Київський столичний університет імені Бориса Грінченка
Факультет інформаційних технологій та математики
Кафедра комп'ютерних наук

Допущено до захисту

Завідувач
кафедри комп'ютерних наук,
_____к.т.н., доцент_____
(науковий ступінь, вчене звання)

_____ Ірина МАШКІНА
(підпис)
« ___ » _____ 202__ р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня «бакалавр»
Спеціальність 122 Комп'ютерні науки
Освітня програма 122.00.01 Інформатика

Тема:
ВЕБ-СИСТЕМА АВТОМАТИЗАЦІЇ БІЗНЕС ПРОЦЕСІВ РЕАЛІЗАЦІЇ НЕРУХОМОГО МАЙНА

Виконав
студент групи ІНб-1-21-4.0д
Криволапов Гліб Ярославович
(ПІБ)

_____
(підпис)

Науковий керівник:
_____к.т.н., доцент_____
(науковий ступінь, вчене звання)

_____ Євген ІВАНІЧЕНКО
(підпис)

Київ – 2025

Київський столичний університет імені Бориса Грінченка
Факультет інформаційних технологій та математики
Кафедра комп'ютерних наук

«Затверджую»

Завідувач
кафедри комп'ютерних наук,
_____к.т.н., доцент_____
(науковий ступінь, вчене звання)

_____ Ірина МАШКІНА
(підпис)
« ___ » _____ 202__ р.

ЗАВДАННЯ НА КВАЛІФІКАЦІНУ РОБОТУ БАКАЛАВРА

«Веб-система автоматизації бізнес процесів реалізації нерухомого майна»
Виконавець – студент групи ІНб-1-21-4.0д,
спеціальності 122 Комп'ютерні наук,
освітньої програми 122.00.01 Інформатика
Криволапов Гліб Ярославович

1. Вихідні дані: *Розроблена вебсистема автоматизації бізнес-процесів реалізації нерухумого майна. Публікації за напрямом дослідження.*
2. Основні завдання: *Здійснити огляд існуючих аналогів за темою роботи. Обґрунтувати мету, об'єкт, предмет та наукові основи дослідження. Розробити завдання, структуру та зміст бакалаврської роботи. Обрати та обґрунтувати методи і засоби дослідження. Підготувати матеріали до розділів роботи відповідно до індивідуального завдання. Розробити вебсистему автоматизації бізнес процесів реалізації нерухомого майна. Розробити алгоритми, обрати апаратні та програмні засоби для розробки вебзастосунку, створити діючий прототип. Провести аналіз результатів дослідження, скласти висновки та оформити бакалаврську роботу відповідно до затверджених на кафедрі вимог.*
3. Пояснювальна записка: *Обсяг – 84 стор. формату А4 комп'ютерного набору з дотриманням вимог стандарту і методичних рекомендацій кафедри.*
4. Графічні матеріали: *не передбачено.*
5. Додатки: *лістинг програми.*
6. Строк подання роботи на кафедру: *«___» _____2025 р.*

Науковий керівник
к.т.н., доцент
_____ Іваніченко Є.В.
дата

Виконавець:
_____ Криволапов Г.Я.
дата

Календарний план

№	Назва етапу дипломної роботи	Строк виконання етапу роботи	Примітка
1	Формування теми дипломної роботи бакалавра	10.11.2024	Виконано
2	Ознайомлення з методичними рекомендаціями до дипломної роботи	25.11.2024	Виконано
3	Складання змісту та календарного плану роботи	09.12.2024	Виконано
4	Підбір літератури, складання завдання	09.12.2024	Виконано
5	Написання реферату та вступу	09.12.2024	Виконано
6	Написання першого розділу	14.12.2024	Виконано
7	Написання другого розділу	29.03.2025	Виконано
8	Написання третього розділу	29.03.2025	Виконано
9	Написання висновків	10.05.2025	Виконано
10	Виправлення зауважень	20.05.2025	Виконано
11	Створення презентації	20.05.2025	Виконано
12	Захист матеріалів дипломної роботи на засіданні кафедри	22.05.2025	Виконано
13	Офіційний захист матеріалів дипломної роботи на засіданні екзаменаційної комісії	18.06.2025	

Здобувач_____

Керівник роботи _____

РЕФЕРАТ бакалаврської роботи

Кваліфікаційна робота: 84 с., 47 рис., 5 табл., 39 джерел.

Актуальність теми роботи лежить у ідеї допомогти забудовнику здійснювати реалізацію нерухомості, а клієнтам – заощаджувати власний час.

Об’єкт дослідження – процес реалізації об’єктів нерухомості.

Предмет дослідження – теоретичні, програмні засоби, підходи та принципи створення програмної системи продажі нерухомості забудовником.

Метою роботи є підвищення ефективності та автоматизації процесу продажу об’єктів нерухомості забудовником за допомогою розробленого програмного забезпечення.

У результаті виконання роботи було опрацьовано теоретичні основи створення сайтів із продажі нерухомості, проаналізовано конкурентні системи, спроєктовано структуру та архітектуру системи, здійснено реалізацію системи продажі нерухомості. Створена система дозволяє автоматизувати основні бізнес-процеси реалізації об’єктів нерухомості забудовником. Для створення вебсистеми було використано мови програмування TypeScript і C# та технології Angular, Angular Material UI, ASP.NET Core, систему керування базами даних PostreSQL.

Практичне значення отриманих результатів полягатиме у тому, що розроблена система дозволяє автоматизувати та цифровізувати, відповідно, підвищуючи ефективність процесів, пов’язаних з реалізацією об’єктів нерухомості, а наведені теоретичні, проєктні та практичні відомості можуть бути використані для побудови аналогічних систем.

Ключові слова: вебзастосунок/сервіс, архітектура вебзастосунку, фреймворк, програмна система, об’єкти нерухомості, електронна комерція.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

AJAX (англ. Asynchronous JavaScript and XML) – набір методів веброзробки, який використовує різні вебтехнології на стороні клієнта для створення асинхронних вебзастосунків

UI (англ. User interface, Інтерфейс користувача) – засіб зручної взаємодії користувача (людини) з інформаційною системою

UX (англ. User Experience, Користувацький досвід) – це загальне враження користувача від взаємодії з інформаційною системою, продуктом або сервісом. UX охоплює зручність, ефективність, емоції, очікування та задоволеність користувача під час використання продукту

API (англ. Application Programming Interface) – це механізми, які дозволяють двом програмним компонентам взаємодіяти один з одним за допомогою набору визначень і протоколів

JWT (англ. JSON Web Token) – це відкритий стандарт (RFC 7519), який визначає компактний і автономний спосіб безпечної передачі інформації між сторонами у вигляді об'єкта JSON

SQL (англ. Structured Query Language, Структурована мова запитів) – це мова запитів, призначена для керування даними в реляційній базі даних

IDE (англ. Integrated development environment , Інтегроване середовище розробки) – комплексне програмне рішення для розробки програмного забезпечення

Бекенд (Backend) – це внутрішня частина продукту, яка знаходиться на сервері та прихована від користувачів

Фронтенд (Frontend) – презентаційна частина застосунків, інтерфейс користувача і пов’язані з ним компоненти

ВСТУП

У сучасному світі цифрові технології стали невід'ємною частиною нашого повсякденного життя. Зі збільшенням кількості користувачів мережі Інтернет, щодня збільшується кількість ресурсів, що надають різноманітну інформацію, включаючи рекламний контент. Ефективність такої реклами постійно зростає. Продавці стикаються з необхідністю підвищення ефективності своїх робочих процесів для якісного, швидкого та продуктивного управління замовленнями та товарними запасами, а також для залучення більшої кількості потенційних клієнтів. Багато власників агентств розуміють, що власна вебсистема може стати потужним інструментом для залучення клієнтів. А у контексті повномасштабної війни в Україні потреба у сучасних цифрових рішеннях зростає ще більше. Багато людей втратили свої домівки або були змушені переїхати в інші, відносно безпечніші регіони. У зв'язку з цим попит на житло значно зріс, особливо в містах із меншою кількістю обстрілів. Це створює нові виклики для забудовників, які повинні оперативно реагувати на зміну ринку.

Актуальність теми роботи лежить у ідеї допомогти забудовнику здійснювати реалізацію нерухомості, а клієнтам – заощаджувати власний час.

Метою роботи є підвищення ефективності та автоматизації процесу продажу об'єктів нерухомості забудовником за допомогою розробленого програмного забезпечення, яке матиме зрозумілу архітектуру, а також буде виконувати потреби більшості користувачів.

Для успішної реалізації мети, необхідно вирішити **наступні завдання**:

- дослідити сучасні підходи до розроблення і впровадження вебсистем;
- порівняти існуючі аналоги, виділити їх основні переваги та недоліки;
- проаналізувати архітектурні рішення і обрати програмні засоби для реалізації вебсистеми;

- спроєктувати модель вебсистеми за допомогою структурних та поведінкових діаграм;
- реалізувати програмно вебсистему для автоматизації процесів реалізації об’єктів нерухомостей забудовником;
- здійснити розгортання вебсистеми з передбаченням можливості моніторингу та інтерпретації вебтрафіку.

Об’єктом дослідження є процес реалізації об’єктів нерухомості.

Предметом дослідження є теоретичні, програмні засоби, підходи та принципи створення програмної системи продажи нерухомості забудовником.

Методами дослідження виступають: моделювання – використовувалось при проєктуванні інформаційної системи; порівняння було використано для зіставлення систем-аналогів та підходів до створення системи автоматизації процесів реалізації об’єктів нерухомості; абстрагування – для виділення ключових елементів дослідження; аналіз та синтез застосовувались при огляді структури системи автоматизації процесів реалізації об’єктів нерухомості, описі її елементів; системний підхід – для ефективного розгляду застосунку реалізації об’єктів нерухомості як цілісної системи.

Практичне значення одержаних результатів полягає у тому, що розроблена система дозволяє автоматизувати та цифровізувати, відповідно, підвищуючи ефективність процесів, пов’язаних з реалізацією об’єктів нерухомості, а наведені теоретичні, проєктні та практичні відомості можуть бути використані для побудови аналогічних систем.

Галузь застосування – ринок нерухомості та цифрові платформи для купівлі, продажу й оренди об’єктів.

Апробація результатів – результати дослідження були представлені на міжнародній науково-практичний конференції «Applied Information Systems and Technologies in the Digital Society» (ISSN 3041-2323).

Бакалаврська робота складається із вступу, трьох розділів, висновку і списку використаних джерел.

1 ОГЛЯД СУЧАСНИХ ПІДХОДІВ ДО РОЗРОБЛЕННЯ І ВПРОВАДЖЕННЯ ВЕБСИСТЕМ

1.1 Роль вебзастосунків у світі

Для людини XXI століття вебзастосунки стали невід'ємною частиною життя, так як вони слугують основою для сучасного зв'язку та цифрової взаємодії. Це, в певному сенсі, комп'ютерні програми, доступ до яких і взаємодія з якими здійснюється через веббраузер з підключенням до мережі Інтернет. Іншими словами, вони не запускаються операційною системою користувача, як традиційні десктопні програми, а доступні через спеціальну комп'ютерну програму – веббраузер, що свідчить про їхню залежність як від браузера, так і вебсервера для управління запитами користувачів [1]. Це означає, що більша частина навантаження прилягає на сторону сервера, відповідального за обробку даних та їх повернення.

Історія вебзастосунків бере свій початок з перших днів існування Всесвітньої павутини, яка спочатку повністю складалася зі статичних сторінок (рис. 1.1) [2].

Рисунок 1.1. Перша у світі вебсторінка [2]

Впровадження CGI (англ. Common Gateway Interface, загальний інтерфейс шлюзу) та інших технологій на початку 1990-х років почало змінювати цю ситуацію.

Пізніші розробки JavaScript і AJAX дозволили зробити сторінки динамічними і швидко реагуючими, фактично перетворивши їх на інтерактивні вебзастосунки, якими ми звикли користуватися сьогодні [3]. Вищезгадані технології дозволили з’явитися онлайн-формам, системам управління контентом і платформам електронної комерції, і незабаром вебзастосунки змінили те, як ми робимо покупки, здійснюємо банківські операції або навчаємося. Завдяки цьому вебзастосунки вирішили незліченну кількість практичних проблем і спростили операції, що вимагали великих ресурсів для виконання. Вони усунули географічні бар’єри для здійснення покупок, навчання та спілкування в режимі реального часу з людьми по всьому світу. Мобільні технології та адаптивний дизайн також зробили вебзастосунки доступними і придатними для використання на різних пристроях, що покращило доступ до них і користувацький досвід загалом. Один з найяскравіших прикладів – це сфера електронної комерції. Без вебзастосунків покупки вимагали поїздок до магазинів, що займало багато часу і обмежувало людей в більшості місцем їхнього проживання та графіком роботи магазинів. З приходом таких сайтів, як Amazon та eBay, покупці отримали можливість переглядати, порівнювати та купувати товари з усього світу в будь-який зручний для них час. Ці платформи не лише дозволити зберегти час і зусилля, але й надали ширший асортимент товарів за більш конкурентоспроможними цінами, до яких користувач може отримати доступ коли і де завгодно.

У майбутньому вебзастосунки продовжуватимуть розвиватися, оскільки в них інтегрується все більше і більше нових технологій, таких як штучний інтелект і машинне навчання, розширюючи межі можливого завдяки автоматизації та оптимізації [4]. Цей прогрес принесе ще більше зручності та підвищить ефективність виконання повсякденних завдань, водночас вимагаючи

від вебзастосунків значних покращень персоналізації, доступності та функціональності.

1.2 Етапи розробки вебзастосунків

Розробка вебзастосунків – це процес створення прикладної програми. Точніше, прикладної програми, яка зберігається на сервері, доставляється користувачам через активне інтернет-з'єднання через веббраузер. Користувачам не потрібно завантажувати або зберігати вебзастосунки на своєму пристрої, як-от на ноутбуці або мобільному пристрої, тому що доступ до вебзастосунків здійснюється через Інтернет.

Існує різноманіття типів вебзастосунків, які використовуються для вирішення різних завдань. Вебзастосунки можуть варіюватися від простого статичного односторінкового інтерфейсу до складних систем, які включають розширені хмарні версії програмного забезпечення, наприклад, Microsoft Office 365, що містить такі програми, як Word, Excel.

Коли відбувається зустріч з командою розробників, проводиться аналіз ринку сьогодення, відбувається детальне ознайомлення середовище бізнесу замовника, його складові та найголовніше визначення цілей до створення цього застосунку. Проводиться брифінг із замовником, у ході якого фіксуються всі вимоги та побажання до майбутнього проєкту, чітко позначаються функції, алгоритм [5]. Потім відбувається підбір технологій, які підходять під зазначене завдання, щоб створити не просто продукт, а якісну річ, яка буде потрібна потенційним користувачам і буде зручна у використанні.

Наступний етап передбачає письмовий аналіз вимог, що дозволяє оцінити проєкт та глибоко вивчити технологічні аспекти. Головне завдання полягає в створенні зрозумілого, функціонального та детально побудованого вебзастосунку, що задовольнятиме потреби замовника [5].

На етапі збору інформації важливо правильно розуміти цільову аудиторію та їх потреби. Наприклад, якщо вебзастосунок призначений для людей похилого

віку, необхідно враховувати їх можливі проблеми із зором чи проблеми з рухами. Організація простого та зрозумілого меню, великі шрифти та інші адаптації можуть значно покращити їх користувацький досвід та зробити взаємодію з вебзастосунком комфортнішою. Отже, потрібно зробити так, щоб команда розробників була орієнтована саме на потенційних користувачів та максимально задовольняла їх потреби. В цьому завданні допоможе тільки ефективне планування та дизайн вебзастосунку.

Другим етапом є формування команди, ескізи та аналіз наявного коду. Цей етап вельми важливий, оскільки від правильно підібраної та організованої команди залежить подальший успіх проєкту. Він буде надсилати завдання, контролювати кожен етап роботи – від планування і проєктування до програмування та підтримки після випуску продукту на ринок. Також цей менеджер буде відповідати за відбір та запрошення найбільш кваліфікованих та відповідних спеціалістів для вашого проєкту. Найголовніше, що він буде тримати вас в курсі звітів про прогрес, а також документувати та ділитися вашими відгуками з командою розробників. Після встановлення мети та завдань проєкту, UI/UX дизайнер робить перший крок у створенні вебзастосунку – підготовку серії ескізів. Ці ескізи, як правило, є каркасами; прості, обрізані діаграми, без використання кольорів, графіки або користувацьких шрифтів. Простіше кажучи ці ескізи розроблені, щоб надати клієнту загальне уявлення про те, як виглядатиме остаточний макет, до початку дизайнерських робіт. Клієнти можуть легше зрозуміти структуру сторінок та функціонал, а також внести свої пропозиції до покращення дизайну в ранній стадії розробки. Якщо у вашій організації є наявні кодові бази, наприклад уже наявний вебзастосунок, команда розробників перевірить ці коди. Це гарантує, що команда почне все з чистого аркуша, одночасно усуваючи будь-які помилки або невідповідності коду, які можуть вплинути на якість вебзастосунку. Вхідні дані клієнта є важливими на цьому етапі, оскільки дані замовника визначатимуть зовнішній вигляд вебзастосунку. Ретельне уточнення вимог замовника є важливим для

успішної реалізації проєкту. Важливо мати чітке уявлення про бажаний дизайн вебзастосунку та вид користувацького досвіду, який замовник прагне забезпечити цільовій аудиторії. Замовнику важливо бути ясним та конкретним у вказівках та побажаннях. При потребі завжди краще уточнити деталі та уточнення, а також звернутися за допомогою до спеціалістів. Команда розробників завжди готова надати необхідну підтримку та поради для прийняття обґрунтованих рішень.

Наступним третім етапом є вдосконалення дизайну та інтерактивних елементів вашого вебзастосунку. Після узгодження ескізів, дизайнер береться до створення різноманітних макетів, кожен з яскраво вираженими індивідуальними характеристиками, такими як колірна гама, шрифтове оформлення, графічні елементи, кнопки, анімаційні переходи, тощо. Клієнт отримує можливість ознайомитися з кожним з варіантів, обрати той, що сподобався найбільше, і дати свій відгук. Незалежно від того, чи є у вас чітке бачення дизайну, чи ви не впевнені, в якому напрямку рухатися, важливо спілкуватися зі своїм дизайнером. Вони вислухають ваші ідеї та допоможуть зробити важкий творчий вибір. Важливим аспектом є також розуміння типу вебзастосунку, який ви прагнете створити, вибір стилю, що відповідає вашій цільовій аудиторії, а також технічні вимоги для Вашого застосунку. Завдяки цьому процесу Ви матимете чітке усвідомлення того, як ваш вибір дизайну буде сприйнятий та виконаний в майбутньому.

Після того, як естетика вебзастосунку встановлена, наступним четвертим етапом є забезпечення його функціональності, що передбачає програмування зовнішніх і внутрішніх аспектів вебзастосунку. Front-end розробка – це використання різних вебтехнологій, таких як HTML, CSS і JavaScript, для створення частин вебзастосунку, які користувач може бачити та взаємодіяти з ними. Адаптивний дизайн, правила завантаження та відображення елементів, клікабельність, наявність анімації, різноманітних компонентів для взаємодії – все це має ключове значення та підбирається індивідуально для кожного кейсу

[5]. Тим часом back-end розробка передбачає створення основної частини функціоналу програми або сайту, який запускається через інтерфейсну частину, але функціонує на стороні сервера, наприклад, на базах даних, внутрішній логіці, інтерфейсах прикладного програмування (API) та архітектурі.

На п’ятому етапі команда перейде до написання контенту та маркування. Чіткий, лаконічний і розбірливий текст грає важливу роль у покращенні взаємодії з користувачем вебзастосунку. Використання належно структурованого тексту сприяє полегшенню розуміння можливостей застосунку та очікуваних результатів від певних дій. За допомогою відмінного тексту ваші користувачі зможуть зрозуміти, що може робити ваш вебзастосунок, куди вони можуть піти і яких результатів вони можуть очікувати, виконуючи певні дії [6].

Різні типи вмісту, який потрібен вашому вебзастосунку, включають заголовки, підзаголовки, описи, мітки кнопок, тощо. Важливим аспектом є також узгоджена робота з художниками та дизайнерами для точного розташування необхідного текстового матеріалу. Наприклад, дослідження показують, що правильно розміщений текст сприяє збільшенню ефективності взаємодії з користувачем та підвищенню його задоволення від використання застосунку. Під час створення контенту розробникам варто звернути особливу увагу на цільову аудиторію. Письмовий матеріал повинен відповідати потребам, проблемам користувачів, надаючи їм зрозумілу та корисну інформацію про можливості та способи використання застосунку для досягнення поставлених цілей. Крім того, основним завданням для розробників є доступна подача інформації не тільки для потенційних користувачів, а й для ширшого кола аудиторії.

Тестування є одним з найважливіших етапів у процесі розробки вебзастосунків. Це підтверджує, що вебзастосунок працює належним чином і відповідає всім відповідним стандартам організації та світу. Тестування зазвичай проводиться спеціальною командою із забезпечення якості (QA). Ця команда застосовує комбінацію ручного та автоматизованого тестування для

виявлення багів, помилок, збоїв та інших проблем. Їх завдання містить перевірку всіх аспектів продукту, таких як користувацький досвід, безпека, продуктивність, функціональність та швидкість відгуку. Наприклад, при автоматизованому тестуванні можуть використовуватися інструменти для створення скриптів тестування, які автоматично виконують послідовність кроків та перевіряють реакцію програми на них. Це дозволяє тестувальнику швидко виявляти та усувати можливі проблеми в роботі програми. Після того, як вебзастосунок відповідає всім стандартам якості, його можна запускати в звичайних браузерах [6].

