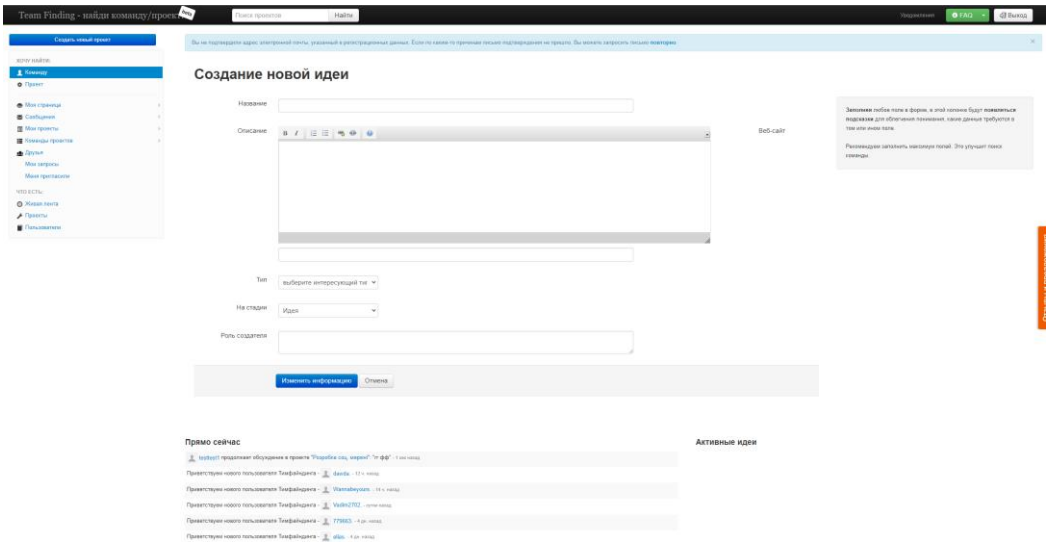


Рис. 1.2 Приклад сторінки створення проекту

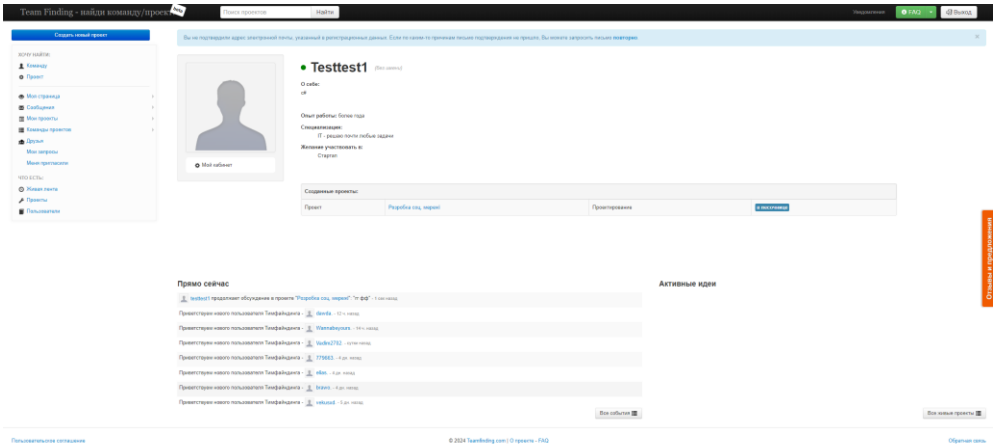


Рис. 1.3 Приклад сторінки користувача

1.2.2 DevPost

«DevPost» є платформою для хакатонів і змагань серед розробників, яка дозволяє організаторам створювати і проводити змагання, а учасникам - подавати свої проекти та ділитися ними з іншими.

«DevPost» створений для сприяння розвитку інновацій та залучення розробників до участі в різноманітних технічних змаганнях. Платформа служить як для організаторів хакатонів, так і для розробників, які шукають нові виклики, можливості для навчання та співпраці.

Переваги:

- Платформа пропонує велику кількість хакатонів з різних тем і технологій, що дозволяє розробникам знайти подію, яка відповідає їхнім інтересам та навичкам.

- Користувачі можуть створювати детальні профілі, де відображаються їхні навички, досвід та проекти, що сприяє побудові професійного портфоліо.

- Платформа дозволяє публікувати проекти, які потім можуть бути переглянуті іншими користувачами та потенційними роботодавцями.

Недоліки:

- Для тих, хто не бере участь у хакатонах, платформа може не пропонувати достатньо функціоналу або мотивації для постійного використання.

- Участь у хакатонах може бути дуже вимогливою щодо часу, що може бути проблемою для розробників, які мають інші професійні чи особисті зобов’язання.

Приклад графічного інтерфейсу платформи «DevPost» рисунки 1.4, 1.5.

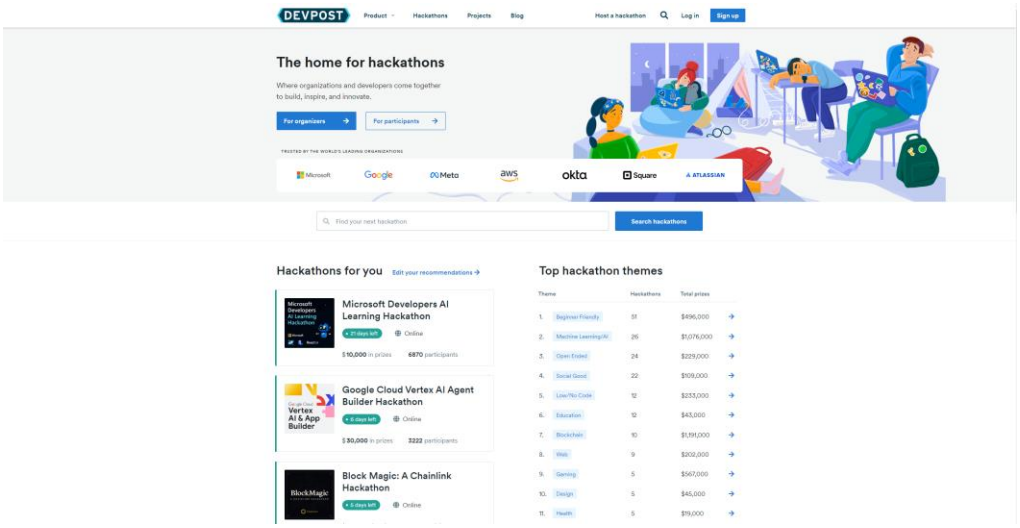


Рис. 1.4 Головна сторінка

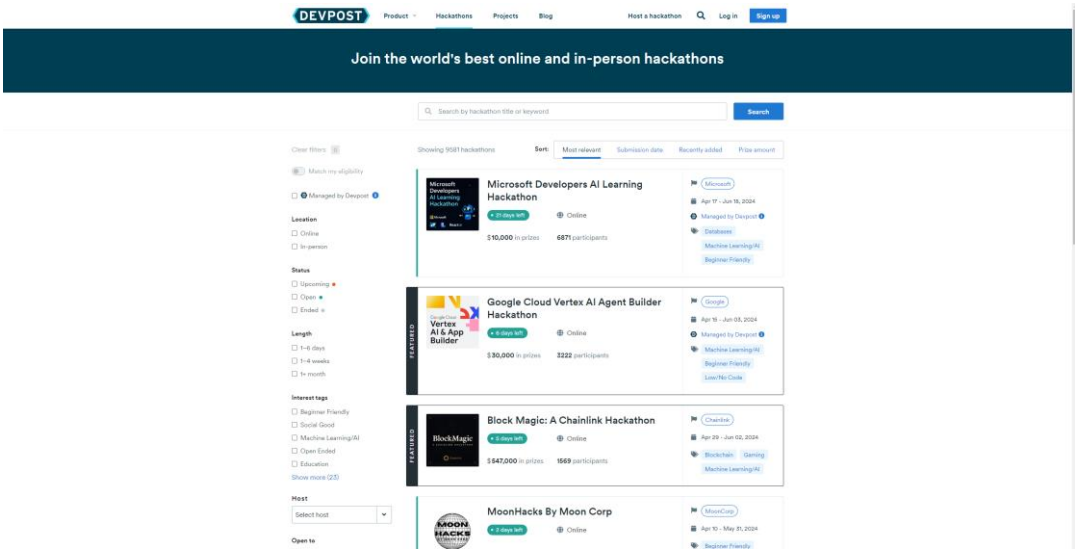


Рис. 1.5 Сторінка пошуку та фільтрації хакатонів

1.2.3 LinkedIn

«LinkedIn» є провідною соціальною мережею для професіоналів, яка дозволяє користувачам створювати професійні профілі, знаходити роботу, створювати мережу контактів і ділитися професійним контентом. LinkedIn створений для того, щоб допомагати професіоналам у всьому світі підтримувати зв’язки, знайти можливості для працевлаштування, ділитися досвідом і знаннями, а також розвивати свою кар'єру.

Переваги:

- Дозволяє користувачам створювати і підтримувати професійні контакти з колегами, партнерами та потенційними роботодавцями.

- Користувачі можуть створювати детальні професійні профілі, включаючи резюме, портфоліо робіт, досягнення та рекомендації.

- Платформа дозволяє приєднуватися до професійних груп, де можна обговорювати нові тенденції, ділитися досвідом і налагоджувати контакти з колегами по галузі.

Недоліки:

- Для ефективного використання платформи необхідно витрачати час на підтримку профілю, участь в обговореннях і нетворкінг.

Приклад графічного інтерфейсу рисунок 1.6.

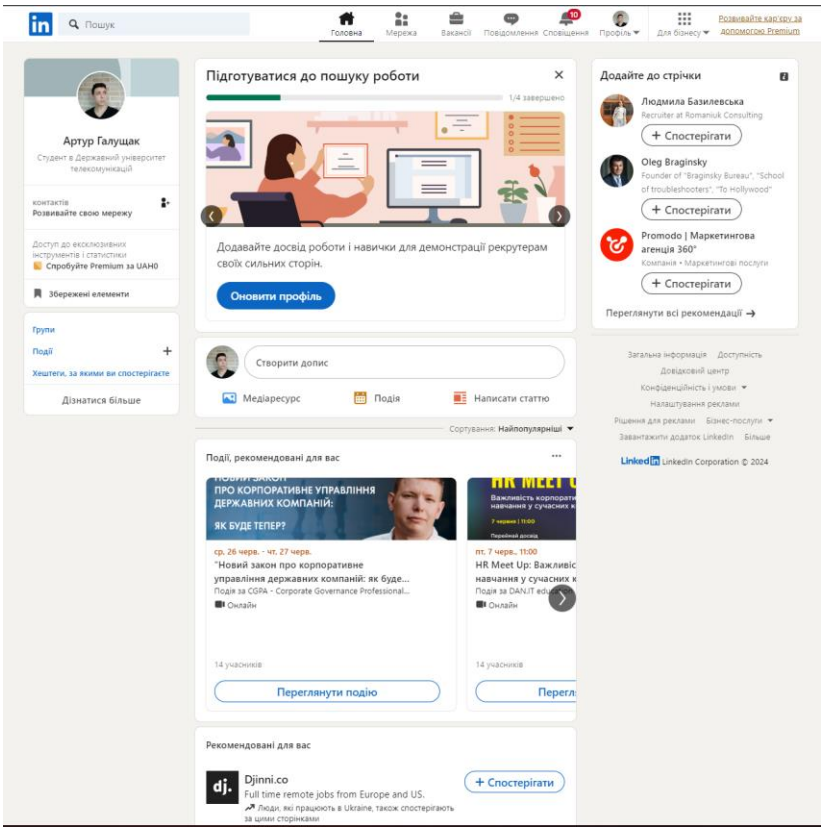


Рис. 1.6 Головна сторінка linkedin

Зведені результати аналізу характеристик платформ для пошуку проектів та об’єднання розробників для співпраці наведено в таблиці 1.1.

Табиця 1.1

Порівняння існуючих програм-аналогів

Функціональні можливості	**Teamfinding**	**DevPost**	**LinkedIn**	**DevTeamUp**
Пошук проектів по технологіям	-	+	-	+
Пошук користувачі по технологіям	-	-	+	+
Сторінки користувачів	+	+	+	+
Сторінки проектів	+	+	-	+
Можливість залишати відгуки на інших користувачів	-	+	+	+

2 ПРОЕКТУВАННЯ ТА РОЗРОБКА ВЕБ-ПЛАТФОРМИ

2.1 Моделювання вимог до програмного забеспечення

Діаграма випадків використання демонструє основні взаємодії між користувачами та системою, підкреслюючи основні функціональні можливості. Вона служить орієнтиром для розуміння загального потоку дій та функціоналу системи. На рисунку 2.1 представлено загальний вигляд діаграми випадків використання платформи для пошуку розробників для співпраці.

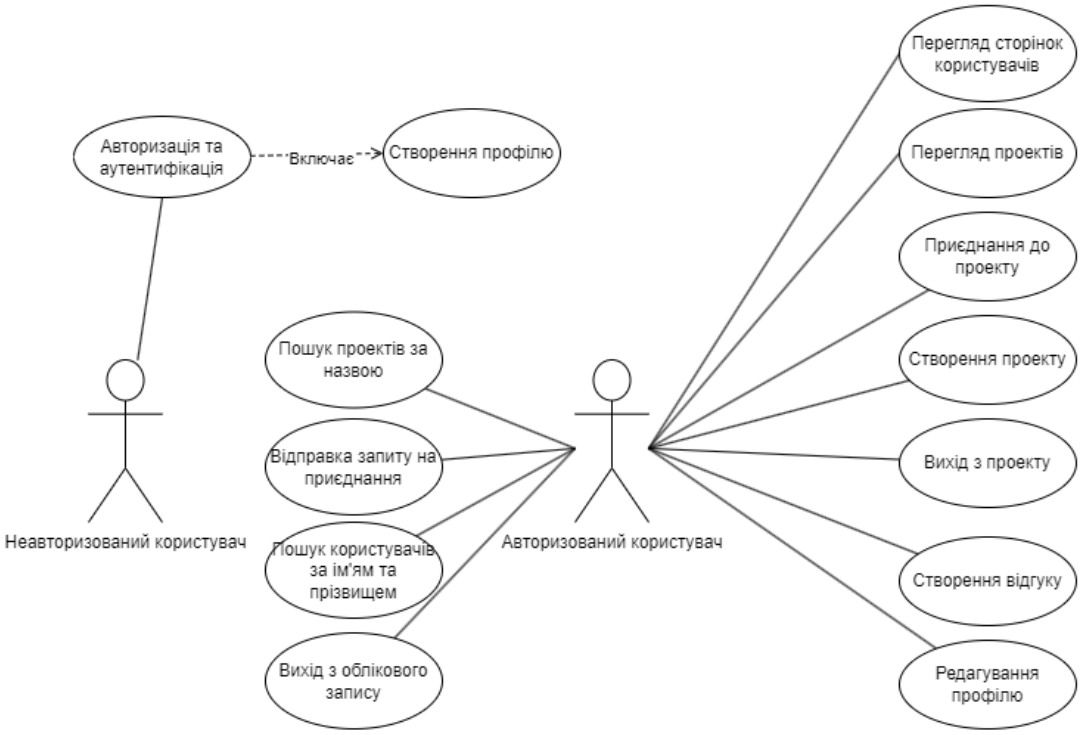


Рис.2.1 Діаграма варіантів використання

Функціонал платформи відрізняється залежно від того, зареєстрований користувач чи ні. Новому користувачу, перед використанням платформи потрібно обов'язково заповнити свій профіль. Відповідно функціонал платформи, а саме:

- Перегляд сторінок користувачів
- Перегляд проектів
- Долучення до проектів
- Створення проекті
- Вихід з проекту
- Написання відгуків на користувачів
- Редагування профілю
- Пошук проектів за назвою
- Пошук користувачів за ім’ям та прізвищем

Буде доступний користувачу, лише після успішної авторизації, та заповненому профілю користувача.

