Комп'ютерна Академія IT STEP
Професійна комп'ютерна освіта
Кафедра розробки програмного забезпечення

ПОЯСНЮВАЛЬНА ЗАПИСКА ДО ДИПЛОМНОГО ПРОЕКТУ
на тему: "Аналог Booking і Airbnb"

Розробила команда гр. ПВ-123 — Вичкін Д.О.
Батюк Д.С.
Кізілпінар С.М.

Ментор ДП — Рубан С.А.

Кривий Ріг
2024

ЗМІСТ

	Вступ	3
1	**Постановка завдання**	4
1.1	Огляд концепції проекту	4
1.2	Обґрунтування вибору теми	4
1.3	Основні функціональні можливості проекту	4
1.4	Архітектура системи	6
1.5	Очікуваний результат	8
2	**Проектування системи**	8
2.1	Система управляння базою даних	9
2.2	Вибір мови програмування	11
2.3	Додаткові технології	14
2.4	GIT	16
2.5	Deployment & CI/CD	17
2.6	Опис API і взаємодії між клієнтом і сервером	19
3	**Інтерфейс користувача**	19
3.1	Дизайн інтерфейсу користувача, включаючи wireframes та макети сторінок	19
3.2	UX/UI розробка та її вплив на взаємодію з користувачем	20
4	**Принцип праці програми**	20
4.1	Створення бази даних	20
4.1.1	Інструкція по використанню Migrations у Entity Framework Core	24
4.2	Загальний опис роботи `MappingProfile`	25
4.3	Загальний опис роботи Models -> DTOs	26
4.4	Загальний опис роботи Functions	30
4.5	Загальний опис роботи AzureClientFactoryBuilderExtensions	32
4.6	Загальний опис роботи Services	33
4.7	Загальний опис роботи Controllers	35
4.8	Загальний опис роботи Frontend на основі React	39
5	**Безпека**	
6	**Деплоймент**	
6.1	Інструкції для розгортання проекту	
6.1.1	Підготовка до розгортання	
6.1.2	Розгортання на сервері	
6.1.3	Перевірка розгортання	

6.2 Опис CI/CD процесів
6.2.1 CI процеси
6.2.2 CD процеси
7 Інструкція користування для користувача
8 Плани на майбутнє
Висновок
Список використаної літератури

Вступ

У сучасному світі платформи для бронювання житла онлайн змінюють наше розуміння подорожей та проживання поза домом. Вони пропонують небачені досі можливості для експлорації нових місць з комфортом, надаючи доступ до мільйонів варіантів помешкань по всьому світу за декілька кліків. Від квартир у міських джунглях до усамітнених шале в горах, ці платформи дозволяють кожному знайти ідеальне місце для відпочинку або роботи, виходячи з індивідуальних потреб та бажань.

Цей дипломний проект, "Аналог Booking і Airbnb", створено з метою розширення можливостей сучасного ринку оренди житла через інтернет. Задум проекту полягає в створенні надійної, інтуїтивно зрозумілої та легкої у використанні платформи, яка задовольнятиме потреби як орендодавців, так і мандрівників, забезпечуючи їм зручний засіб для здійснення транзакцій.

Ми пропонуємо сучасні рішення для вирішення традиційних завдань, зокрема:

- Автоматизація процесу бронювання та управління нерухомістю;
- Покращення інтерфейсу та користувацького досвіду;
- Використання передових технологій для забезпечення безпеки транзакцій;
- Запровадження інтелектуальних алгоритмів для особистих рекомендацій та пропозицій.

У наступних розділах цього звіту ми розглянемо ключові аспекти проекту, включаючи його концепцію, технологічний стек, архітектуру системи, а також механізми забезпечення безпеки та приватності даних. Також будуть описані основні функціональні можливості платформи та подальші перспективи її розвитку.

1 Постановка завдання

1.1 Огляд концепції проекту

Проект "Аналог Booking і Airbnb" є веб-платформою, створеною для забезпечення вигідного та зручного способу знайти та забронювати житло в різних куточках світу. Платформа дозволяє власникам нерухомості публікувати оголошення про оренду, а мандрівникам - легко знаходити оптимальні варіанти на основі власних критеріїв, таких як ціна, розташування, комфортність та наявність певних зручностей.

1.2 Обґрунтування вибору теми

Тема проекту була обрана через стрімкий розвиток цифрової економіки та зростання попиту на альтернативне житло, що може пропонувати більш особистісний і гнучкий досвід проживання порівняно з традиційними готелями. Це також відповідає сучасним тенденціям у сфері туризму, де гості шукають унікальні, автентичні варіанти проживання, що дозволяють глибше зануритися в місцеву культуру.

1.3 Основні функціональні можливості проекту

Пошук та фільтрація об'єктів: Користувачі можуть шукати об'єкти за різними параметрами, включно з розташуванням, ціною, типом житла, кількістю кімнат та іншими умовами. Система фільтрації дозволяє звузити пошук до найбільш підходящих варіантів.

Бронювання онлайн: Проект надає простий інтерфейс для бронювання житла. Користувачі можуть вибрати дати перебування та здійснити бронювання з автоматичним підтвердженням.

Управління об'єктами: Орендодавці мають змогу керувати своїми об'єктами через платформу, оновлювати інформацію про житло, умови проживання та доступність.

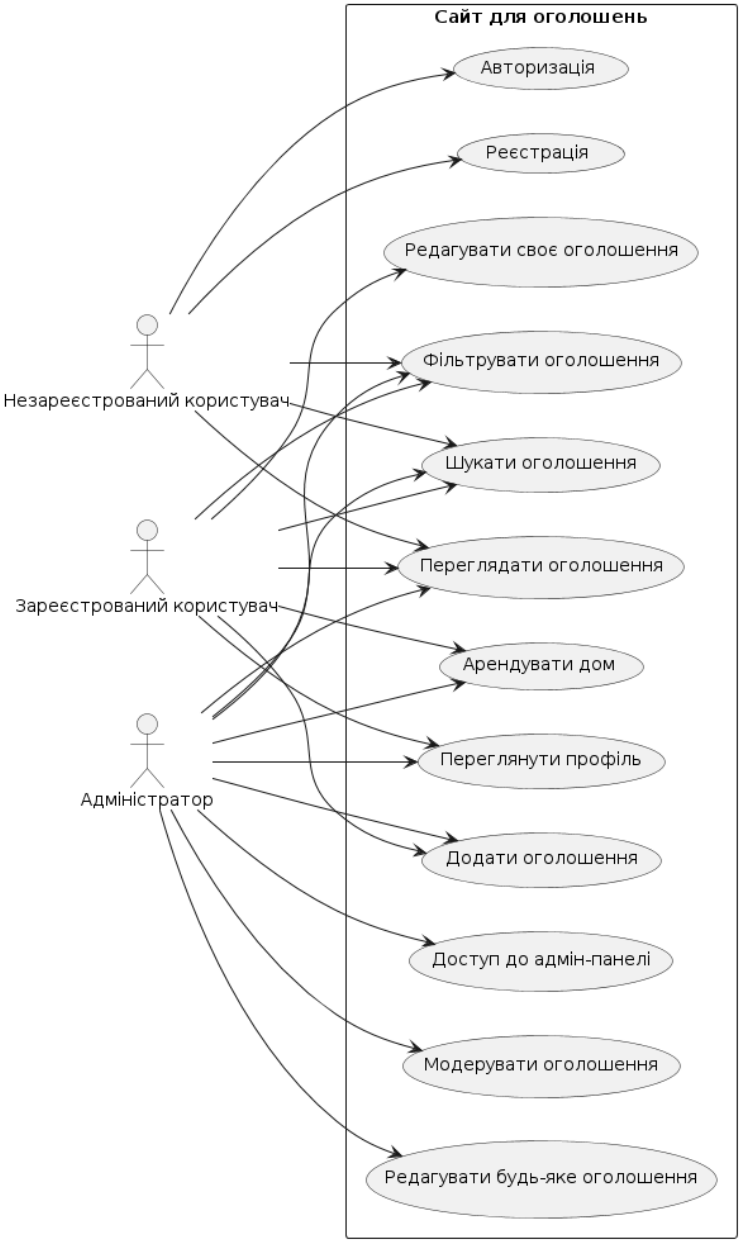
Відгуки та рейтинги: Платформа надає систему відгуків та рейтингів, що дозволяє користувачам залишати відгуки про своє перебування та переглядати відгуки інших для прийняття обґрунтованих рішень.

Профілі користувачів: Користувачі можуть створювати та керувати своїми профілями, вносити необхідну особисту інформацію та переглядати історію бронювань.

Безпечні платежі: Інтегрована платіжна система забезпечує безпечні та зручні способи оплати за бронювання.

Сповіщення та повідомлення: Платформа надає систему сповіщень, яка інформує користувачів про актуальні стадії бронювання, зміни в умовах або нові повідомлення.

Мобільна адаптація: Проект забезпечений повноцінною мобільною адаптацією, що дозволяє користуватися всіма функціями через смартфон або планшет.



Мал.1.3.1 Діаграма використання

1.4 Архітектура системи

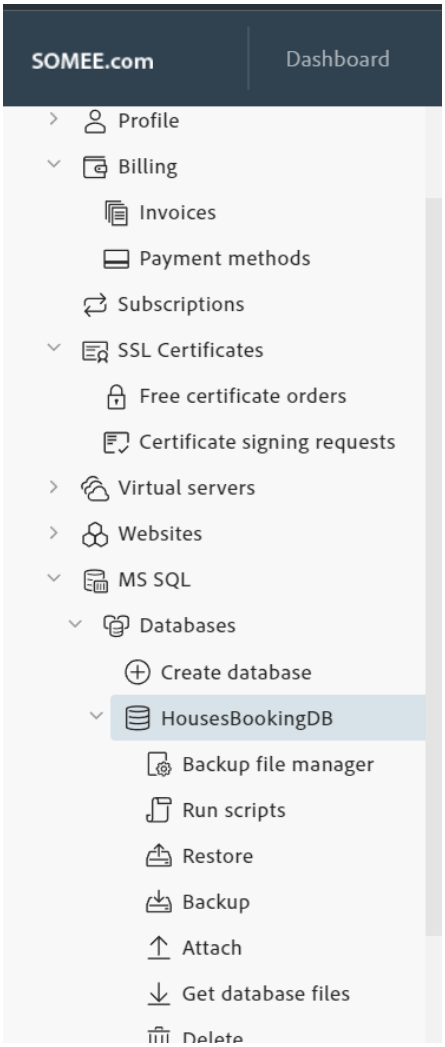
Архітектура проекту "Аналог Booking і Airbnb" використовує перевірену структуру, яка ділиться на фронтенд та бекенд для гнучкості та ефективності.