Технічне обслуговування та оновлення після запуску є заключним етапом розробки вебзастосунку. Цей етап містить регулярне технічне обслуговування, направлене на забезпечення безперебійної роботи та високої продуктивності вебзастосунку. Розробники зазвичай випускають оновлення, які містять виправлення помилок, покращення безпеки та нові функції.

Наприклад, регулярні патчі можуть вирішувати виявлені проблеми та підвищувати загальну ефективність вебзастосунку. Більш значні оновлення можуть додавати нові можливості, які поліпшують користувацький досвід та розширюють функціонал програми [5].

Завдяки постійній увазі до технічного обслуговування та оновлень, вебзастосунок залишається актуальним і конкурентоспроможним на ринку. Такий підхід допомагає забезпечити високу якість продукту та задоволення користувачів. Звичайно, при регулярному технічному обслуговуванні можна знайти та усунути вразливість безпеки до того, як станеться серйозне порушення безпеки.

Плани безперервної реалізації та вдосконалення повинні бути розроблені на ранніх стадіях життєвого циклу проєкту. Цей план має враховувати менші, незначні виправлення та покращення зручності використання, а також потенціал для більш значних оновлень, таких як впровадження нових функцій.

1.3 Підходи до розроблення сучасних вебзастосунків

Сьогодні сфера веброзробки характеризується великою різноманітністю і перебуває у стані постійної трансформації. Тому від сучасних вебзастосунків очікується, що вони повинні бути адаптивними на різних пристроях і з різним доступом до мережі, а також забезпечувати приємний користувацький досвід, який можна порівняти з нативними застосунками. Цей фактор спричинив появу цілої низки практик і архітектур розробки – від серверних застосунків до клієнтських односторінкових застосунків і прогресивних вебзастосунків.

Основою будь-якого вебзастосунку є його базові технології: HTML5, CSS3 та JavaScript. HTML5 – це остання версія мови гіпертекстової розмітки, яка має більше семантичних елементів для пояснення зміст у набагато простіший спосіб, таких як: <article>, <section>, <nav> і <footer>, що підвищує доступність і робить застосунки набагато надійнішими і зрозумілішими як для розробників, так і для пристроїв (рис. 1.2) [7].

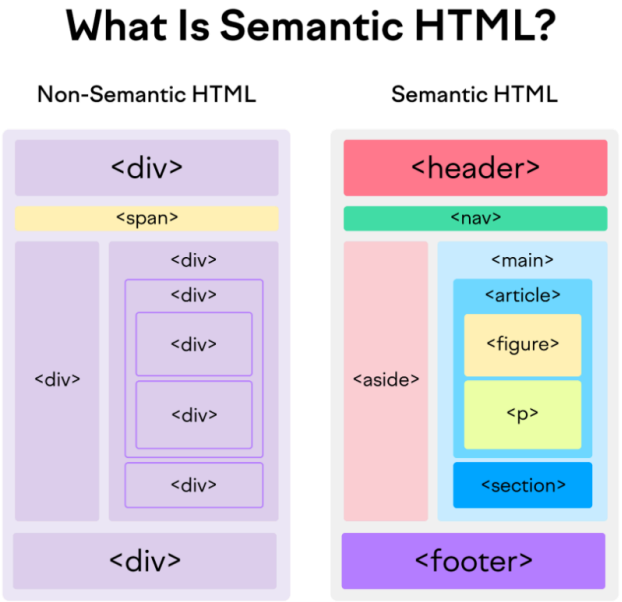

Рисунок 1.2. Порівняння Semantic та Non-Semantic HTML [7]

Доповнена потужним синтаксисом CSS3, мова стилів надає розробникам потужний засіб для створення візуально привабливих та адаптивних користувацьких інтерфейсів за допомогою розширених селекторів, анімації та

функцій адаптивного дизайну [8]. JavaScript, який колись був інструментом для написання простих сценаріїв на стороні клієнта, став основою для інтерактивних і динамічних функцій у вебзастосунках. Набагато більше функціональності було додано з впровадженням AJAX, а пізніше фреймворків, які дозволили створювати складні застосунки у більш керований та ефективний спосіб.

На стороні сервера Node.js зробив революцію у використанні JavaScript, дозволивши розробникам запускати JavaScript на сервері. Його неблокуюча архітектура, керована подіями, ідеально підходить для створення масштабованих мережевих застосунків (рис. 1.3) [9].

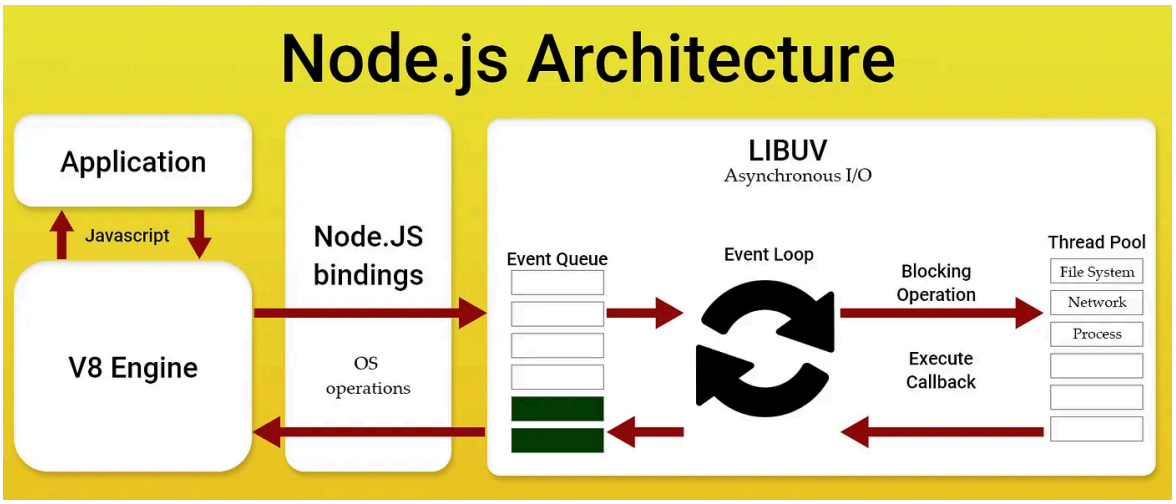

Рисунок 1.3. Архітектура бекенд фреймворку Node.js [9]

Динамічні вебзастосунки відкривають ширший спектр можливостей порівняно зі статичними вебсайтами. Вони можуть інтегруватися з різноманітними зовнішніми сервісами, такими як платіжні системи, соціальні мережі або API інших застосунків. Це дає змогу розширити функціональність застосунка та забезпечити користувачам більше можливостей.

REST (RESTful) – набір загальних принципів, які використовуються для організації взаємодії між клієнтом і сервером через протокол HTTP. Особливість REST в тому, що сервер не запам’ятовує стан користувача між запитами – в

кожному запиті передається інформація, що ідентифікує користувача (наприклад, token, отриманий через OAuth-авторизацію) і всі параметри, необхідні для виконання операції.

Взаємодія із сервером зазвичай зводиться до чотирьох операцій (4 – не-обхідний і достатній мінімум, а конкретної реалізації типів операцій може бути й більше) (рис. 1.4):

- отримання даних з сервера (як правило, у JSON форматі, або XML);
- додавання нових даних на сервер;
- модифікація (зміна) вже існуючих даних на сервері;
- видалення даних з сервера.

Безумовно, отримання даних не приводить до зміни стану сервера.

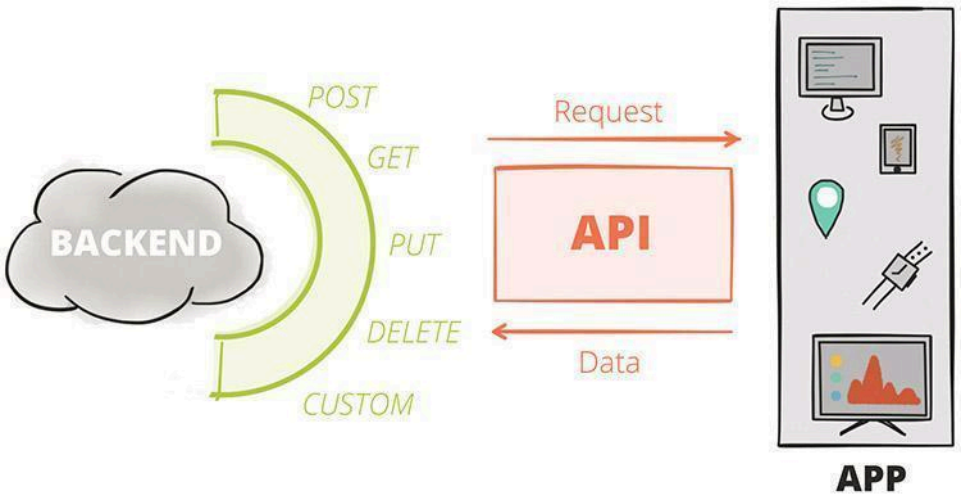

Рисунок 1.4. Структура REST API [10]

Для кожного типу операції використовується відповідний метод HTTP-запиту:

- отримання – GET (запити з використанням цього методу можуть тільки отримувати дані);
- додавання – POST (використовується для відправки сутностей до певного ресурсу. Часто викликає зміну стану або якісь побічні ефекти на сервері);

- модифікація – PUT (замінює всі поточні уявлення ресурсу даними запиту);
- видалення – DELETE (видаляє вказаний ресурс).

Також, варто зазначити, що на сьогоднішній день дедалі більше компаній відмовляються від придбання дорогого обладнання на користь хмарних сервісів. Такий підхід є зручним, адже користувачі отримують доступ до обчислювальних ресурсів через Інтернет, що дозволяє значно скоротити витрати на утримання власної інфраструктури. Провайдери хмарних технологій укладають угоди з клієнтами, пропонуючи їм потужні інструменти для роботи без потреби у купівлі та технічному обслуговуванні обладнання. Явні переваги використання хмарних сервісів порівняно з локальними серверами можна простежити при їх порівняльному аналізі (табл. 1.1) [11].

Таблиця 1.1

Порівняння хмарних сервісів та локальних серверів

Критерій	Хмарні сервіси	Локальні сервери
Вартість	Сплата за користування (модель pay-as-you-go), у більшості випадків передбачає нижчі початкові витрати	Високі початкові витрати на обладнання, ліцензії, встановлення та налаштування
Масштабованість	Легко масштабувати ресурси вгору чи вниз залежно від потреб	Складніше та дорожче масштабування, вимагає придбання та встановлення додаткового обладнання
Доступність	Доступ до даних та застосунків з будь-якого місця за умови підключення до мережі Інтернет	Обмежено доступ фізичним розташуванням сервера
Обслуговування	Провайдер хмарних послуг несе відповідальність за обслуговування, оновлення та безпеку	Потрібна власна IT-команда для обслуговування, оновлень та забезпечення безпеки
Безпека	Провайдери хмарних послуг інвестують значні кошти в безпеку, проте дані зберігаються на зовнішніх серверах	Більший контроль над безпекою, проте вимагає експертизи та ресурсів для захисту від потенційних загроз
Надійність	Висока надійність завдяки резервному копіюванню та відновленню даних	Надійність залежить від якості обладнання та резервного копіювання

1.4 Поняття вебсистем

Вебсистеми – це потужні вебзастосунки, що об’єднують кілька окремих програмних компонентів, кожен з яких виконує свою роль. Вони забезпечують гнучкий розподіл прав доступу та надають користувачам персоналізований доступ до ресурсів. Такі системи здатні обслуговувати досить велику кількість користувачів одночасно, враховуючи їхню роль, рівень доступу та потреби.

У кожного користувача в такій системі є своя роль – це може бути гість, клієнт або адміністратор. Система розмежовує їхні права. Наприклад, клієнт не має доступу до управління товарами, а адміністратор, навпаки, може змінювати параметри системи, додавати нових користувачів або редагувати інформацію. Це забезпечує як безпеку, так і зручність використання – кожен бачить лише те, що йому потрібно.

Вебсистеми працюють за принципом “клієнт-сервер”. Користувач взаємодіє з клієнтською частиною (вебінтерфейсом у браузері), яка надсилає запити на сервер. Сервер обробляє ці запити, звертається до бази даних, формує відповідь і надсилає її назад клієнту. Такий підхід дозволяє швидко обробляти великий потік запитів і масштабувати систему при збільшенні кількості користувачів (рис. 1.5).

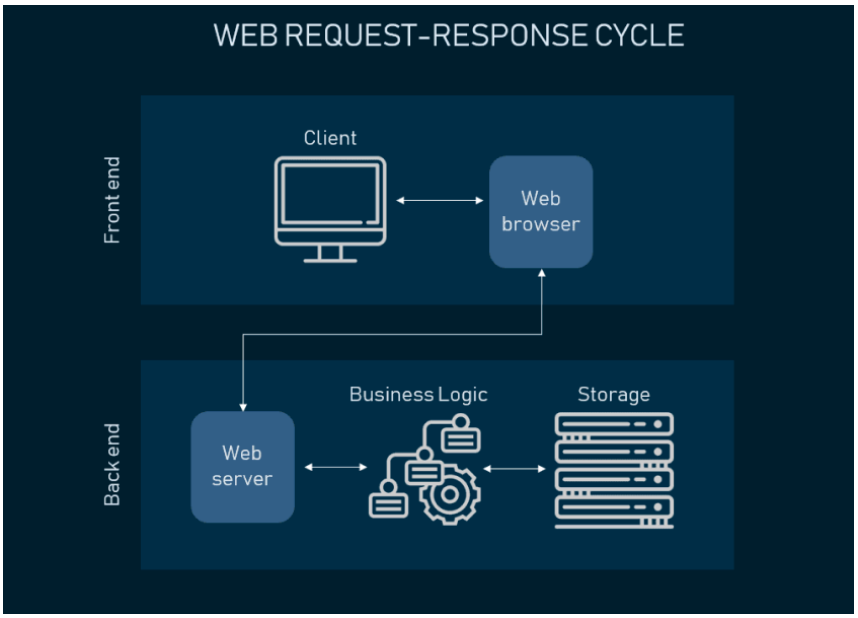

Рисунок 1.5. Компоненти архітектури вебсистеми [12]

Сучасні вебсистеми також інтегрують засоби безпеки – наприклад, автентифікацію (перевірку логіна і пароля), шифрування даних, логування дій користувачів. Це особливо важливо, коли мова йде про конфіденційну інформацію, наприклад, персональні дані або фінансову звітність.

1.5 Огляд деяких існуючих рішень автоматизації бізнес-процесів реалізації нерухомого майна

Процес реалізації об’єктів нерухомості традиційно супроводжувався великою кількістю ручної роботи, паперових документів та неефективної комунікації між забудовником, агентами з продажу та потенційними покупцями. Такі підходи часто призводили до затримок, втрати клієнтів, помилок у документації та складнощів з контролем за актуальним станом об’єктів. У сучасних умовах, коли цифрові технології активно впроваджуються у всі сфери бізнесу, з’явилася потреба в автоматизованих рішеннях, що дозволяють оптимізувати внутрішні процеси компанії та покращити взаємодію з клієнтами. Розвиток інтернету, вебтехнологій та мобільного доступу відкрив нові можливості для забудовників. Вебсистеми автоматизації дають змогу

централізовано керувати об’єктами нерухомості, відстежувати статус продажу, вести облік клієнтів, контролювати бронювання та платежі.

Однією з таких вебсистем є flatfy.ua [13], що агрегує оголошення про продаж та оренду житла з різних джерел, надаючи користувачам можливість шукати квартири, будинки та інші об’єкти нерухомості по всій Україні (рис. 1.6).

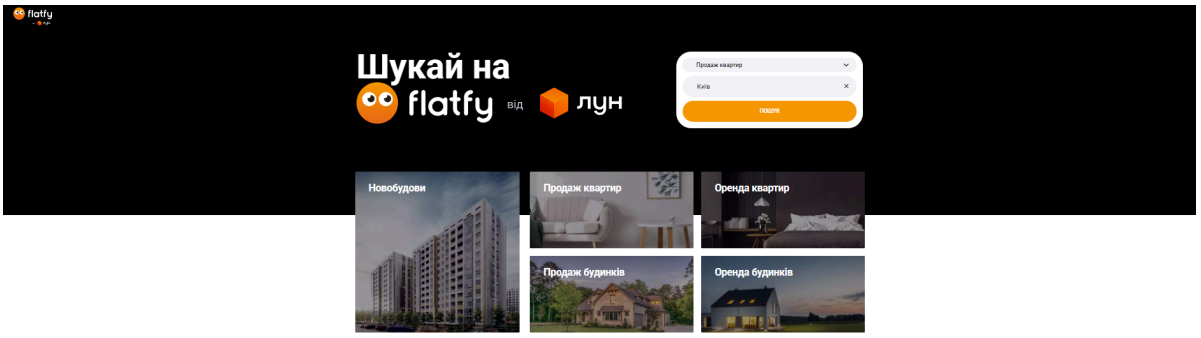

Рисунок 1.6. Головна сторінка платформи flatfy.ua

Користувачі можуть переглядати оголошення з фотографіями, описами та контактною інформацією продавців або орендодавців. Платформа також пропонує інтерактивну карту, яка допомагає візуалізувати розташування об’єктів нерухомості (рис. 1.7).

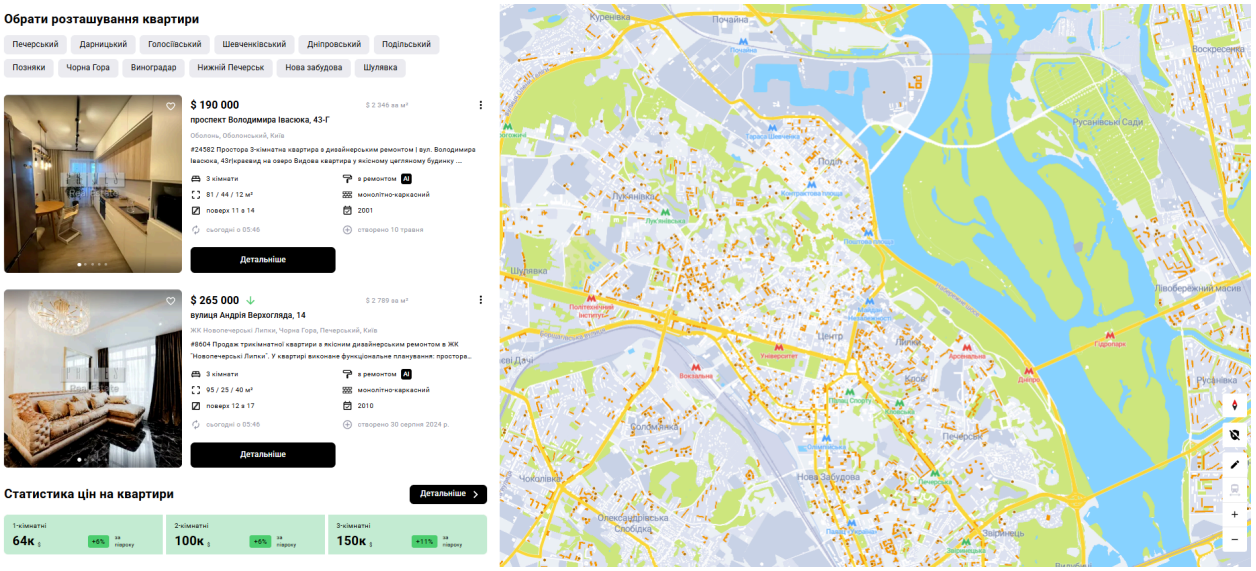

Рисунок 1.7. Сторінка купівлі квартир платформи flatfy.ua

Також у застосунку flatfy користувач може вказати ключові слова й за ними здійснити пошук нерухомості. Надається можливість вказувати параметри сортування для більш зручного та швидкого пошуку (рис. 1.8).