2.2 Опис архітектури MVC

Для досягнення масштабованості, зручної розробки та підтримки платформи була обрана архітектура MVC. При розробці з використанням MVC, програма складається з трьох основних складових:

- Модель (Model). Представляє дані та логіку що стоїть за ними. А також модель може включати роботу з базою даних, бізнес-логіку та інші обробки даних.
- Вид (View). Відповідає за відображення інформації користувачеві та взаємодіє з ним. Вид відповідає за представлення даних моделі та інтерфейс користувача
- Контролер (Controller). Відповідає за обробку вхідних даних від користувача, взаємодіє з компонентом моделлю (Model) та обирає відповідне представлення (View) для відображення. Контролер реагує на дії користувача та виконує відповідні дії, наприклад оновлення даних чи відображення нового представлення (View).

Архітектура MVC сприяє розділенню відповідальностей між компонентами програми, що полегшує розробку, тестування та підтримку

програмного забезпечення. Вона також сприяє повторному використанню коду та забезпечує більшу гнучкість у зміні функціональності програми. Графічний приклад архітектури MVC зображенний на рисунку 2.2.

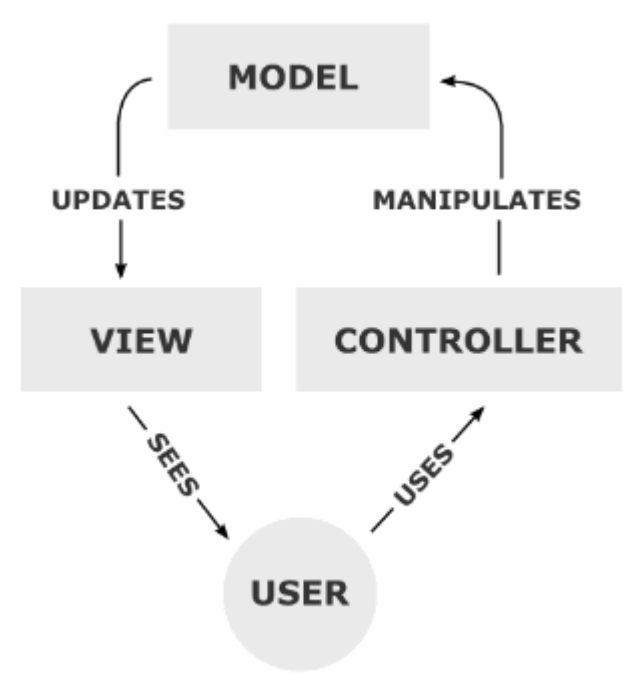


Рис. 2.2 Схема MVC

2.2.1 Модель в архітектурі MVC

У рамках архітектурного патерну MVC, компонент "Модель" (Model) виконує ключову роль у представленні даних і бізнес-логіки додатку. Вона є основним елементом, що відповідає за зберігання, обробку та управління даними, а також взаємодію з іншими компонентами системи.

Доступ до даних: Модель забезпечує доступ до даних, які можуть бути збережені в різних джерелах, таких як бази даних, файлові системи або веб-служби. Вона обробляє запити на отримання, оновлення, видалення та вставку даних, гарантуючи, що всі дії відповідають поточному стану додатку.

Бізнес-логіка: Крім зберігання даних, модель також містить бізнес-логіку, яка включає обробку та маніпулювання даними. Наприклад, це може бути логіка перевірки правильності введених даних, проведення розрахунків або

формування звітів. Ця логіка забезпечує коректність і актуальність даних у додатку.

Взаємодія з контролером: Модель тісно взаємодіє з контролером (Controller), який виступає посередником між користувачем і системою. Контролер використовує модель для виконання операцій з даними, таких як отримання інформації або оновлення записів. Модель, у свою чергу, передає результати цих операцій назад контролеру для подальшої обробки та відображення.

Повідомлення про зміни: У разі зміни даних модель повідомляє про це контролер, що дозволяє останньому відповідно оновлювати відображення на вигляді (View). Це забезпечує динамічне та актуальне відображення інформації користувачам. Незалежність від інтерфейсу користувача: Модель повинна бути розроблена так, щоб не залежати від інтерфейсу користувача. Це означає, що вона повинна функціонувати незалежно від того, як дані будуть представлені на вигляді. Такий підхід забезпечує гнучкість і можливість повторного використання моделі в різних контекстах.

Тестування: Модель є легкою для тестування, оскільки вона містить усю бізнес-логіку і доступ до даних, незалежно від інтерфейсу користувача. Це дозволяє розробникам проводити юніт-тестування моделі, виявляючи і виправляючи помилки на ранніх стадіях розробки, що підвищує загальну якість додатку.

Незалежність від технологій: Модель повинна бути розроблена таким чином, щоб не залежати від конкретних технологій або інфраструктури, наприклад, від конкретної бази даних або веб-служби. Це забезпечує гнучкість у виборі технологічних рішень і дозволяє легко адаптувати додаток до змін у технологічному середовищі.

Незалежність від інших компонентів: Модель повинна залишатися незалежною від інших компонентів додатку, таких як вигляд і контролер. Це дозволяє досягти високого рівня модульності і масштабованості додатку, що полегшує його подальший розвиток і підтримку.

Таким чином, модель у патерні архітектури MVC є критично важливим компонентом, який забезпечує управління даними і бізнес-логікою, підтримуючи незалежність від інших елементів системи і технологій. Це дозволяє створювати гнучкі, масштабовані і легкотестовані додатки.

2.2.2 Вид в архітектурі MVC

У контексті архітектурного патерну MVC (Model-View-Controller) компонент "Вид" (View) виконує ключову роль у відображенні даних і взаємодії з користувачем. Вид відповідає за візуальне представлення інформації, що отримується від моделі, та забезпечує інтерфейс, через який користувач може взаємодіяти із системою.

Відображення даних: Основна функція виду полягає у відображенні даних, що надаються моделлю. Вид отримує дані від контролера і відтворює їх у зрозумілому і зручному для користувача форматі. Це можуть бути таблиці, графіки, форми або будь-які інші елементи інтерфейсу, які необхідні для представлення інформації.

Реакція на дії користувача: Вид також відповідає за обробку взаємодії користувача з інтерфейсом. Це включає натискання кнопок, введення тексту, вибір елементів з випадаючих списків тощо. Коли користувач виконує якусь дію, вид передає відповідні запити контролеру для обробки і оновлення даних.

Оновлення інтерфейсу: Вид повинен динамічно оновлювати інтерфейс у відповідь на зміни, що відбуваються у моделі. Наприклад, якщо дані в моделі були змінені, вид має відобразити ці зміни без потреби перезавантаження всієї сторінки. Це забезпечує більш плавну і інтерактивну взаємодію користувача з системою.

Відокремлення логіки від презентації: Вид повинен бути ізольованим від бізнес-логіки додатку, яка реалізується у моделі. Це означає, що вид не повинен містити складних алгоритмів або логіки обробки даних. Він лише відображає інформацію і реагує на дії користувача, забезпечуючи чітке розмежування між логікою і презентацією.

Тестування виду: Вид також повинен бути легко тестованим. Це досягається шляхом написання тестів, які перевіряють коректність відображення даних і взаємодії з елементами інтерфейсу. Тестування виду допомагає виявити і виправити помилки на ранніх стадіях розробки, забезпечуючи високу якість користувацького інтерфейсу.

Незалежність від інших компонентів: Вид повинен бути максимально незалежним від інших компонентів системи, таких як модель і контролер. Це забезпечує можливість легкого оновлення або заміни виду без впливу на інші частини додатку. Такий підхід підвищує гнучкість і масштабованість системи.

Застосування шаблонів: У сучасних веб-додатках для створення виду часто використовуються шаблони, які дозволяють розробникам розділяти HTML-код і логіку відображення. Це робить код більш організованим і легким для підтримки. Шаблони можуть бути використані для повторного використання елементів інтерфейсу в різних частинах додатку.

Використання фреймворків: Для розробки виду часто використовуються фреймворки, такі як React, Angular або Vue.js, які спрощують процес створення інтерактивних і динамічних інтерфейсів. Ці фреймворки надають розширені можливості для управління станом компонента і реактивного оновлення інтерфейсу.

Таким чином, компонент "Вид" в архітектурі MVC виконує важливу роль у забезпеченні взаємодії користувача з додатком. Він відповідає за відображення даних, реакцію на дії користувача і динамічне оновлення інтерфейсу, забезпечуючи при цьому чітке розмежування між логікою і презентацією, що робить систему більш гнучкою і легкою для підтримки.

2.2.3 Контролер в архітектурі MVC

Контролер (Controller) у патерні архітектури MVC (Model-View-Controller) виконує ключову роль в обробці запитів від користувачів, взаємодії з моделлю та передавання відповідних даних до виду для відображення. Контролер є центральною ланкою, яка пов’язує модель і вид, забезпечуючи координацію між ними.

Обробка запитів користувачів: Контролер відповідає за отримання запитів від користувачів. Це можуть бути запити, які надходять через веб-інтерфейс, наприклад, натискання кнопок, введення даних у форму або навігація по сторінках сайту. Контролер аналізує ці запити і визначає, які дії необхідно виконати.

Взаємодія з моделлю: Після отримання запиту контролер взаємодіє з моделлю для виконання необхідних операцій. Це може включати доступ до бази даних, обробку бізнес-логіки, валідацію даних або інші дії, які повинні бути виконані для задоволення запиту користувача. Контролер викликає відповідні методи моделі, отримує дані та обробляє їх.

Передача даних до виду: Після виконання операцій з моделлю контролер передає отримані дані до виду. Контролер вирішує, які дані необхідно відобразити і як вони повинні бути представлені. Це дозволяє контролеру оновлювати вигляд відповідно до змін у даних або виконаних операцій.

Розподіл логіки: Контролер відповідає за розподіл логіки між моделлю та видом. Вся бізнес-логіка і обробка даних знаходиться в моделі, тоді як вид відповідає за візуальне представлення інформації. Контролер забезпечує правильну взаємодію між цими компонентами, підтримуючи чітке розмежування їхніх обов'язків.

Обробка виключень: Контролер також відповідає за обробку виключень та помилок, що можуть виникати під час обробки запитів. Якщо під час взаємодії з моделлю виникає помилка, контролер повинен обробити її та передати

відповідне повідомлення до виду, щоб користувач отримав зрозумілу інформацію про проблему.

Роутинг: Контролер зазвичай включає функціональність роутингу, яка визначає, як URL-запити зіставляються з конкретними методами контролера. Це дозволяє системі правильно обробляти різні типи запитів і направляти їх до відповідних дій. У фреймворках, таких як ASP.NET Core, ця функціональність вбудована і дозволяє легко налаштовувати маршрути для різних дій.

Забезпечення безпеки: Контролер також може виконувати функції, пов'язані з безпекою, наприклад, автентифікацію та авторизацію користувачів. Він перевіряє, чи має користувач право доступу до певних даних або функцій додатку, і виконує відповідні дії на основі результатів цієї перевірки.

Тестування: Контролери є важливою частиною тестованого коду. Вони можуть бути протестовані окремо від моделі та виду для перевірки правильності обробки запитів, взаємодії з моделлю і передачі даних до виду. Це допомагає виявити помилки на ранніх етапах розробки та забезпечує високу якість коду.

Взаємодія з іншими контролерами: У складних додатках може бути кілька контролерів, кожен з яких відповідає за свою частину функціональності. Контролери можуть взаємодіяти один з одним для виконання складних сценаріїв і забезпечення координації між різними частинами додатку.

Таким чином, контролер в архітектурі MVC є центральним компонентом, який координує взаємодію між моделлю та видом. Він обробляє запити користувачів, виконує операції з моделлю та передає дані до виду, забезпечуючи правильне функціонування додатку і підтримуючи чітке розмежування логіки та представлення.

2.2.4 Переваги архітектури MVC

Архітектура MVC (Model-View-Controller) має багато переваг, які роблять її популярною серед розробників програмного забезпечення. Вона забезпечує ефективний поділ обов'язків між різними компонентами додатку, що сприяє організованій та гнучкій розробці.

Чітке розділення обов’язків: Однією з головних переваг архітектури MVC є чітке розділення обов’язків між її компонентами. Модель (Model) відповідає за управління даними та бізнес-логікою, Вид (View) займається відображенням даних, а Контролер (Controller) обробляє запити користувачів і координує взаємодію між Моделлю та Видом. Це дозволяє розробникам зосередитися на конкретних аспектах додатку, що підвищує ефективність роботи та зменшує ризик виникнення помилок.

Підвищення можливостей повторного використання коду: Завдяки чіткому розмежуванню між компонентами MVC архітектури, код стає більш повторно використовуваним. Наприклад, Модель, яка містить бізнес-логіку та управління даними, може бути використана в різних частинах додатку без змін. Це дозволяє економити час та зусилля розробників, зменшує дублювання коду і сприяє підтримці високої якості програмного забезпечення.

Спрощення тестування: Оскільки кожен компонент MVC архітектури є окремою сутністю з власними функціями, їх можна тестувати незалежно один від одного. Це спрощує процес створення тестів та забезпечує виявлення помилок на ранніх етапах розробки. Розробники можуть проводити юніт-тестування для моделі та контролера, а також тестування інтерфейсу користувача для виду.

Підтримка одночасної розробки: Архітектура MVC дозволяє кільком розробникам працювати над різними частинами додатку одночасно. Один розробник може працювати над логікою додатку в моделі, інший – над інтерфейсом користувача у виді, а третій – над обробкою запитів у контролері. Це значно підвищує продуктивність команди та зменшує час, необхідний для розробки додатку.

Полегшення обслуговування та розширення: Завдяки чіткому розділенню завдань у архітектурі MVC, обслуговування та розширення додатку стає значно простішим. Зміни в одному компоненті можуть бути внесені без впливу на інші компоненти. Наприклад, зміна у відображенні даних не вимагає змін у бізнес-

логіці або управлінні даними. Це сприяє легшому впровадженню нових функціональних можливостей та виправленню помилок.