Фронтенд реалізований за допомогою *React*, що забезпечує сучасний інтерфейс з відмінною взаємодією користувача. Додатково, використання *TypeScript* додає строгу типізацію для покращення читабельності та

надійності коду. Стилізація фронтенду відбувається через систематизовані *CSS-модулі*, які дозволяють легко масштабувати та модифікувати зовнішній вигляд застосунку.

Бекенд побудований на *ASP.NET Core*, ідеально підходить для створення потужних *веб-API* завдяки своїй продуктивності та масштабованості. Безпека користувачів гарантована через використання *ASP.NET Identity* для аутентифікації та авторизації, а також шифрування паролів та захищеного управління токенами доступу.

База даних використовує *Microsoft SQL Server* з *Entity Framework Core* для *ORM*, забезпечуючи ефективну роботу з даними та їх цілісність.



Мал.1.4.1 Somee.com MS SQL

Весь код розгортання та управління проектом відбувається через стандартні інструменти розробки без контейнеризації *Docker*. Бекенд хоститься на *Azure*, який пропонує надійні та масштабовані веб-служби. Фронтенд розміщується на *Vercel* для забезпечення швидкого доступу до статичного контенту.

CI/CD процеси налаштовані через *GitHub Actions*, де автоматичне тестування та розгортання додатку відбувається при кожному коміті. Це забезпечує неперервну інтеграцію змін та мінімізує можливість помилок при деплойменті.

1.5 Очікуваний результат

Очікуваним результатом розробки проекту "Аналог Booking і Airbnb" є запуск надійної та інтуїтивно зрозумілої онлайн-платформи для оренди житла. Платформа має забезпечити користувачам швидкий та легкий доступ до широкого вибору житла, а орендодавцям — ефективний спосіб промоції та здачі їхньої нерухомості.

2 Проектування системи

Проектування системи для проекту "Аналог Booking і Airbnb" включає ретельно продуману архітектуру для забезпечення зручності, швидкодії та безпеки платформи онлайн-бронювання житла. Важливим аспектом є ефективне зберігання та керування обсягами даних, що включають інформацію про користувачів, об'єкти, бронювання, відгуки та інші ключові параметри.

1 Довгострокове зберігання даних: Наша база даних буде зберігати всю необхідну інформацію надійно та безпечно. Це охоплює профілі користувачів, деталі об'єктів нерухомості, історію бронювань, та відгуки клієнтів.

2 Оптимізований доступ до даних: Система забезпечуватиме швидке відновлення даних для виведення на користувацький інтерфейс, гарантуючи, що користувачі матимуть актуальну інформацію в реальному часі.

3 Ефективне управління даними: З використанням реляційної бази даних, дані будуть систематизовані за допомогою зв'язків, нормалізації та індексів для поліпшення продуктивності та уникнення дублікації.

4 Безпека: Ми застосуємо найкращі практики шифрування, аутентифікації, та авторизації для забезпечення безпеки користувацьких даних і транзакцій.

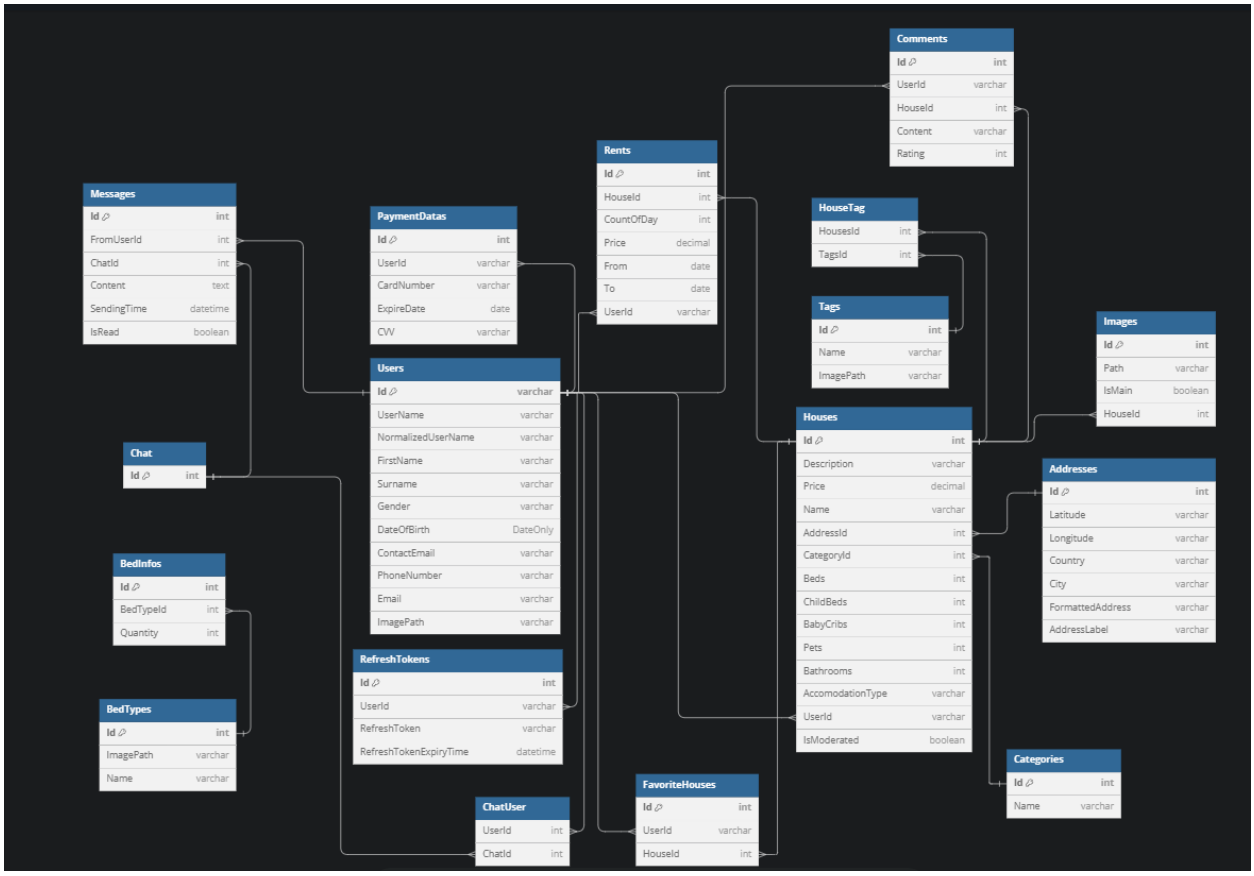
5 Масштабованість: База даних буде спроектована з урахуванням можливого розширення проекту, що дозволить легко масштабувати ресурси під зростаючий трафік та кількість даних.

6 Інтеграція компонентів: Використання *API* для зв'язку між фронтендом та бекендом забезпечить гнучке управління даними та інтеграцію з зовнішніми сервісами, такими як системи платежів чи зовнішні *API* для карти.

Система використовуватиме сучасні технології та підходи, включаючи мікросервісну архітектуру, для покращення відокремленості компонентів та забезпечення гнучкості розвитку проекту. Це дозволить забезпечити стабільну та швидку роботу платформи, а також легко впроваджувати нові функції в міру зростання та розвитку проекту.

2.1 Система управляння базою даних

Проект "Аналог Booking і Airbnb" використовує Microsoft SQL Server як систему управління базою даних (СУБД), що надає як численні переваги, так і деякі недоліки, які потрібно врахувати для повноцінного розуміння вибору технології.



Мал. 2.1.1 Структура таблиць бази даних додатка

Переваги SQL Server:

1 Інтеграція з .NET: Тісна інтеграція з .NET Framework та .NET Core забезпечує легкість розробки та оптимізацію продуктивності.

2 Продуктивність: Висока продуктивність при обробці транзакцій та аналітичних запитів завдяки вбудованим технологіям і оптимізації запитів.

3 Безпека: Розширені можливості шифрування, аудиту та авторизації забезпечують надійний захист даних.

4 Масштабованість: Можливість легкого масштабування від малих до великих систем відповідає потребам зростаючих проектів.

5 Підтримка та спільнота: Сильна підтримка від Microsoft і активна спільнота користувачів сприяють вирішенню проблем.

Недоліки SQL Server:

1 Вартість: Хоча існують безкоштовні версії, комерційні версії SQL Server можуть бути дорогими для масштабування і додаткових функцій.

2 Ресурси системи: SQL Server може бути ресурсоємним, вимагаючи значних обчислювальних та пам'ятних ресурсів для оптимальної роботи.

3 Складність управління: Для користувачів без досвіду роботи з Microsoft продуктами управління SQL Server може здатися складним.

4 Прив'язка до Microsoft екосистеми: Оптимальне використання SQL Server часто передбачає інвестиції у інші продукти Microsoft, що може обмежувати гнучкість вибору технологічного стеку.

Вибір SQL Server для "Аналог Booking і Airbnb" засновано на високому рівні взаємодії з .NET, продуктивності та безпеці, які важливі для онлайн-сервісу бронювання. Водночас, команда розробників має бути готова вирішувати можливі складнощі з масштабуванням та управлінням системи, використовуючи доступні ресурси та підтримку.

2.2 Вибір мови програмування

Вибір мови програмування для проекту "Аналог Booking і Airbnb" базувався на потребах проекту, з урахуванням специфічних вимог до функціональності, продуктивності та швидкості розробки.

1 C# (.NET Framework):

Мова програмування *C#* та *.NET Framework* були вибрані для розробки бекенду цього проекту через їх високу продуктивність, сильні можливості управління пам'яттю та велику кількість бібліотек та розробницьких інструментів. Ця мова та платформа дозволяє створювати надійні та масштабовані веб-додатки.

2 JavaScript/TypeScript:

JavaScript з використанням *TypeScript* був вибраний для клієнтської частини завдяки своїй гнучкості, підтримці асинхронних операцій та широкому спектру інструментів для розробки інтерактивних веб-інтерфейсів.

TypeScript додає строгу типізацію та покращує підтримку великих проектів зі складною архітектурою.

3 React:

Бібліотека *React* була обрана для створення користувацького інтерфейсу, завдяки її ефективності в побудові динамічних *SPA* (S*ingle-Page Applications*), які забезпечують швидку та гладку взаємодію з користувачем без перезавантаження сторінки.

Для підвищення продуктивності розробки та забезпечення сучасного вигляду користувацького інтерфейсу, проект також інтегрує *Material-UI (Mui)*, яка є популярною бібліотекою *React UI* компонентів. *Mui* пропонує готовий набір дизайн-компонентів, які відповідають принципам матеріального дизайну, що дозволяє швидко реалізувати естетично привабливий та функціонально зручний інтерфейс. Використання *Mui* сприяє уніфікації вигляду компонентів і поліпшує користувацький досвід, надаючи інтуїтивно зрозумілу та відзивчиву взаємодію.