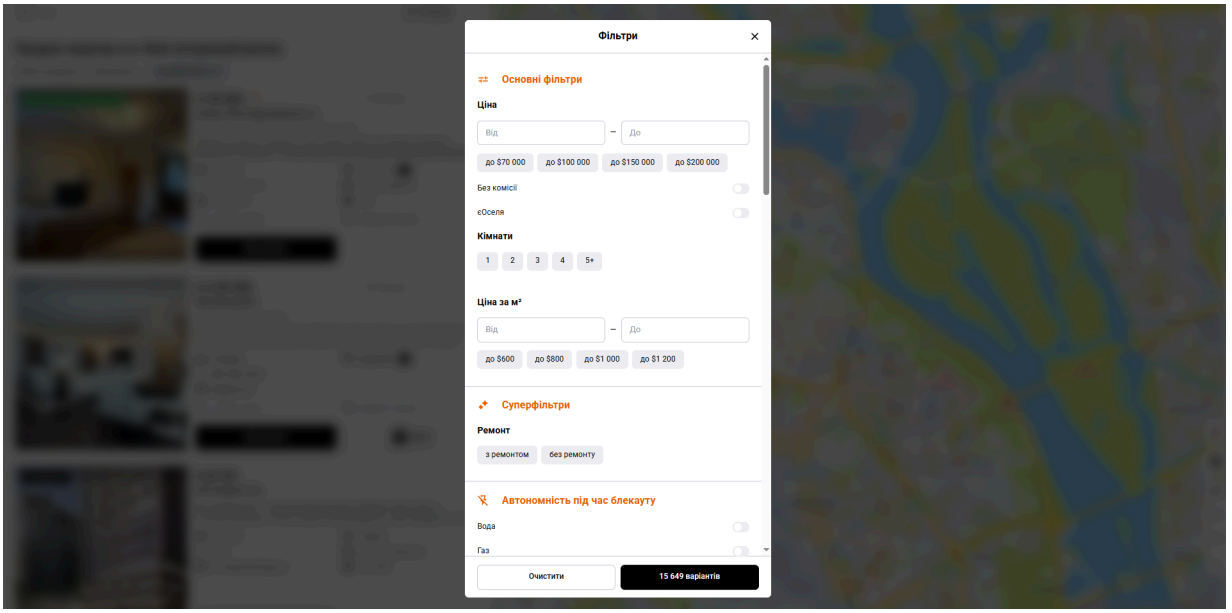

Рисунок 1.8. Фільтри пошуку нерухомостей платформи flatfy.ua

При виборі конкретної нерухомості відкривається її сторінка, де міститься детальна інформація, що є важливою для клієнта: ціна, адреса, опис, фото, відео, площа, розташування на мапі, можливі інші оголошення від цього ж власника тощо. (рис. 1.9).

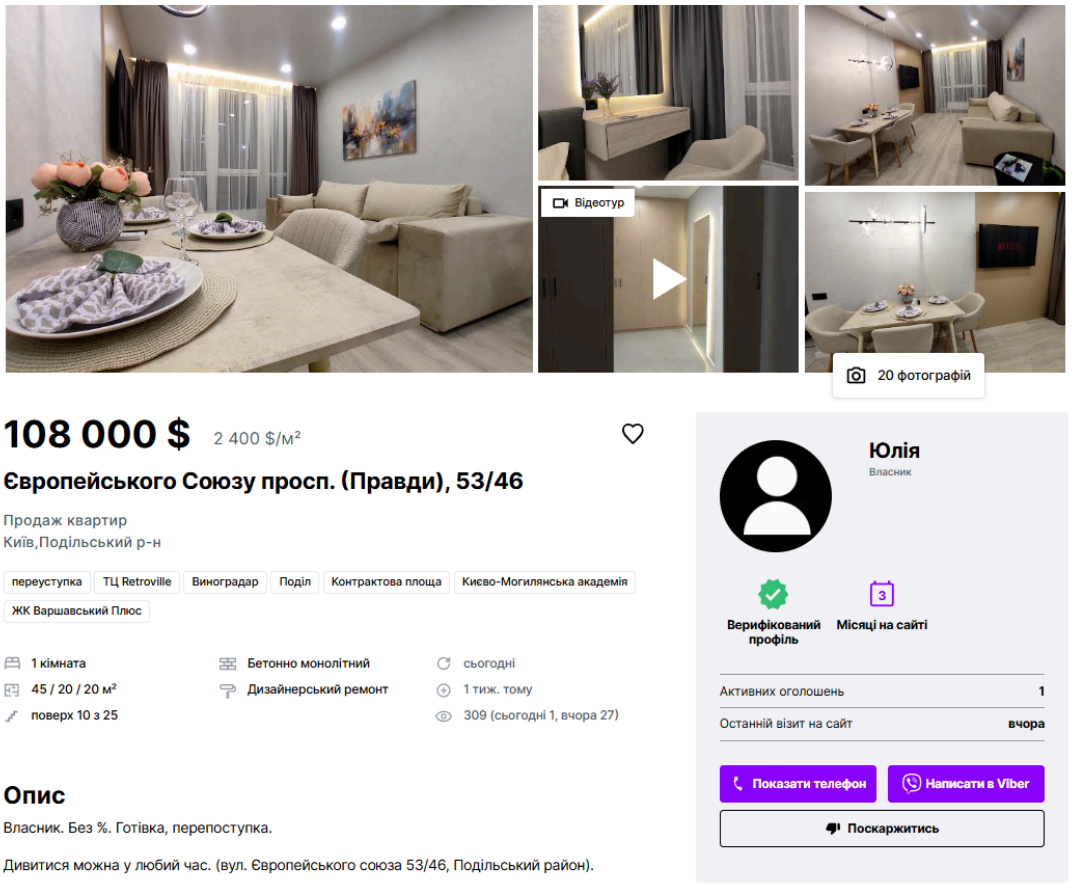

Рисунок 1.9. Сторінка конкретної нерухомості платформи flatfy.ua

Як можна побачити, зв’язок між потенційним покупцем і продавцем здійснюється шляхом прямого перенаправлення – за вказаним номером телефону або через посилання на профіль у месенджері. Такий підхід не передбачає захищеного чи зафіксованого каналу комунікації всередині самої платформи. Це створює ризики для користувача, оскільки після встановлення зв’язку між клієнтом та продавцем сайт більше не контролює хід домовленостей і не несе відповідальності за зміст розмов, умови угод або можливі шахрайські дії з боку власника нерухомості.

Серед сучасних платформ для продажі нерухомості також величезною популярністю користується dom.ria.com [14] (рис. 1.10).

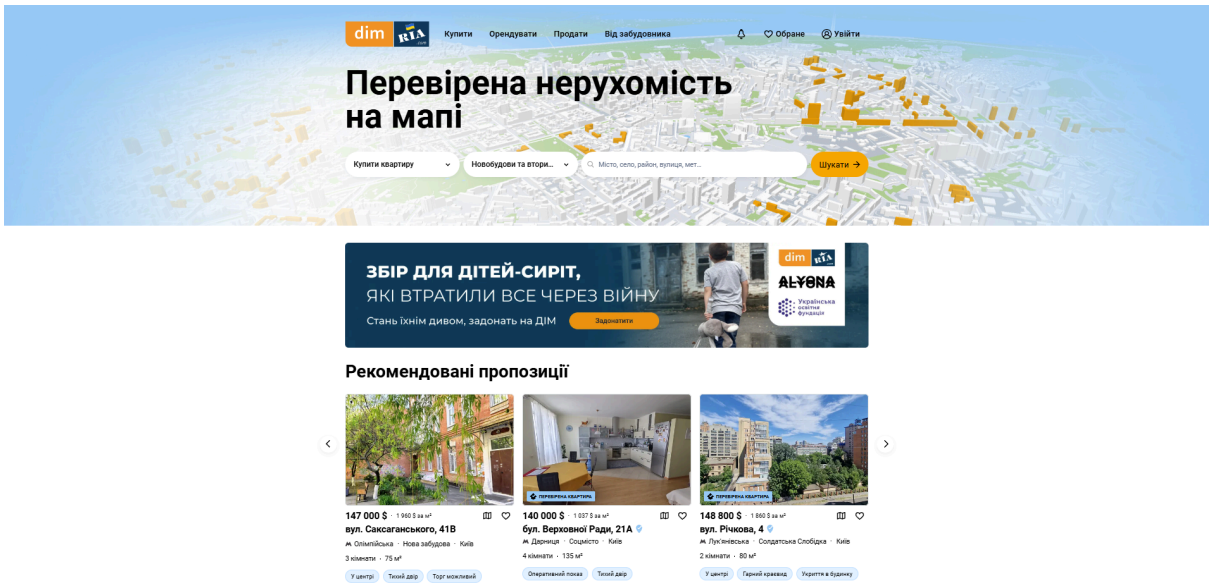

Рисунок 1.10. Головна сторінка платформи dom.ria.com

На відміну від flatfy, який є агрегатором оголошень з різних джерел і не завжди гарантує актуальність та достовірність інформації, DOM.RIA фокусується на якості та перевіреності контенту. Однією з ключових особливостей DOM.RIA є система перевірки оголошень. Платформа пропонує багаторівневу перевірку об’єктів нерухомості, яка включає в себе відеоогляди, перевірку документів та особисту інспекцію об’єктів (рис. 1.11).

Рисунок 1.11. Інформація про перевірку оголошень на dom.ria.com

Аналогічно flatfy.ua, на сторінці нерухомості міститься необхідна інформація про неї зі схожими складовими (рис. 1.12).

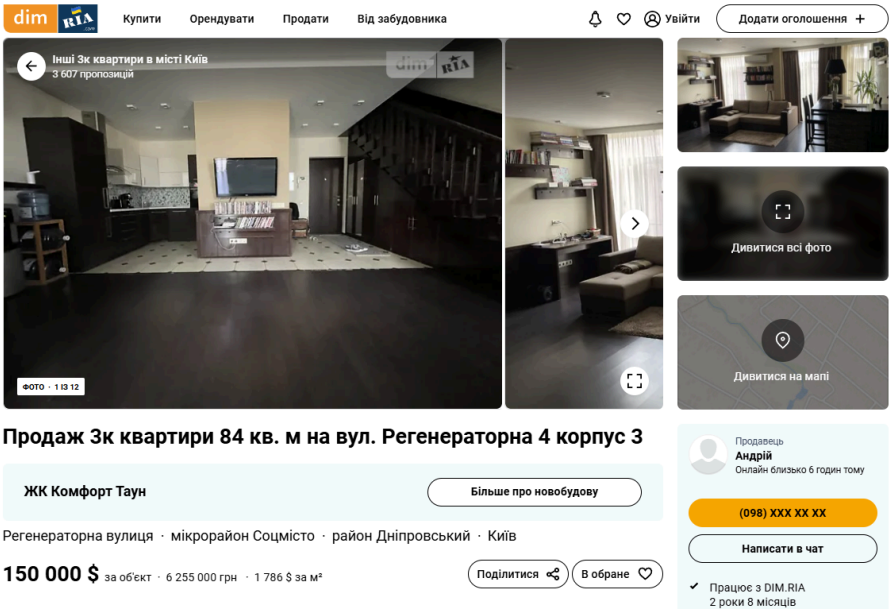

Рисунок 1.12. Сторінка конкретної нерухомості на dom.ria.com

Проаналізувавши наявну інформацію на сторінці випадково обраної нерухомості мною було виявлено наступний недолік: при перегляді місцерозташування житла на мапі виділено весь житловий комплекс, натомість немає мітки біля самого корпусу, у якому знаходиться квартира. Власник звісно надав цю інформацію в назві товару, але міг цього й не робити (рис. 1.13).

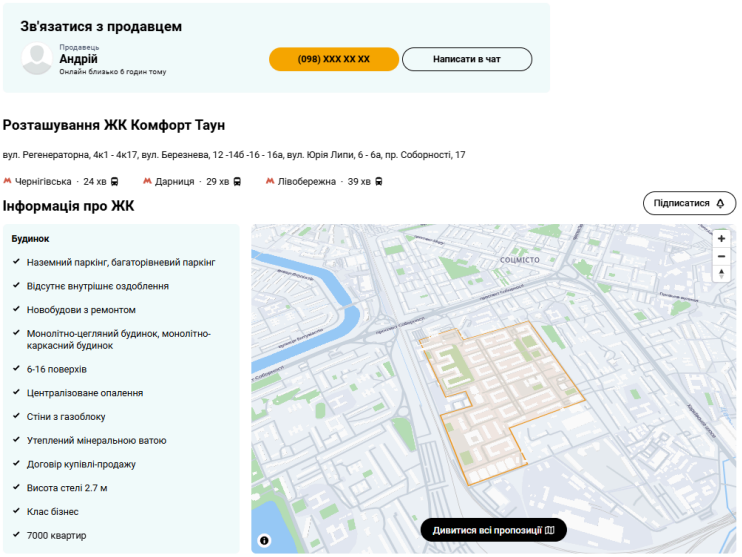

Рисунок 1.13. Розташування конкретної нерухомості на dom.ria.com

2 АНАЛІЗ АРХІТЕКТУРНИХ РІШЕНЬ І ВИБІР ПРОГРАМНИХ ЗАСОБІВ ДЛЯ РЕАЛІЗАЦІЇ ВЕБСИСТЕМИ

2.1 Принципи автентифікації у веброзробці

У веброзробці концепція авторизації, що включає в себе захист застосунку і забезпечення доступу користувачів лише до тих ресурсів, до яких вони мають дозвіл, має першочергове значення. Авторизація відрізняється від процесу автентифікації, який насправді відноситься до перевірки особи, тому що це специфікація того, що ці автентифіковані користувачі можуть робити. Концепція авторизації може бути реалізована різними способами, залежно від вимог програми та архітектури. Популярними методами є управління доступом на основі ролей, OAuth та більш динамічні підходи, такі як управління доступом на основі атрибутів [15]. Основою авторизації в сучасному світі веброзробки є JSON Web Tokens (JWT), що визначають відкритий стандарт (RFC 7519) для компактної та автономної захищеної передачі інформації між двома сторонами у вигляді JSON-об’єктів. Ці дані підписуються цифровим підписом, а отже, їм можна довіряти і їх можна перевірити. Як правило, JWT можуть бути підписані за допомогою секретного (за допомогою алгоритму HMAC) або відкритого/закритого ключа з використанням RSA або ECDSA. Вебтокен JSON складається з трьох частин: заголовка, корисного навантаження і підпису. Заголовок зазвичай складається з двох частин: тип токена – тобто JWT – і алгоритму підпису, який використовується, наприклад, HMAC SHA256 або RSA (рис. 2.1).

Рисунок 2.1. Структура заголовка JWT токена

Друга частина токена, яка містить твердження, називається корисним навантаженням. Реквізити – це твердження про сутність, зазвичай користувача, і додаткові метадані. Існує три типи реквізитів: зареєстровані, публічні та приватні реквізити [16]. Підпис призначений для захисту токена від підробки. Наприклад, токен, підписаний алгоритмом HMAC SHA256, буде згенерований таким чином, щоб зашифрувати заголовок і корисне навантаження. Загалом, вебтокени є дуже зручним інструментом для авторизації, оскільки дозволяють власнику токена отримати доступ до ресурсу без численних перевірок облікових даних користувача кожного разу.

Також, одним із найбільш розповсюджених підходів на сьогоднішній день є використання автентифікації за допомогою cookies. Це метод, при якому після входу користувача на сайт сервер створює сесію і зберігає її ідентифікатор у cookie-файлі браузера. При кожному наступному запиті браузер автоматично надсилає cookie на сервер, що дозволяє розпізнати користувача. Це забезпечує безперервність сеансу та дозволяє реалізувати доступ до захищених сторінок без повторного входу. Cookies можуть зберігатися до закриття браузера або на тривалий термін, залежно від налаштувань.

2.2 Вибір технологічного стеку для розробки вебсистеми

2.2.1 Вибір засобів для реалізації клієнтської частини

Під час проєктування та програмування вебсистем для електронної комерції необхідно враховувати низку важливих чинників, які безпосередньо

впливають на ефективність онлайн-бізнесу. Одним із ключових елементів є користувацький досвід (UX). Щоб зацікавити та утримати відвідувачів, сайт повинен мати зрозумілий і привабливий інтерфейс, зручну структуру, високу швидкість завантаження та простий процес оформлення замовлення. Для цього потрібно пройти декілька важливих етапів (рис. 2.2).

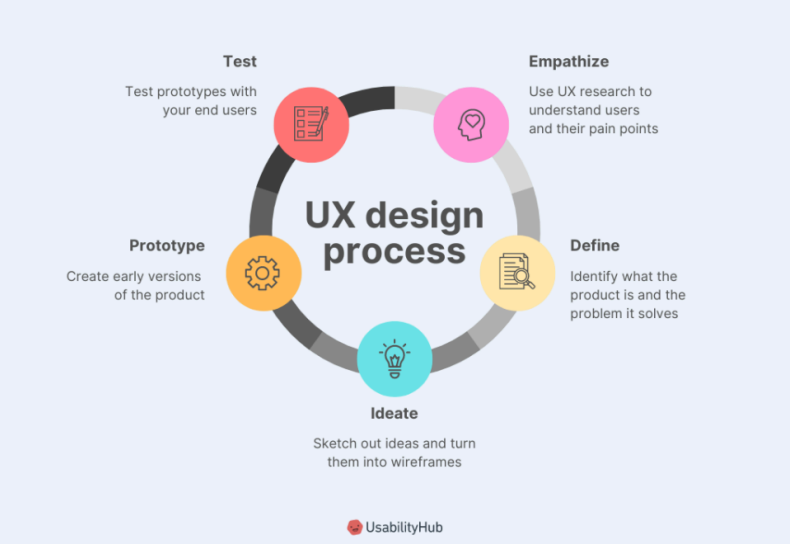

Рисунок 2.2. Етапи впровадження користувацького досвіду [17]

Надзвичайно важливо забезпечити адаптивність ресурсу до різних типів пристроїв – зокрема, смартфонів і планшетів. З огляду на зростаючу популярність оформлення онлайн-замовлень з мобільних пристроїв, сайт має коректно відображатися та функціонувати на будь-якому екрані, забезпечуючи комфортну взаємодію для кожного користувача.

Також, варто зазначити, що неабияку роль при розробці вебсистеми відіграє аналіз інформації щодо її користування. Впровадження аналітичних інструментів на кшталт Google Analytics дає змогу відслідковувати дії користувачів вебсистеми, що сприяє підвищенню її ефективності та оптимізації маркетингових стратегій [18].

Для розробки клієнтської частини однозначно знадобиться використати такі технології, як HTML, CSS, JavaScript. На цьому можна було б вже зупинитися, але я прагну, щоб дані на сторінці оновлювалися без необхідності перезавантажувати її. Подібні вебсайти називаються SPA (Single-page application).

SPA – це буквально одна сторінка, яка постійно взаємодіє з користувачем, динамічно переписуючи поточну сторінку, а не завантажуючи цілі нові сторінки з сервера (рис. 2.3). Ми бачимо приклади одно сторінкових застосунків кожен день: Trello, Facebook, Gmail, Twitter – ось кілька прикладів SPA.

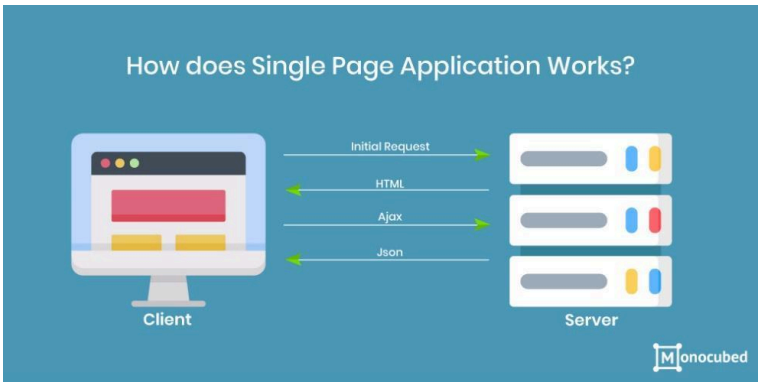

Рисунок 2.3. Принцип роботи односторінкових застосунків [19]

Особливість архітектури SPA полягає в тому, що всі елементи, необхідні для роботи сайту знаходяться на одній сторінці. Вони завантажуються при ініціалізації. Також даний вид вебзастосунків завантажує додаткові модулі після запиту від користувача. Будь-яка призначена для користувача активність фіксується для зручності навігації. Це дозволяє скопіювати посилання і відкрити сайт на тому ж етапі взаємодії на іншій вкладці, браузері або пристрої [20].

При завантаженні нових модулів в SPA контент на них оновлюється тільки частково, так як немає необхідності повторно завантажувати незмінні елементи. Це збільшує швидкість відповіді і скорочує передається обсяг даних між браузером і сервером. Для підтвердження доцільності рішення про

створення саме односторінкового застосунку, здійснімо порівняльний аналіз його із класичними багатосторінковими застосунками (табл. 2.1) [11].

Таблиця 2.1

Порівняння різних підходів до створення вебзастосунків

Критерії	**Односторінкові застосунки (SPA)**	**Багатосторінкові застосунки (MPA)**
Швидкість завантаження	Завантаження лише одного разу, динамічне оновлення даних	Кожна сторінка завантажується окремо

Продовження Таблиці 2.1

Інтерактивність	Висока, плавний користувацький досвід	Можлива затримка під час переходу між сторінками
Навантаження на сервер	Мінімізована завдяки асинхронним запитам	Висока через необхідність завантаження сторінок
SEO	Складнощі через відсутність окремих URL для сторінок	Легше оптимізується завдяки унікальним сторінкам
Масштабованість	Легко масштабуються завдяки компонентній архітектурі	Вимагає оновлення кожної сторінки окремо
Реалізація та підтримка	Вимагає більше навичок розробки, складніша підтримка	Простіша реалізація, легше підтримувати

Даний вид сайтів за способом взаємодії з користувачем найбільше схожий на роботу десктопних застосунків, але на сервері. Хоча і можливо створити SPA за допомогою лише JavaScript, HTML, CSS, на даний момент найкращим підходом для швидкої і якісної розробки таких вебзастосунків буде використання готових рішень. Найпопулярнішими на даний момент є Angular, React, Vue.js.

Vue.js є наймолодшим з трьох вказаних мною рішень. Довгий час його не підтримувала жодна з корпорацій-гігантів, але він все одно зміг гідно триматися на рівні Angular і React завдяки спільноті.

Також варто зазначити, що не буде зовсім правильно називати Vue.js і React фреймворками, адже вони скоріше – бібліотеки. Основна відмінність у тому, що Angular має все прямо з «коробки», в той час як React і Vue.js досягають такого результату за допомогою встановлення необхідних додаткових модулів.