Підтримка різних видів інтерфейсу: MVC архітектура дозволяє легко підтримувати різні види інтерфейсу користувача для одного і того ж додатку. Оскільки види є незалежними від моделі та контролера, можливо створювати різні інтерфейси для веб-додатків, мобільних додатків або десктопних додатків, використовуючи одну і ту ж бізнес-логіку та управління даними.

Підвищення модульності та масштабованості: Завдяки модульності компонентів MVC архітектура сприяє легкій масштабованості додатку. Додавання нових функціональних можливостей або розширення існуючих компонентів може бути виконане без значного впливу на інші частини системи. Це забезпечує гнучкість у розробці та підтримці великих і складних додатків.

Покращена структура коду: Архітектура MVC сприяє організованій та зрозумілій структурі коду. Чіткий поділ між моделлю, видом та контролером забезпечує логічне розташування коду відповідно до його функціональних обов’язків. Це полегшує читання, розуміння та обслуговування коду, що особливо важливо при роботі в команді або при передачі проекту новим розробникам.

Таким чином, архітектура MVC має безліч переваг, які сприяють ефективній, організованій та гнучкій розробці програмного забезпечення. Вона забезпечує чітке розділення обов’язків, можливість повторного використання коду, спрощення тестування, підтримку одночасної розробки, легке обслуговування та розширення, підтримку різних видів інтерфейсу, підвищення модульності та масштабованості, а також покращену структуру коду.

2.2.5 Недоліки архітектури MVC

Хоча архітектура MVC (Model-View-Controller) має багато переваг, вона не позбавлена своїх недоліків. Нижче розглянуто основні проблеми та обмеження, з якими можуть зіткнутися розробники при використанні цієї архітектури.

Складність навчання: Архітектура MVC може бути складною для розуміння новачками. Вона вимагає розуміння трьох окремих компонентів (Модель, Вид, Контролер) та взаємодії між ними. Це може бути викликом для розробників, які тільки починають працювати з цією архітектурою. Навчання може зайняти більше часу, що може вплинути на загальну продуктивність команди.

Збільшена кількість файлів та класів: Використання MVC призводить до збільшення кількості файлів та класів у проекті. Кожен компонент (Модель, Вид, Контролер) зазвичай представлений окремими файлами, що може ускладнити навігацію та управління проектом, особливо в великих додатках. Розробники можуть витрачати більше часу на пошук потрібних файлів та розуміння їх взаємодії.

Підвищена складність розробки: Поділ обов’язків між моделлю, видом та контролером додає додаткового рівня складності при розробці додатків. Розробникам потрібно координувати взаємодію між цими компонентами, що може вимагати більше зусиль та часу. Написання коду для взаємодії між моделлю та контролером, а також між контролером та видом може бути більш складним, ніж в архітектурах з меншою кількістю рівнів.

Потенційні проблеми з продуктивністю: Архітектура MVC може створювати додаткові накладні витрати на продуктивність через необхідність обробки даних у різних компонентах. Наприклад, дані можуть бути спочатку оброблені моделлю, потім передані контролеру, а потім виду для відображення. Цей додатковий крок може вплинути на швидкість роботи додатку, особливо якщо дані обробляються інтенсивно або великими обсягами.

Труднощі в обслуговуванні малих проектів: Для невеликих проектів використання архітектури MVC може бути надмірним і ускладнювати розробку та обслуговування. В таких випадках розробники можуть вважати MVC занадто складним і громіздким, що може привести до збільшення часу та зусиль, необхідних для завершення проекту.

Високий поріг входження: Через складність архітектури MVC, нові учасники команди можуть стикатися з труднощами при входженні в проект. Це може уповільнити процес адаптації нових розробників, що може бути критичним для проектів з високою плинністю кадрів або великими командами.

Залежність між компонентами: Незважаючи на те, що MVC сприяє розділенню обов'язків, часто компоненти стають сильно залежними один від одного. Наприклад, зміна в моделі може вимагати змін в контролері і навпаки. Це може ускладнити внесення змін та збільшити ризик виникнення помилок.

Труднощі з масштабуванням контролерів: У великих додатках контролери можуть стати надто великими та складними через необхідність обробки різних запитів користувачів і управління взаємодією між моделлю та видом. Це може привести до труднощів у підтримці коду та збільшення часу на його розробку та тестування. Отже, хоча архітектура MVC має численні переваги, її недоліки також слід враховувати. Вибір цієї архітектури вимагає ретельного аналізу потреб проекту, врахування складності та масштабів розробки, а також готовності команди до роботи з нею.

2.2.6 Розробка архітектури платформи

Платформа DevTeamUp створена на основі архітектури MVC (Model-View-Controller), яка використовується для розробки веб-додатків на мові C# з використанням фреймворка ASP.NET. Основні компоненти цієї архітектури — це модель, представлення та контролер.

Модель (Model) в нашому додатку представлена класами, розташованими в каталозі Models. Ці класи відповідають за обробку даних та бізнес-логіку. Наприклад, класи CreateProjectViewModel.cs, LoginViewModel.cs, ProfileInitVM.cs забезпечують взаємодію з базою даних для отримання та збереження інформації.

Представлення (View) розташовані у каталозі Views і відповідають за відображення даних, отриманих з моделі. Вони дозволяють користувачам взаємодіяти з додатком, надаючи можливість редагувати, видаляти та додавати

дані. Приклади представлень включають файли Login.cshtml, Index.cshtml, CreateProject.cshtml.

Контролери (Controller) розташовані у каталозі Controllers і відповідають за обробку запитів користувачів та взаємодію з моделлю та представленнями. Вони отримують запити від користувачів, виконують необхідну логіку, та передають дані до відповідних представлень. Приклади контролерів включають AccountController.cs, ProfileController.cs, ProjectController.cs.

Загалом, архітектура MVC дозволяє чітко розділити відповідальність між різними частинами додатку, що покращує підтримку та розширення коду.

2.3 Вибір засобів програмної реалізації

2.3.1 Мова C#

C# є сучасною об'єктно-орієнтованою мовою програмування, яка функціонує в середовищі .NET. Її вибір для реалізації платформи пошуку розробників обґрунтований кількома ключовими перевагами, що роблять цю мову ідеальною для нашого проекту.

По-перше, C# забезпечує високу продуктивність та ефективність. Завдяки можливостям компіляції в байт-код, що виконується на платформі .NET, C# забезпечує швидке виконання коду. Це особливо важливо для веб-додатків, де швидкість відгуку є критичною для користувачів. Також C# підтримує паралельне та асинхронне програмування, що дозволяє оптимізувати роботу з великими обсягами даних і підвищити продуктивність системи.

По-друге, C# надає розвинену підтримку об'єктно-орієнтованого програмування (ООП). Завдяки потужній системі типів і широкому спектру інструментів для роботи з об'єктами, розробники можуть створювати складні та модульні програми. Це забезпечує легкість підтримки та масштабованості коду, що є важливим для довгострокового розвитку платформи.

Третя перевага – безпека та надійність. C# включає в себе механізми, такі як автоматичне збирання сміття (Garbage Collection), що допомагає уникати

проблем з управлінням пам'яттю. Крім того, C# має розвинену систему обробки винятків, що дозволяє ефективно управляти помилками та зменшувати ймовірність збоїв у роботі програми.

Ще однією значною перевагою C# є його інтеграція з платформою .NET, яка забезпечує доступ до великої кількості бібліотек і фреймворків. Це значно полегшує процес розробки, дозволяючи використовувати готові рішення для багатьох завдань, таких як робота з базами даних, мережеві операції, аутентифікація та багато іншого. Багатий набір інструментів і бібліотек значно зменшує час на розробку та тестування.

Крім того, C# є кросплатформною мовою програмування завдяки .NET Core, що дозволяє розробляти програми для різних операційних систем, включаючи Windows, Linux і macOS. Це забезпечує широку доступність платформи для різних користувачів, що сприяє її популярності та зростанню.

Нарешті, C# має активну спільноту розробників і велику кількість навчальних ресурсів, що полегшує навчання та підвищення кваліфікації розробників. Це дозволяє швидко знайти відповіді на питання та отримати допомогу у вирішенні проблем, що виникають під час розробки.

Таким чином, вибір C# для реалізації платформи пошуку розробників забезпечує високу продуктивність, надійність, безпеку та зручність розробки, що робить його оптимальним рішенням для нашого проекту.

2.3.2 ASP.NET Core MVC

ASP.NET Core MVC – це потужний і гнучкий фреймворк для розробки веб-додатків на основі архітектури Model-View-Controller (MVC). Його вибір для реалізації нашої платформи обґрунтований низкою переваг, які забезпечують ефективну та масштабовану розробку веб-застосунків.

По-перше, ASP.NET Core MVC є кросплатформним фреймворком, що дозволяє створювати веб-додатки для різних операційних систем, включаючи Windows, Linux і macOS. Це забезпечує широке охоплення аудиторії та дає

можливість розгортання додатків у різних середовищах, що є важливим для сучасних веб-додатків.

Другою значною перевагою є висока продуктивність ASP.NET Core MVC. Завдяки оптимізації та легковажній архітектурі, цей фреймворк забезпечує швидке виконання коду та низький час відгуку. Це особливо важливо для веб-додатків, де продуктивність є критичною для забезпечення якісного користувацького досвіду.

ASP.NET Core MVC також підтримує модульність і розширюваність. Архітектура MVC дозволяє розділяти логіку програми на різні компоненти: моделі, представлення та контролери. Це спрощує розробку, тестування та підтримку коду, забезпечуючи можливість легкого внесення змін і додавання нових функцій.

Ще однією важливою перевагою є інтеграція з сучасними стандартами веб-розробки. ASP.NET Core MVC підтримує розробку RESTful API, що дозволяє створювати масштабовані та легко інтегровані веб-сервіси. Крім того, фреймворк забезпечує підтримку таких технологій, як Razor Pages, що спрощують розробку сторінок з динамічним вмістом.

Безпека є ще одним важливим аспектом вибору ASP.NET Core MVC. Фреймворк включає в себе численні механізми безпеки, такі як захист від CSRF-атак, інструменти для управління аутентифікацією та авторизацією, підтримку SSL та багато інших. Це забезпечує захист додатків від широкого спектру загроз і забезпечує надійну роботу веб-застосунків.

Крім того, ASP.NET Core MVC має активну спільноту розробників і розвинену екосистему, що включає численні бібліотеки, інструменти та ресурси для навчання. Це спрощує процес розробки, дозволяючи швидко знайти рішення для типових завдань і отримати підтримку у разі виникнення проблем.

Таким чином, вибір ASP.NET Core MVC для реалізації платформи пошуку розробників забезпечує кросплатформеність, високу продуктивність, безпеку та зручність розробки, що робить цей фреймворк ідеальним рішенням для нашого проекту.

2.3.3 Entity Framework Core

Entity Framework Core (EF Core) – це сучасний об'єктно-реляційний мапер (ORM) для .NET, який забезпечує ефективний спосіб роботи з базами даних, використовуючи об'єктно-орієнтовані принципи програмування. Вибір EF Core для нашої платформи обґрунтований кількома ключовими перевагами.

По-перше, EF Core значно спрощує роботу з базами даних завдяки концепції об'єктно-реляційного мапінгу. Це дозволяє розробникам працювати з базами даних, використовуючи об'єкти і класи, замість написання складних SQL-запитів. Це підвищує продуктивність і зменшує ймовірність помилок при роботі з базою даних.

Другою важливою перевагою EF Core є його підтримка різних баз даних. Це забезпечує гнучкість у виборі бази даних для нашої платформи і дозволяє легко змінювати або оновлювати базу даних без значних змін у коді програми. EF Core підтримує такі бази даних, як SQL Server, MySQL, PostgreSQL, SQLite та інші.

EF Core також забезпечує високу продуктивність і оптимізацію запитів. Завдяки таким функціям, як відкладене завантаження (lazy loading), попереднє завантаження (eager loading) та кешування, розробники можуть оптимізувати роботу з даними і забезпечити швидкий доступ до необхідної інформації. Це важливо для масштабованих додатків, де ефективна робота з базою даних є критичною.

Ще однією важливою перевагою EF Core є підтримка міграцій. Це дозволяє легко управляти змінами в структурі бази даних протягом всього циклу розробки. Міграції дозволяють автоматизувати процес оновлення бази даних, забезпечуючи узгоджені.

2.3.4 Microsoft SQL Server (MSSQL)

Microsoft SQL Server (MSSQL) є одним із провідних реляційних систем управління базами даних (РСУБД), який забезпечує високу продуктивність, надійність і безпеку для управління великими обсягами даних. Його вибір для нашої платформи обґрунтований низкою важливих переваг.

По-перше, MSSQL забезпечує високу продуктивність і масштабованість. Завдяки сучасним алгоритмам оптимізації запитів, індексації та паралельному виконанню запитів, MSSQL може ефективно обробляти великі обсяги даних та забезпечувати швидкий доступ до інформації. Це критично важливо для платформи пошуку розробників, де швидкість відгуку і доступ до даних грають ключову роль.

Другою значною перевагою MSSQL є його надійність і відмовостійкість. MSSQL підтримує різні механізми резервного копіювання та відновлення даних, що забезпечує захист від втрати інформації. Крім того, функції реплікації і кластеризації дозволяють створювати відмовостійкі системи з високою доступністю.

MSSQL також забезпечує високий рівень безпеки. Він підтримує різноманітні механізми захисту даних, включаючи шифрування на рівні стовпців, шифрування баз даних (Transparent Data Encryption), а також контроль доступу на основі ролей і прав користувачів. Це забезпечує захист конфіденційної інформації і відповідність стандартам безпеки.

Ще однією важливою перевагою MSSQL є його інтеграція з іншими продуктами Microsoft. MSSQL легко інтегрується з Microsoft Visual Studio, Entity Framework, а також іншими інструментами і сервісами, що полегшує процес розробки і підтримки додатків. Це забезпечує зручність і ефективність роботи розробників.

MSSQL також має розвинену екосистему інструментів для адміністрування і моніторингу. SQL Server Management Studio (SSMS) надає зручний інтерфейс для управління базами даних, створення запитів,

налаштування безпеки і виконання адміністративних завдань. Крім того, інструменти для моніторингу продуктивності допомагають виявляти і усувати проблеми в роботі бази даних.

2.3.5 AutoMapper

AutoMapper є потужним інструментом для автоматичного мапінгу об'єктів, який значно спрощує процес трансформації даних між різними шарами додатку. Вибір AutoMapper для нашої платформи обґрунтований кількома важливими перевагами.

По-перше, AutoMapper забезпечує автоматизацію процесу мапінгу об'єктів. Це дозволяє уникнути написання великої кількості коду для ручного мапінгу, що підвищує продуктивність розробників і зменшує ймовірність помилок. AutoMapper автоматично зіставляє поля між об'єктами з однаковими іменами, що значно спрощує процес трансформації даних.

Другою важливою перевагою AutoMapper є його гнучкість і налаштовуваність. Розробники можуть налаштовувати правила мапінгу, включаючи специфічні перетворення даних, ігнорування полів або обробку складних типів даних. Це дозволяє адаптувати AutoMapper під конкретні потреби проекту і забезпечити точність мапінгу.