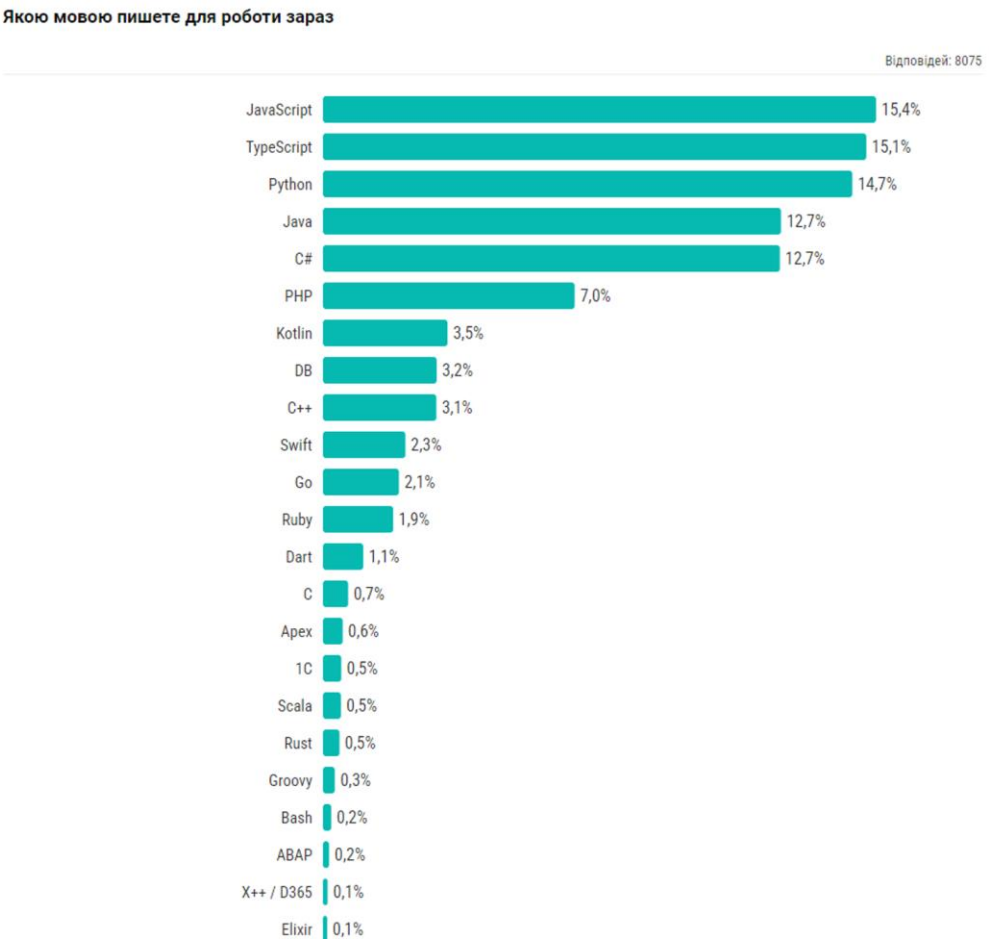
4 MS SQL Server:

Microsoft SQL Server був обраний як система управління базою даних через тісну інтеграцію з *.NET* платформою, сильні засоби забезпечення безпеки, транзакцій та високу продуктивність при обробці запитів та аналітичних операцій.

5 Entity Framework:

Entity Framework, ORM (*Object-Relational Mapping*) інструмент для *.NET*, був вибраний для роботи з базою даних, щоб мінімізувати кількість ручної роботи з *SQL* та покращити швидкість розробки.

Розглянемо на мал. 2.2.1 статистичну діаграму використання мов програмування у всьому світі станом на 2024:



Мал. 2.2.1 Статистичні дані використання мов програмування у світі станом на 2024 рік

Статистичний аналіз використання мов програмування (за даними діаграми) показує, що *JavaScript* (15.4%) і *Python* (14.7%) є найбільш популярними у всьому світі. В той час як використання *C#* не є найвищим на глобальному рівні, його вибір для цього проекту був зумовлений специфічними потребами системи та вимогами до інтеграції та продуктивності, а не загальносвітовою популярністю. Вибір спирався на потреби в швидкій розробці, легкій підтримці та надійності, які *C#* і *.NET* можуть забезпечити.

2.3 Додаткові технології

Мал. 2.3.1 Технології програмування

1 Frontend:

Проект інтегрує передові технології та бібліотеки для створення інтуїтивно зрозумілого та відзивчивого користувацького інтерфейсу. Однією з ключових інтеграцій є Material-UI (Mui), бібліотека компонентів React UI, яка відповідає принципам матеріального дизайну. Використання Mui дозволяє швидко реалізувати естетично привабливий інтерфейс з готовим набором дизайн-компонентів. Такий підхід не тільки поліпшує зовнішній вигляд додатку, але й забезпечує уніфікований та зручний користувацький досвід.

Додатково, для розширення функціональності геолокації, в проекті використовується Radar API, сучасна технологія для інтеграції геолокаційних сервісів. Radar API надає інструменти для точного визначення місцезнаходження користувачів, геокодування, а також для пошуку та планування маршрутів. Це значно полегшує взаємодію користувачів з картами та локаціями, надаючи можливість більш зручного пошуку об'єктів нерухомості відповідно до їхнього реального розташування.

Інтеграція Radar API є важливою для платформ, що пропонують сервіси з прив'язкою до локацій, таких як бронювання житла чи туристичні сервіси. Це дозволяє надати користувачам точну інформацію про місцезнаходження об'єктів, покращити пошук та вибір, а також оптимізувати логістику і планування подорожей.

Використання цих технологій разом гарантує, що проект буде не тільки візуально привабливим, але й функціонально розширеним, забезпечуючи користувачам всі необхідні інструменти для зручної та ефективної взаємодії з платформою.

2 Backend:

Робота з базою даних у проекті є вкрай важливою, і для цього використовуються ***Entity Framework Migrations***. *Migrations* дозволяють керувати змінами в базі даних шляхом версіонування схеми бази даних і є ідеальним інструментом для роботи з кодовою базою, що постійно еволюціонує. Це забезпечує гладке і контрольоване впровадження нових функцій та змін, а також підтримує синхронізацію структури бази даних між розробниками і виробничим середовищем.

Автентифікація користувачів у проекті здійснюється за допомогою ***JWT (JSON Web Tokens)***. *JWT* надає безпечний і гнучкий механізм для обміну даними між клієнтом і сервером. Токени *JWT* забезпечують перевірку автентичності користувачів і дозволяють зберігати необхідний стан сесії без створення додаткового навантаження на базу даних.

Для зберігання зображень та інших великих об'єктів проект використовує ***Azure Blob Storage***, який надає надійне, масштабоване та оптимізоване рішення для зберігання об'ємних даних у хмарі. Azure Blob Storage ідеально підходить для обробки статичного контенту, як-от зображення та документи, і дозволяє зменшити навантаження на основний сервер, прискорюючи час відгуку системи.

2.4 GIT

При розробці проекту наша команда використовує систему контролю версій *GIT*, яка є необхідною для синхронізації роботи розробників та забезпечення цілісності коду. Застосування *GIT* дозволяє нам ефективно управляти змінами, відстежувати історію розробки та колаборативно

працювати над проектом, надаючи можливість кожному учаснику вносити власні внески через віддалені репозиторії.

У нашому випадку, для зберігання коду проекту використовуються репозиторії на *GitHub*, які служать централізованим сховищем для всієї команди. Це надає нам наступні переваги:

– ***Колаборація:*** GitHub забезпечує платформу, де розробники можуть одночасно працювати над різними фрагментами проекту, використовуючи гілки для експериментів та розробки нових функцій, мінімізуючи ризик конфліктів коду.

– ***Відстеження прогресу:*** Всі коміти та пул-реквести супроводжуються описами, що дозволяє нам відстежувати зміни в проекті, розуміти контекст та причини змін.

– ***Відновлення та резервне копіювання:*** *GIT* забезпечує безпеку коду завдяки збереженню історії змін, що дозволяє відновлювати попередні стани проекту у разі помилок або втрати даних.

– ***Контроль версій:*** Можливість мітити версії релізів, інтегрувати зміни з різних гілок, та управляти залежностями та релізами.

На даний момент, весь код проекту розділено на два основних репозиторії: бекенд та фронтенд, що дозволяє нам краще організувати розробку та тестування різних частин системи.

Фронтенд: Це та частина, з якою взаємодіє користувач. Вона включає клієнтську логіку, представлення даних та клієнтські запити до сервера. Код фронтенду знаходиться у репозиторії GitHub за посиланням:

Frontend Repo

Бекенд: Включає логіку сервера, обробку даних, аутентифікацію та авторизацію, управління базою даних і відповідає на запити клієнта. Код бекенду знаходиться у репозиторії GitHub за посиланням:

Backend Repo

Кожен репозиторій містить README файл, що надає детальну інформацію про проект, інструкції зі встановлення, використання, та контрибуції, а також інші важливі деталі, які спрощують розуміння та роботу з проектом для всієї команди розробників.

Такий підхід забезпечує ефективний процес розробки та постійну готовність коду до деплойменту, враховуючи найновіші зміни та удосконалення, внесені командою.

2.5 Deployment & CI/CD

Процес розгортання (*deployment*) та неперервна інтеграція/неперервне розгортання (*CI/CD*) є ключовими аспектами сучасної розробки програмного забезпечення, які забезпечують швидке впровадження змін та стабільність роботи системи.

Deployment:

- ***База даних:*** Використання хостингу somee.com дозволяє нам безкоштовно розгортати та управляти базою даних, забезпечуючи доступність та стабільність даних в режимі реального часу. Завдяки інструментам міграції, ми маємо можливість оперативно оновлювати схему бази даних без перерв у роботі сервісу.
- ***Бекенд:*** Для бекенду використовується сервіс хмарного хостингу Azure, що забезпечує високу доступність, масштабованість та безпеку серверних додатків.
- ***Фронтенд:*** Клієнтська частина розгортається через платформу Vercel, яка надає оптимізоване середовище для статичних сайтів та додатків на React, з автоматизованими процесами збірки та розгортання.

CI/CD:

- ***GitHub Actions:*** Цей інструмент використовується для автоматизації робочих процесів CI/CD, дозволяючи здійснювати автоматичну збірку,

тестування та розгортання проекту при кожному коміті в основну гілку або при внесенні змін через пул-реквести.

– ***Стратегії розгортання:*** Застосовуються різні стратегії для забезпечення безперебійного розгортання, як-от canary releases або blue-green deployment, для мінімізації ризиків та впливу на кінцевих користувачів під час оновлень.