Vue.js – це JavaScript-фреймворк для створення вебінтерфейсів з використанням шаблону архітектури MVVM (Model-View-ViewModel). Оскільки Vue працює тільки на «рівні уявлення» і не використовується для проміжного програмного забезпечення і бекенда, він може легко інтегруватися з іншими проєктами і бібліотеками. Vue.js містить широку функціональність для рівня уявлень і може використовуватися для створення потужних вебзастосунків.

До можливостей Vue.js входять:

- реактивні інтерфейси;
- декларативний рендеринг;
- зв’язування даних;
- директиви (всі директиви мають префікс «V-». В директиву передається значення стану, а в якості аргументів використовуються html атрибути або Vue JS події);
- логіка шаблонів і компоненти;
- обробка подій;
- властивості, переходи і анімація CSS, фільтри.

Через можливі певні складнощі з масштабуванням і значним розширенням системи в майбутньому, було прийнято рішення відкинути цей варіант.

React – найпопулярніша бібліотека JavaScript для розробки користувальницького інтерфейсу (UI). React пропонує відмінний відповідь на вхідні дані користувача, використовуючи новий метод візуалізації вебсайтів.

Компоненти цього інструменту були розроблені Meta (раніше Facebook). Він був запущений як інструмент JavaScript з відкритим вихідним кодом в 2013 році. В даний час React випереджає своїх основних конкурентів, таких як Angular і Vue.js.

React використовується сотнями великих компаній по всьому світу, включаючи Netflix, Airbnb, American Express, Facebook, WhatsApp, eBay і Instagram. Це доказ того, що у інструменту є ряд переваг, з якими важко конкурувати.

Розглянемо переваги React:

- React легко освоїти розробникам з базовими знаннями JavaScript. Це робить його доступним для швидкого старту;
- компоненти можна повторно використовувати в різних частинах застосунку, що спрощує розробку і тестування;
- JSX дозволяє писати HTML-подібний код в JavaScript, що спрощує створення компонентів і робить код зрозумілішим;
- Virtual DOM мінімізує маніпуляції з реальним DOM, що покращує продуктивність застосунку;
- дозволяє використовувати React для створення мобільних застосунків на Android та iOS.

А як щодо недоліків? Звісно, вони також існують:

- React – це бібліотека для побудови інтерфейсів, тому для повноцінної розробки застосунків часто потрібно додатково використовувати інші бібліотеки (React Router, Redux);
- JSX, хоча й зручний, але може бути важким для розуміння новачками;
- великі проєкти потребують знання багатьох додаткових інструментів і бібліотек, що підвищує складність;
- неправильне налаштування компонентів може призвести до зниження продуктивності;
- у великих застосунках управління станом може бути складним без додаткових інструментів (наприклад, Redux).

Хоча React є потужною і популярною бібліотекою для розробки користувацьких інтерфейсів, його обмежений функціонал “з коробки”, складність встановлення та налаштування додаткових модулів, а також потреба в численних ручних налаштуваннях роблять його менш привабливим вибором для мого проєкту.

Angular є єдиним з трьох розглянутих рішень, яке можна по праву назвати фреймворком. Він включає все необхідне для створення високофункціонального вебзастосунку прямо “з коробки”. Angular надає готові модулі для маршрутизації, роботи з сервером, формами, а також забезпечує зручну передачу даних між окремими компонентами. Підтримка Angular здійснюється такими гігантами, як Google і Microsoft, що гарантує стабільність і довготривалу підтримку фреймворку.

Ось деякі з причин, щоб використовувати саме його:

- підтримується великими корпораціями, що гарантує стабільність і регулярні оновлення фреймворку;
- Angular CLI забезпечує легке створення, налаштування і управління проєктами, що значно спрощує процес розробки;
- пропонує чітко визначену структуру проєкту, що сприяє кращій організації коду і спрощує його підтримку;
- використовує TypeScript, що забезпечує додаткову перевірку типів, покращує продуктивність розробників і зменшує кількість помилок;
- інтегрується з RxJS, що дозволяє ефективно управляти асинхронними подіями і потоками даних у зв’язці з HTTP-клієнтом Axios;
- має вбудовану підтримку Dependency Injection, що спрощує управління залежностями і робить код більш модульним і тестованим;
- використовує шаблони на основі HTML, що робить розробку інтерфейсів більш зручною і інтуїтивно зрозумілою;
- забезпечує підтримку Shadow DOM, що покращує ізоляцію стилів і підвищує безпеку застосунків;
- підтримує сучасні вебтехнології, що дозволяє створювати високопродуктивні і що важливо – добре масштабовані вебзастосунки.

Тепер, після того, як розглянуто найбільш популярні рішення, можна порівняти їх за основними критеріями для прийняття остаточного рішення (табл. 2.2).

Таблиця 2.2

Порівняльна характеристика Angular, React та Vue

Критерій	Angular	React	Vue
Архітектура	Повна, компонентно-орієнтована архітектура з вбудованими інструментами для масштабування та модульності	Бібліотека для UI, потребує додаткових бібліотек для повного стеку	Гнучка, мінімалістична, легка для інтеграції, але вимагає додаткових модулів
Масштабованість	Висока масштабованість завдяки строгій структурі та модульності	Масштабування можливе, але залежить від вибору сторонніх бібліотек	Підтримує середні проєкти, складніше масштабувати великі системи
Інструменти для автоматизації	Вбудовані інструменти для автоматизації тестування, інтеграції API та підтримки високого навантаження	Потребує інтеграції сторонніх інструментів для автоматизації та тестування	Мінімальна підтримка вбудованих рішень для автоматизації, потребує додаткових модулів
Підтримка великих проектів	Ідеально підходить для корпоративних і масштабних систем з великим обсягом коду	Підходить для середніх та великих проєктів, але структура менш визначена	Більш підходить для малих і середніх проєктів, не ідеально для великих систем
Навчальна крива	Відносно висока, але виправдана для великих проєктів завдяки повноцінній архітектурі	Середня крива, легший старт для UI-розробників, але складніша інтеграція додаткових рішень	Низька крива навчання, але обмежені можливості для великих систем
Підтримка та спільнота	Підтримується Google, активна спільнота, регулярні оновлення та довгострокова підтримка	Підтримується Meta (Facebook), велика спільнота, активні оновлення	Відкрите джерело з активною, але меншою спільнотою

Продовження Таблиці 2.2

Продуктивність	Оптимізована для масштабних систем, добре працює з великим	Висока продуктивність для UI, але можливі проблеми з	Легка і швидка, але може знизити продуктивність у великих проєктах

	обсягом даних і взаємодій у режимі реального часу	продуктивністю при обробці великих даних	
Безпека	Високий рівень безпеки, включаючи вбудовані засоби захисту, такі як шифрування та захист від XSS-атак	Потребує додаткових бібліотек для реалізації повної безпеки	Потребує додаткових налаштувань для підвищення безпеки

Враховуючи зазначені переваги, Angular виявився найкращим вибором для мого проєкту, оскільки він надає повний набір інструментів та готову структуру проєкту з самого початку, що значно спрощує процес розробки і дозволяє зосередитися на реалізації функціональності без додаткових налаштувань та оптимізацій. А також зважаючи на те, що в мене вже був певний досвід роботи з цім фреймворком, я вирішив зупинитися на цьому рішенні для розробки мого SPA.

Також, не можна не відзначити досить цікавий модуль, що також буде використано при розробці – Angular Material UI. Це дуже зручний інструментарій для створення стильного та функціонального користувацького інтерфейсу в застосунках, розроблених на Angular. Використання Angular Material UI значно полегшує процес розробки інтерфейсу, завдяки готовим компонентам і розширеним можливостям налаштування.

Однією із головних переваг Angular Material UI є його готові компоненти, які можна легко і швидко використовувати для створення різноманітних елементів інтерфейсу. Наприклад, такі компоненти як кнопки, тексти, таблиці, форми, діалогові вікна та багато інших є доступними “з коробки” (рис. 2.4). Це дозволяє розробникам зосередитися на функціональності застосунку, не витрачаючи час на написання коду для базових елементів інтерфейсу.

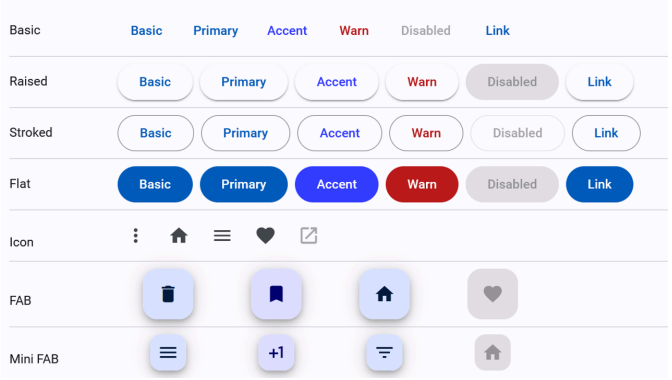

Рисунок 2.4. Приклад кнопок зроблених з допомогою Angular Material

Angular Material UI значно полегшив розробку інтерфейсу. Наприклад, для створення складних форм мною було вирішено використовувати готові компоненти Angular Material, такі як матеріальні тексти, випадаючи списки та перемикачі. Це дозволило швидко розробити інтерактивні та зручні форми без необхідності писати величезну кількість коду вручну. Крім того, завдяки можливостям налаштування стилів, я легко впровадив досить стильній і мінімалістичний дизайн у застосунок, забезпечуючи єдність в зовнішньому вигляді. Загалом, Angular Material надає величезну кількість компонентів (рис. 2.5).

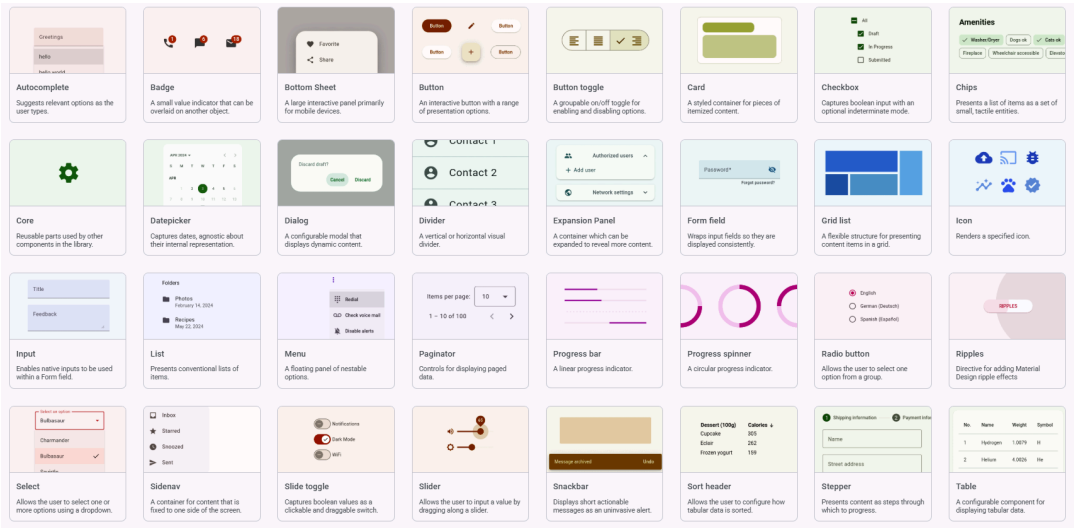

Рисунок 2.5. Деякі компоненти Angular Material

Також, при стилізації сторінок було вирішено не обходити стороною Sass (Syntactically Awesome Style Sheets) – CSS-препроцесор, який розширює

можливості стандартного CSS і допомагає створювати більш гнучкі, організовані й підтримувані стилі. Він дозволяє використовувати змінні, вкладеність, функції, оператори, умовні конструкції та цикли, що значно спрощує розробку великих і складних інтерфейсів. Sass має дві основні форми синтаксису: оригінальний (Sass) із відступами замість дужок і крапок з комою, та SCSS (Sassy CSS), який зберігає знайомий синтаксис звичайного CSS, додаючи при цьому можливості Sass. SCSS став більш популярним через свою зручність і сумісність із наявним CSS-кодом, тому саме його й було використано при оформленні сторінок сайту. Після написання код SCSS компілюється у звичайний CSS, який без проблем можна використовувати у вебзастосунках.

2.2.2 Вибір засобів для реалізації серверної частини

Я обрав платформу C# ASP.NET Core для розробки бекенду свого проєкту з кількох важливих причин.

ASP.NET Core доволі проста у використанні. Вона надає багато готових інструментів, які значно полегшують роботу. Наприклад, є вбудовані шаблони проєктів, що дозволяють швидко почати роботу. Також є зручні механізми для обробки запитів та перевірки даних. Ця технолоігя має багато корисних функцій “з коробки”. Наприклад, вона підтримує автентифікацію та авторизацію користувачів, роботу з сесіями, кешування і навіть роботу з базами даних через Entity Framework. Це дуже зручно, бо можна зосередитися на розробці основних функцій застосунку, не витрачаючи багато часу на налаштування базових речей.

ASP.NET Core працює на Windows, Linux та macOS, що дозволяє розгортати застосунки на різних платформах. А вбудована підтримка DI (Dependency Injection) – спрощує тестування та побудову масштабованої архітектури.

Більш того, ASP.NET Core забезпечує високу продуктивність і масштабованість. Вона добре справляється з великими обсягами даних і високими навантаженнями. Завдяки підтримці асинхронної обробки запитів і інтеграції з хмарними сервісами, можна створювати масштабовані рішення для сучасних вебсистем.

Також, наявні вбудовані механізми безпеки, які захищають від різних видів атак, таких як SQL Injection, XSS і CSRF. Це дуже важливо, адже безпека є одним з ключових аспектів у розробці вебсистем [21].

У якості системи керування базами даних для вебсистеми було обрано PostgreSQL – потужну, стабільну та безкоштовну реляційну СКБД з відкритим вихідним кодом. Її часто використовують як у стартапах, так і у великих корпоративних проєктах завдяки високій надійності, продуктивності та широким можливостям. Вона повністю підтримує стандарт SQL, забезпечуючи при цьому розширені можливості, як-от збережені процедури, тригери, користувацькі типи даних, індекси, реплікацію та повнотекстовий пошук.

Одна з ключових переваг PostgreSQL – це її здатність масштабуватись горизонтально та вертикально, що дозволяє ефективно працювати як з невеликими, так і з надзвичайно великими обсягами даних. Крім того, система забезпечує підтримку транзакцій з гарантіями ACID, що важливо для збереження цілісності та надійності даних, особливо в комерційних або фінансових застосунках. Завдяки активній спільноті розробників PostgreSQL постійно оновлюється та вдосконалюється, що робить її актуальним рішенням на роки вперед [18].

СКБД легко інтегрується з більшістю сучасних мов програмування (зокрема C#, JavaScript, Python, Java), а також з ORM-бібліотеками, як-от Entity Framework або Sequelize. У веброзробці PostgreSQL чудово поєднується з

фреймворками типу ASP.NET Core, Node.js чи Django, забезпечуючи високу швидкість доступу до даних та зручне адміністрування.

PostgreSQL при розробці системи буде використано у поєднанні з технологією Entity Framework Core – ORM (Object-Relational Mapping) від Microsoft, що значно спрощує роботу з базою даних у середовищі .NET.

Entity Framework дозволяє розробникам працювати з базами даних на рівні об’єктів замість написання SQL-запитів вручну. Це особливо зручно під час розробки складних застосунків, оскільки дозволяє будувати бізнес-логіку в термінах звичних класів і об’єктів, а не в термінах таблиць, рядків і стовпців. EF Core автоматично трансформує запити LINQ у відповідні SQL-запити, що значно спрощує підтримку й тестування коду, робить його більш чистим і читабельним (рис. 2.6).

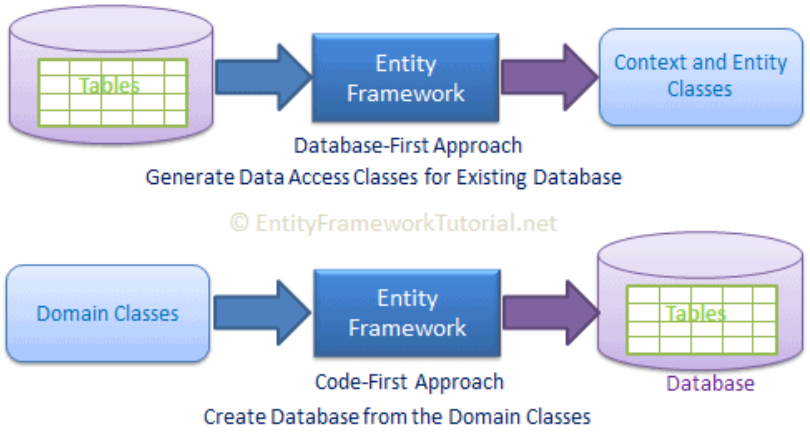

Рисунок 2.6. Принцип роботи Entity Framework Core [22]

Ще однією сильною стороною Entity Framework є підтримка міграцій – механізму, що дозволяє поступово змінювати структуру бази даних у процесі розробки. Це усуває необхідність створювати SQL-скрипти вручну для зміни структури таблиць, адже всі зміни можна описати в C# коді.

Для зручної роботи з даними у форматі JSON буде використано бібліотеку Newtonsoft.Json. Однією з її основних функцій є серіалізація та десеріалізація об’єктів.

Серіалізація – це процес перетворення об’єкта (наприклад, класу в C# з полями, властивостями та значеннями) у формат, зручний для зберігання або передачі, наприклад, у текстовий формат JSON (рис. 2.7).

Рисунок 2.7. Приклад серіалізації у Newtonsoft.Json [23]

Десеріалізація, відповідно, – це зворотній процес. Вона дозволяє взяти JSON-рядок і перетворити його назад у об’єкт C#. Тобто, з тексту знову створюється об’єкт, який можна використовувати в програмному коді, викликати його методи, змінювати значення атрибутів тощо (рис. 2.8).

Рисунок 2.8. Приклад десеріалізації у Newtonsoft.Json [23]

У контексті веброзробки ці процеси є надзвичайно важливими. Серіалізація використовується для відправки даних із сервера на клієнт у форматі JSON, а десеріалізація – щоб обробити отримані ззовні (наприклад, з форми, API або іншого сервера) дані і представити їх як об’єкти, з якими можна працювати у коді. Newtonsoft.Json дозволяє робити це швидко, зручно і гнучко – з підтримкою форматування, атрибутів, обробки вкладених структур, і навіть правил перетворення, що можна налаштовувати.

Також, у проєкті буде використано Swagger – інструмент для документування та тестування API. Він автоматично генерує зручний вебінтерфейс, у якому можна переглядати всі доступні запити, їхні параметри, типи повертаємих даних та можливі відповіді. Це значно спрощує розробку, тестування та інтеграцію сторонніми клієнтами, оскільки вся необхідна інформація про API доступна в одному місці [24]. Swagger також дозволяє виконувати запити прямо з браузера, що особливо зручно на етапах налагодження. У рамках проєкту його використання допоможе пришвидшити взаємодію між фронтендом і бекендом, забезпечивши зрозумілу та актуальну документацію. Приклад вікна запиту в сервісі Swagger подано на рис. 2.9.

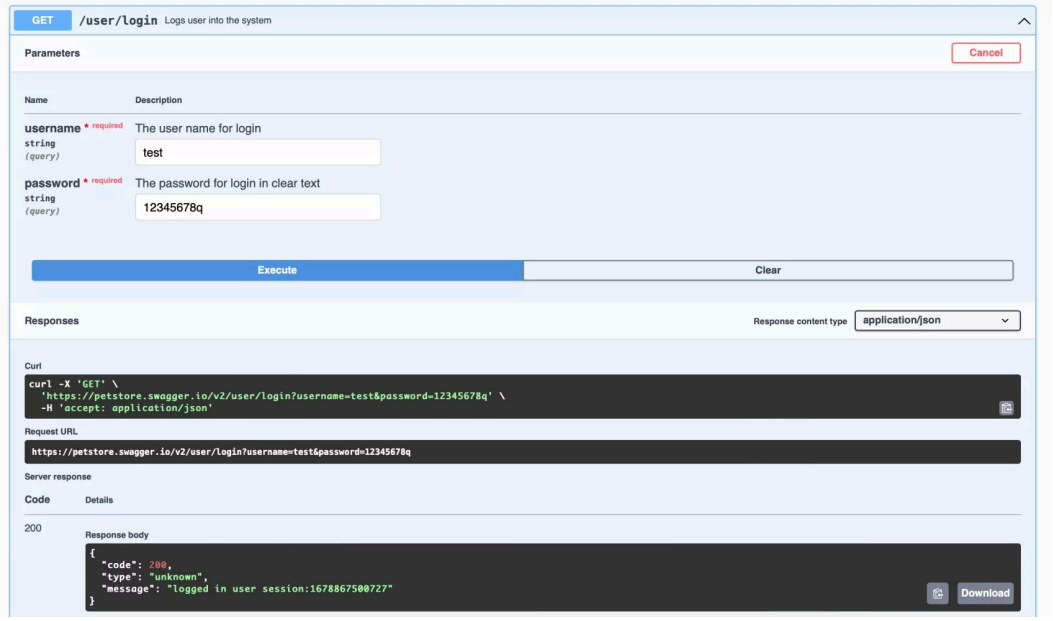

Рисунок 2.9. Приклад GET-запиту у Swagger [24]

Загалом, принцип роботи бекенду вебсистеми, враховуючи обраний технологіний стек, матиме приблизно таку структуру, як продемонстровано на схемі (рис. 2.10).