AutoMapper також підтримує роботу з колекціями та ієрархічними об'єктами. Це забезпечує можливість мапінгу складних структур даних, що є важливим для платформи пошуку розробників, де необхідно обробляти великі обсяги даних з різними структурами.

Ще однією перевагою AutoMapper є його інтеграція з іншими компонентами .NET. AutoMapper легко інтегрується з Entity Framework, ASP.NET Core та іншими фреймворками, що забезпечує зручність використання і узгодженість у всій архітектурі додатку.

Крім того, AutoMapper підтримує профілі мапінгу, що дозволяють організувати і управляти правилами мапінгу в масштабних проектах. Це

забезпечує можливість легко додавати нові мапінги або змінювати існуючі, не порушуючи при цьому інші частини коду.

Таким чином, вибір AutoMapper для реалізації платформи пошуку розробників забезпечує автоматизацію, гнучкість і зручність у роботі з мапінгом об'єктів, що підвищує продуктивність розробників і забезпечує точність та ефективність трансформації даних.

2.3.6 Visual Studio

Visual Studio є потужним інтегрованим середовищем розробки (IDE), яке надає розробникам всі необхідні інструменти для створення, тестування і розгортання додатків. Його вибір для розробки нашої платформи обґрунтований кількома ключовими перевагами.

По-перше, Visual Studio підтримує широкий спектр мов програмування, включаючи C#, що є основною мовою для нашого проекту. Це забезпечує зручність і ефективність розробки, оскільки всі необхідні інструменти зосереджені в одному середовищі.

Visual Studio надає розширені можливості для написання і налагодження коду. Інструменти для автодоповнення коду (IntelliSense), рефакторингу і статичного аналізу коду допомагають підвищити продуктивність і якість розробки. Інтегрований налагоджувач дозволяє легко виявляти і виправляти помилки в коді, що зменшує час на тестування і відладку.

Ще однією важливою перевагою Visual Studio є підтримка розподіленої розробки і інтеграція з системами контролю версій, такими як Git і Azure DevOps. Це дозволяє команді розробників ефективно співпрацювати, відслідковувати зміни в коді і автоматизувати процеси збірки і розгортання.

Visual Studio також надає інструменти для тестування додатків. Інтегровані засоби для створення юніт-тестів, тестування інтерфейсів і навантажувального тестування допомагають забезпечити високу якість і надійність програмного забезпечення.

Крім того, Visual Studio підтримує розширення, що дозволяють додавати нові функції і інтегруватися з іншими інструментами і сервісами. Це забезпечує гнучкість і розширюваність середовища розробки, дозволяючи адаптувати його під конкретні потреби проекту.

3 ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Розробка структури бази даних

У даному проекті використовується підхід Code First з фреймворком Entity Framework Core (EF Core) для створення та управління структурою бази даних. Цей підхід дозволяє спочатку визначати моделі даних у коді C#, а потім автоматично створювати базу даних на основі цих моделей. Схему бази зображено на рисунку 3. Основними таблицями є:

- таблиця користувачів (User) – зберігає дані про користувачів, такі як ім’я, прізвище, пошту, опис користувача ті інші дані;
- таблиця навичків (Skill) – зберігає дані про навички, такі як назва навички;
- таблиця проектів (Project) – зберігає дані про проекти на платформі, такі як опис проекту, короткий опис проекту, автор проекту;
- таблиця відгуків (Reviews) – зберігає дані про коментарі на сторінці інших користувачів, такі як сам коментар, час створення та інші.
- таблиця запитів на приєднання до проекту (ProjectApplication) – зберігає данні, про запити на приєднання до проекту, такі як повідомлення для автора проект, статус запиту, проект до якого відправлено запит на приєднання та інші.

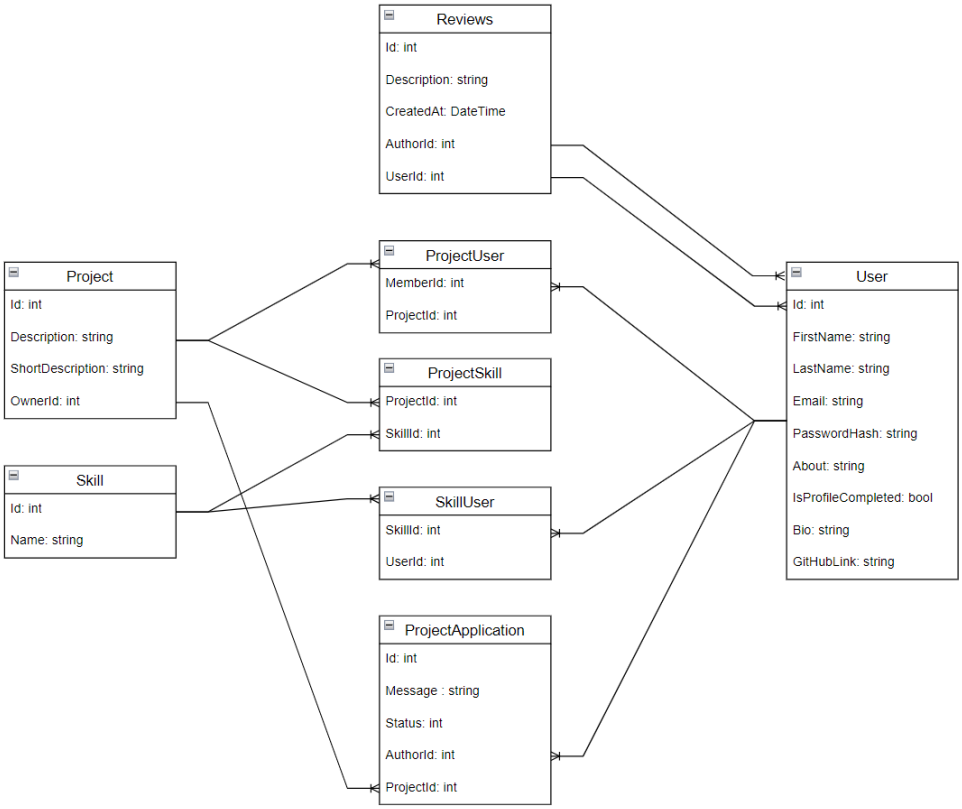


Рис. 3.1 Схема бази даних

3.2 Мапа сайту веб-платформи та порядок роботи з сайтом

Для ефективного розуміння структури та функціоналу веб-платформи, представлено карту сайту на рисунку 3.2, яка включає основні сторінки та їх взаємозв'язки. На основі наданої схеми, розглянемо кожну сторінку детальніше та опишемо порядок роботи з сайтом.

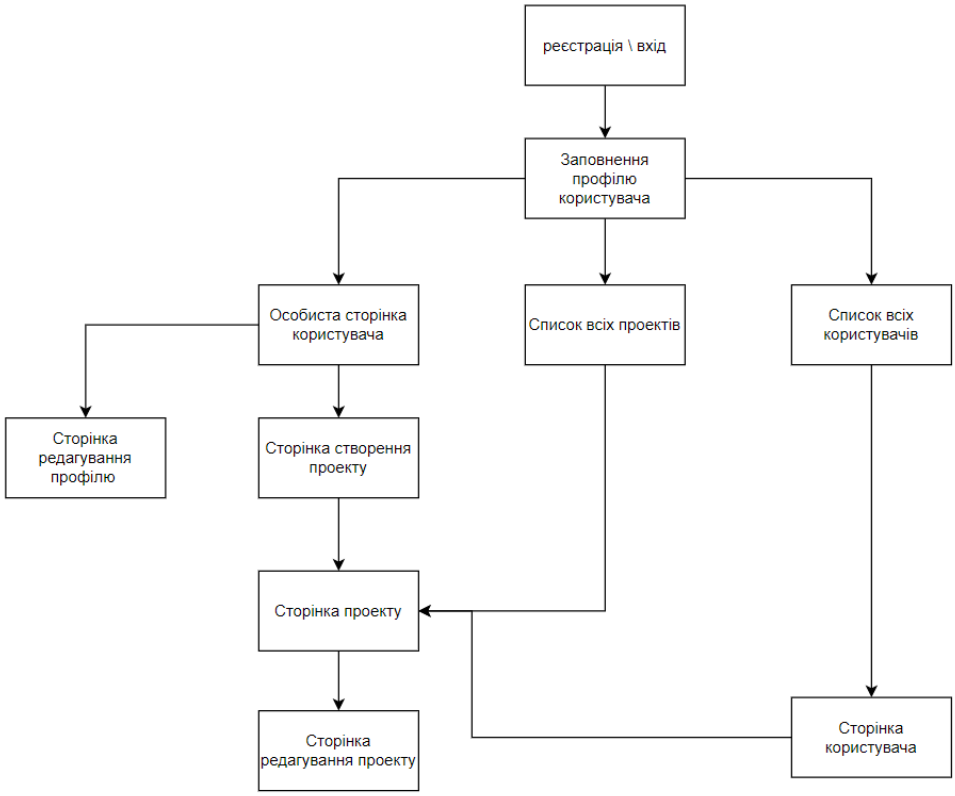


Рис. 3.2. Мапа сайту веб-платформи

Сторінка реєстрації/входу - це перший крок для користувачів, де вони можуть створити новий обліковий запис або увійти до існуючого. Користувачеві потрібно ввести основні дані, такі як ім'я, електронна пошта та пароль (рис. 3.3).

Рис. 3.3. Сторінка авторизації

Після успішної реєстрації або входу, користувач перенаправляється на сторінку Заповнення профілю (рис. 3.4). Тут користувач заповнює додаткову інформацію про себе, яка буде використовуватися для персоналізації досвіду на сайті.

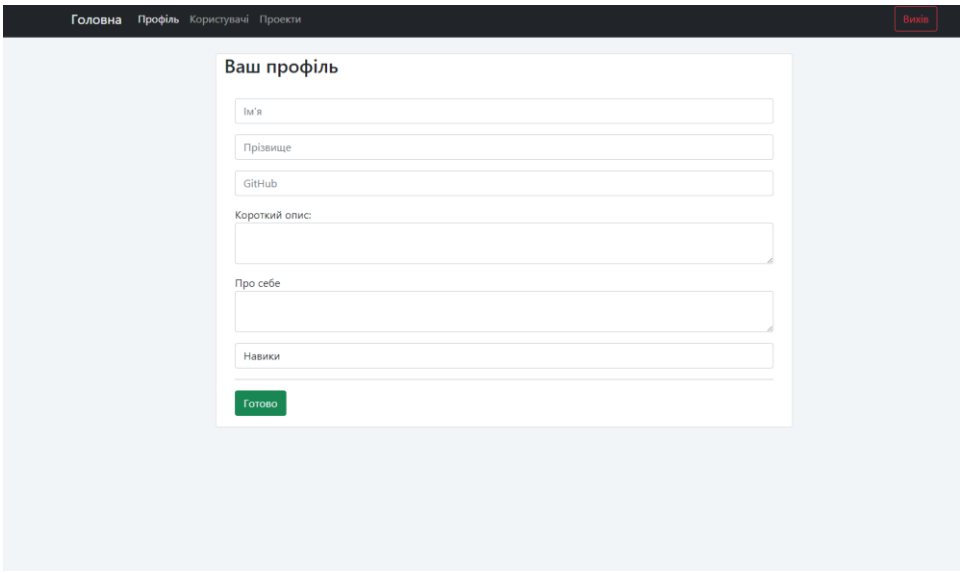


Рис. 3.4 Сторінка заповнення профілю користувача

Особиста сторінка користувача містить інформацію про поточного користувача, включаючи його проекти та персональні дані (рис. 3.5). Тут користувач може редагувати свої дані та створювати нові проекти. А також на сторінці користувача відображаються проекти до яких користувач долучений та відгуки інших користувачів.

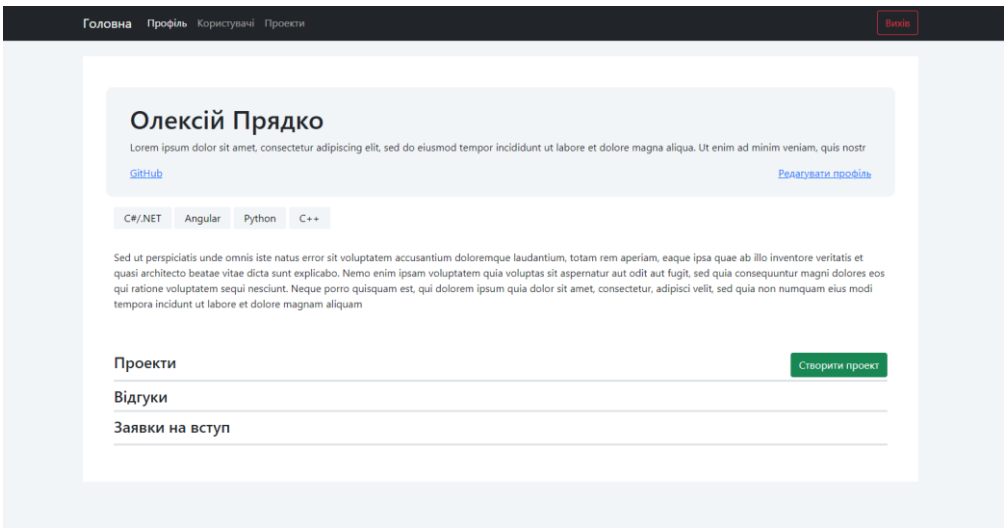


Рис. 3.5 Особиста сторінка користувача

На Сторінці редагування профілю користувач може змінювати свої персональні дані, такі як ім'я, прізвище, опис про себе, навички та інше (рис. 3.6). Це дозволяє підтримувати актуальність інформації користувача.

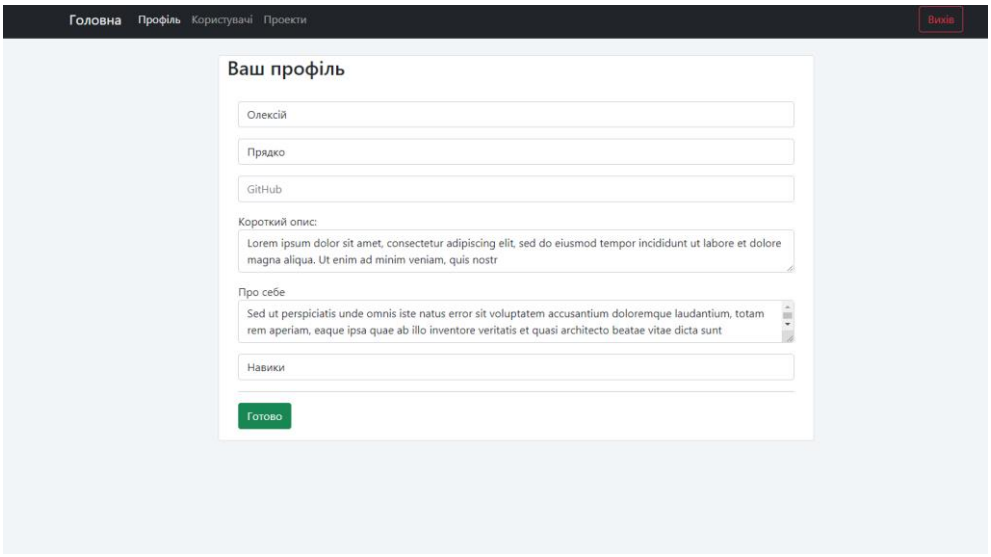


Рис. 3.6 Сторінка редагування профілю користувача

Сторінка створення проекту дозволяє користувачам створювати нові проекти, вводячи детальну інформацію про проект, включаючи назву, опис та інші параметри (рис. 3.7).