– ***Моніторинг та логування:*** Інтеграція з системами моніторингу та логування для забезпечення оперативного виявлення та вирішення проблем в роботі додатку.

Використання перелічених вище технологій та підходів забезпечує гнучке та ефективне розгортання проекту, підтримку високої якості коду та безперервне вдосконалення продукту без затримок чи перебоїв у наданні послуг користувачам.

2.6 Опис API і взаємодії між клієнтом і сервером

Бекенд пропонує *RESTful API*, яке дозволяє фронтенду запитувати, відправляти, оновлювати, та видаляти дані в уніфікованому форматі. Система авторизації та аутентифікації користувачів реалізована за допомогою *AspNetCore.Identity* з використанням *JWT* для безпечної передачі та верифікації користувацьких сесій. Такий підхід забезпечує високий рівень безпеки та надійності платформи.

3 Інтерфейс користувача

Дизайн інтерфейсу користувача є ключовим аспектом в процесі розробки проекту "Аналог Booking і Airbnb". Розуміння того, як користувачі взаємодіють з платформою, і створення інтуїтивно зрозумілого та естетично

приємного інтерфейсу є пріоритетним завданням для нашої команди дизайнерів.

3.1 Дизайн інтерфейсу користувача, включаючи wireframes та макети сторінок

Для забезпечення високої якості дизайну наша команда дизайнерів використовує сучасні інструменти та методології. Вони розробляють детальні wireframes, які слугують основою для макетів сторінок. Ці wireframes допомагають візуалізувати структуру інформації, розміщення елементів на сторінці, та шляхи взаємодії користувача з платформою. Після затвердження wireframes, дизайнери створюють високодеталізовані макети, які відображають кінцевий вигляд сторінок та інтерфейсу користувача.

3.2 UX/UI розробка та її вплив на взаємодію з користувачем

UX (User Experience) розробка зосереджена на оптимізації взаємодії користувача з платформою, гарантуючи, що користувачі можуть легко та ефективно досягати своїх цілей. UI (User Interface) розробка забезпечує, що кожен елемент інтерфейсу виглядає привабливо та є консистентним з загальною дизайн-системою платформи. Комбінація UX та UI розробки дозволяє нам створити продукт, який не тільки виглядає добре, але й є простим у використанні, забезпечуючи позитивний досвід для користувачів нашого сервісу.

Завдяки злагодженій роботі між дизайнерами, фронтенд-розробниками та продуктовими менеджерами, ми гарантуємо, що кожен аспект нашої платформи буде не тільки функціональним, але й візуально привабливим, що є вирішальним для забезпечення високої впізнаваності та конкурентоспроможності на ринку.

4 Принцип праці програми

Принцип роботи програми для дипломного проекту "Аналог Booking і Airbnb" базується на взаємодії між клієнтською стороною (frontend), розробленою на React, та серверною стороною (backend), що виконана на ASP.NET Core.

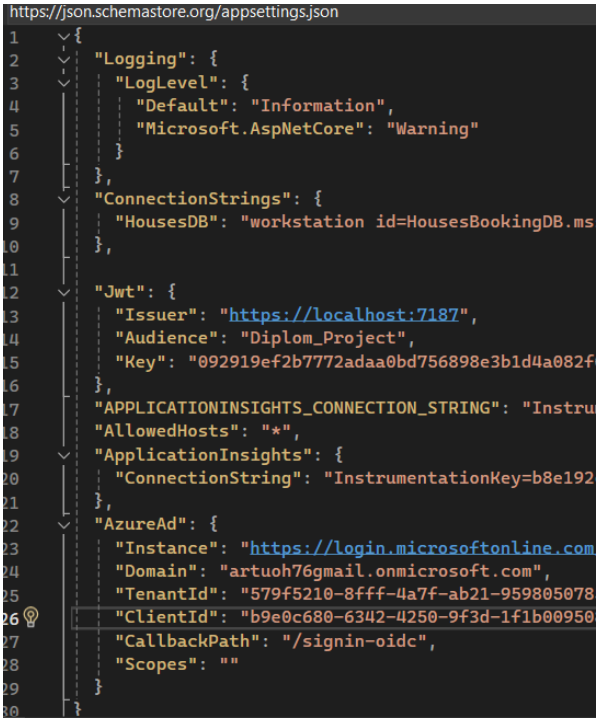
4.1 Створення бази даних

У рамках розробки платформи для онлайн-бронювання житла, основну роль у збереженні та керуванні даними грає реляційна база даних. У нашому випадку, використовується Microsoft SQL Server як СУБД, а сама база даних розгорнута на сервісі somee.com, що дозволяє отримати безкоштовний хостинг на стартовому етапі проекту.

Структура бази даних і схема даних проектуються за допомогою класів у мові програмування C#, які визначають сутності та відносини між ними. Ці класи потім мапляться на таблиці в базі даних за допомогою Entity Framework Core - ORM системи, яка спрощує процес роботи з даними.

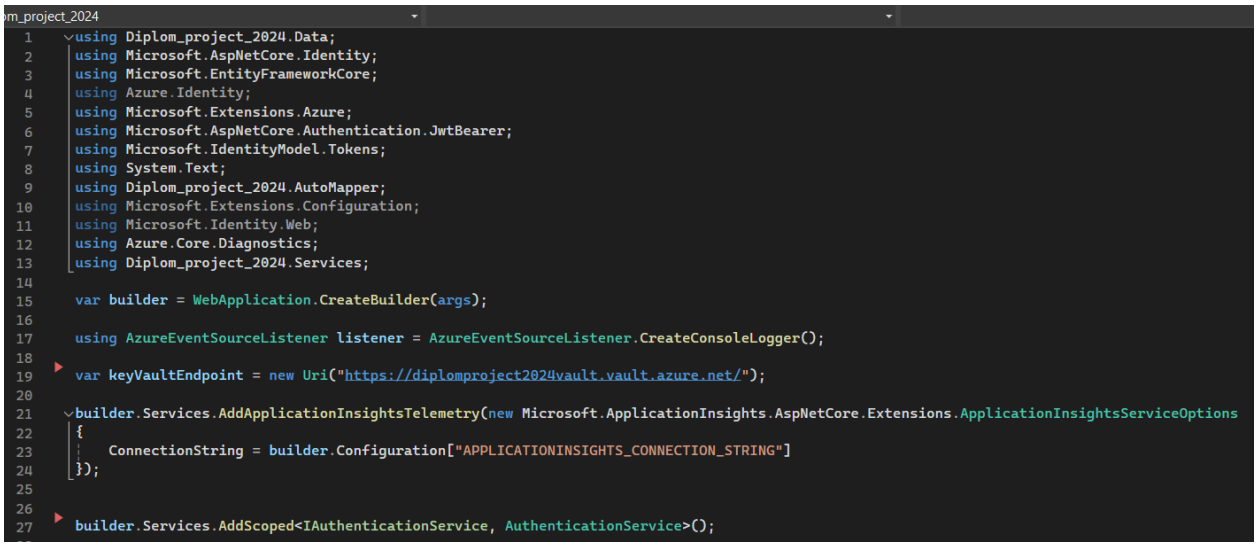
Застосування міграцій в Entity Framework Core дозволяє нам керувати змінами в структурі бази даних відповідно до модифікацій у доменних моделях нашого додатку. Кожна міграція відображає окремий набір змін, які були застосовані до класів, і переводить ці зміни у відповідні SQL інструкції, які виконуються на сервері бази даних.

Конфігурація підключення до бази даних зберігається у файлі `appsettings.json`. Це дозволяє централізовано керувати параметрами підключення і легко адаптувати їх під різні середовища розгортання, наприклад, для розробки, тестування, чи продакшену.



Мал. 4.1.1 Appsettings.json

Основний код проекту розміщений у файлі `Program.cs`, де відбувається конфігурація та ініціалізація всіх необхідних сервісів. Тут задаються налаштування для аутентифікації, CORS політик, підключення до бази даних, а також реєструються сервіси, необхідні для роботи API.



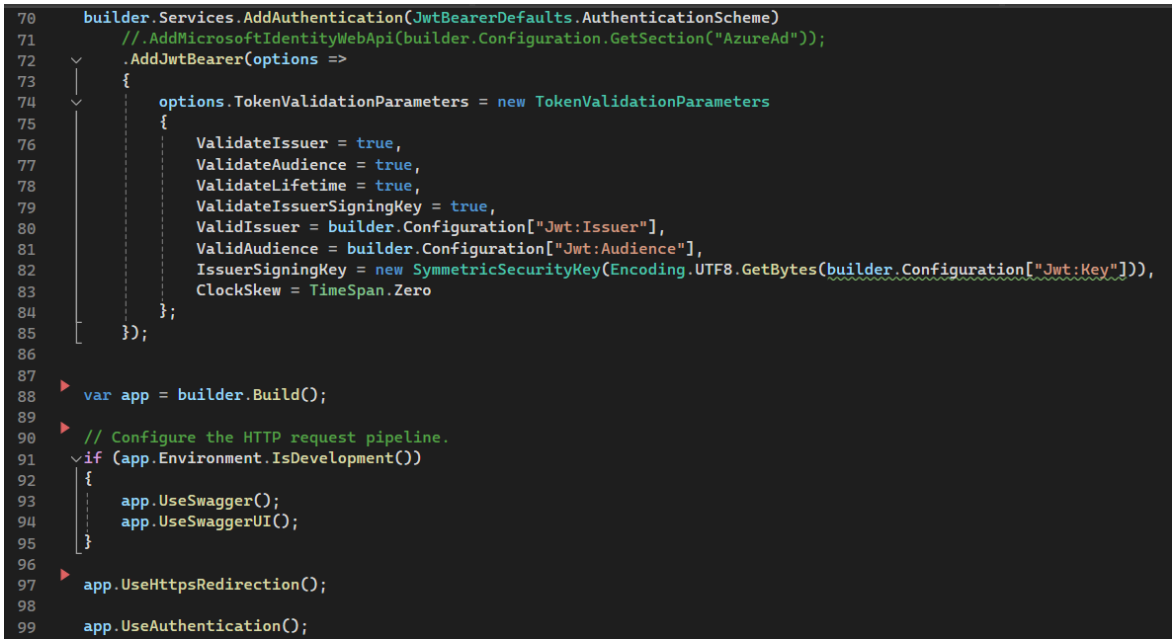
Мал. 4.1.2 Programm.cs

```
builder.Services.AddCors(options =>
{
    options.AddPolicy("AllowAllOrigins",
        build =>
        {
            build
                .AllowAnyOrigin()
                .AllowAnyMethod()
                .AllowAnyHeader();
        });
});
 // Add services to the container.

 builder.Services.AddControllers();

 //Add AutoMapper
 builder.Services.AddAutoMapper(typeof(MappingProfile));

 builder.Services.AddIdentityApiEndpoints<User>().AddRoles<IdentityRole>()
     .AddEntityFrameworkStores<HousesDBContext>();
builder.Services.AddDbContext<HousesDBContext>(options =>
{
    options.UseSqlServer(builder.Configuration.GetConnectionString("HousesDb"));
    //options.UseSqlServer(builder.Configuration["HousesDB"]);
});
```