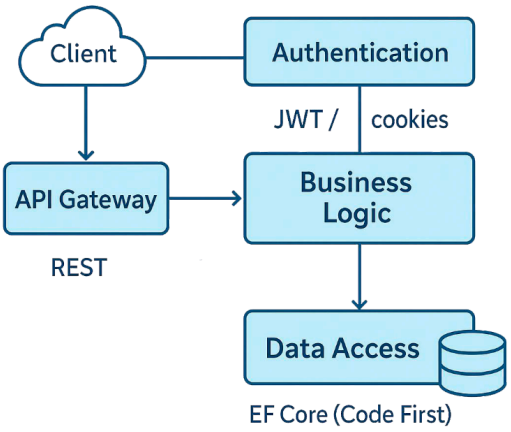

Рисунок 2.10. Обраний підхід до розробки бекенду

2.2.3 Вибір засобів для написання коду

Одним з найвідоміших інструментів, що використовується для написання коду при розробці вебзастосунків є Visual Studio Code – надійний та гнучкий редактор коду. Однією з головних переваг VS Code є його легкість і швидкодія, що дозволяє працювати навіть на менш потужних комп'ютерах. Інтерфейс VS Code інтуїтивно зрозумілий, підтримує розділення вікон, вкладки, пошук по проєкту та інтегрований термінал. Але найважливішою перевагою звісно є величезна кількість доступних розширень – сотні плагінів з найрізноманітніших сфер: від підтримки мов програмування, інтеграції з системами контролю версій, до інструментів для тестування, налагодження, аналізу коду та автодоповнення.

Visual Studio Code підтримує GitHub Copilot – інтелектуального асистента для розробників, створеного компаніями GitHub і OpenAI. Copilot інтегрується прямо в редактор коду і надає пропозиції в реальному часі на основі введеного

контексту. Він аналізує вже написаний код, коментарі та назви змінних, щоб передбачити, що саме ви хочете написати далі, і пропонує відповідні фрагменти коду або навіть цілі функції (рис. 2.11).

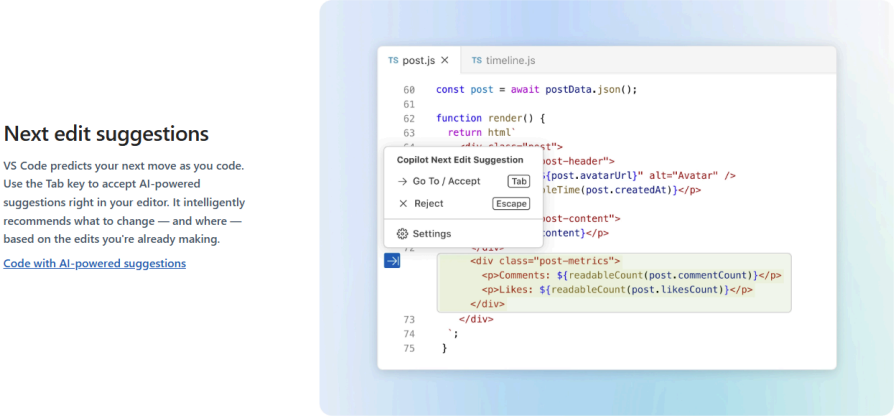

Рисунок 2.11. VS Code передбачає наступний крок написання коду [25]

Підтримка Copilot у VS Code значно підвищує продуктивність, особливо під час написання шаблонного або повторюваного коду, а також дає змогу швидше засвоювати нові технології, підказуючи синтаксис і правильні підходи. Крім того, Copilot “навчається” у процесі, підлаштовуючись під стиль і звички конкретного розробника.

Також, було прийнято рішення також використовувати й інтегроване середовище розробки (IDE) Visual Studio – переважно для розробки серверної частини. Це пов’язано із тим, що в якості основного фреймворку для створення бекенду було обрано ASP.NET Core, а Visual Studio максимально адаптоване для роботи з платформою .NET.

Особливість Visual Studio полягає в тому, що це офіційне середовище, розроблене тією ж компанією, що створила .NET, тому воно завжди підтримує найновіші можливості платформи раніше за інші IDE. Наприклад, нові шаблони проєктів, підтримка останніх версій SDK, інтеграція з Azure, інструменти для тестування та профілювання – усе це працює “з коробки”, без додаткових

налаштувань. Саме тому для розробки на .NET Visual Studio є найзручнішим і найефективнішим вибором, особливо для великих або корпоративних проєктів.

Visual Studio надає розширену підтримку для C#, автоматичне доповнення коду (IntelliSense), зручний відлагоджувач, інтегроване керування пакетами NuGet і візуальні редактори інтерфейсів. Функція IntelliSense забезпечує інтелектуальне завершення на основі типів змінних, визначень функцій та імпортованих модулів, підвищуючи ефективність і точність кодування. Інтерфейс IDE, що налаштовується, дозволяє пристосовувати середовище розробки до різних потреб, підвищуючи продуктивність програміста [26]. Інтерфейс пропонує кілька основних вікон для роботи (рис. 2.12).

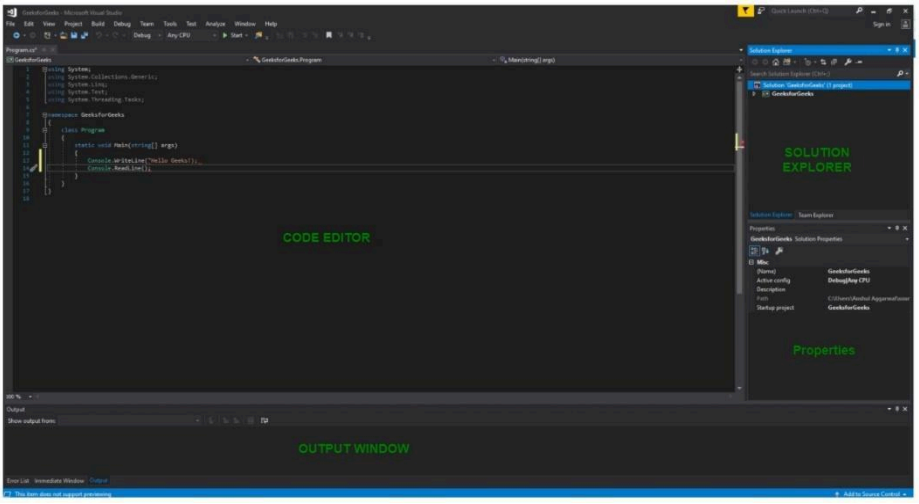

Рисунок 2.12. Вигляд робочого середовища Visual Studio [26]

Підсумовуючи, даний стек відмінно підходить для створення вебсистеми автоматизації бізнес-процесів реалізації нерухомого майна, забезпечуючи збалансоване поєднання практичності, продуктивності та масштабованості. За допомогою схеми архітектури застосунку можна продемонструвати обрані технології та принцип взаємодії основних складових системи (рис. 2.13).

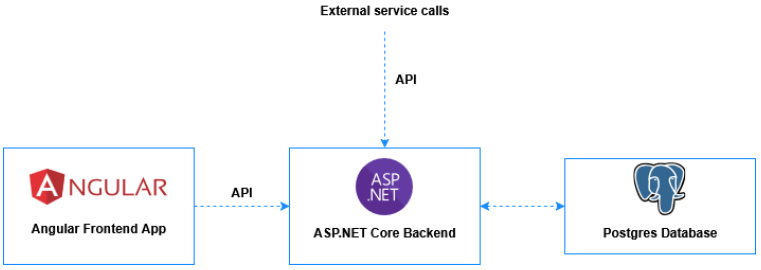

Рисунок 2.13. Схема архітектури застосунку

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБСИСТЕМИ ДЛЯ ПРОДАЖУ НЕРУХОМОСТІ

3.1 Визначення вимог до розробки системи

Зважаючи на функціонал наявних вебзастосунків з продажі об’єктів нерухомості та їх аналіз, проведений у розділі 1.5, визначимо функціональні вимоги до вебсистеми, що розробляється (табл. 3.1).

Таблиця 3.1

Функціональні вимоги до розробки системи

Вимога	Складові вимоги
Зберігання нерухомостей	Забезпечення можливості додавання нової нерухомості з детальною інформацією (назва, адреса, ціна, поверх, площа тощо), перегляду доступних нерухомостей та їх деталей, редагування інформації про нерухомість, видалення нерухомості з бази даних.
Робота із замовленнями	Створення замовлень, зміна статусу замовлень.
Оновлення нерухомостей	Автоматичне оновлення доступної нерухомості після кожного продажу, ручне оновлення інформації про товари.
Фільтрація нерухомостей	Реалізація фільтрації нерухомості за декількома параметрами одночасно.
Безпека доступу	Управління ролями та правами доступу користувачів до різних функцій системи, автентифікація та авторизація користувачів.

3.2 Проєктування моделі вебсистеми засобами CASE-технологій

Проблеми, пов’язані з автоматизацією проєктування інформаційних систем, призвели до появи спеціалізованих програмно-технологічних інструментів – так званих CASE-засобів (Computer Aided Software Engineering). CASE-технологія охоплює методи розробки програмного забезпечення та забезпечує набір інструментів, які дозволяють зручно моделювати предметну область, проводити аналіз цієї моделі на всіх етапах створення та супроводу програм, а також розробляти рішення, що відповідають інформаційним потребам користувачів.

До основних типів CASE-засобів належать: програми для керування конфігурацією, засоби моделювання даних, інструменти для аналізу та проєктування систем, засоби трансформації моделей, редактори програмного коду, інструменти для рефакторингу, генератори коду, а також засоби для створення UML-діаграм [27].

UML (Unified Modeling Language) – це уніфікована мова моделювання, яка широко застосовується в об’єктно-орієнтованому програмуванні та є важливою складовою уніфікованого процесу створення програмного забезпечення. Вона базується на використанні діаграм, які дозволяють наочно зобразити структуру та поведінку системи, що значно спрощує подальшу реалізацію у вигляді програмного коду.

На основі функціональних вимог до застосунку представимо діаграму варіантів використання, що демонструє набір варіантів використання і діючих осіб-акторів, а також їх зв’язки (рис 3.1).

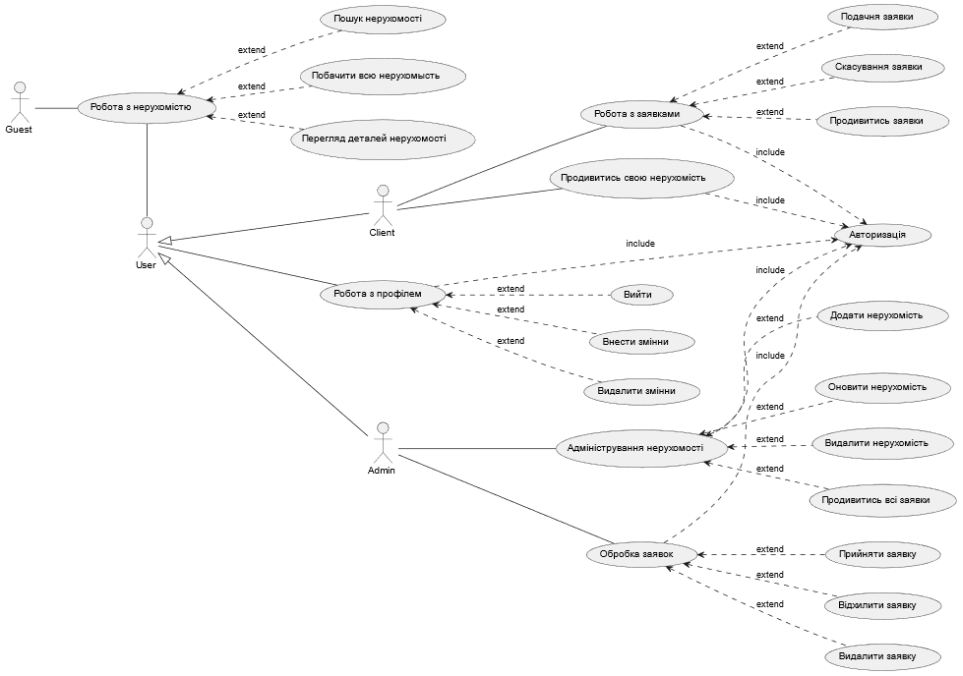

Рисунок 3.1. Діаграма варіантів використання

Основними учасниками системи виступають три типи користувачів: гість, зареєстрований користувач та адміністратор. Гість має доступ до базової взаємодії з нерухомістю, тобто може здійснювати пошук об’єктів, переглядати наявну нерухомість та вивчати її деталі.

Після авторизації користувач отримує розширені можливості, зокрема може переглядати та керувати власними об’єктами нерухомості, подавати заявки й скасовувати їх. Окрім цього, зареєстрований користувач має змогу працювати з профілем: вносити зміни, виходити з акаунту, видаляти свої дані.

У свою чергу, адміністратор отримує повний контроль над усіма об’єктами та заявками в системі. Йому доступна адміністрація нерухомості, включаючи додавання, оновлення, видалення об’єктів, а також перегляд усіх заявок. У межах обробки заявок адміністратор може приймати, відхиляти або повністю видаляти їх.

Наступним кроком буде моделювання логіки виконання процесів у вигляді послідовностей дій, умов і розгалужень (тобто, опис алгоритму роботи

застосунку). Отже, спочатку користувач заходить на сайт і починає шукати нерухомість. Якщо він ще не має облікового запису, система дозволяє зареєструватися. У випадку, якщо обліковий запис вже є, користувач виконує вхід до системи. Після цього він продовжує перегляд і, за потреби, вирішує подати заявку на певний об’єкт. Користувач обирає об’єкт нерухомості, заповнює відповідну заявку та надсилає її на розгляд. У цей момент об’єкт отримує статус “на розгляд”, і дія переходить до адміністратора. Адміністратор перевіряє заявку, після чого ухвалює рішення: або приймає, або відхиляє її. Далі адміністратор оновлює статус нерухомості – або до “продано”, якщо угода завершена. Якщо ж заявку було відхилено, користувача інформують про це, і процес завершується.

Описану вище послідовність дій представлено за допомогою діаграми діяльності (рис. 3.2).

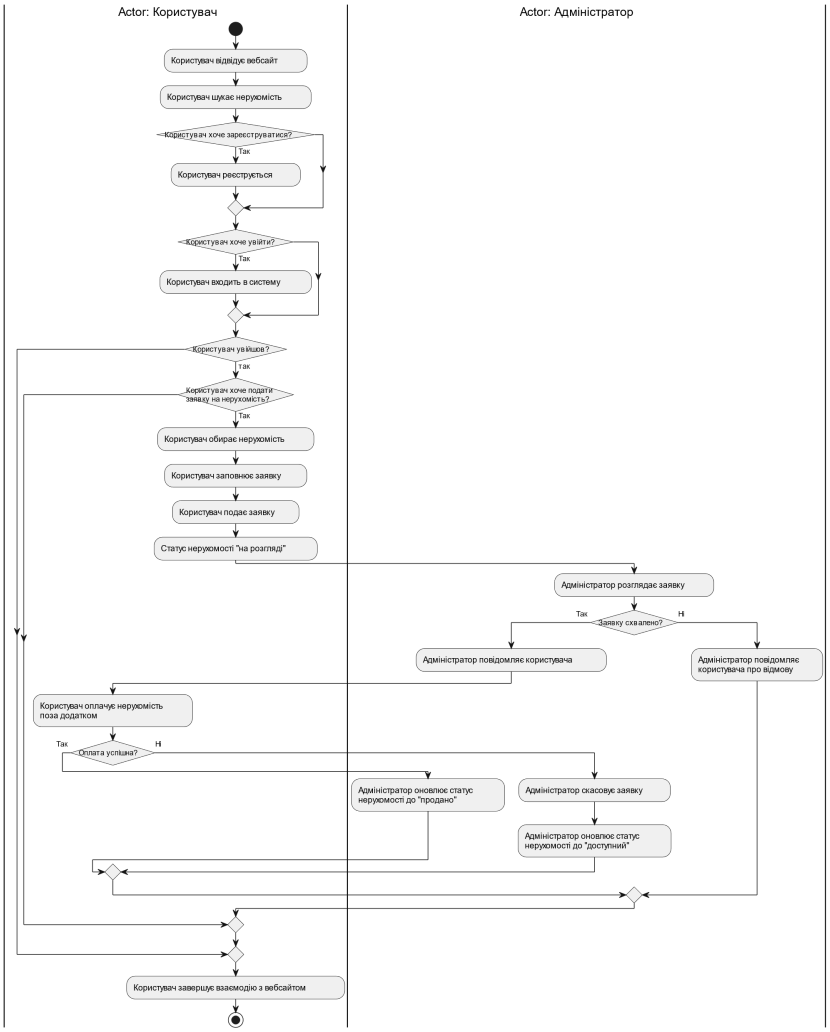

Рисунок 3.2. Діаграма діяльності

Тепер, проаналізуємо поведінку замовлень в залежності від їхнього стану. Для цього побудуємо діаграму станів (рис. 3.3).

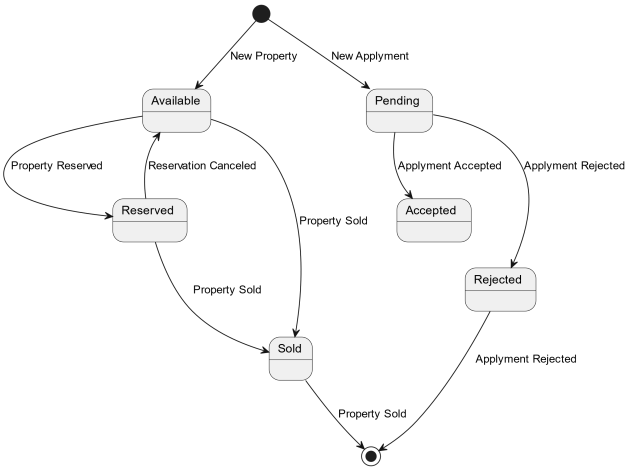

Рисунок 3.3. Діаграма станів замовлень

3.3 Розробка серверної частини

На основі вже наявних знань про сценарії використання та поведінку системи було розроблено серверну частину вебсистеми, структуру якої подано у вигляді діаграми класів (рис. 3.4).

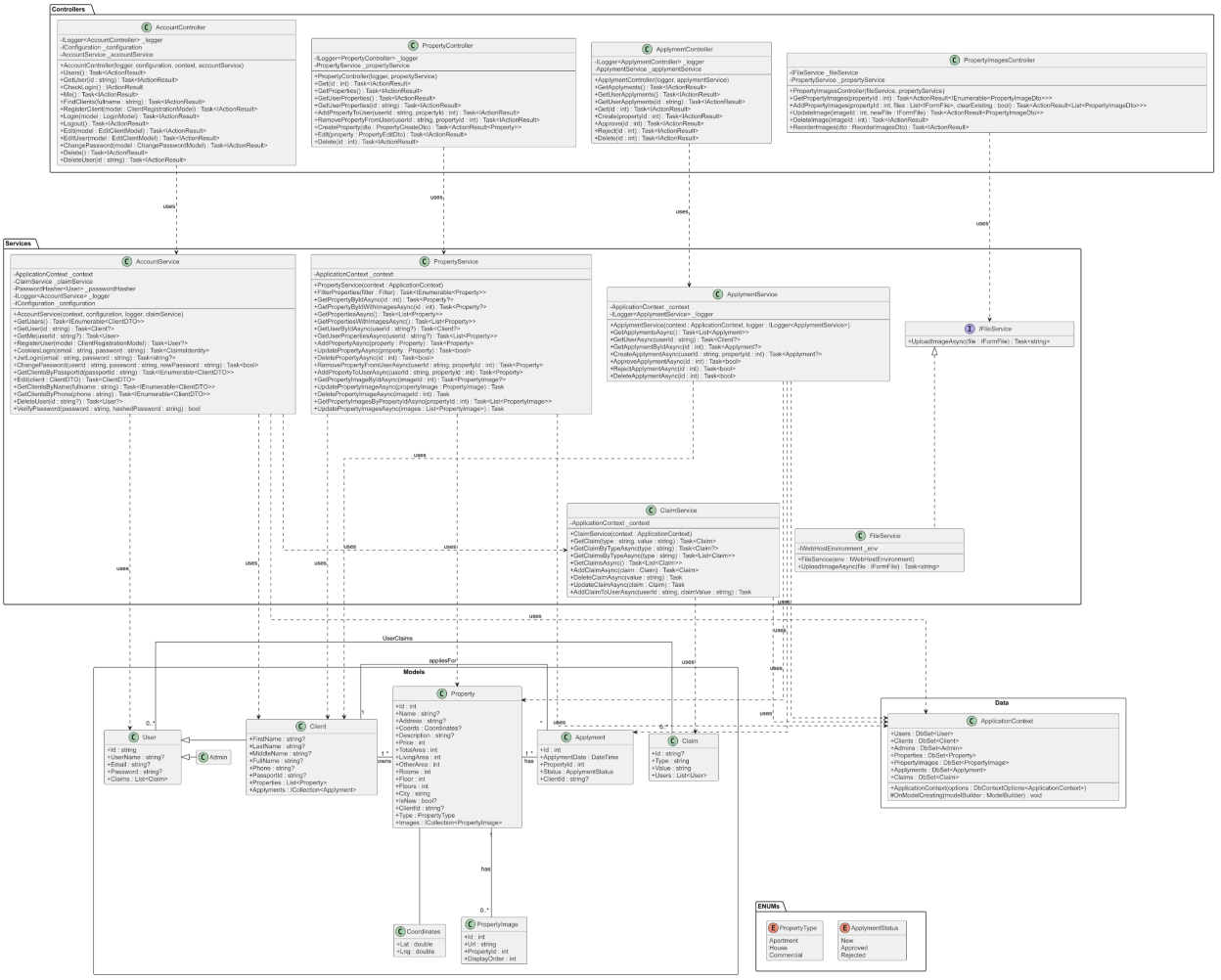

Рисунок 3.4. Діаграма класів

3.3.1 Проєктування бази даних системи

Необхідною умовою для досягнення ефективного створення системи є опис структури бази даних (БД) та усіх зв’язків між її компонентами. Спочатку

проєктується логічна схема бази даних, для більш загального розуміння її структури (рис 3.5).