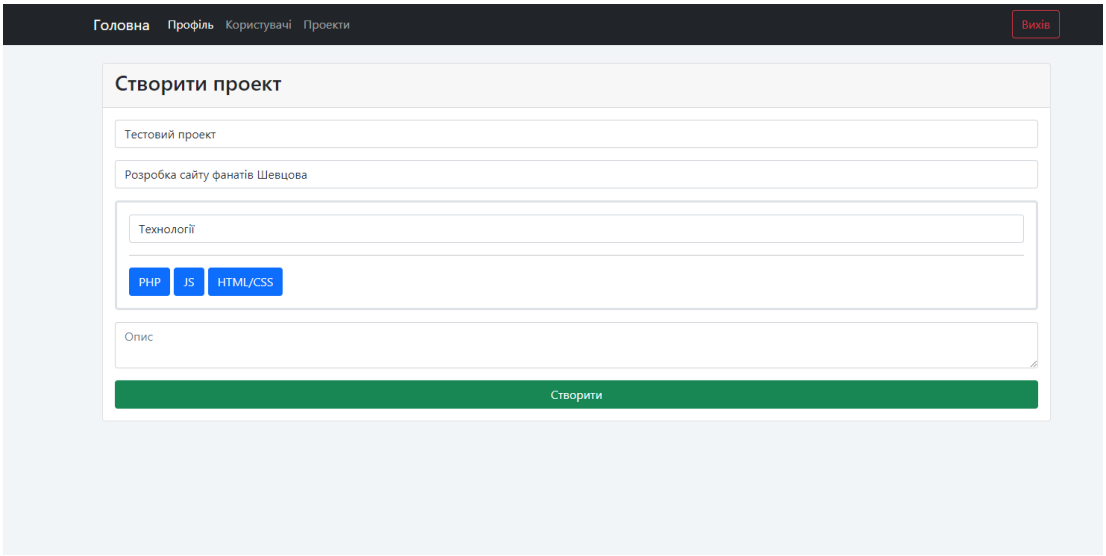


Рис. 3.7 Сторінка створення проекту

На Сторінці проекту користувачі можуть переглядати детальну інформацію про конкретний проект, включаючи опис, учасників, технології які використовуються на проекті (рис. 3.8). Тут також можуть бути доступні дії для взаємодії з проектом, такі як редагування або вихід з проекту.

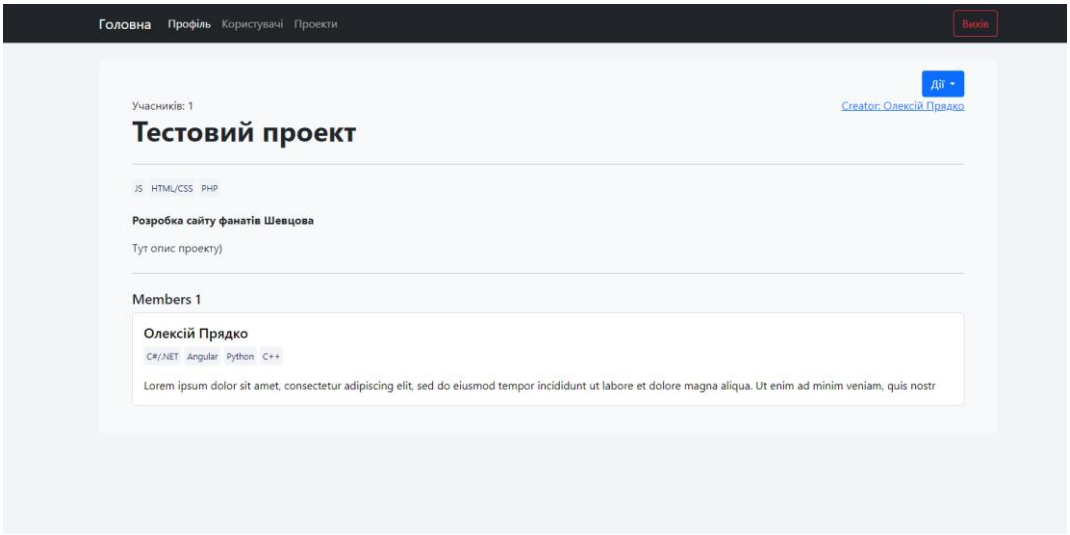


Рис. 3.8 Сторінка проекту

Сторінка з всіма проектами, дозволяє користувачам переглядати всі наявні проекти на платформі (рис. 3.9). Тут представлені всі проекти, створені користувачами, з можливістю перегляду детальної інформації про кожен проект, а також пошук за назвою.

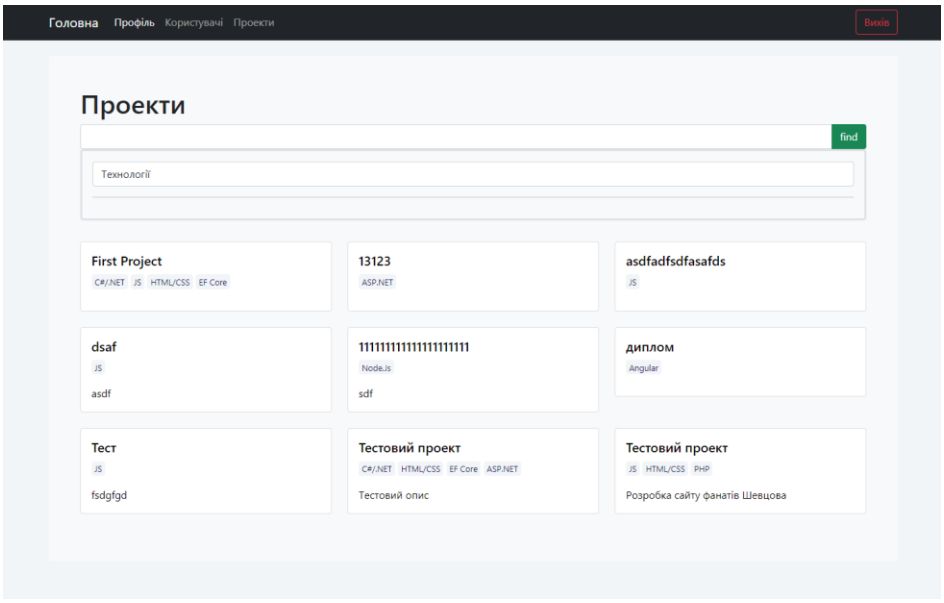


Рис. 3.9 Сторінка з всіма проектами

4 ТЕСТУВАННЯ

4.1 Тестування кросбраузерності

У сучасному світі користувачі використовують різноманітні пристрої та браузери для доступу до веб-сайтів. Щоб забезпечити правильне відображення та функціонування веб-сайту на всіх платформах, необхідно враховувати кросбраузерність.

Кросбраузерність означає, що веб-сайт повинен виглядати та функціонувати однаково добре у всіх браузерах. Це включає оптимізацію сайту для використання як у Microsoft Edge, так і у Google Chrome. Важливо протестувати сайт у різних браузерах, щоб переконатися, що дизайн і функціональність працюють належним чином для всіх користувачів.

Тестування є важливою частиною розробки веб-сайту, оскільки дозволяє перевірити його функціональність перед публікацією. Для перевірки кросбраузерності використовуються популярні браузери, такі як Google Chrome, Microsoft Edge для десктопних версій веб-сайту.

Для тестування кросбраузерності проекту вибираються сторінки з найбільшим функціоналом, наприклад, головна сторінка або сторінка зі складним інтерфейсом користувача. Використання медіа-запитів дозволяє налаштовувати відображення сторінок сайту відповідно до розмірів різних пристроїв. За допомогою CSS-правил можна контролювати стилізацію на основі ширини та висоти екрана. Після написання коду з медіа-запитами сторінка проекту буде належним чином відображатися на різних екранах.

У процесі розробки цієї платформи використовується Bootstrap, популярний фреймворк для створення адаптивних веб-інтерфейсів. Bootstrap забезпечує сумісність з різними браузерами та пристроями, що значно спрощує тестування кросбраузерності. Основні переваги використання Bootstrap включають:

- Адаптивний дизайн: Bootstrap дозволяє створювати веб-сайти, що автоматично адаптуються до різних розмірів екранів і пристроїв.
- Попередньо стилізовані компоненти: Bootstrap надає великий набір готових стилів для різних елементів інтерфейсу, що допомагає швидко створювати професійні та узгоджені дизайни.
- Сумісність з різними браузерами: Bootstrap розроблений з урахуванням сумісності з усіма основними браузерами, що зменшує ризик виникнення проблем при відображенні веб-сайту.

Рисунки для демонстрації кросбраузерності: 4.1, 4.2

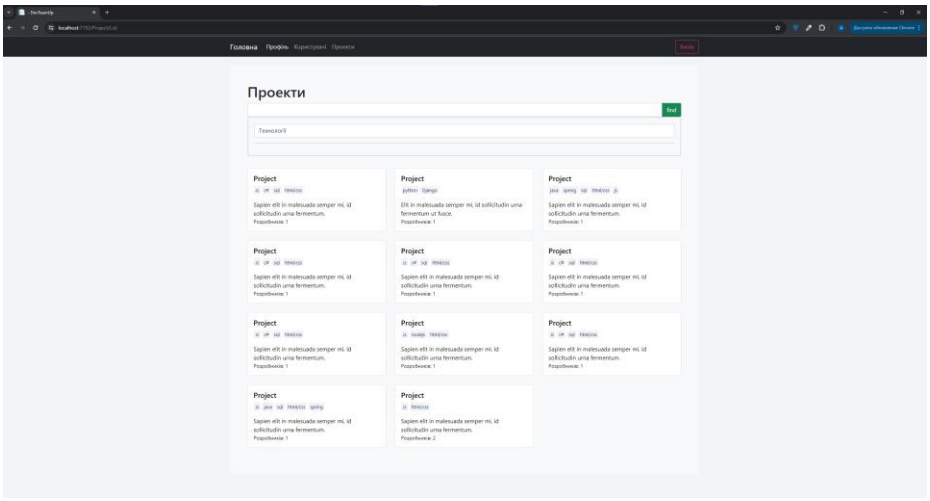


Рис. 4.1 Відображення на браузері Google Chrome

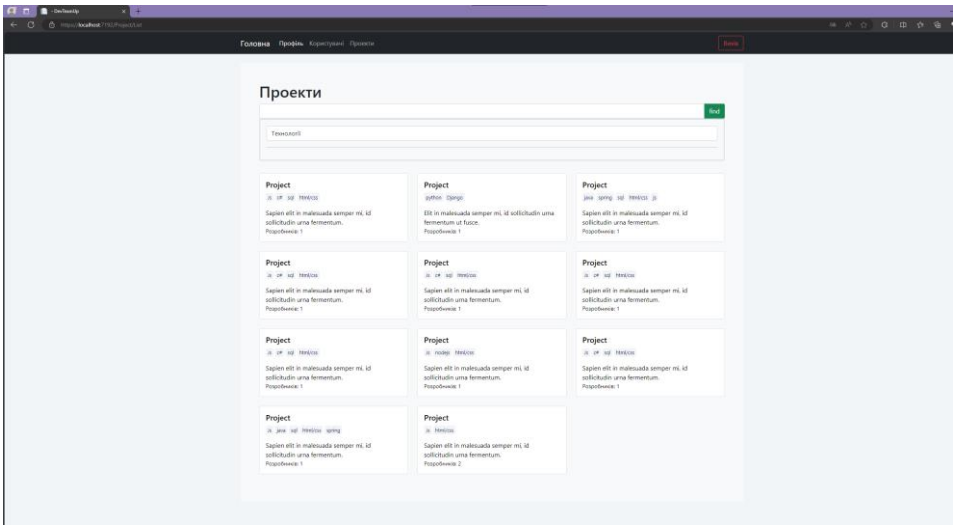


Рис. 4.2 Відображення на браузері Microsoft Edge

4.2 Тестування адаптивності веб-сайту

Адаптивність веб-сайту означає його здатність коректно відображатися та функціонувати на різних пристроях і розмірах екранів, таких як настільні комп'ютери, планшети та смартфони. У сучасному світі, де більшість користувачів використовують мобільні пристрої для доступу до інтернету, адаптивний дизайн є критично важливим для успіху веб-сайту. Адаптивний веб-дизайн дозволяє забезпечити оптимальний користувацький досвід незалежно від пристрою, який використовується для доступу до сайту.

Основні принципи адаптивного дизайну включають використання медіа-запитів CSS, гнучких макетів та гнучких зображень. Це дозволяє автоматично змінювати вигляд сайту в залежності від розмірів екрану. Використання адаптивного дизайну дозволяє уникнути створення окремих версій сайту для різних пристроїв, що значно спрощує процес розробки і підтримки.

В роботі для досягнення адаптивності використовується фреймворк Bootstrap. Bootstrap – це популярний CSS фреймворк, який забезпечує інструменти для створення адаптивних, мобільних сайтів. Він містить готові компоненти та стильові шаблони, які значно прискорюють процес розробки.

Основні можливості Bootstrap:

- Гнучка сітка (Grid System): Bootstrap використовує систему сіток, яка дозволяє створювати макети, що автоматично підлаштовуються під різні розміри екранів. Це забезпечує гарний вигляд сайту на будь-якому пристрої, будь то мобільний телефон або настільний комп'ютер.
- Медіа-запити (Media Queries): Bootstrap містить вбудовані медіа-запити, які дозволяють легко змінювати стилі в залежності від розміру екрана. Це допомагає налаштовувати вигляд сайту для різних пристроїв.
- Готові компоненти (Components): Bootstrap надає безліч готових компонентів, таких як навігаційні панелі, кнопки, форми, модальні

вікна тощо. Всі ці компоненти оптимізовані для адаптивного використання і можуть легко інтегруватися у будь-який проект.

Процес тестування адаптивності Тестування адаптивності – це важливий етап у розробці веб-сайту, який допомагає переконатися, що сайт коректно відображається на різних пристроях. У процесі тестування адаптивності використовуються наступні методи:

- Зміна розміру вікна браузера: Це простий спосіб перевірити, як сайт адаптується до різних розмірів екранів. Зміна ширини та висоти вікна браузера дозволяє спостерігати, як змінюється вигляд сайту. Важливо перевіряти сайт на різних розмірах екранів, від невеликих мобільних пристроїв до великих десктопів.
- Фізичні пристрої: Якщо є доступ до різних пристроїв, можна фізично перевірити адаптивність сайту на кожному з них. Перегляд сайту на смартфонах, планшетах, ноутбуках та інших пристроях дозволяє переконатися, що він належним чином пристосовується до різних екранів.