Мал. 4.1.3 Programm.cs

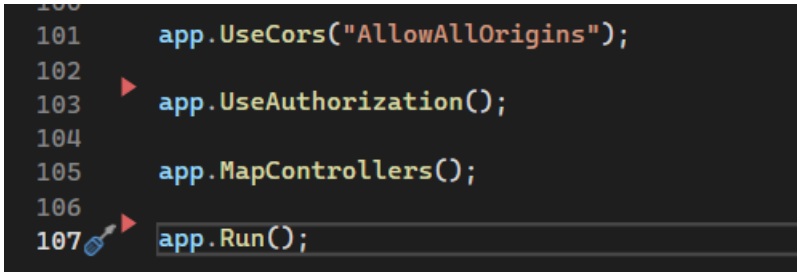
```
builder.Services.AddAzureClients(clientBuilder =>
{
    clientBuilder.AddBlobServiceClient(builder.Configuration["blob-string-proj"]!, preferMsi: true);
    clientBuilder.AddQueueServiceClient(builder.Configuration["blob-string-proj"]!, preferMsi: true);

});


// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

builder.Services.AddAuthorization(); //authorization
```

Мал. 4.1.4 Programm.cs



```
builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    //.AddMicrosoftIdentityWebApi(builder.Configuration.GetSection("AzureAd"));
    .AddJwtBearer(options =>
    {
        options.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateIssuer = true,
            ValidateAudience = true,
            ValidateLifetime = true,
            ValidateIssuerSigningKey = true,
            ValidIssuer = builder.Configuration["Jwt:Issuer"],
            ValidAudience = builder.Configuration["Jwt:Audience"],
            IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(builder.Configuration["Jwt:Key"])),
            ClockSkew = TimeSpan.Zero
        };
    });


var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();

app.UseAuthentication();
```

Мал. 4.1.5 Programm.cs



```
app.UseCors("AllowAllOrigins");

app.UseAuthorization();

app.MapControllers();

app.Run();
```

Мал. 4.1.6 Programm.cs

Розгортання бази даних на сервісі somee.com відбувається шляхом виконання міграцій через командний рядок або відповідні інструменти у середовищі розробки. Це забезпечує гнучкість та контрольованість розгортання, дозволяє швидко впроваджувати нові функціональні можливості та зміни, а також здійснювати відкат до попередніх версій схеми даних, якщо це потрібно.

Завдяки інтеграції з Azure Application Insights та Azure Active Directory, проект має розширені можливості моніторингу та забезпечення безпеки, що є критично важливим для комерційного застосування на платформі подібній до Airbnb.

4.1.1 Інструкція по використанню Migrations у Entity Framework Core

1. Відкрийте консоль диспетчера пакетів:

Перейдіть до `Середства > Диспетчер пакетів NuGet > Консоль диспетчера пакетів` у вашому Visual Studio.

2. Ініціалізація бази даних з першою міграцією:

У консолі введіть команду `Add-Migration InitialCreate`, де `InitialCreate` — це ім’я вашої першої міграції, яке семантично описує набір змін, які ви вводите.

3. Застосування міграції до бази даних:

Після створення міграції виконайте команду `Update-Database`, щоб застосувати міграцію та створити базу даних з визначеннями ваших моделей.

4. Внесення подальших змін до моделей:

Якщо ви внесли зміни до моделей і хочете оновити базу даних, використовуйте `Add-Migration` з новим ім’ям, що відображає ці зміни, наприклад: `Add-Migration AddSomeChangesToBook`.

5. Застосування нових міграцій:

Використовуйте `Update-Database`, щоб застосувати нові міграції до вашої бази даних.

6. Відкат міграції:

Якщо ви хочете відкотити останню застосовану міграцію, можна використати `Remove-Migration`, щоб видалити останні зміни з коду.

4.2 Загальний опис роботи `MappingProfile`

`MappingProfile` у бібліотеці *AutoMapper* використовується для встановлення відповідностей між об'єктами різних класів, щоб автоматизувати процес перетворення одного типу об'єктів в інший. У контексті веб-розробки, це особливо корисно при перетворенні даних між

доменними моделями, які використовуються в базі даних, та *DTO* (Data Transfer Objects), які передаються клієнтам через API.

Для нашого проекту *`MappingProfile`* забезпечує таку функціональність:

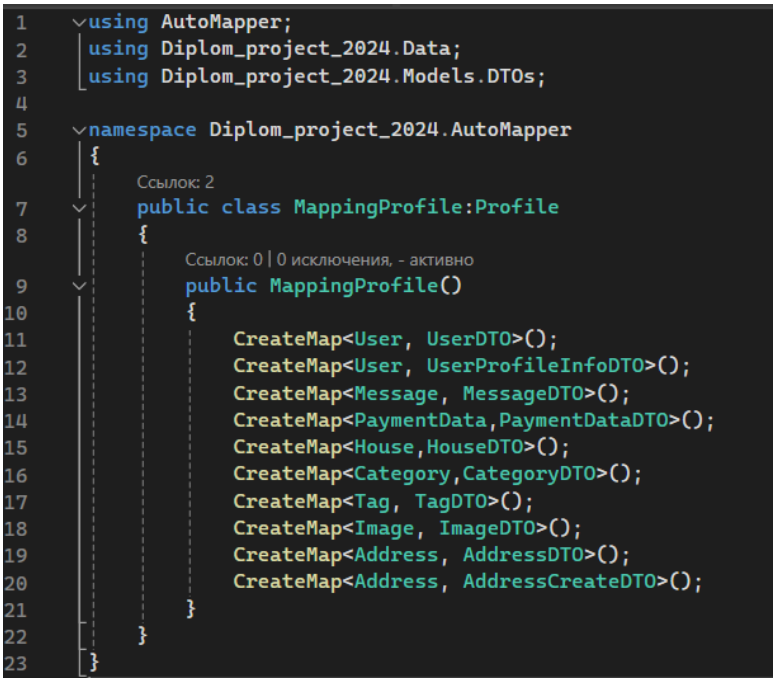
1 Узагальнення даних: *DTO* використовуються для упаковки даних, що передаються між сервером і клієнтом, дозволяючи обмежити витік внутрішніх деталей моделей і безпечно відправляти тільки необхідну інформацію.

2 Оптимізація відгуків API: Замість того, щоб відправляти великі доменні моделі, які можуть містити зайву або чутливу інформацію, *`MappingProfile`* дозволяє створювати спрощені версії цих об'єктів з тільки тими властивостями, які потрібні для конкретної операції.

3 Гнучкість: Коли модель даних змінюється, не потрібно ручно оновлювати кожне місце, де ці моделі перетворюються в *DTO* або навпаки. *`MappingProfile`* дозволяє централізовано управляти цими перетвореннями, що забезпечує легкість управління кодом та його підтримку.

4 Зменшення повторення коду: Замість того, щоб писати багаторазовий код для присвоєння властивостей між об'єктами, *`MappingProfile`* дозволяє визначити одиничне відображення, яке можна використовувати в усьому додатку.

5 Ефективність: *AutoMapper* може збільшити швидкість розробки, автоматизуючи рутинне завдання відображення даних, та зменшити ймовірність помилок, що часто виникають під час ручного копіювання даних.



```
using AutoMapper;
using Diplom_project_2024.Data;
using Diplom_project_2024.Models.DTOs;

namespace Diplom_project_2024.AutoMapper
{
    Ссылок: 2
    public class MappingProfile:Profile
    {
        Ссылок: 0 | 0 исключения, - активно
        public MappingProfile()
        {
            CreateMap<User, UserDTO>();
            CreateMap<User, UserProfileInfoDTO>();
            CreateMap<Message, MessageDTO>();
            CreateMap<PaymentData,PaymentDataDTO>();
            CreateMap<House,HouseDTO>();
            CreateMap<Category,CategoryDTO>();
            CreateMap<Tag, TagDTO>();
            CreateMap<Image, ImageDTO>();
            CreateMap<Address, AddressDTO>();
            CreateMap<Address, AddressCreateDTO>();
        }
    }
}
```

Мал. 4.2.1 Клас MappingProfile

У проекті, `*MappingProfile*` визначає перетворення між моделями сутностей, як `*User*` та `*House*`, та їхніми *DTO* версіями, такими як `*UserDTO*` та `*HouseDTO*`, які можуть бути використані для передачі даних з сервера на клієнт і навпаки без необхідності викривлення чутливих або непотрібних даних.

4.3 Загальний опис роботи Models -> DTOs

DTO (Data Transfer Objects) визначають згортки даних, які містять лише необхідну інформацію для конкретного запиту або відповіді між різними шарами системи. Вони використовуються для оптимізації передачі даних між бекендом та фронтендом або між сервером та клієнтом у веб-додатках. Головна мета використання DTO - це забезпечення безпеки, ефективності та структурованості взаємодії між різними компонентами системи.

При створенні DTO важливо враховувати контекст використання конкретного об'єкта. Наприклад, при створенні об'єкта *AddressCreateDTO*,

необхідно включити лише поля, які є обов'язковими для створення нової адреси, такі як широта, довгота, країна, місто тощо. В той час як об'єкт *AddressDTO* може містити ідентифікатор адреси та повний набір полів для відображення даних про адресу. Це дозволяє зменшити обсяг переданих даних, підвищити продуктивність та забезпечити безпеку, обмежуючи доступ до чутливої інформації.

Крім того, *DTO* допомагають у розділенні відповідальностей між різними компонентами системи. Вони визначаються незалежно від внутрішньої структури даних та бізнес-логіки, що дозволяє змінювати внутрішню логіку, не впливаючи на зовнішній інтерфейс.

Крім перерахованих переваг, використання *DTO* сприяє підтримці розширюваності та масштабованості системи, оскільки вони дозволяють змінювати структуру даних без необхідності змінювати клієнтські або серверні компоненти. Таким чином, *DTO* є важливою складовою у побудові зручних, безпечних та ефективних веб-додатків.

Перелік класів *DTO:*
Звісно, ось повний загальний опис роботи Models -> DTOs для дипломного проекту:

1. AddressCreateDTO: Представляє об'єкт адреси для створення. Містить інформацію про широту, довготу, країну, місто, форматовану адресу та мітку адреси.

2. AddressDTO: *DTO* для адреси з бази даних. Містить ідентифікатор, координати, країну, місто, форматовану адресу та мітку адреси.