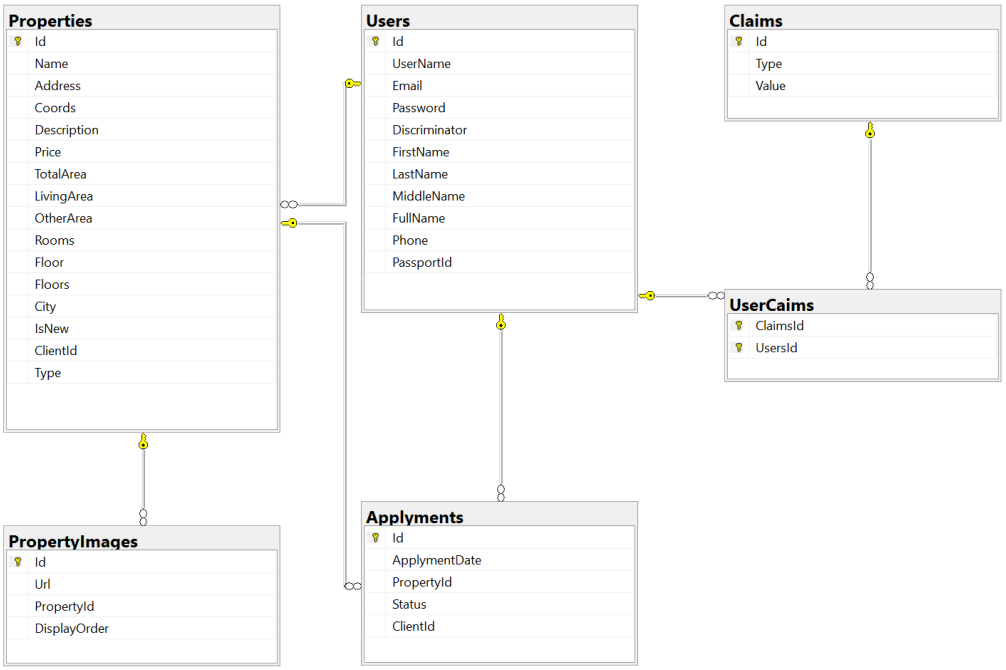

Рисунок 3.5. Логічна схема бази даних

Структуру БД визначено у класі ApplicationContext, що використовує підхід Entity Framework Core Code-First, де класи C# представляють таблиці бази даних, а поля (властивості) відповідних класів – стовпці цих таблиць. Реалізовано основні таблиці: Users (Користувачі), Properties (Нерухомості), PropertyImages (Зображення нерухомості), Applyments (Замовлення), Claims (Ролі).

Між таблицями Users і Claims реалізовано зв’язок “багато-до-багатьох” (користувач може мати декілька ролей, і одну й ту саму роль може бути призначено багатьом користувачам), а для цього було створено додатково таблицю-зв’язку UserCaims. Також, у користувача може бути багато замовлень, однак одне замовлення пов’язане лише з одним користувачем та одним об’єктом нерухомості.

Entity Framework може зіставляти ієрархію типів .NET з базою даних. Це дозволяє створювати .NET-сутності в коді, використовуючи при цьому базові та похідні типи, а також EF безперешкодно може створювати відповідну схему БД.

EF Core підтримує кілька стратегій спадкування, серед яких найпоширенішою є TPH (Table-per-Hierarchy). У цьому випадку вся ієрархія класів зберігається в одній таблиці, а для розрізнення типів використовується спеціальне дискримінаторне поле. Насамперед, ця стратегія забезпечує високу продуктивність, оскільки всі об’єкти ієрархії зберігаються в одній таблиці, що дозволяє уникнути складних об’єднань (JOIN) між кількома таблицями під час запитів. Це значно зменшує навантаження на базу даних і покращує швидкодію, особливо при читанні великих обсягів даних [28]. У проєкті використовується такий тип спадкування для таблиці Users. Тобто, і гості, і клієнти, і адміністратори будуть зберігатися в одній таблиці, а відповідна роль буде зазначена в полі Discriminator.

У результаті було спроєктовано базу даних, що відповідає наступній фізичній схемі (рис. 3.6).

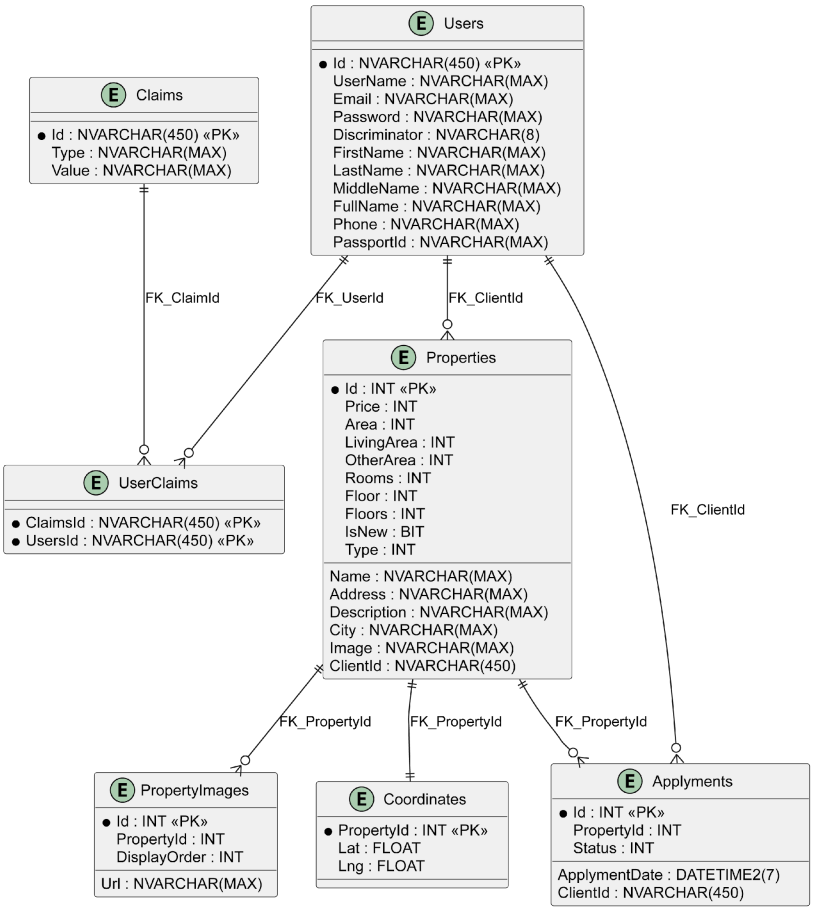

Рисунок 3.6. Фізична схема бази даних

3.3.2. Реалізація моделей даних

Моделі даних є основою для роботи з інформацією у застосунку. Вони визначають структуру даних, що будуть зберігатися та використовуватися в системі. У рамках проєкту було розроблено такі моделі:

- User – базова модель, яка містить загальні властивості користувача, такі як ідентифікатор, username, електронна пошта та пароль. Також тут є колекція Claims, що використовується для зберігання додаткової інформації про користувача;
- Client – спеціалізована модель, яка успадковує властивості моделі User і додає додаткові поля, такі як ім’я, прізвище, по батькові, телефон та

паспортні дані. Клієнт також може мати кілька об’єктів нерухомості, якими він володіє, та заявки на нерухомість;

- Admin – також успадковується від User і представляє адміністратора системи. Вона може не мати додаткових властивостей, але її наявність дозволяє розмежувати права доступу між різними типами користувачів;
- Property – представляє об’єкт нерухомості з такими властивостями, як назва, адреса, опис, ціна, площа, кількість кімнат, поверх та тип нерухомості (квартира, будинок, комерційне приміщення). , а ще – зберігається ідентифікатор клієнта, який є власником цієї нерухомості. Також, модель містить колекцію зображень нерухомості;
- Applyment – замовлення, яке клієнт може подати для придбання нерухомості, що містить ідентифікатор, дату подання, статус, ідентифікатори клієнта та нерухомості;
- PropertyImage – модель для фотографії об’єкту нерухомості, що має ідентифікатор, посилання на саме зображення, ідентифікатор нерухомості та порядковий номер у каруселі зображень при відображенні на сторінці перегляду детальної інформації про товар;
- Coordinates – модель, що містить координати розташування об’єкту нерухомості для його подальшого відображення на сторінці перегляду детальної інформації про товар засобами Google Maps.

3.3.3 Реалізація сервісів для взаємодії з базою даних

Реалізація сервісів дозволяє абстрагувати бізнес-логіку від контролерів і забезпечує зручний доступ до даних через централізовані компоненти. Мною було створено такі сервіси:

- AccountService – відповідає за управління обліковими записами користувачів, забезпечуючи функції для автентифікації, реєстрації та керування користувацькими даними;
- PropertyService – відповідає за керування об’єктами нерухомості. Надає широкий спектр функцій для роботи з товарами, включаючи фільтрацію, додавання, оновлення та видалення;
- ApplymentService – відповідає за керування замовленнями. Сервіс взаємодіє з контекстом бази даних (клас ApplicationContext) для виконання CRUD операцій із замовленнями;
- ClaimService – призначений для роботи з ролями;
- IFileService – інтерфейс (визначає “контракт”, що повинно бути реалізовано). Будь-який клас, який реалізує IFileService, повинен надати реалізацію методу UploadImageAsync, що асинхронно завантажує файл і повертає URL у вигляді текстового рядка;
- FileService – конкретна реалізація інтерфейсу IFileService, що надає логіку для завантаження файлу на сервер.

3.3.4 Розробка API для взаємодії з сервісами

Було розроблено API для взаємодії з різними сервісами, що дозволяє керувати даними користувачів, їхніми замовленнями та нерухомістю. Основні контролери, що були реалізовані, включають:

- AccountController – забезпечує керування обліковими записами;

- PropertyController – керує операціями з нерухомістю;

- ApplymentController – відповідає за керування замовленнями користувачів;

- PropertyImagesController – призначений для керування зображеннями нерухомостей.

Кожен з цих контролерів реалізує CRUD-операції (створення, читання, оновлення й видалення), що дозволяє виконувати всі необхідні операції для взаємодії з відповідними даними. Операції реалізовані таким чином, що відповіді на запити формуються у форматі JSON, що спрощує інтеграцію з клієнтською частиною та забезпечує гнучкість у розробці фронтенду. Вся логіка контролерів заснована на принципах асинхронного програмування, що підвищує продуктивність та масштабованість застосунку.

У проєкті використовується принцип Dependency Injection (DI) для управління залежностями між компонентами. Такі сервіси, як AccountService, PropertyService і ApplymentService, впроваджуються в контролери, що робить код більш модульним і тестованим.

Система використовує автентифікацію на основі файлів cookie для входу користувача та JWT (JSON Web Tokens) для подальшої авторизації. AccountController обробляє реєстрацію користувачів, вхід до системи та вихід із системи. AccountService забезпечує базову логіку для цих операцій, включаючи хешування та перевірку пароля.

3.4 Розробка клієнтської частини

3.4.1. Структура клієнтської частини

Розроблена вебсистема складається з файлів, що розташовані в певній ієрархії та працюють між собою в користувальницькому режимі. На схемі

представлено загальну файлову структуру програми на етапі розробки, коли проєкт ще не було «збудовано» у повноцінний, готовий до деплою вебзастосунок (рис. 3.7).

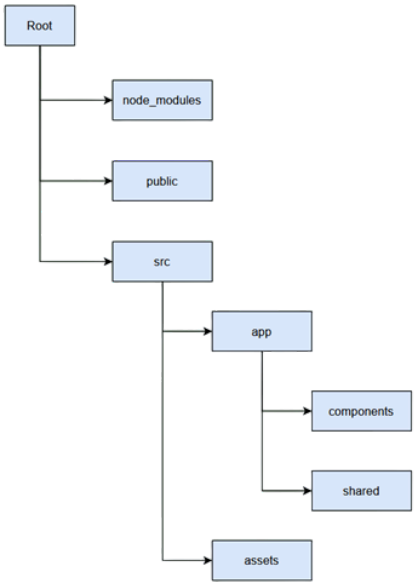

Рисунок 3.7. Файлова структура вебзастосунку

Розглянемо детальніше призначення основних каталогів (табл. 3.2).

Таблиця 3.2

Каталоги файлової структури вебзастосунку

Назва	**Опис**
Root	Папка, в якій знаходяться всі файли вебзастосунку
node_modules	Папка, що містить всі необхідні для роботи вебзастосунку модулі. Це включаю в себе як і чисту бібліотеку Angular, так і будь-які сторонні модулі, встановлені розробником
public	Папка, що містить кінцеву сторінку з розміткою HTML, іконки, які будуть використовуватися на вкладці сайту, логотип, а також певні конфігурації файлу

Продовження таблиці 3.2

src	Папка, в якій знаходиться весь основний код вебзастосунку, а також будь-які дані, необхідні для розробки

components	Папка, в якій зберігаються код всіх компонентів вебзастосунку, включаючи файли стилів
shared	Папка, в якій зберігаються компоненти та інший код, який можна перевикористовувати в різних частинах застосунку
assets	Папка, в якій зберігаються статичні файли наприклад картини та логотип

На етапі, коли застосунок вже був повністю готовий до розміщення в мережі, структура складових клієнтської частини відповідала діаграмі компонентів (рис. 3.8).

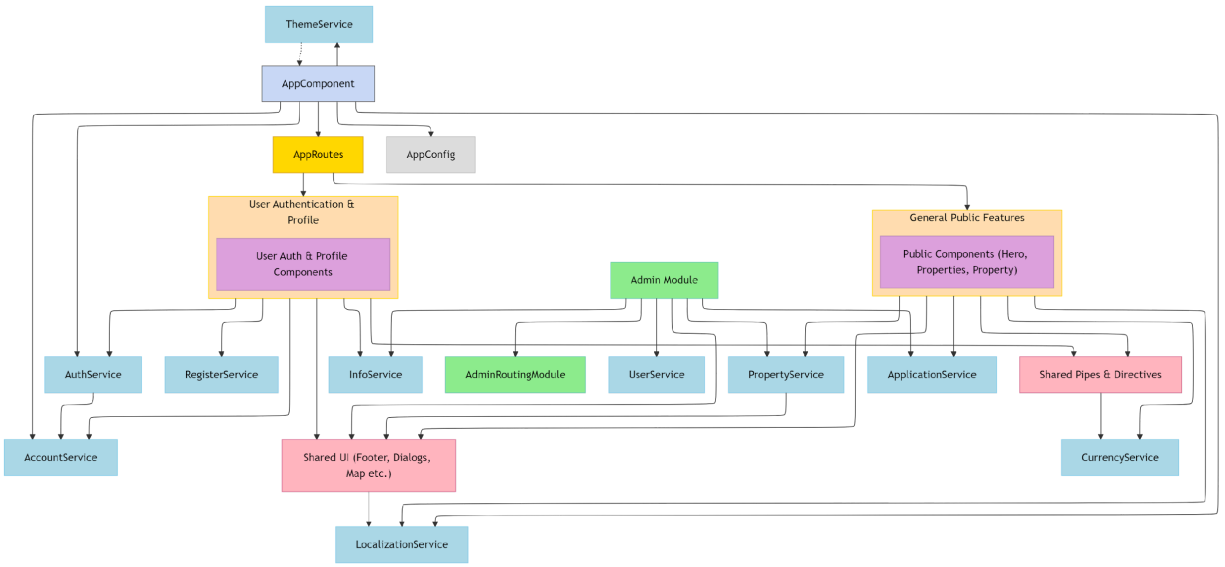

Рисунок 3.8. Діаграма компонентів клієнтської частини

3.4.2 Реалізація основних компонентів і сервісів для взаємодії з API

У ході розробки було створено ряд компонентів, кожен з яких відповідає за окрему частину функціоналу. Головний HTML файл – Index.html, який містить тег app-root, у який вставляється HTML файл app.component.html. Точкою входу для Angular застосунку слугує файл main.ts. Він завантажує головний модуль застосунку та ініціалізує платформу, необхідну для запуску застосунку в браузері. А головний модуль app.module.ts імпортує та оголошує необхідні модулі й компоненти, визначаючи основні налаштування застосунку

через декоратор @NgModule (declarations, imports, providers, bootstrap). Налаштування маршрутизації подано в файлі app-routing.module.ts. Логіка головного компоненту прописана в файлі app.component.ts. Він визначає клас AppComponent з основними властивостями та методами, а також використовує декоратор @Component для визначення метаданих компоненту. А безпосередньо за візуальне представлення інтерфейсу користувача відповідає app.component.html.

Також у застосунку є кілька компонентів для взаємодії з користувачем. Наприклад, компонент Home відповідає за відображення домашньої сторінки. Компонент Application керує створенням та переглядом замовлень, дозволяючи користувачам подавати їх. Компонент Properties відображає список всіх доступних об’єктів нерухомості, дозволяючи переглядати деталі та виконувати дії з об’єктами. Login та Register – відповідають за автентифікацію, надаючи інтерфейс для входу та реєстрації нових користувачів. PropertyDetails забезпечує відображення детальної інформації про конкретний об’єкт нерухомості, включаючи його характеристики та стан, дозволяючи користувачам детальніше ознайомитись з об’єктом перед подачею заявки на купівлю.

Для взаємодії з API в Angular можна використовувати різні сервіси, які забезпечують різні функціональні можливості для роботи з даними. Розглянемо деякі важливі сервіси, що були створені. PropertyService надає функціональність для роботи з об’єктами нерухомості: отримання списку всіх об’єктів, об’єктів конкретного користувача, створення нових об’єктів, оновлення та видалення існуючих, а також додавання та видалення об’єктів у користувача. AccountService відповідає за управління інформацією про користувача, зокрема за автентифікацію, оновлення профілю та управління сесіями. Сервіс також реалізує автоматичне оновлення профілю кожні 5 хвилин для забезпечення актуальності даних. AuthService використовується для реєстрації, входу та виходу користувачів. Він взаємодіє з AccountService для управління сесіями та перевірки адміністративних прав користувача. При вході користувача, сервіс

запускає фонове оновлення профілю через AccountService. UserService забезпечує можливості для управління користувачами. Сервіс включає функції для отримання всіх користувачів, отримання інформації про конкретного користувача за його ID, оновлення даних користувача та видалення користувача. А за роботу із замовленнями відповідає ApplicationService.

3.5 Огляд функціоналу розробленої вебсистеми

Розроблена програмна система має досить інтуїтивно-зрозумілий інтерфейс. Головна сторінка відображена на рис. 3.9.

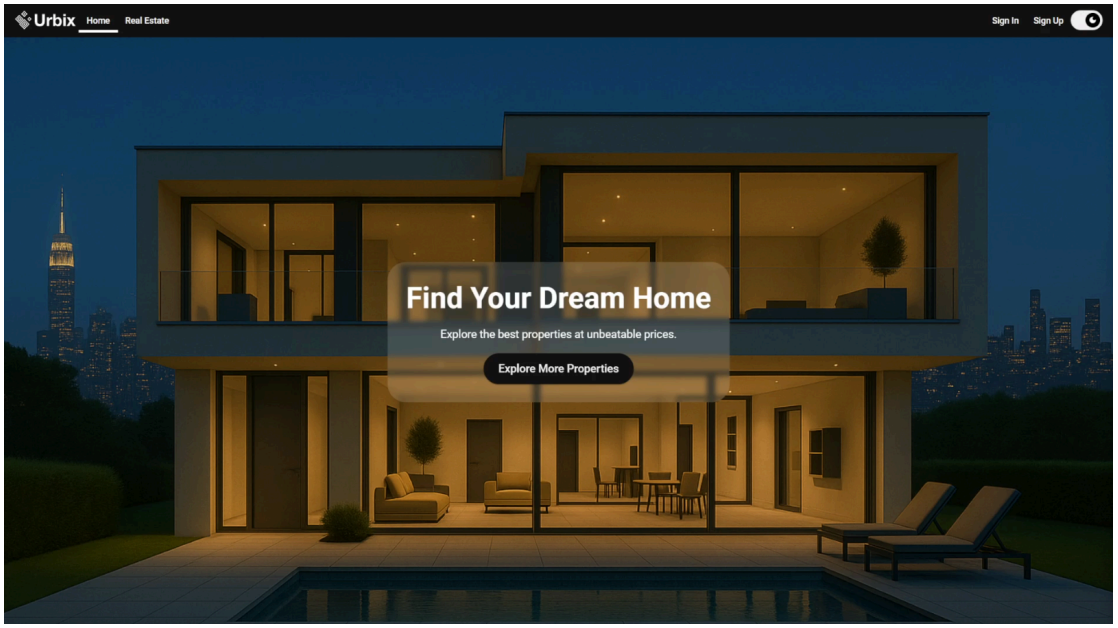

Рисунок 3.9. Головна сторінка застосунку

Меню навігації (зверху) містить список посилань на окремі сторінки (домашню та сторінку з доступними для продажу об’єктами нерухомості), а також посилання переходу на сторінки реєстрації та авторизації. По центру міститься кнопка для переходу до сторінки пошуку об’єктів нерухомості. Внизу головної сторінки міститься footer, де міститься контактна інформація, відповіді на популярні питання, посилання на сторінки в соціальних мережах для комунікації тощо. Також, можна змінювати мову інтерфейсу (обирати українську або англійську) (рис.3.10).