Практичні приклади адаптивності Приклад адаптації веб-сайту можна побачити на малюнках 4.3-4.5. На малюнку 4.3 показано відображення сайту на екрані шириною 320 пікселів, що відповідає типовому смартфону. На малюнку 4.4 відображається сайт на екрані шириною 768 пікселів, що підходить для планшетів. На малюнку 4.5 показано адаптацію сайту для екранів шириною 1440 пікселі, що є типовим для ноутбуків та настільних комп'ютерів.

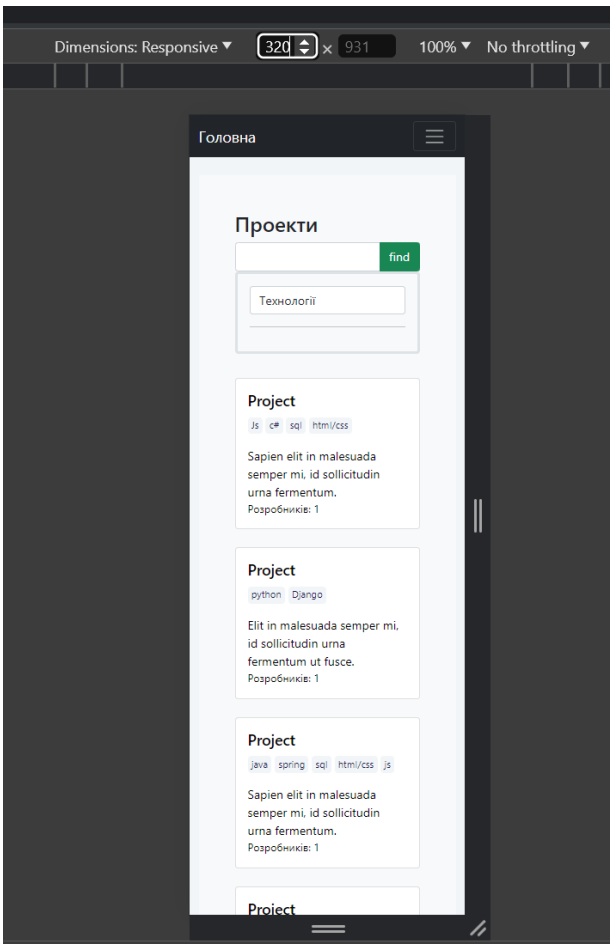


Рис. 4.3 Екрані шириною 320 пікселів

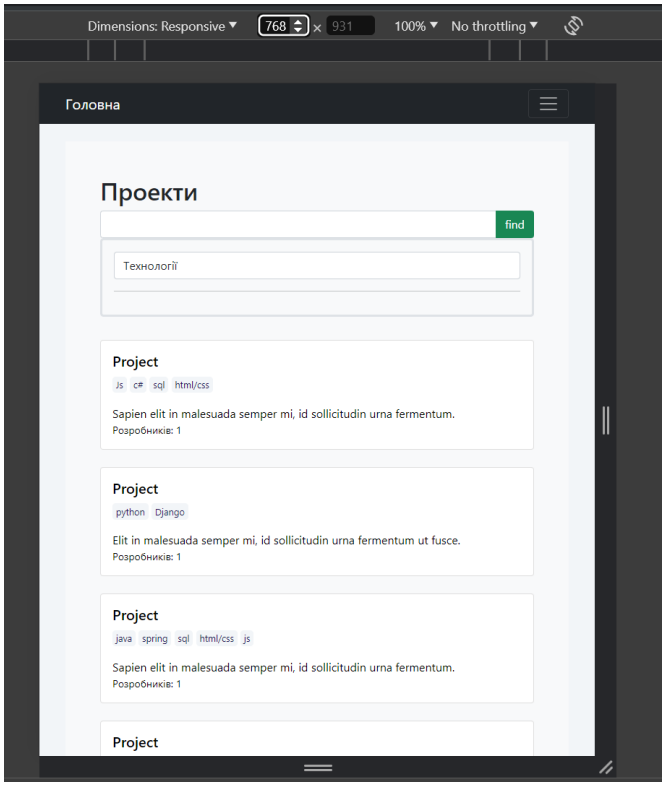


Рис. 4.4 екран шириною 768 пікселів

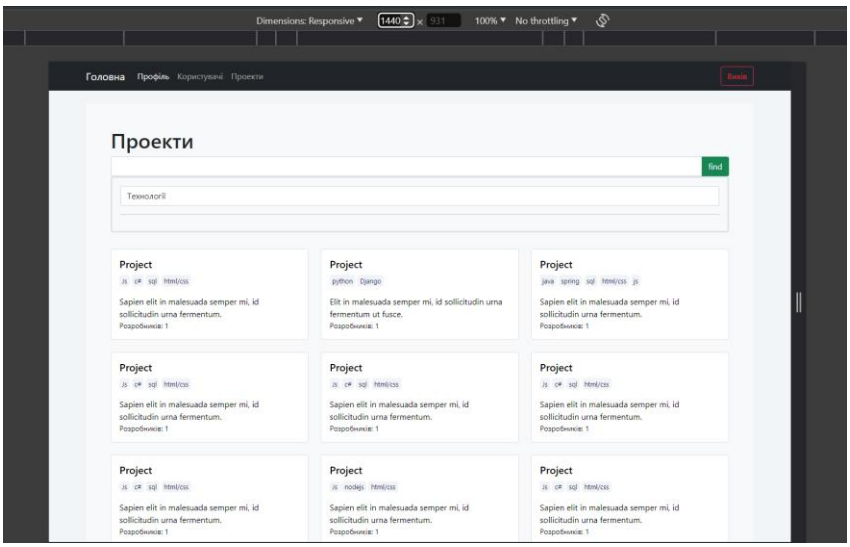


Рис. 4.5 Екран шириною 1440 пікселі

Адаптивний дизайн є ключовим аспектом сучасної веб-розробки. Використання Bootstrap дозволяє легко створювати адаптивні веб-сайти, що коректно відображаються на різних пристроях. Тестування адаптивності допомагає переконатися, що користувачі отримують найкращий досвід незалежно від того, який пристрій вони використовують. Це важливо для забезпечення зручності користувачів та успішного функціонування веб-сайту.

ВИСНОВКИ

1. Проведено аналіз обраної предметної області у сфері налагодження комунікації між розробниками.
2. Проаналізовано наявні засоби, визначено основні функції, переваги та недоліки веб-додатків.
3. Спроектовано архітектуру системи для веб-платформи.
4. Спроектовано та розроблено базу даних з використанням MSSQL.
5. Реалізовано основні функції веб-платформи, такі як створення облікового запису, пошук користувачів та проектів за технологіями, редагування профілю користувача, створення проекту з можливістю редагування, можливість долучитись до конкретного проекту, можливість залишати відгук на сторінці користувача з яким вилась співпраця над проектом. Для реалізації використовувались такі засоби як: C#, ASP.NET Core MVC, EF Core, MSSQL.
6. Проведено тестування з метою виявлення та усунення недоліків.
7. Робота пройшла апробацію на наукових конференціях, за результатами апробації опубліковано тези доповідей: Галущак А.П. Визначення вимог до вебресурсу для знаходження команди та проектів для спільної роботи над проектами // Міжнародна науково-практичної інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи» ВНТУ, 2024.

ПЕРЕЛІК ПОСИЛАНЬ

1. Overview of ASP.NET Core MVC. [Електронний ресурс] : Режим доступу до ресурсу: https://learn.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-8.0
2. Entity Framework Core. [Електронний ресурс] : Режим доступу до ресурсу: https://learn.microsoft.com/en-us/ef/core/
3. C# Docs. [Електронний ресурс] : Режим доступу до ресурсу: https://learn.microsoft.com/ru-ru/dotnet/csharp/
4. Introducing SQL Server 2022. [Електронний ресурс] : Режим доступу до ресурсу: https://www.microsoft.com/en-us/sql-server
5. Introduction to Razor Pages in ASP.NET Core. [Електронний ресурс] : Режим доступу до ресурсу: https://learn.microsoft.com/en-us/aspnet/core/razor-pages/?view=aspnetcore-7.0&tabs=visual-studio
6. Зашко Б. Переваги використання технології ASP.NET Core для створення веб-серверу [Електронний ресурс] / Б. Зашко – Режим доступу до ресурсу: https://elartu.tntu.edu.ua/bitstream/lib/34293/2/VIII_NTK_2020_Zashko_B-Advantages_of_using_ASP_NET_141.pdf
7. Razor Pages in ASP.NET Core [Електронний ресурс] // Microsoft. – 2023. – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/aspnet/core/razorpages/?view=aspnetcore-7.0&tabs=visual-studio.
8. Taylor R. "Principled Design of the Modern Web Architecture" / R. Taylor, R. Fielding. // ACM. – 2002. – С. 115–150.
9. Bootstrap Docs [Електронний ресурс]: https://getbootstrap.com/docs/5.0/getting-started/introduction/
10. ORM. [Електронний ресурс]. Режим доступу: - https://ua.wikipedia.org/wiki/ORM
11. Model–view–controller. [Електронний ресурс]. Режим доступу: - https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

12. Галущак А.П. Визначення вимог до вебресурсу для знаходження команди та проектів для спільної роботи над проектами. *Міжнародній науково-практичній інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи»* 20 травня 2024р. м. Вінниця, Вінницький Національний Технічний Університет. https://conferences.vntu.edu.ua/index.php/mn/mn2024/paper/view/21743/18000

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка Web-платформи для пошуку розробників для спільної роботи над проектами з використанням ASP.NET Core MVC

Виконав студент 4 курсу
Групи ПД-42
Галущак Артур Павлович
Керівникроботи
К.т.н., доц., доцент кафедри ІПЗ Золотухіна Оксана Анатоліївна

Київ – 2024

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи** – спростити пошуку команди в проект для спільної розробки для розробників-початківців за рахунок веб-платформи створеної мовою C#.
- **Об'єкт дослідження –** процес пошуку команди в проект для спільної розробки для розробників-початківців.
- **Предмет дослідження –** веб-платформа для пошуку команди в проект для спільної розробки для розробників-початківців.

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Провести огляд та аналіз схожих веб-додатків для пошуку розробників в проект.
2. Розробити вимоги до веб-платформи на основі попереднього аналізу аналогів.
3. Проаналізувати та обрати засоби реалізації веб-платформи для пошуку розробників для спільної роботи над проектами.
4. Спроектувати архітектури веб-платформи для пошуку розробників для спільної роботи над проектами.
5. Спроектувати та розробити базу даних веб-платформи.
6. Спроектувати та розробити веб-платформи для пошуку розробників для спільної роботи над проектами
7. Провести тестування веб-платформи.

3

АНАЛІЗ АНАЛОГІВ

Функціональні можливості	Teamfinding	DevPost	LinkedIn	DevTeamUp
Пошук проектів по технологіям	-	+	-	+
Пошук користувачі по технологіям	-	-	+	+
Сторінки користувачів	+	+	+	+
Сторінки проектів	+	+	-	+
Можливість залишати відгуки на інших користувачів	-	+	+	+

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

1. Створення нового облікового запису.
2. Доступ до власного облікового запису.
3. Пошук проектів за назвою.
4. Сторінка з інформацією про обраний проект.
5. Пошук користувачів за назвою .
6. Реєстрація та автентифікація користувача.
7. Наявність сторінки користувача .
8. Можливість залишати відгук на сторінці інших користувачів.
9. Долучення до проектів.

Нефункціональні вимоги:

1. Кросбраузерність.
2. Архітектура веб-платформи повинна забезпечувати можливість впровадження нових функцій без значних змін в інших частинах застосунку та без значних перебоїв у роботі.

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

6

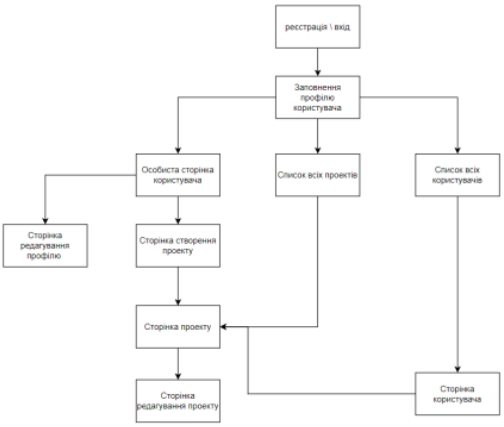
USE CASE ДІАГРАМА

7

Схема бази даних

8

Мапа сайту веб-платформи



9

ЕКРАННІ ФОРМИ

Перегляд проектів

Сторінка користувача

10

ЕКРАННІ ФОРМИ

Сторінка проекту

Сторінка створення проекту

11

ЕКРАННІ ФОРМИ

Сторінка заповнення профілю

12

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Галущак А.П. Визначення вимог до вебресурсу для знаходження команди та проектів для спільної роботи над проектами // Міжнародна науково-практичної інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи» ВНТУ, 2024.

ВИСНОВКИ

1. Проведено аналіз обраної предметної області у сфері налагодження комунікації між розробниками.
2. Проаналізовано наявні засоби, визначено основні функції, переваги та недоліки веб-додатків.
3. Спроектовано архітектуру системи для веб-платформи.
4. Спроектовано та розроблено базу даних з використанням MSSQL.
5. Реалізовано основні функції веб-платформи, такі як створення облікового запису, пошук користувачів та проектів за технологіями, редагування профілю користувача, створення проекту з можливістю редагування, можливість долучитись до конкретного проекту, можливість залишати відгук на сторінці користувача з яким вилась співпраця над проектом. Для реалізації використовувались такі засоби як: C#, ASP.NET Core MVC, EF Core, MSSQL.
6. Проведено тестування з метою виявлення та усунення недоліків.