3. CategoryDTO: Представляє категорію об'єкту. Містить ідентифікатор та назву.

4. ChatDTO: Використовується для передачі даних про чат. Містить ідентифікатор, інформацію про співбесідника, останнє повідомлення та кількість непрочитаних повідомлень.

5. HouseCreateDTO: *DTO* для створення об'єкту житла. Включає інформацію про категорію, тип проживання, адресу, кількість ліжок, ванних кімнат, та інші параметри.

6. HouseDTO: Представляє об'єкт житла з бази даних. Включає опис, ціну, кількість ліжок, адресу, категорію, користувача, статус модерації, теги, зображення та рейтинг.

7. HouseUpdateDTO: Використовується для оновлення інформації про житло. Містить поля для оновлення опису, ціни, площі, кількості кімнат, тощо.

8. ImageDTO: *DTO* для зображення. Містить ідентифікатор, шлях до зображення та прапорець основного зображення.

9. MessageDTO: Представляє повідомлення в чаті. Включає ідентифікатор, вміст, дату відправлення та інформацію про відправника.

10. MessageSendDTO: Використовується для надсилання нового повідомлення. Містить ідентифікатор чату, ідентифікатор отримувача та вміст повідомлення.

11. PasswordChangeDTO: *DTO* для зміни паролю користувача. Містить новий та старий пароль.

12. PaymentDataDTO: Представляє дані про оплату. Включає номер картки, дату закінчення та *CVV-код*.

13. RefreshTokenDTO: *DTO* для оновлення токена доступу. Містить оновлюваний токен.

14. RentCreationDTO: Використовується для створення запису оренди. Містить інформацію про користувача, об'єкт оренди, тривалість, ціну тощо.

15. RentDTO: Представляє запис оренди з бази даних. Містить ідентифікатор, тривалість, ціну, дати початку та закінчення, інформацію про користувача та об'єкт оренди.

16. RentUpdateDTO: Використовується для оновлення запису оренди. Містить поля для оновлення тривалості, ціни, дат початку та закінчення.

17. TagCreateDTO: *DTO* для створення тегу. Містить назву та зображення.

18. TagDTO: Представляє тег з бази даних. Містить ідентифікатор, назву та шлях до зображення.

19. TagEditDTO: Використовується для редагування тегу. Містить ідентифікатор, нову назву та, при необхідності, нове зображення.

20. TokenDTO: *DTO* для передачі токенів доступу та оновлення. Містить токен доступу та оновлюваний токен.

21. UserBasicInfoDTO: Представляє основну інформацію про користувача. Містить ім'я, прізвище, стать та дату народження.

22. UserContactInfoDTO: Використовується для передачі контактної інформації користувача. Містить електронну пошту та номер телефону.

23. UserDTO: Представляє користувача з бази даних. Включає ідентифікатор, електронну пошту, ім'я, прізвище та список улюблених об'єктів житла.

24. UserLoginDTO: *DTO* для аутентифікації користувача. Містить електронну пошту, пароль та прапорець "Запам'ятати мене".

25. UserProfileInfoDTO: Представляє повну інформацію про користувача. Містить особисті дані, контактну інформацію, дані про картку, кількість домів, кількість коментарів та шлях до зображення профілю.

26. UserProfileInfoSetDTO: Використовується для оновлення інформації про користувача. Містить поля для оновлення особистих даних та контактної інформації.

27. UserRegisterDTO: *DTO* для реєстрації нового користувача. Містить пароль та електронну пошту.

28. SetFirstNameModel, SetSurnameModel, SetGenderModel, SetDateOfBirthModel, SetPhoneNumber, SetContactEmail, AddFavoriteHouse: Допоміжні класи для оновлення окремих полів користувача.

Ці класи *DTO* дозволяють передавати дані між різними частинами системи, забезпечуючи структурованість та зручність у взаємодії між клієнтом та сервером.

4.4 Загальний опис роботи Functions

`Functions` — це класи і методи, які виконують певні операції та обробку даних в рамках додатку. Вони призначені для відокремлення логіки виконання завдань від основного коду, що забезпечує кращу організацію коду та його перевикористання.

В проекті є два основних типи функцій:

1. BlobContainerFunctions: Ці функції управляють завантаженням та видаленням зображень у сховищі *Azure Blob Storage*, яке є хмарним рішенням для зберігання файлів великого розміру, таких як медіафайли.

– *`UploadImage`* — метод, який завантажує зображення до Azure Blob Storage та повертає URL адресу зображення. Це важливо для того, щоб користувачі могли завантажувати зображення для їх об'єктів або профілів.

– *`DeleteImage`* — метод для видалення зображення з Azure Blob Storage, що забезпечує можливість управління зображеннями, наприклад, коли користувач хоче оновити або видалити своє фото.

```
using Azure.Storage.Blobs;
using Diplom_project_2024.Data;

namespace Diplom_project_2024.Functions
{
    Ссылок: 8
    public static class BlobContainerFunctions
    {
        Ссылок: 4 | 0 исключения, - активно
        public static async Task<string> UploadImage(BlobContainerClient container, IFormFile image)
        {
            var blob = container.GetBlobClient($"{Guid.NewGuid()}{Path.GetExtension(image.FileName)}");
            await blob.UploadAsync(image.OpenReadStream());
            return blob.Uri.AbsoluteUri;
        }
        Ссылок: 4 | 0 исключения, - активно
        public static async void DeleteImage(BlobContainerClient container, string ImagePath)
        {
            var blob = container.GetBlobClient(Path.GetFileName(ImagePath));
            await blob.DeleteIfExistsAsync();
        }
    }
}
```

Мал. 4.4.1 class BlobContainerFunctions

2. UserFunctions: Ці функції використовуються для управління даними користувача в *Identity Framework*, який є системою управління ідентифікацією і доступом у *.NET*.

– *`GetUser`* — метод, який дозволяє отримати об'єкт користувача з *UserManager* на основі даних, які містяться в *ClaimsPrincipal*. Це необхідно для різних операцій, пов'язаних з аутентифікацією та авторизацією, таких як отримання інформації про поточного користувача.

```
using Diplom_project_2024.Data;
using Microsoft.AspNetCore.Identity;
using System.Security.Claims;

namespace Diplom_project_2024.Functions
{
    Ссылок: 21
    public static class UserFunctions
    {
        Ссылок: 21 | 0 исключения, - активно
        public static async Task< User> GetUser(UserManager<User> userManager, ClaimsPrincipal User)
        {
            return await userManager.FindByNameAsync(User.Identity.Name);
        }
    }
}
```

Мал. 4.4.2 class UserFunctions

Загалом, використання *`Functions`* в коді дозволяє зробити бізнес-логіку більш модульною та чистою, також спрощує процес тестування окремих компонентів системи.

4.5 Загальний опис роботи AzureClientFactoryBuilderExtensions

Клас *`AzureClientFactoryBuilderExtensions`* розширює можливості *`AzureClientFactoryBuilder`*, що є частиною бібліотеки *Azure SDK* для *.NET*. Ці методи розширення надають додаткову функціональність при створенні клієнтів для сервісів *Azure Blob Storage* та *Azure Queue Storage*, які є ключовими компонентами для зберігання і керування даними в хмарі.

Ось їхні основні функції:

1. AddBlobServiceClient: Цей метод розширення використовується для створення екземпляру *`BlobServiceClient`*. Він дозволяє підключитися до

Azure Blob Storage за допомогою рядка підключення або *URI* сервісу. Якщо встановлено параметр *`preferMsi` (Managed Service Identity)*, то метод намагається створити клієнт, використовуючи *URI* сервісу, що забезпечує безпечніше підключення через *Azure Managed Identities*.

2. AddQueueServiceClient: Схожий на *`AddBlobServiceClient`*, цей метод розширення створює екземпляр *`QueueServiceClient`*, який дозволяє взаємодіяти з *Azure Queue Storage*. За допомогою цього клієнта можна управляти чергами повідомлень, які часто використовуються для асинхронної обробки задач у розподілених системах.

```
using Azure.Core.Extensions;
using Azure.Storage.Blobs;
using Azure.Storage.Queues;
using Microsoft.Extensions.Azure;

Ссылок: 0
internal static class AzureClientFactoryBuilderExtensions
{
    public static IAzureClientBuilder<BlobServiceClient, BlobClientOptions>
        Ссылок: 1 | 0 исключения, - активно
        AddBlobServiceClient(this AzureClientFactoryBuilder builder, string serviceUriOrConnectionString, bool preferMsi)
    {
        if (preferMsi && Uri.TryCreate(serviceUriOrConnectionString, UriKind.Absolute, out Uri? serviceUri))
        {
            return builder.AddBlobServiceClient(serviceUri);
        }
        else
        {
            return builder.AddBlobServiceClient(serviceUriOrConnectionString);
        }
    }

    public static IAzureClientBuilder<QueueServiceClient, QueueClientOptions>
        Ссылок: 1 | 0 исключения, - активно
        AddQueueServiceClient(this AzureClientFactoryBuilder builder, string serviceUriOrConnectionString, bool preferMsi)
    {
        if (preferMsi && Uri.TryCreate(serviceUriOrConnectionString, UriKind.Absolute, out Uri? serviceUri))
        {
            return builder.AddQueueServiceClient(serviceUri);
        }
        else
        {
            return builder.AddQueueServiceClient(serviceUriOrConnectionString);
        }
    }
```

Мал. 4.5.1 class AzureClientFactoryBuilderExtensions

Обидва методи розширення дозволяють легко інтегрувати *Azure Storage services* у *.NET* додатки з можливістю перемикання між різними моделями автентифікації. Це сприяє гнучкості та забезпечує розробникам зручні інструменти для конфігурації підключення до хмарних сервісів *Azure*.

4.6 Загальний опис роботи Services

Сервіси в цьому проекті забезпечують логіку бізнес-процесів і є ключовою частиною системи, вони інкапсулюють всю роботу, пов'язану з аутентифікацією та авторизацією користувачів.