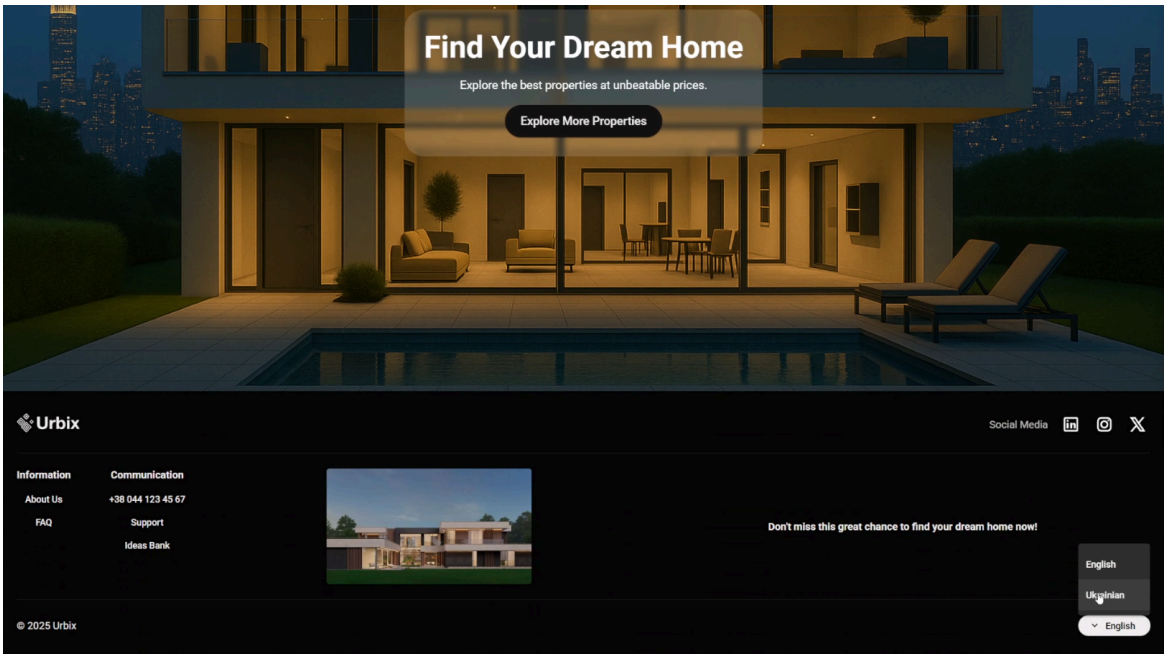

Рисунок 3.10. Footer застосунку

Тему інтерфейсу також можна змінювати за бажанням (рис. 3.11).

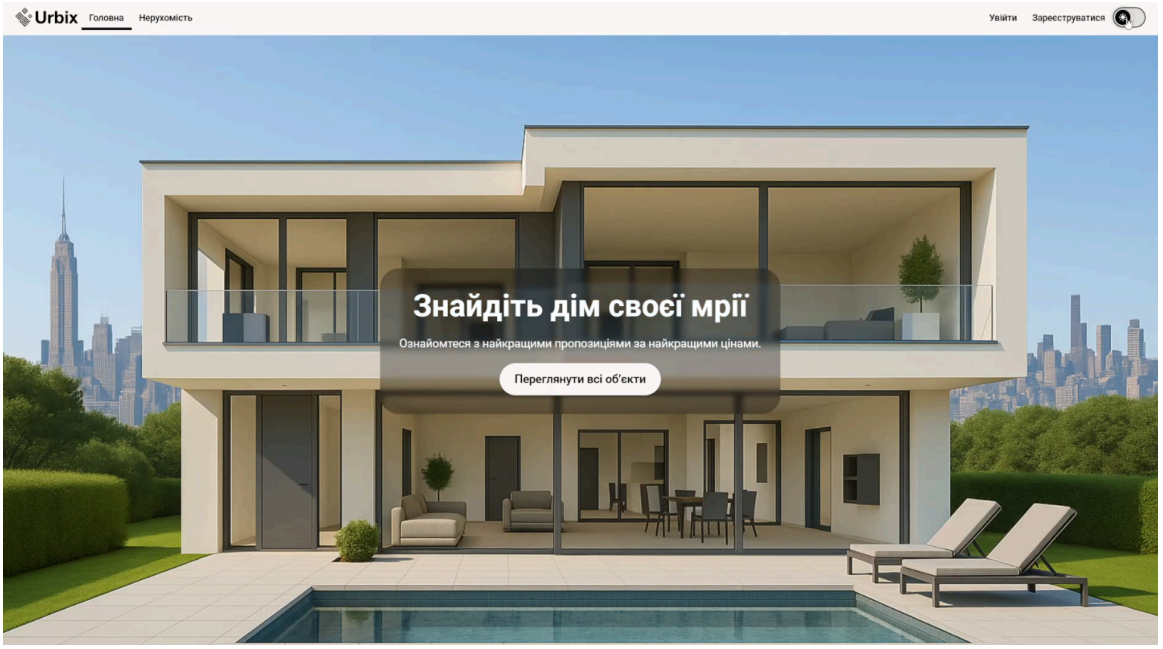

Рисунок 3.11. Вигляд інтерфейсу при виборі світлої теми

Щоб потрапити на головну сторінку чи переглядати доступні нерухомості для продажу, реєстрація чи авторизація не є обов’язковими. Сторінка реєстрації

користувача містить 8 полів для заповнення даних користувача, кнопку підтвердження реєстрації та посилання на сторінку авторизації (рис. 3.12).

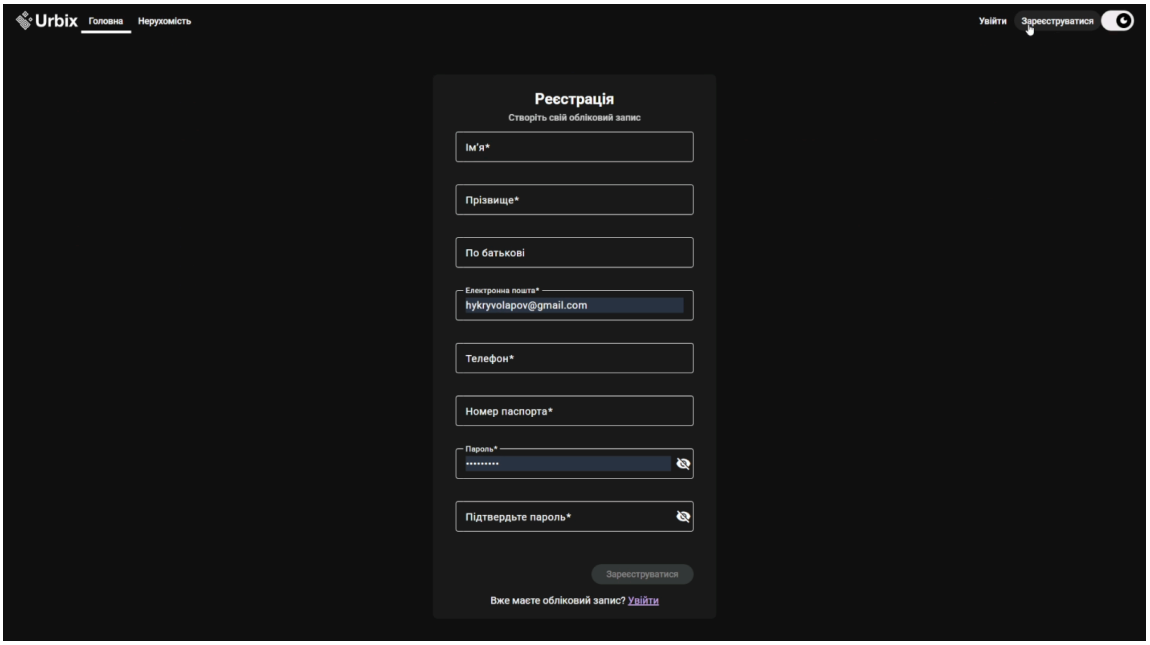

Рисунок 3.12. Сторінка реєстрації

У разі успішної валідації користувач додається до відповідної таблиці бази даних. Усі поля окрім «По батькові» обов’язкові для заповнення. Введена інформація валідується при втраті елементом фокусу, і якщо дані будь-якого поля не відповідають критеріям, інформація не зберігається до бази даних, а біля помилкової інформації з’явиться повідомлення про помилку та підказка. У випадку, коли обов’язкове поле не є заповненим, кнопка підтвердження не буде активною.

Сторінка авторизації має аналогічний інтерфейс. Вона містить 2 поля для введення даних: електронну адресу та пароль. Також під полем введення паролю є чекбокс «Запам’ятати мене», при виборі якого при наступній реєстрації вказані дані будуть автоматично заповнені. Після натискання кнопки підтвердження відбувається перевірка введеної інформації, і якщо даних, введених користувачем, немає в базі даних, виводиться повідомлення про проблему, а поля оновлюються для наступної спроби. В іншому випадку,

відображається головна сторінка інтерфейсу користувача, а вікно авторизації закривається.

Сторінка із доступними об’єктами нерухомості відображена на рис. 3.13.

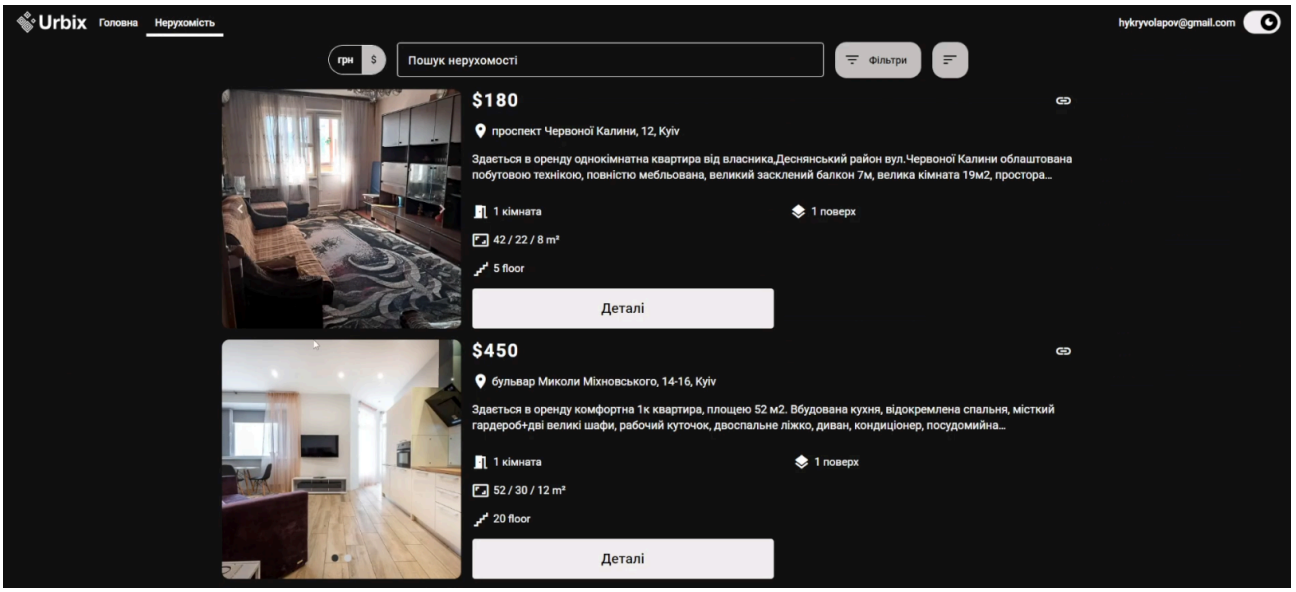

Рисунок 3.13. Список доступних для купівлі об’єктів нерухомості

Можна вказувати, у якій валюті відображати ціни: українська гривня чи американський долар (рис. 3.14).

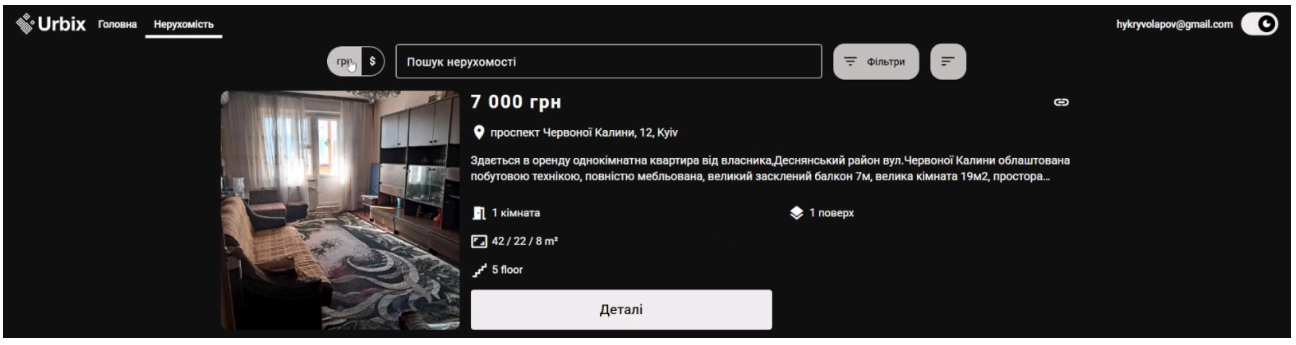

Рисунок 3.14. Вибір валюти «українська гривня»

Також для більш ефективного пошуку об’єкту нерухомості можна застосовувати фільтри, вказуючи діапазон ціни, кількість кімнат, площу тощо. За потреби усі фільтри можна скинути натисканням лише однієї кнопки (рис. 3.15).

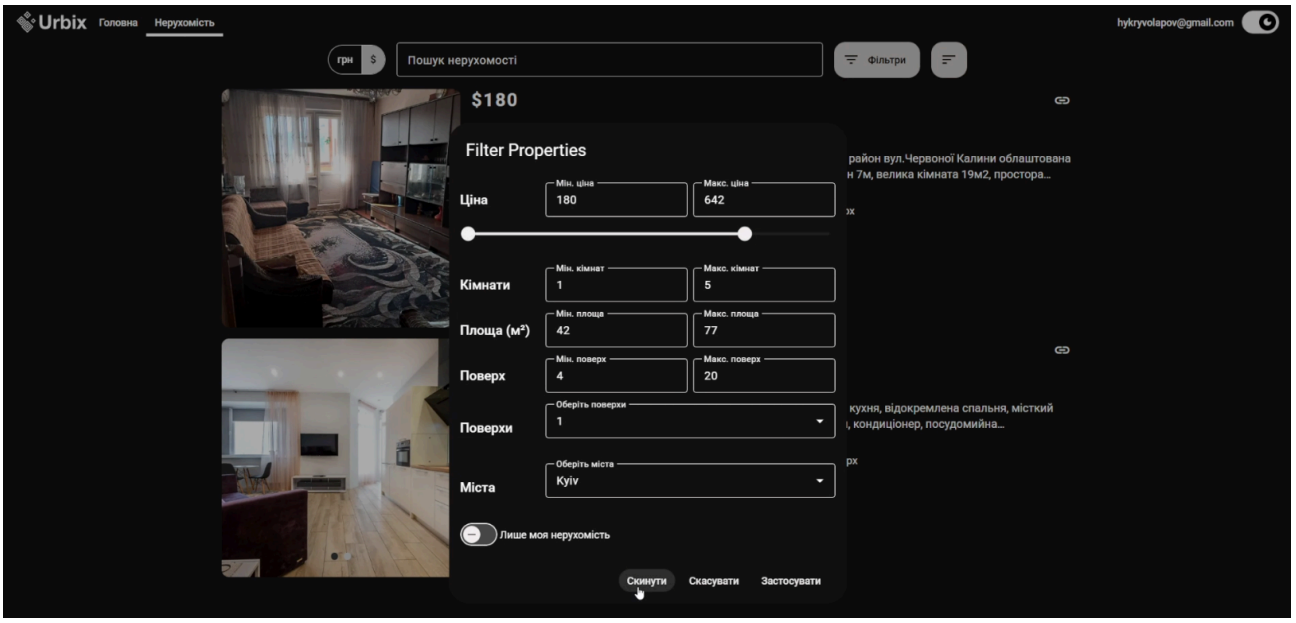

Рисунок 3.15. Фільтрація пошуку об’єктів нерухомості

Наявне й сортування: за замовчуванням, за ціною, за площею (рис. 3.16).

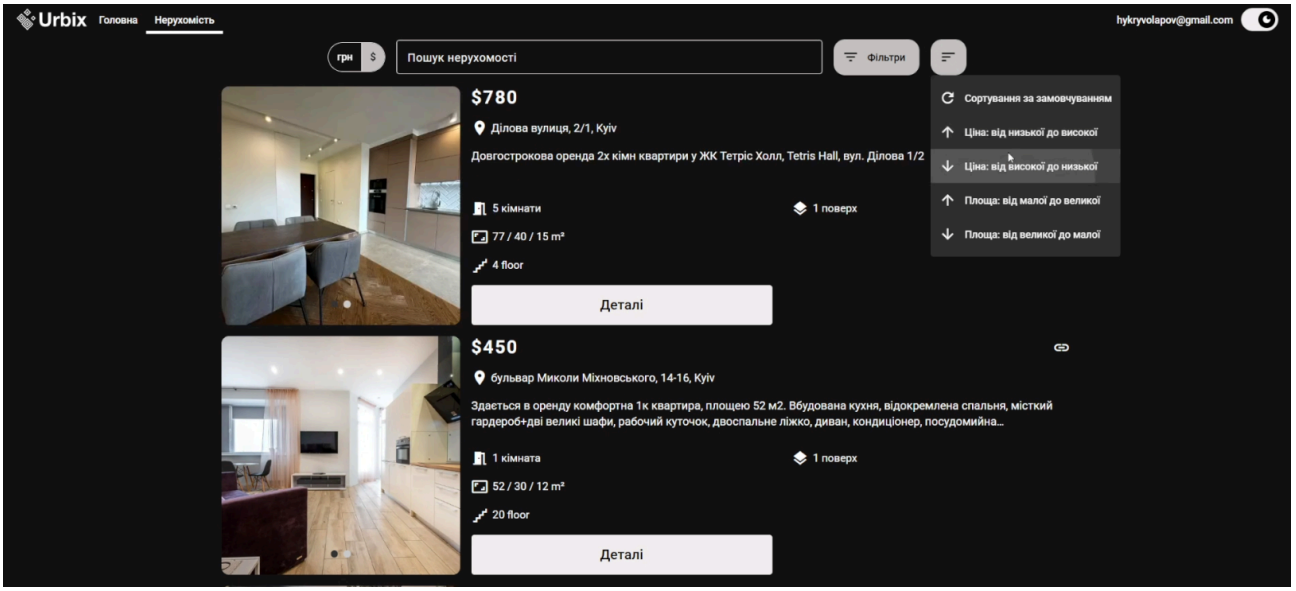

Рисунок 3.16. Приклад сортування – за збільшенням ціни

При переході на сторінку об’єкту нерухомості (після натискання кнопки «Деталі») можна побачити таку інформацію, як: фото нерухомості (може бути декілька, відображаються у вигляді “каруселі зображень”), адреса, кількість кімнат, площа, опис, кнопка для створення замовлення тощо.

Також, показується розташування об’єкту нерухомості на карті (рис. 3.17).

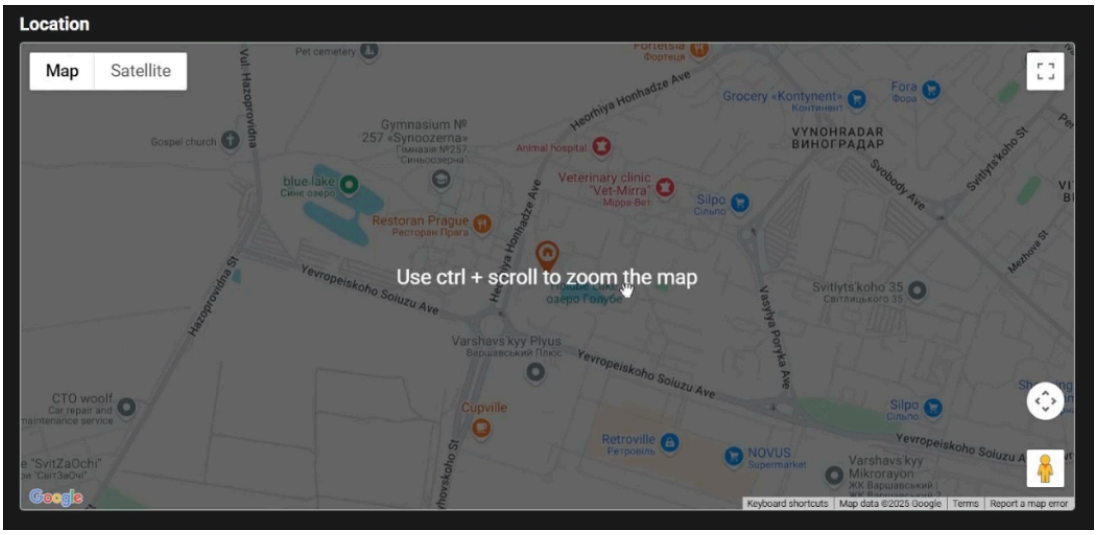

Рисунок 3.17. Перегляд нерухомості в Google Maps

Кожен користувач може переглянути інформацію щодо свого профілю, внести зміни до особистих даних й деяких параметрів інтерфейсу (рис. 3.18).