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

Program.cs:

```
//using DevTeamUp.BLL.AutoMapper;
using DevTeamUp.AutoMapper;
using DevTeamUp.BLL.AutoMapper;
using DevTeamUp.BLL.Services;
using DevTeamUp.DAL.EF;
using DevTeamUp.DAL.EF.Entities;
using DevTeamUp.Filters;
using Microsoft.AspNetCore.Authentication.Cookies;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc.Authorization;
using Microsoft.EntityFrameworkCore;


namespace DevTeamUp
{


   public class Program
   {
      public static void Main(string[] args)
      {
         var builder =
WebApplication.CreateBuilder(args);


         builder.Services.AddControllersWithViews(
config =>
         {
            var policy = new
AuthorizationPolicyBuilder().RequireAuthenticatedUs
er().Build();
            config.Filters.Add(new
AuthorizeFilter(policy));

config.Filters.Add<ProfileCompletionFilter>();
         }).AddRazorRuntimeCompilation();


builder.Services.AddAuthentication(CookieAuthenticat
ionDefaults.AuthenticationScheme)
            .AddCookie(op =>
            {
               //TODO: write login path
               op.LogoutPath = new
PathString("/Account/Login");
            });


builder.Services.Configure<IdentityOptions>(options
=>
         {
            options.Password.RequireDigit = false;
            options.Password.RequireLowercase = false;
            options.Password.RequireNonAlphanumeric
= false;
            options.Password.RequireUppercase = false;
            options.Password.RequiredLength = 1; //
Минимальная длина пароля
            options.Password.RequiredUniqueChars = 0;
// Минимальное количество уникальных символов
в пароле
         });

         builder.Services.AddAuthorization();
         builder.Services.AddIdentity<User,
IdentityRole<int>>()

.AddEntityFrameworkStores<DataContext>()
            .AddDefaultTokenProviders();


         // DB

builder.Services.AddDbContext<DataContext>(op =>
            op.UseLazyLoadingProxies()

.UseSqlServer(builder.Configuration.GetConnectionStr
ing("cs1"))
         );

         builder.Services.AddAutoMapper(
            typeof(MappingProfile).Assembly,
            typeof(ViewModelMapper).Assembly);

         // services

builder.Services.AddTransient(typeof(ProjectService));

builder.Services.AddTransient(typeof(SkillService));

builder.Services.AddTransient(typeof(UserService));


         var app = builder.Build();

         // Configure the HTTP request pipeline.
         if (!app.Environment.IsDevelopment())
         {
            app.UseExceptionHandler("/Home/Error");

            app.UseHsts();
         }

         app.UseHttpsRedirection();
         app.UseStaticFiles();

         app.UseRouting();

         app.UseAuthentication();
         app.UseAuthorization();

         app.MapControllerRoute(
            name: "default",
            pattern:
"{controller=Home}/{action=Index}/{id?}");
```

```
            app.Run();
        }
    }

                }
```

AccountController.cs:

```
using DevTeamUp.BLL.Services;
using DevTeamUp.DAL.EF.Entities;
using DevTeamUp.Filters;
using DevTeamUp.Models;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;

namespace DevTeamUp.Controllers
{
    //[ServiceFilter(typeof(ProfileCompletionFilter))]
    [AllowAnonymous]
    public class AccountController : Controller
    {

        private readonly UserManager<User>
userManager;
        private readonly SignInManager<User>
signInManager;
        private readonly UserService userService;

        public AccountController(UserManager<User>
userManager, SignInManager<User> signInManager,
UserService userService)
        {
            this.userManager = userManager;
            this.signInManager = signInManager;
            this.userService = userService;
        }


        public IActionResult Login()
        {

            return View();
        }

        [HttpPost]
        public IActionResult Login(LoginViewModel
model)
        {

            var signInResult =
signInManager.PasswordSignInAsync(model.Email,
model.Password, true, false).Result;
            if (signInResult.Succeeded)
            {
                return RedirectToAction("Index", "Home");
            }

            ViewBag.Msg = "Невірний логін або
пароль";
            return View();
        }


        public IActionResult Register()
        {
            return View();
        }

        [HttpPost]
        public IActionResult
Register(RegisterViewModel model)
        {

            var newUser = new User
            {
                Email = model.Email,
                UserName = model.Email,
            };

            var result =
userManager.CreateAsync(newUser,
model.Password).Result;
            _ = result;

            if (result.Succeeded)
            {
                signInManager.SignInAsync(newUser,
true).Wait();
                //return RedirectToAction("Index", "Home");
                return RedirectToAction("ProfileInit",
"Profile");
            }

            ViewBag.Msg = "Схоже такий користувач
вже існує";
            return View();
        }

        public IActionResult Logout()
        {

            signInManager.SignOutAsync();

            return RedirectToAction("Login", "Account");
        }
    }
}
```

ProfileController.cs:

```
using AutoMapper;
using DevTeamUp.BLL.DTOs;
using DevTeamUp.BLL.Filters;
using DevTeamUp.BLL.Services;
using DevTeamUp.DAL.EF;
using DevTeamUp.DAL.EF.Entities;
using DevTeamUp.Models;
using DevTeamUp.Models.Profile;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
```

```
using Microsoft.AspNetCore.Mvc.Rendering;
using
Microsoft.EntityFrameworkCore.Metadata.Convention
s;

namespace DevTeamUp.Controllers
{
   public class ProfileController : Controller
   {
      private readonly ProjectService projectService;
      private readonly SkillService skillService;
      private readonly UserService userService;
      private readonly UserManager<User>
userManager;
      private readonly IMapper mapper;
      private readonly DataContext dataContext;
      //private UserDto _currentUser;
      private UserDto? currentUser
      {
         get
         {
            return
userService.GetUser(int.Parse(userManager.GetUserId
(User)));
         }
      }
      public ProfileController(ProjectService
projectService, UserManager<User> userManager,
IMapper mapper, UserService userService,
SkillService skillService, DataContext dataContext)
      {
         this.projectService = projectService;
         this.userManager = userManager;
         this.mapper = mapper;
         this.userService = userService;
         this.skillService = skillService;
         this.dataContext = dataContext;
      }

      public IActionResult Index(int? uId)
      {
         if(uId == null)
         {
            var userId =
int.Parse(userManager.GetUserId(User));
            var userProfile = userService.Profile(userId);
            userProfile.IsProfileOwner = true;

            ViewBag.MembershipApplications =
projectService.MembershipApplications(userId);

            return View(userProfile);
         }
         else
         {
            var userProfile =
userService.Profile(uId.Value);
            userProfile.Id = uId.Value;
            return View(userProfile);
         }

         //var userDto = userService.GetUser(userId);
         //var profileModel = new ProfileViewModel
         //{
         //    Username = userDto.Username,
         //};
         //profileModel.Skills =
profileModel.toSelectListItem(userDto.Skill);
         //var availableSkills =
skillService.GetSkills().ExceptBy(userDto.Skill.Select(
i=> i.Id), u => u.Id );
         //profileModel.AvailableSkills =
profileModel.toSelectListItem(availableSkills);




      }

      //[HttpGet("[controller]/{id}", Order =
int.MaxValue)]
      //public IActionResult UserProfile(int id)
      //{
      //    if(currentUser.Id == id)
      //        return RedirectToAction("Index");
      //    // мб делать проверку на собственную
страницу
      //    return View();
      //}

      public IActionResult ProfileInit()
      {

         var profile =
userService.SelfProfile(currentUser.Id);
         ProfileInitVM model =
mapper.Map<ProfileInitVM>(profile);

         model.AvailableSkills =
skillService.GetSkills().Select(s =>
               new SelectListItem(s.Name,
s.Id.ToString()))
            .ToList();


         _ = model;

         return View(model);
      }

      [HttpPost]
      public IActionResult ProfileInit(ProfileInitVM
model)
      {
         _ = model;
         model.AvailableSkills =
skillService.GetSkills().Select(s =>
               new SelectListItem(s.Name,
s.Id.ToString()))
            .ToList();
         var dto = mapper.Map<ProfileDTO>(model);
         userService.ProfileInit(dto, currentUser.Id);
         return View(model);
```

```
        }

        public IActionResult List(ProfileFilter? filter)
        {
            _ = filter;
            var profiles = userService.GetProfiles(filter);
            _ = profiles;
            return View(profiles);
        }

        public IActionResult JoinResult(int reqId, bool
status)
        {

            projectService.JoinResult(reqId, status);

            return Redirect("Index");
        }

        [HttpPost]
        public IActionResult
CreateReview(ReviewViewModel model)
        {

            Review newReview = new Review()
            {
                AuthorId = currentUser.Id,
                Description = model.Description,
                RecipientId = model.ProfileId,
                CreatedAt = DateTime.Now,
            };

            dataContext.Reviews.Add(newReview);
            dataContext.SaveChanges();

            return Redirect("Index");
        }
    }
}
```

ProjectController.cs:

```
using AutoMapper;
using DevTeamUp.BLL.DTOs;
using DevTeamUp.BLL.Filters;
using DevTeamUp.BLL.Services;
using DevTeamUp.DAL.EF.Entities;
using DevTeamUp.Models;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;

namespace DevTeamUp.Controllers
{
    public class ProjectController : Controller
    {
        private readonly ProjectService projectService;
        private readonly SkillService skillService;
        private readonly UserManager<User>
userManager;
        private readonly IMapper mapper;
        public ProjectController(ProjectService
projectService, UserManager<User> userManager,
IMapper mapper, SkillService skillService)
        {
            this.projectService = projectService;
            this.userManager = userManager;
            this.mapper = mapper;
            this.skillService = skillService;
        }

        [HttpGet("[controller]/{id}", Order =
int.MaxValue)]
        public IActionResult Index(int id)
        {
            try
            {

                var dto = projectService.GetProject(id);
                var model=
mapper.Map<ProjectPageViewModel>(dto);
                _ = model;


                var userId =
int.Parse(userManager.GetUserId(this.User));
                if (userId == model.OwnerProfile.Id)
                {
                    model.IsOwner = true;
                    model.IsMember = true;
                }
                else if(model.Members.Any(u => u.Id ==
userId))
                {
                    model.IsMember= true;
                }
                return View(model);
            }
            catch (Exception)
            {

                return Ok("Project not found");

            }

        }

        public IActionResult Test()
        {
            return RedirectToRoute("[controller]", new { id
= 11 });
        }

        public IActionResult List()
        {
            //_ = filter;
            //ProjectsListDTO projectsListDTO =
projectService.GetPage(page, filter);
            //ProjectsListViewModel model = new()
            //{
            //    Count = projectsListDTO.Projects.Count,
```

```
        //    TotalCount = projectsListDTO.TotalCount,
        //    TotalPages = projectsListDTO.TotalPages,
        //    Projects =
projectsListDTO.Projects.Select(dto => new
ProjectViewModel
        //    {
        //        Id = dto.Id,
        //        Name = dto.Name,
        //        Description = dto.Description,
        //        OwnerId = dto.OwnerId,
        //        Stack = dto.Stack.Select(skill => new
SkillViewModel {
        //            Id = skill.Id,
        //            Name = skill.Name,
        //        })
        //        .ToList()

        //    }).ToList()
        //};

        //ViewBag.AvailableTechnologies =
listItemsAvailableTechnologies();
        //return View(model);


        var projects = projectService.GetAllProjects();


        return View(projects);
    }

    public IActionResult CreateProject()
    {
        CreateProjectViewModel model = new()
        {
            Skills = listItemsAvailableTechnologies()
        };

        return View(model);
    }

    [HttpPost]
    public IActionResult
CreateProject(CreateProjectViewModel model)
    {
        if(ModelState.IsValid)
        {
            var userId =
int.Parse(userManager.GetUserId(this.User));
            //var projectDto = new CreatedProjectDTO
            //{
            //    Name = model.Name,
            //    Description = model.Description,
            //    SkillsIds = model.SelectedSkillsIds
            //};

            var dto =
mapper.Map<CreatedProjectDTO>(model);
            _ = dto;
            var newProject =
projectService.CreateProject(dto, userId);
            _ = newProject;
            return RedirectToAction("Index");
        }

        return View(model);
    }

    [HttpPost]
    public IActionResult
JoinToProject(RequestJoinViewModel model)
    {
        try
        {
            _ = model;
            var userId =
int.Parse(userManager.GetUserId(this.User));

projectService.JoinToProject(model.ProjectId, userId,
model.Message);


            return RedirectToAction("Index", "Profile");
            //return RedirectToRoute("[controller]", new
{ id = project.Id });
        }
        catch (Exception)
        {


        }

        return Ok("error");
    }

    private IList<SelectListItem>
listItemsAvailableTechnologies()
    {
        var availableTechnologies = skillService
            .GetSkills()
            .Select(s => new SelectListItem
            {
                Text = s.Name,
                Value = s.Id.ToString()
            }).ToList();
        return availableTechnologies;
    }

    public IActionResult Leave(int projectId)
    {
        var userId =
int.Parse(userManager.GetUserId(this.User));
        projectService.LeaveProject(userId, projectId);
        return RedirectToAction("List");
    }
  }
}
```

Login.cshtml:

```
@{
  Layout = "_Anonim";
}
@model LoginViewModel;
```

```
<div class="container mt-5">

   <div class="card">
      <div class="card-header">
         <h2>Вхід</h2>
      </div>
      <div class="card-body">
         <div class="card-subtitle text-danger mb-3">
            @ViewBag.Msg
         </div>
         <form asp-action="Login" method="post">
            <div class="mb-3">
               <label for="exampleInputEmail1"
class="form-label">Пошта</label>
               <input name="email" type="email"
class="form-control" id="exampleInputEmail1" aria-
describedby="emailHelp">

            </div>
            <div class="mb-3">
               <label for="exampleInputPassword1"
class="form-label">Пароль</label>
               <input name="password"
type="password" class="form-control"
id="exampleInputPassword1">
            </div>
            <div class="mb-3">

               <a asp-action="Register">Реєстрація</a>
            </div>
            <button type="submit" class="btn btn-
primary">Війти</button>
         </form>

      </div>
   </div>


</div>
```

```
      </div>
      <div class="card-body">
         <div class="card-subtitle text-danger mb-3">
            @ViewBag.Msg
         </div>
         <form asp-action="Register" method="post">
            @*<div class="mb-3">
            <label class="form-label">Username</label>
            <input name="username" type="email"
class="form-control" id="Username" >
            <div id="emailHelp" class="form-
text">We'll never share your email with anyone
else.</div>
            </div>*@
            <div class="mb-3">
               <label for="exampleInputEmail1"
class="form-label">Пошта</label>
               <input name="email" type="email"
class="form-control" id="exampleInputEmail1" aria-
describedby="emailHelp">

            </div>
            <div class="mb-3">
               <label for="exampleInputPassword1"
class="form-label">Пароль</label>
               <input name="password"
type="password" class="form-control"
id="exampleInputPassword1">
            </div>
            <div class="mb-3">
               <a asp-action="Login">Вхід</a>
            </div>
            <button type="submit" class="btn btn-
primary">Реєстрація</button>
         </form>



      </div>
   </div>


</div>
```

Register.cshtml:

```
@model RegisterViewModel

@{
   Layout = "_Anonim";
}


<div class="container mt-5">

   <div class="card">
      <div class="card-header">
         <h2>Реєстрація</h2>
```