AuthenticationService – це сервіс, який відповідає за створення та валідацію токенів, реєстрацію користувачів, та оновлення доступу через refresh токени. Ось ключові моменти цього сервісу:

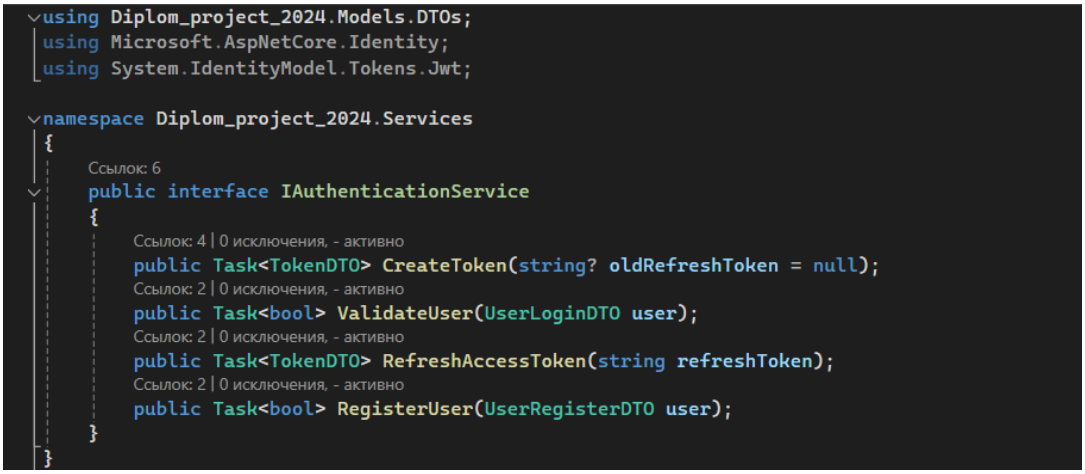
1. Створення JWT (JSON Web Tokens): Сервіс використовує клас ***`JwtSecurityToken`*** для генерації токенів, які використовуються для авторизації на *API*. Токени містять інформацію про користувача та роль, що дозволяє контролювати доступ до ресурсів.

2. Управління Refresh Tokens: Сервіс зберігає *refresh* токени в базі даних і використовує їх для генерації нових *access* токенів, коли старі закінчуються. Це забезпечує безперервний доступ користувачів до системи без потреби повторного входу.

3. Реєстрація нових користувачів: Використовує *`UserManager`* для створення нових записів користувачів в базі даних, забезпечуючи захист паролів та збереження облікових даних.

4. Валідація користувачів: Перевіряє правильність логіну та паролю, використовуючи *`UserManager`*, і забезпечує логіку входу в систему.

Сервіси інтерфейсують з клієнтом (наприклад, веб-фронтенд) і передають дані через *DTO* (D*ata Transfer Objects*), які є простішими версіями доменних моделей. Це дозволяє забезпечити відділення внутрішньої структури даних від зовнішнього представлення та знижує ризики пов'язані з безпекою, в той же час забезпечуючи необхідний рівень інформації для виконання функціоналу.



Мал. 4.6.1 interface IAuthenticationService

Використання сервісно-орієнтованої архітектури сприяє кращому розподілу відповідальності, полегшує тестування окремих компонентів системи та покращує масштабованість та підтримку проекту.

4.7 Загальний опис роботи Controllers

Контролери в проекті виконують роль посередників між користувацьким інтерфейсом і бізнес-логікою додатку, вони відповідають за обробку HTTP-запитів, валідацію вхідних даних, і виклик відповідних сервісів чи методів доступу до даних. Ось деякі приклади їх функцій:

1. CRUD Операції: Багато контролерів надають стандартні *CRUD* (створення, читання, оновлення, видалення) операції для різних сутностей в базі даних, таких як *`Addresses`*, *`Categories`*, *`Chats`*, *`Messages`*, *`Rents`*, *`Tags`*, та інші. Вони використовують методи *`GET`*, *`POST`*, *`PUT`*, *`DELETE`* для взаємодії з клієнтами.

2. Авторизація і Аутентифікація: Контролер *`AuthorizationController`* виконує логіку пов'язану з входом користувачів, реєстрацією, та управлінням рефреш-токенами.

3. Взаємодія з Azure Blob Storage: Контролер *`TagsController`* імплементує логіку для завантаження та видалення зображень в *Azure Blob Storage* за допомогою бібліотеки *Azure SDK*.

4. Управління користувацьким профілем: Контролер *`UserController`* забезпечує функції для оновлення профільної інформації користувача, управління платіжними даними, зміни паролю, та видалення акаунта.

5. Забезпечення Інформації для Головної Сторінки: Контролер *`HousesController`* виконує запити для отримання даних, які будуть відображені на головній сторінці, включаючи списки нерухомості за рейтингом та популярністю.

```
namespace Diplom_project_2024.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    Ссылок: 1
    public class AddressesController : ControllerBase
    {
        private readonly HousesDBContext _context;

        Ссылок: 0 | 0 исключения, - активно
        public AddressesController(HousesDBContext context)
        {
            _context = context;
        }

        //GET: api/Addresses
        [HttpGet]
        Ссылок: 0 | 0 запросы, - активно | 0 исключения, - активно
        public async Task<ActionResult<IEnumerable<Address>>> GetAddresses()
        {
            return await _context.Addresses.ToListAsync();
        }
```

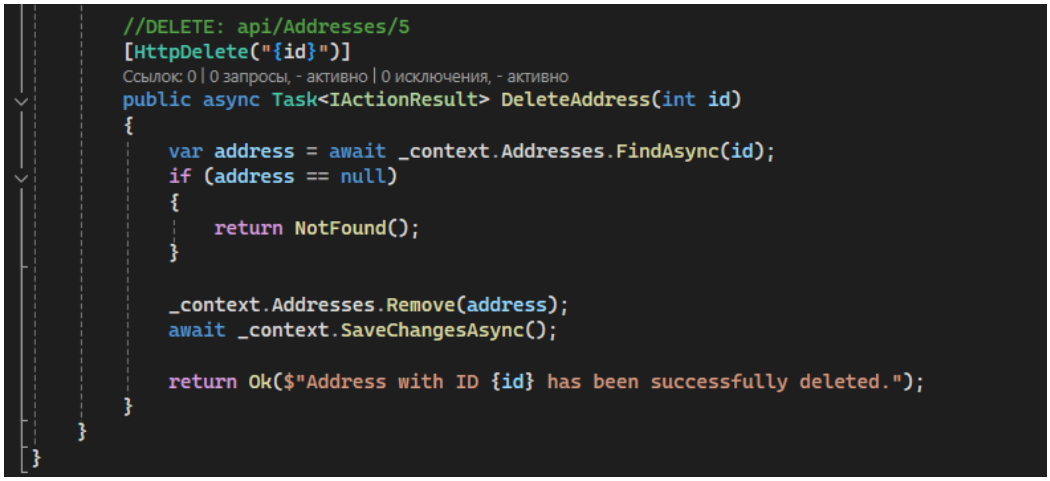
Мал. 4.7.1 class AddressesController

```
namespace Diplom_project_2024.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    Ссылок: 1
    public class AddressesController : ControllerBase
    {
        private readonly HousesDBContext _context;

        Ссылок: 0 | 0 исключения, - активно
        public AddressesController(HousesDBContext context)
        {
            _context = context;
        }

        //GET: api/Addresses
        [HttpGet]
        Ссылок: 0 | 0 запросы, - активно | 0 исключения, - активно
        public async Task<ActionResult<IEnumerable<Address>>> GetAddresses()
        {
            return await _context.Addresses.ToListAsync();
        }
```

Мал. 4.7.2 class AddressesController

```
        [HttpPut("{id}")]
        Ссылок: 0 | 0 запросы, - активно | 0 исключения, - активно
        public async Task<ActionResult<Address>> PutAddress(int id, Address address)
        {
            if (id != address.Id)
            {
                return BadRequest("Address ID mismatch");
            }

            var addressFromDB = await _context.Addresses.FindAsync(id);
            if (addressFromDB == null)
            {
                return NotFound();
            }

            _context.Entry(addressFromDB).CurrentValues.SetValues(address);

            try
            {
                await _context.SaveChangesAsync();
            }
            catch (DbUpdateConcurrencyException)
            {
                if (!_context.Addresses.Any(e => e.Id == id))
                {
                    return NotFound();
                }
                else
                {
                    throw;
                }
            }

            await _context.Entry(addressFromDB).ReloadAsync();

            // Return the updated address
            return addressFromDB;
        }
```

Мал. 4.7.3 class AddressesController



Мал. 4.7.4 class AddressesController

Контролери використовують *DTO (Data Transfer Objects)* для передачі даних до і з клієнта, забезпечуючи, що внутрішня структура моделей бази даних не буде експонована зовнішнім користувачам. Це допомагає підтримувати безпеку даних та упрощує процес мапінгу даних між внутрішніми моделями і форматами, які передаються користувачам.

Контролери також можуть застосовувати авторизацію на рівні методів, використовуючи атрибути, як *`Authorize`*, для контролю доступу до певних операцій, забезпечуючи, що лише авторизовані користувачі можуть виконувати чутливі дії.

Загалом, контролери є ключовою частиною *RESTful API*, що дозволяє забезпечити чисту та ефективну архітектуру, яка легко масштабується та адаптується до змін у бізнес-процесах.

5 Безпека

В рамках дипломного проекту безпека даних користувачів є високопріоритетним завданням, оскільки вона є основою для встановлення довіри та забезпечення безпечного використання сервісу. Для досягнення цього мети були впроваджені наступні заходи забезпечення безпеки:

1. Хешування паролів з сіллю: Всі паролі користувачів зберігаються у захешованому вигляді з унікальною сіллю, що запобігає злому та витоку інформації.

2. Використання протоколу HTTPS: Усі дані, що передаються між клієнтом та сервером, зашифровані за допомогою протоколу *HTTPS*, що гарантує конфіденційність інформації та запобігає перехопленню трафіку.

3. Захист від вразливостей: Система забезпечена від таких типів атак, як *Cross-Site Scripting (XSS)*, *Cross-Site Request Forgery (CSRF)* та *SQL injection*, що зменшує ризик компрометації даних користувачів.

4. Контроль доступу: Кожному користувачеві надається обмежений доступ до функцій та даних відповідно до його ролі в системі, що мінімізує можливість несанкціонованого доступу до конфіденційної інформації.

Описані заходи допомагають забезпечити високий рівень безпеки даних користувачів та забезпечують довіру до сервісу.

Окрім того, система авторизації та аутентифікації також має велике значення для забезпечення безпеки даних. У проекті використовується *ASP.NET Core Identity*, який надає надійний фреймворк для управління користувацькими акаунтами. Ключові компоненти цієї системи включають:

1. JWT (JSON Web Tokens): Для авторизації користувачів використовуються токени JWT, що дозволяють зберігати стан авторизації на клієнтській стороні.

2. Двофакторна автентифікація: Підвищення безпеки за рахунок використання додаткових методів верифікації, таких як SMS-повідомлення або електронні листи.

3. Refresh Tokens: Використання токенів оновлення для підтримки безпечного входу користувачів без необхідності повторного введення логіна та пароля.

4. Обмеження невдалих спроб входу: Автоматичне блокування акаунта після певної кількості невдалих спроб входу для запобігання брутфорс-атак.

5. Ролева модель доступу: Визначення прав доступу до ресурсів відповідно до ролі користувача, що забезпечує ефективне управління доступом до даних.

Загалом, реалізація цих заходів забезпечує високий рівень безпеки та довіри до системи, зменшуючи ризики порушення конфіденційності та цілісності даних користувачів.

6 Деплоймент

Деплоймент - це процес виведення програмного забезпечення в експлуатацію, який включає в себе розгортання програми на сервері або іншому середовищі, необхідному для її функціонування. Це фінальний етап в розробці програмного продукту, коли програма готова до використання користувачами або клієнтами. Деплоймент включає в себе підготовку програми до розгортання, налаштування середовища для її роботи, перевірку функціональності після розгортання та, за потреби, автоматизацію процесу. Коректний деплоймент забезпечує безперебійну роботу програми та задоволення потреб користувачів.

6.1 Інструкції для розгортання проекту

Розгортання проекту включає в себе кілька кроків, які забезпечують ефективне і безпечне виведення програми в експлуатацію:

6.1.1 Підготовка до розгортання

1. Переконайтеся, що всі останні зміни коду були злиті в головну гілку репозиторію.

2. Проведіть повний аудит коду та випробування, щоб переконатися в його стабільності.

3. Забезпечте резервне копіювання існуючих баз даних перед їх оновленням.

6.1.2 Розгортання на сервері

1 Налаштуйте сервер *Azure* для *API*:

1.1 Створіть новий ресурс в *Azure* для хостингу *API*.

1.2 Налаштуйте базу даних *SQL*, наприклад на *somee.com*, якщо вона ще не створена.

1.3 Задайте налаштування безпеки, включно з брандмауером та правилами доступу до бази даних.

2 Налаштуйте середовище *Vercel* для клієнтської частини:

2.1 Створіть акаунт на *Vercel* та інтегруйте його з вашим *GitHub* репозиторієм.

2.2 Налаштуйте проект на *Vercel*, вказавши необхідні змінні середовища.

3. Впровадіть міграції бази даних:

3.1 Використовуйте *Entity Framework Core* міграції для розгортання останньої версії схеми бази даних на *Azure SQL* або, як в нашому випадку на *somee.com.*

6.1.3 Перевірка розгортання

1. Після розгортання перевірте доступність *API* та клієнтської частини через інтернет.

2. Запустіть всі функціональні та інтеграційні тести, щоб переконатися у правильності роботи системи.

3. Перевірте логи та системи моніторингу на наявність помилок або попереджень.

6.2 Опис CI/CD процесів

Неперервна інтеграція *(CI)* та неперервне розгортання *(CD)* відіграють ключову роль в автоматизації процесів розробки та розгортання в проекті:

6.2.1 CI процеси

При кожному коміті в головну гілку (зазвичай *`main`* або *`master`*), запускається автоматична збірка та тести.

Код проходить статичний аналіз та оцінюється на наявність помилок або потенційних проблем.

Успішна збірка стає кандидатом на розгортання і готується до *CD.*

6.2.2 CD процеси

За допомогою *GitHub Actions* або іншого інструменту *CD* здійснюється автоматичне розгортання збірки на тестовий сервер для додаткового випробування.

Після вдалого тестування, збірка автоматично розгортається на продуктивний сервер.

Всі етапи *CD* документуються та мають ручні точки зупину за необхідності зворотного відкату.

Впровадження *CI/CD* дозволяє проекту швидко реагувати на зміни, поліпшувати якість продукту та мінімізувати ризики, пов'язані з розгортанням нового коду.

7 **Інструкція користування для користувача**

1. Реєстрація та Вхід

1.1 Відкрийте додаток та натисніть на кнопку "Реєстрація". Введіть необхідні дані, такі як електронна пошта, пароль та основну інформацію про себе. Підтвердіть свою електронну адресу через отриманий лист.

1.2 Якщо у вас вже є акаунт, виберіть "Вхід" та введіть вашу електронну пошту та пароль.

2. Пошук житла

2.1 На головній сторінці використовуйте поле пошуку для введення міста або адреси, де ви хочете знайти житло.

2.2 Вкажіть дати прибуття та від'їзду для визначення доступності житла.

2.3 Оберіть кількість дорослих, дітей та домашніх тварин, якщо це необхідно.

3. Бронювання житла

3.3 Перегляньте список доступних об'єктів і виберіть той, який вам найбільше підходить.

3.4 Клацніть на обраному житлі, щоб дізнатися більше про умови, ціну, зручності та переглянути фотографії.

3.5 Натисніть на кнопку "Бронювати" та введіть необхідну інформацію для завершення бронювання. Підтвердіть ваше бронювання, виконавши оплату.

4. Управління бронюваннями

4.1 Активні та минулі бронювання можна переглянути у вашому профілі під розділом "Мої бронювання".

4.2 Виберіть бронювання, яке хочете змінити або скасувати, та слідуйте інструкціям для внесення змін або відміни бронювання.

5. Взаємодія з господарем

5.1 Використовуйте систему повідомлень у додатку для спілкування з господарем щодо деталей вашого перебування.

5.2 Після завершення вашого перебування залиште відгук про житло та господаря, що допоможе іншим користувачам у їх виборі.

6. Налаштування та підтримка

6.1 Ви можете змінити особисту інформацію, фотографію профілю та контактні дані у налаштуваннях вашого профілю.

6.2 У разі виникнення питань або проблем використовуйте розділ "Допомога" для отримання інструкцій або звернення до служби підтримки.

8 Плани на майбутнє

1. Розширення функціональності

– *Інтеграція з іншими сервісами:* Планується розробка *API* для інтеграції з популярними платформами бронювання та туристичними сервісами, що дозволить користувачам отримувати більш широкий спектр послуг.

– *Мобільний додаток:* Розробка мобільних додатків для iOS та Android для забезпечення кращого доступу і зручності користувачів.

2. Покращення інтерфейсу користувача

– *Персоналізація досвіду користувачів:* Впровадження функцій персоналізації на основі поведінки користувачів для надання рекомендацій, що відповідають їхнім інтересам.

– *Оновлення дизайну:* Модернізація веб-інтерфейсу та мобільних додатків для забезпечення більш інтуїтивного і сучасного дизайну.

3. Збільшення бази пропозицій

– *Додавання нових типів помешкань:* Розширення асортименту нерухомості, включаючи унікальні помешкання, як-от замки, хижі та ексклюзивні вілли.

– *Глобалізація:* Розширення ринку за межі поточних географічних обмежень для включення нових країн і регіонів.

4. Безпека та приватність

– *Підвищення рівня безпеки:* Вдосконалення систем безпеки для захисту даних користувачів та запобігання будь-яким формам кібератак.

– *Регулярний аудит та відповідність:* Проведення регулярних аудитів безпеки та забезпечення відповідності міжнародним стандартам приватності та захисту даних.

5. Розширення партнерської мережі

– Співпраця з місцевими постачальниками: Налагодження взаємодій з місцевими туристичними агентствами та постачальниками послуг для створення вигідних пропозицій для користувачів.

– Корпоративні програми: Розвиток корпоративних програм, які залучатимуть бізнес-клієнтів до використання платформи для організації ділових поїздок.

6. Підтримка та обслуговування клієнтів

– Підтримка багатомовності: Додавання підтримки додаткових мов для поліпшення досвіду користувачів з різних країн.

– Розвиток системи підтримки: Розробка більш ефективної системи обслуговування клієнтів, включаючи чат-ботів та штучний інтелект для автоматизації відповідей на часті запитання.

Ці плани спрямовані на забезпечення сталого розвитку проекту, розширення ринкової частки та покращення якості обслуговування користувачів, забезпечуючи високий рівень задоволення та відданості клієнтів.

Висновок

Цей проект є сучасним веб-рішенням, яке забезпечує користувачам зручний інтерфейс для бронювання житла в різних куточках світу. Завдяки інтеграції передових технологій та впровадженню сучасних заходів безпеки, платформа гарантує надійність, безпеку даних та зручність у використанні.

Проект активно використовує автоматизовані процеси та *API* для забезпечення плавності та ефективності сервісів, що робить його адаптивним до потреб користувачів та здатним швидко реагувати на змінні вимоги ринку. Система авторизації та аутентифікації з двофакторною перевіркою і використанням JSON Web Tokens забезпечує високий рівень захисту акаунтів користувачів.

Майбутнє проекту обіцяє подальше розширення функціональності, впровадження додаткових модулів для персоналізації досвіду користувачів та використання новітніх технологій для оптимізації процесів. Планується також розвиток мобільних додатків для забезпечення кращої доступності та зручності використання сервісу на різних пристроях.

Завершуючи, проект може продемонструвати свою ефективність і популярність серед користувачів, пропонуючи надійні та безпечні рішення для пошуку та бронювання житла по всьому світу. Він заслуговує на подальшу увагу та інвестиції для реалізації запланованого потенціалу та забезпечення якісного обслуговування своїх користувачів.

Список використаної літератури

1 ***Документація Microsoft по ASP.NET Core*****:**

- Microsoft. (2023). *ASP.NET Core documentation*. Знайдено на https://docs.microsoft.com/aspnet/core/

2 ***Матеріали по React та Node.js*****:**

- Facebook. (2023). *React – A JavaScript library for building user interfaces*. Знайдено на https://reactjs.org/
- OpenJS Foundation. (2023). *Node.js*. Знайдено на https://nodejs.org/en/docs/

3 ***Документація Material-UI*****:**

- Material-UI. (2023). *Material-UI: A popular React UI framework*. Знайдено на https://material-ui.com/

4 ***Книги та підручники по веб-розробці*****:**

- Flanagan, D. (2021). *JavaScript: The Definitive Guide*. O'Reilly Media.
- Mead, A. (2022). *The Complete Node.js Developer Course*. Udemy.

5. ***Ресурси з безпеки даних*****:**

- OWASP. (2023). *OWASP Top Ten Web Application Security Risks*. Знайдено на https://owasp.org/www-project-top-ten/

6 ***Руководства по CI/CD*****:**

- Forsgren, N., Humble, J., & Kim, G. (2018). *Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations*. IT Revolution Press.
- Walls, M. (2017). *Continuous Delivery with Docker and Jenkins: Delivering software at scale*. Packt Publishing.

7 ***Статті та блоги з веб-розробки*****:**

- Smashing Magazine. (2023). *Articles on web development*. Знайдено на https://www.smashingmagazine.com/articles/

8. ***Інформація з масштабування веб-додатків*****:**

- Martin, R. C. (2017). *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Prentice Hall.

9 *Курси та відеоуроки*:

- Codecademy. (2023). *Learn to code*. Знайдено на https://www.codecademy.com/
- Coursera. (2023). *Computer Science courses.* Знайдено на https://www.coursera.org/browse/computer-science