Project/Index.cshtml:

```
@model ProjectPageViewModel

<div class="container bg-light p-3 px-5">
   <div class="row">
      <div class="col ">
         <div class="d-flex justify-content-end ">


            <div class="btn-group ">
```

```
                <button type="button" class="btn btn-
primary dropdown-toggle" data-bs-toggle="dropdown"
aria-expanded="false">
                    Дії
                </button>
                <ul class="dropdown-menu">
                    <li><a class="dropdown-item"
href="#">Action</a></li>
                    @if (Model.IsOwner)
                    {
                    <li><a class="dropdown-item"
href="#">Редагувати проект</a></li>

                    }
                    <li><hr class="dropdown-
divider"></li>
                    @if (Model.IsOwner)
                    {
                        <li><a class="dropdown-item text-
danger" href="#">Delete</a></li>
                    }
                    @if (Model.IsMember)
                    {
                        <li><a class="dropdown-item text-
danger"
href="/Project/Leave?projectId=@Model.Id">Leave</
a></li>
                    }

                </ul>
            </div>
        </div>

    </div>

</div>
<div class="row">
    <div class="col">
        <span>Учасників:
@Model.Members.Count</span>
        <div>
            <h1 class="fw-bold">@Model.Name</h1>

        </div>
    </div>
    <div class="col d-flex justify-content-end ">
        <a
href="/Profile/@Model.OwnerProfile.Id">Creator:
@Model.OwnerProfile.FirstName
@Model.OwnerProfile.LastName</a>
    </div>
</div>
<hr />
<div class="row">
    <div class="col">
        <div class="stack">
            <ul class="project-skills ps-0 mt-1">
                @foreach (var item in Model.Stack)
                {
                    <li class="project-skill-item p-
1">@item.Name</li>
                }
            </ul>
        </div>
    </div>
</div>
<div class="row ">
    <div class="shortDescription fw-bolder">
        @Model.ShortDescription
    </div>
    <div class="description mt-3" >
        @Model.Description
    </div>
</div>
<div class="row">
    <div class="col">
        <hr class="my-4"/>
        <h5>Members @Model.Members.Count </h5>

        <div class="members">
            @foreach (var member in Model.Members)
            {
                <a href="/Profile?uId=@member.Id"
class="text-decoration-none text-black">

                    <div class="profile-list-item card mb-4
box-shadow position-relative ">
                        <div class="card-body">
                            <h5 class="card-
title">@member.FirstName
@member.LastName</h5>
                            <div>
                                <ul class="project-skills ps-0
mt-1">
                                    @foreach (var item in
member.Skills)
                                    {
                                        <li class="project-skill-item
p-1">@item.Name</li>
                                    }
                                </ul>
                            </div>
                            <div class="card-subtitle">
                                @member.Bio
                            </div>
                            <div class="d-flex justify-content-
between align-items-center">


                            </div>
                        </div>
                    </div>
                </a>

            }
        </div>
    </div>
</div>
@if (!Model.IsMember)
{

    <div class="row mt-5">
        <div class="col">
            <div class="card">
                <div class="card-header">
```

```
                <h3 class="card-title">Запит на
вступ</h3>
            </div>
            <div class="card-body">
              <form asp-action="JoinToProject" asp-
controller="Project" method="post">
                <textarea class="form-control"
name="message"></textarea>
                <input class="btn btn-primary mt-2
w-100" type="submit" value="Надіслати запит" />
                <input type="hidden"
value="@Model.Id" name="ProjectId"  />
              </form>
            </div>
          </div>


        </div>
      </div>
    }

</div>
```

ProjectService:

```
using AutoMapper;
using Azure.Core;
using DevTeamUp.BLL.DTOs;
using DevTeamUp.BLL.Filters;
using DevTeamUp.DAL.EF;
using DevTeamUp.DAL.EF.Entities;
using Microsoft.EntityFrameworkCore;
using Microsoft.IdentityModel.Tokens;
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DevTeamUp.BLL.Services
{
  public class ProjectService
  {
    private const int pageSize = 5;

    private readonly DataContext _dataContext;
    private readonly IMapper _mapper;
    public ProjectService(DataContext dataContext,
IMapper mapper)
    {
      _dataContext = dataContext;
      _mapper = mapper;
    }

    public ProjectDTO
CreateProject(CreatedProjectDTO createdProjectDTO,
int userId)
    {
      var user = _dataContext.Users.First(u => u.Id
== userId);
      var eProject =
_mapper.Map<Project>(createdProjectDTO);
      eProject.OwnerId = userId;
      eProject.Stack = _dataContext.Skills
        .Where(s =>
createdProjectDTO.SkillsIds.Contains(s.Id))
        .ToList();

      _dataContext.Projects.Add(eProject);
      eProject.Members = new List<User>(new[] {
user });
      _dataContext.SaveChanges();


      var createdProject = _dataContext.Projects
        .Include(p => p.Stack)
        .Include(p => p.Owner)
        .First(p => p.Id == eProject.Id);

      var projectDTO =
_mapper.Map<ProjectDTO>(createdProject);
      return projectDTO;
    }

    public IEnumerable<ProjectDTO>
GetAllProjects()
    {
      //return _dataContext.Projects.Select(p => new
ProjectDTO {
      //    Id = p.Id,
      //    Name = p.Name,
      //    Description = p.Description,
      //    OwnerId = p.OwnerId,
      //}).ToList();

      return
_mapper.Map<IList<ProjectDTO>>(_dataContext.Proj
ects.Include(p => p.Stack).ToList());

    }
    public IEnumerable<ProjectDTO>
OwnersProjects(int userId)
    {

      var projects = _dataContext.Projects
        .Include(p => p.Stack)
        .Where(p => p.OwnerId == userId).ToList();
      return
_mapper.Map<IList<ProjectDTO>>(projects);

    }

    public ProjectsListDTO GetPage(int pageIndex,
ProjectsFilter? filter)
    {
      // Пример того, как должно работать. SQL
запрос работает отлично
      //  select*
      //  from Projects as p, ProjectSkill as ps
      //  where p.id = ps.ProjectsId
      //      and ps.StackId = (select Id from Skills
where Skills.[Name] = N'C#/.NET')
            //      and p.[Name] like '%dima1%'
```

```
        var queryExpression = _dataContext.Projects
            .Include(p => p.Stack)
            .AsQueryable();

        if
(!String.IsNullOrWhiteSpace(filter?.Keyword))
            queryExpression = queryExpression.Where(p
=> p.Name.Contains(filter.Keyword));

        // Выбираем все проекты, если у них есть
хотя бы 1 технология из фильтра
        if (filter?.TechnologyIds != null &&
filter.TechnologyIds.Any())
            queryExpression = queryExpression.Where(p
=> p.Stack.Any(t =>
filter.TechnologyIds.Contains(t.Id)));

        // Хахахаха, у меня просто в фильтры не
заполнялся Keyword
        //if
(!String.IsNullOrWhiteSpace(filter?.Keyword))
        //{
        //    queryExpression =
queryExpression.Where(p =>
p.Name.Contains(filter.Keyword));
        //    if (filter?.TechnologyIds != null &&
filter.TechnologyIds.Any())
        //    {
        //        queryExpression =
queryExpression.Where(p => p.Stack.Any(t =>
filter.TechnologyIds.Contains(t.Id)));
        //    }
        //}
        //else if (filter?.TechnologyIds != null &&
filter.TechnologyIds.Any())
        //{
        //    queryExpression =
queryExpression.Where(p => p.Stack.Any(t =>
filter.TechnologyIds.Contains(t.Id)));
        //}

        var projectsPage =
queryExpression.Skip(pageSize * (pageIndex -
1)).Take(pageSize).ToList();

        var result =
_mapper.Map<IList<ProjectDTO>>(projectsPage);

        // TODO: add total items, totalPages and
pageSize
        var totalCount = _dataContext.Projects.Count();
        return new ProjectsListDTO
        {
            Projects = result,
            TotalCount = totalCount,
            TotalPages =
(int)Math.Ceiling((double)totalCount / pageSize)
        };
    }

    public void JoinToProject(int projectId, int userId,
string message)
    {
        var project =
_dataContext.Projects.FirstOrDefault(p => p.Id ==
projectId);
        if (project == null)
            throw new ArgumentException("Такого
проекту не існує");

        var user = _dataContext.Users.FirstOrDefault(u
=> u.Id == userId);

        if (user == null) throw new
AggregateException("Невідомий користувач");
        if (project.Members.Contains(user)) throw new
ArgumentException("Ви вже в цьому проекті");


        ProjectApplication projectApplication = new
ProjectApplication()
        {
            AuthorId = userId,
            Message = message,
            ProjectId = projectId,
        };

_dataContext.ProjectApplications.Add(projectApplicat
ion);

        _dataContext.SaveChanges();
    }

    public IList<ProjectApplication>
MembershipApplications(int userId)
    {
        var user = _dataContext.Users.FirstOrDefault(u
=> u.Id == userId);

        return
_dataContext.ProjectApplications.Where(r =>
user.ProjectsOwner.Select(p =>
p.Id).Contains(r.ProjectId)).Where(r => r.Status ==
ProjectApplicationStatus.pending).ToList();
    }

    public void JoinResult(int requestId , bool status)
    {
        var request =
_dataContext.ProjectApplications.First(j => j.Id ==
requestId);
        if(status == true)
        {
            request.Status =
ProjectApplicationStatus.accepted;
            var user = _dataContext.Users.First(u => u.Id
== request.AuthorId);
            _dataContext.Projects.First(p => p.Id ==
request.ProjectId).Members.Add(user);
        }
        else
        {
            request.Status =
ProjectApplicationStatus.rejected;
        }
```

```
            _dataContext.SaveChanges();
        }

        public IEnumerable<ProjectDTO>
GetProjectByUser(int userId)
        {
            var user = _dataContext.Users
                .Include(u => u.ProjectsMember)
                .FirstOrDefault(u => u.Id == userId);
            if(user == null)
                throw new ArgumentException("User not
found");

            var projects =
_mapper.Map<IEnumerable<ProjectDTO>>(user.Proje
ctsMember);

            return projects;
        }

        public bool Delete(int projectId)
        {


            throw new NotImplementedException();
        }

        public ProjectPageDTO GetProject(int projectId)
        {
            var project = _dataContext.Projects.First(p =>
p.Id == projectId);

            var dto =
_mapper.Map<ProjectPageDTO>(project);

            return dto;
        }

        public void LeaveProject(int userId, int projectId)
        {
            var project =
_dataContext.Projects.FirstOrDefault(p => p.Id ==
projectId);
            if (project == null)
                throw new ArgumentException("Такого
проекту не існує");

            var user = _dataContext.Users.FirstOrDefault(u
=> u.Id == userId);

            if (user == null) throw new
AggregateException("Невідомий користувач");

            if(user.Id != project.OwnerId)
            {
                project.Members.Remove(user);
                _dataContext.SaveChanges();
            }
        }
    }
}
```

UserService.cs:

```
using AutoMapper;
using DevTeamUp.BLL.DTOs;
using DevTeamUp.BLL.Filters;
using DevTeamUp.DAL.EF;
using DevTeamUp.DAL.EF.Entities;
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Security;
using System.Text;
using System.Threading.Tasks;

namespace DevTeamUp.BLL.Services
{
    public class UserService
    {
        private readonly DataContext _dataContext;
        private readonly IMapper _mapper;

        public UserService(DataContext dataContext,
IMapper mapper)
        {
            _dataContext = dataContext;
            _mapper = mapper;
        }

        public UserDto? GetUser(int id)
        {

            // Может быть проблема при мапинге User to
            // UserDTO в части UserDTO.ProjectsMember
-> ProjectDTO.Stack
            // Возможно слишком глубокий маппинг
            var user = _dataContext.Users
                .Include( u => u.Skills)
                .Include( u => u.ProjectsMember)
                .FirstOrDefault(x => x.Id == id);

            if (user == null)
                return null;

            var dto = _mapper.Map<UserDto>(user);
            return dto;
        }

        public ProfilePageDTO Profile(int userId)
        {
            var user = _dataContext.Users
                .Include(u => u.Skills)
                .Include(u => u.CommentsForUser )
                .Include(u => u.ProjectsMember)
                    .ThenInclude(project => project.Stack)

                .FirstOrDefault(x => x.Id == userId);

            if (user == null) throw new
ArgumentException("User not found");
            var profile = new ProfilePageDTO();
            profile.FirstName = user.FirstName;
            profile.LastName = user.LastName;
```

```
            profile.About = user.About;
            profile.Bio = user.Bio;
            profile.Skills = user.Skills.Select(s => new
SkillDTO
            {
                Id = s.Id,
                Name = s.Name,
            });

            profile.Projects =
_mapper.Map<IEnumerable<ProjectDTO>>(user.Proje
ctsMember);
            profile.Reviews =
_mapper.Map<IEnumerable<ReviewDTO>>(user.Com
mentsForUser);

            return profile;
        }

        public ProfileDTO SelfProfile(int userId)
        {
            var user = _dataContext.Users.FirstOrDefault(u
=> u.Id == userId);
            if (user == null)
                throw new ArgumentException();

            return _mapper.Map<ProfileDTO>(user);
        }


        public UserDto ChangeSkills(int id, IList<int>
skillsIds)
        {
            var user = _dataContext.Users.Include( u =>
u.Skills).First(x => x.Id == id);

            user.Skills.Clear();
            user.Skills = _dataContext.Skills.Where(s =>
skillsIds.Contains(s.Id)).ToList();


            _dataContext.SaveChanges();

            return _mapper.Map<UserDto>(user);
        }

        public bool IsProfileCompleted(int userId)
        {
            var user = _dataContext.Users.FirstOrDefault(u
=> u.Id == userId);

            if (user == null)
                throw new ArgumentException();

            return user.IsProfileCompleted;
        }

        public void ProfileInit(ProfileDTO profile, int
userId)
        {
            var currentUser = _dataContext.Users.First(u
=> userId == u.Id);
            currentUser.FirstName = profile.FirstName;
            currentUser.LastName = profile.LastName;
            currentUser.About = profile.About;
            currentUser.GitHubLink = profile.GitHubLink;
            currentUser.Bio = profile.Bio;
            //currentUser.Skills =

            var skillIds = profile.Skills.Select(s =>
s.Id).ToList();
            var res = _dataContext.Skills
                .Where(s => skillIds.Contains(s.Id))
                .ToList();



            _ = res;
            currentUser.Skills.Clear();
            currentUser.Skills = res;
            currentUser.IsProfileCompleted = true;
            _dataContext.SaveChanges();
            //currentUser.FirstName = profile.FirstName;
            //currentUser.LastName = profile.LastName;
            //currentUser.About = profile.About;
            //currentUser.Skills = _dataContext.Skills
            //    .Where(s =>
profile.TechnologiesIds.Contains(s.Id))
            //    .ToList();

            //currentUser.IsProfileCompleted = true;
            //_dataContext.SaveChanges();
        }

        public IList<ProfileListItemDTO >
GetProfiles(ProfileFilter? filter)
        {
            var query = _dataContext.Users.Where(u =>
u.IsProfileCompleted);
            if(filter != null &&
!string.IsNullOrEmpty(filter?.Query))
            {
                query = query.Where(u =>
u.FirstName.ToLower().Contains(filter.Query) ||

u.LastName.ToLower().Contains(filter.Query)
                );
            }

            var users = query.ToList();

            var profiles =
_mapper.Map<IList<ProfileListItemDTO>>(users);

            return profiles;
        }
    }
}
```

DataContext.cs:

```
using DevTeamUp.DAL.EF.Entities;
using Microsoft.AspNetCore.Identity;
```

```
using
Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DevTeamUp.DAL.EF
{
   public class DataContext : IdentityDbContext<User,
IdentityRole<int>, int >
   {
      public DataContext(DbContextOptions op)
         :base(op)
      {

      }

      public DbSet<Skill> Skills { get; set; }
      public DbSet<Review> Reviews { get; set; }
      public DbSet<Project> Projects { get; set; }
      public DbSet<ProjectApplication>
ProjectApplications { get; set; }
   }
}
```