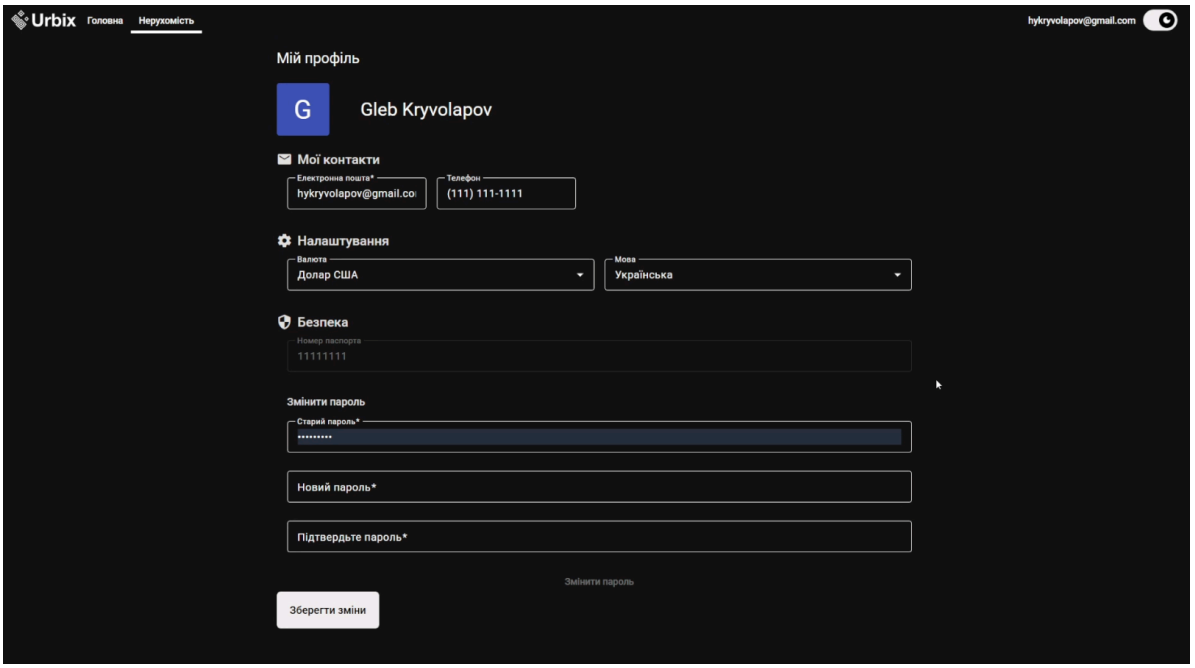

Рисунок 3.18. Сторінка профілю користувача

З облікового запису адміністратора у навігаційній панелі (зверху) з’являється кнопка для переходу на сторінку панелі адміністратора, де можна

переглядати замовлення користувачів і переглядати список усіх людей, що були зареєстровані в системі за весь час. Адміністратор може схвалити, відхилити чи видалити замовлення (рис. 3.19).

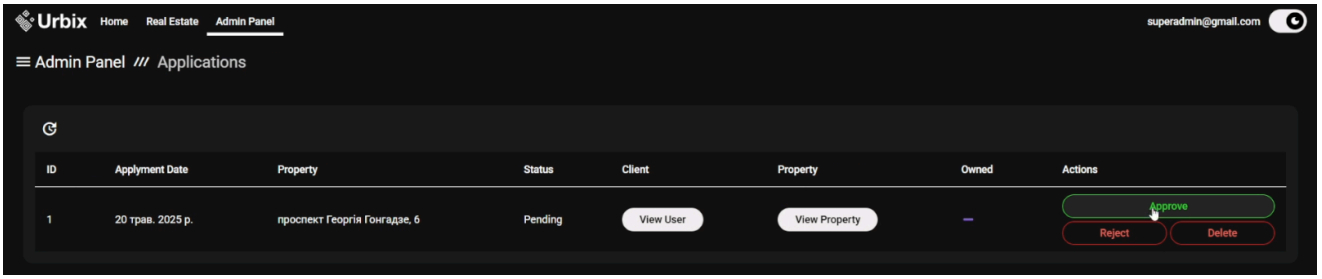

Рисунок 3.19. Сторінка панелі адміністратора – перегляд замовлень

Якщо адміністратор погодив угоду, то у користувача в розділі «Мої нерухомості» (рис. 3.20) з’явиться відповідна нерухомість (рис. 3.21).

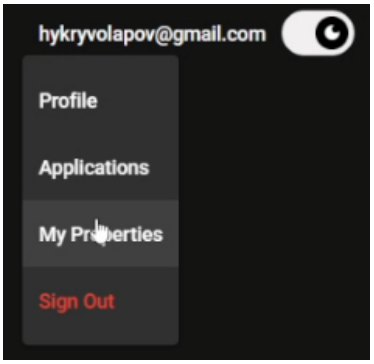

Рисунок 3.20. Перехід у розділ «Мої нерухомості»

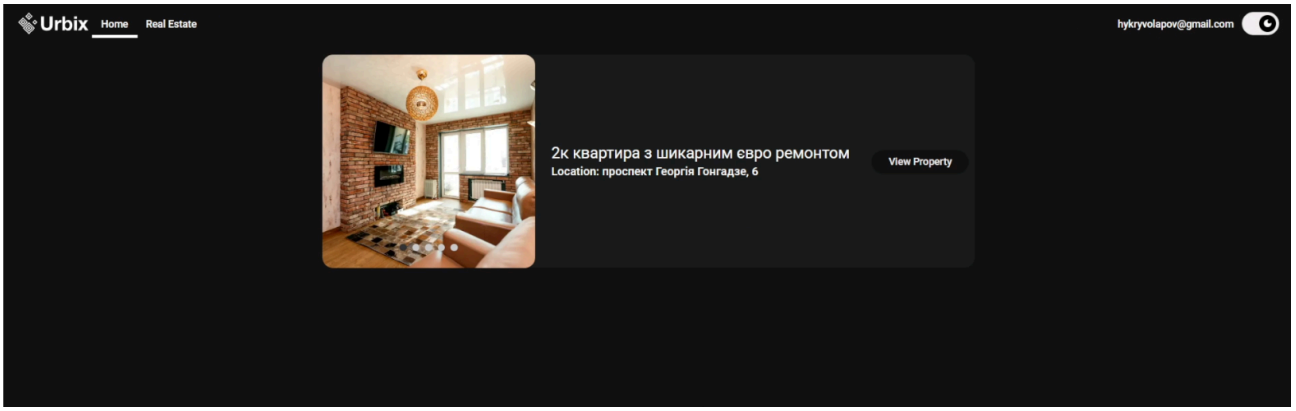

Рисунок 3.21. Сторінка нерухомостей користувача

3.6 Розгортання вебсистеми

Для розгортання розробленої вебсистеми було обрано хмарну платформу Google Cloud Platform (GCP). Такий вибір зумовлений її широким набором керованих сервісів та можливістю масштабування ресурсів відповідно до поточних потреб. Проєкт знаходиться за посиланням: https://urbix.pp.ua/

Оркестрація контейнеризованих застосунків здійснюється за допомогою Kubernetes, а управління інфраструктурою як кодом (IaC) реалізовано з використанням Terraform.

На першому етапі за допомогою Terraform описується та створюється вся необхідна інфраструктура в GCP. Створюється кластер Kubernetes, який буде середовищем для запуску frontend та backend частин застосунку. Terraform дозволяє декларативно визначити конфігурацію кластера, включаючи тип та кількість вузлів, мережеві налаштування та параметри безпеки. Для розміщення бази даних PostgreSQL, обраної на етапі проєктування, використовується керований сервіс Cloud SQL. Terraform відповідає за створення екземпляра бази даних, налаштування параметрів (тип машини, обсяг сховища, регіон), правил доступу та резервного копіювання. Для зберігання Docker-образів використовується приватний репозиторій Google Container Registry (GCR). Варто зазначити, що Terraform також використовується для налаштування VPC (Virtual Private Cloud), підмереж, правил брандмауера та статичних IP-адрес, якщо це необхідно.

Обидві частини вебсистеми – клієнтська (Angular) та серверна (ASP.NET Core) – контейнеризуються за допомогою Docker. Створюється Dockerfile, який описує процес збірки образу застосунку ASP.NET Core. Цей образ включає всі необхідні залежності та скомпільований код серверної частини. Після збірки образ завантажується до GCR. Клієнтська частина, розроблена на Angular, компілюється у статичні файли (HTML, CSS, JavaScript). Для їх обслуговування

створюється Docker-образ на основі легковісного вебсервера. Цей образ також завантажується до GCR.

Після підготовки інфраструктури та Docker-образів, розгортання застосунку в кластері Google Kubernetes Engine (GKE) здійснюється за допомогою маніфестів Kubernetes. Для клієнтської та серверної частини створюються окремі Kubernetes Deployments. Вони визначають, які Docker-образи використовувати, кількість реплік (екземплярів) кожного застосунку для забезпечення відмовостійкості та масштабованості. Для забезпечення доступу до backend-застосунку всередині кластера та ззовні (якщо потрібно) створюється Kubernetes Service (наприклад, типу ClusterIP для внутрішнього доступу або LoadBalancer для зовнішнього). Фронтенд також отримує свій Service. Для маршрутизації зовнішнього HTTP/HTTPS трафіку до відповідних сервісів (в першу чергу до frontend) використовується Kubernetes Ingress. Він може бути інтегрований з Google Cloud Load Balancer, забезпечуючи єдину точку входу, SSL-термінацію та розподіл навантаження. Конфігураційні дані (наприклад, рядки підключення до бази даних, ключі API) зберігаються в Kubernetes ConfigMaps та Secrets, що дозволяє безпечно передавати їх у контейнери без жорсткого кодування в образах.

Загальний процес розгортання передбачає таку послідовність дій:

1. Визначається вся необхідна інфраструктура GCP.
2. Застосування конфігурації Terraform: Командою terraform apply створюються або оновлюються ресурси в GCP.
3. Збираються Docker-образи для frontend та backend частин.
4. Зібрані образи завантажуються у приватний репозиторій Google Container Registry.
5. Застосовуються маніфести Kubernetes (або через kubectl apply, або керовані Terraform) для розгортання додатків в кластері GKE. Kubernetes завантажує образи з GCR та запускає контейнери відповідно до конфігурації Deployments.

У контексті моніторингу та інтерпретації вебтрафіку було використано Plausible Analytics – повноцінну, орієнтовану на конфіденційність альтернативу Google Analytics. Це програмне забезпечення з відкритим вихідним кодом мінімізує обсяг зібраних даних, не застосовує файли cookie й забезпечує відповідність вимогам GDPR (General Data Protection Regulation), що особливо актуально для європейського правового поля. На відміну від громіздких комерційних рішень, Plausible пропонує досить легкий скрипт, не навантажує сторінку та усуває ризик блокування рекламними фільтрами, водночас надаючи ключові метрики (унікальні відвідування, тривалість сеансу, конверсії тощо) у реальному часі через інтуїтивну панель керування. Завдяки можливості самостійного хостингу дослідник отримує повний контроль над зібраною інформацією, що сприяє дотриманню етичних принципів обробки персональних даних, а спрощена модель зберігання агрегованих показників полегшує аналітичну інтерпретацію результатів роботи системи.

ВИСНОВОК

У результаті виконання роботи було спроєктовано та розроблено вебсистему для автоматизації бізнес-процесів реалізації нерухомого майна, що дозволяє забудовнику швидко здійснювати реалізацію об’єктів нерухомості, а потенційним клієнтам – заощаджувати свій час..

Під час виконання роботи було проведено дослідження, в ході якого були визначені найкращі інструменти та технологічні рішення для забезпечення необхідного функціоналу, зручного та інтуїтивного користувацького інтерфейсу.

Вебсистема для продажу нерухомості була реалізована з використанням Angular, фреймворку для створення односторінкових застосунків (SPA). Мовою програмування була TypeScript, а також використовувалися HTML та SCSS для оформлення інтерфейсу. Для написання коду проєкту використовувалася інтегрована середа розробки Visual Studio і редактор коду Visual Studio Code. Для керування залежностями та встановлення додаткових модулів використовувався пакетний менеджер npm. Вебсистема взаємодіє з REST API для отримання необхідних даних про нерухомості, таких як зображення, опис, ціна та інша інформація. Реалізація цього проєкту на Angular дозволила легко керувати станом застосунку, забезпечити масштабованість, використовувати компоненти та модулі Angular Material для швидкої і зручної розробки користувацького інтерфейсу. А реалізація серверної частини на ASP.NET Core дозволило забезпечити кросплатформеність, підтримку Dependency Injection і використання підходу Code First для проєктування бази даних.

У якості можливих перспектив розвитку можна розглядати додавання можливості арендування об’єктів нерухомості та їх розділення за категоріями для більш швидкого й ефективного пошуку, впровадження функції створення візуального туру для об’єктів нерухомості, додаткові тестування системи на стійкість до кібератак і підвищення рівня безпеки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. A short history of the Web. *CERN*. URL: https://home.cern/science/computing/birth-web/short-history-web
2. Shontell A. FLASHBACK: This Is What The First-Ever Website Looked Like. *Business Insider*. URL: https://www.businessinsider.com/flashback-this-is-what-the-first-website-ever-looked-like-2011-6
3. Engineering G. A Complete Guide to S3 File Upload using pre-signed POST URLs. *Medium*. URL: https://medium.com/@Games24x7Tech/a-complete-guide-to-s3-file-upload-using-pre-signed-post-urls-9cb2d6cfc0ab
4. Karnani B. How Generative AI Is Transforming The Software Growth Landscape. *Forbes*. URL: https://www.forbes.com/councils/forbestechcouncil/2025/04/01/how-generative-ai-is-transforming-the-software-growth-landscape/
5. Етапи розробки вебзастосунків. URL: https://terentevdesignstudio.com/blog/etapi-rozrobki-veb-dodatkiv/
6. Web App Development: 8 Critical Stages and 12 Best Practices Need To Know. URL: https://www.smartosc.com/guide-to-web-app-development/
7. Chuck’s Academy. Introduction to HTML Semantics | Semantic HTML5 | Chuck’s Academy. Chuck's Academy. URL: https://www.chucksacademy.com/en/topic/html-semantic/intro
8. Types of Web Development for Beginners. *Maryville University Online*. URL: https://online.maryville.edu/online-bachelors-degrees/computer-science/careers/types-of-web-development/
9. Shikha R. Understanding Node.js Ecosystem and Tooling. *Medium*. URL: https://medium.com/@shikha.ritu17/understanding-node-js-ecosystem-and-tooling-fe7466d686c9

10. What is an api: how does it work and why do you need it. *Lvivity.* URL: https://lvivity.com/what-is-an-api-and-how-does-it-work
11. Плескач, В., Криволапов, Я., Криволапов, Г., Нагорний, В. (1 жовтня, 2024). Роль вебтехнологій у довгостроковій оптимізації та стратегії розвитку бізнесу. *Прикладні інформаційні системи та технології в цифровому суспільстві*. URL: https://aistis.knu.ua/wp-content/uploads/2024/10/Text_Book_confernce_01.10.2024.pdf
12. How to Create a Modern Web Application Architecture? URL: https://integrio.net/blog/modern-web-application-architecture
13. flatfy. *flatfy*. URL: https://flatfy.ua/uk/
14. DIM.RIA™ – вся нерухомість України. Продаж і оренда будь-якої нерухомості. *DOM.RIA.com*. URL: https://dom.ria.com/uk/
15. Web Application Authentication: How It Works and How to Implement It - Authgear. *Simplified Identity & Access Management, Built for Dev with Security - Authgear*. URL: https://www.authgear.com/post/web-application-authentication-guide
16. JWT.IO - JSON Web Tokens Introduction. *JSON Web Tokens - jwt.io*. URL: https://jwt.io/introduction
17. Singh S. Mastering User Experience Design: Elevate Your Product Management Game!. *LinkedIn*. URL: https://www.linkedin.com/pulse/mastering-user-experience-design-elevate-your-product-sourabh-singh
18. Pleskach V., Kryvolapov Y., Kryvolapov H., Zelikovska O. (20-21 листопада, 2024). Investigating E-Commerce Systems for Book Sales: From Theoretical Foundations to Software Development. *CEUR Workshop Proceedings – Information Technology and Implementation 2024*. URL: https://ceur-ws.org/Vol-3909/Paper_21.pdf
19. What is Single Page Application? 3 Examples, Pros and Cons. *Monocubed*. URL: https://www.monocubed.com/blog/what-is-single-page-application/

20. Patel B. SPA vs MPA: 8 Key Comparisons for Your Web Development. *Space-O Technologies*. URL: https://www.spaceotechnologies.com/blog/singlepage-application-vs-multi-page-application/

21. В Машкіна, ВО Абрамов, ТІ Носенко, ЄВ Іваніченко.Навчально-методичний посібник до виконання і захисту кваліфікаційної роботи бакалавра спеціальностей 122 Комп'ютерні науки для здобувачів вищої освіти денної форми навчання, Київський столичний університет імені Бориса Грінченка. Київ, 2024.- 43 с.

22. Теоретичні та практичні аспекти використання математичних методів та інформаційних технологій в освіті й науці:моногр. / за заг. ред. О. Литвин. — К.: Київ. ун-т ім. Б. Грінченка,2021. — 332 с.

23. Яскевич, Владислав Олександрович (2023) *JavaScriptбібліотека React Навчальний посібник для студентів спеціальності Комп'ютерні науки* Київський університет імені Бориса Грінченка, Україна, Київ. https://elibrary.kubg.edu.ua/id/eprint/48587/

24. ASP.NET Core, an open-source web development framework | .NET. *Microsoft*. URL: https://dotnet.microsoft.com/en-us/apps/aspnet

25. Karia R. Entity Framework Core. *Entity Framework Tutorial*. URL: https://www.entityframeworktutorial.net/efcore/entity-framework-core.aspx

26. Json.NET – Newtonsoft. *Json.NET* – Newtonsoft. URL: https://www.newtonsoft.com/json

27. Тестування API з Swagger: як перевірити функціональність API. *Корисні матеріали: Статті та новини IT-індустрії | Комп'ютерна школа Hillel*. URL: https://blog.ithillel.ua/articles/api-testing-with-swagger

28. Microsoft. Visual Studio Code – Code Editing. Redefined. *Visual Studio Code – Code Editing. Redefined*. URL: https://code.visualstudio.com/

29. Introduction to Visual Studio – GeeksforGeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/introduction-to-visual-studio/

30. Contributors to Wikimedia projects. Computer-aided software engineering - Wikipedia. *Wikipedia, the free encyclopedia*. URL: https://en.wikipedia.org/wiki/Computer-aided_software_engineering
31. Inheritance – EF Core. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/en-us/ef/core/modeling/inheritance
32. Angular. *Home • Angular*. URL: https://angular.dev/
33. Angular. Angular Material. *Angular Material*. URL: https://material.angular.io/
34. JavaScript With Syntax For Types. *TypeScript: JavaScript With Syntax For Types*. URL: https://www.typescriptlang.org/
35. Eloquent JavaScript, 3d edition/ Marijn Haverbeke, 2018; 237 c.
36. Kotonya G., Sommerville I., Requirements Engineering: Processes and Techniques. Нью-Йорк: Wiley, 1998 – 306 с.
37. Doguhan Uluca. Angular for Enterprise Applications. 3rd Ed. Packt Publishing. 2024.
38. Roberto Heckers. Effective Angular. Packt Publishing. 2024.
39. Ben Beattie-Hood. Modern TypeScript: A Practical Guide to Accelerate Your Development Velocity. Apress Media. 2023.

ДОДАТКИ

Додаток А

Код сервісу auth.service клієнтської частини

```
import { Injectable } from '@angular/core';
import { BehaviorSubject, Observable, tap } from 'rxjs';
import { APIEndpoints } from '@shared/api.endpoints';
import { HttpClient } from '@angular/common/http';
import { map, shareReplay } from 'rxjs/operators';
import { AccountService } from './account.service';

@Injectable({
  providedIn: 'root',
})
export class AuthService {
  constructor(
    private http: HttpClient,
    private accountService: AccountService
  ) {}

  login(formData: any): Observable<any> {
    return this.http.post<any>(APIEndpoints.login, formData).pipe(
      tap((data) => {
        //  this.setSession(data);
        this.accountService.startBackgroundFetch();
      }),
      shareReplay()
    );
  }

  logout(): Observable<any> {
    this.accountService.logout();
    return this.http.post(APIEndpoints.logout, '');
  }

  isAdmin() {
    return this.accountService.isAdmin();
  }

  checkLogin() {
    this.http.get<any>(APIEndpoints.checkLogin).subscribe({
      error: (error) => {
        console.log(error);
        if (error.status === 401) this.logout();
      },
    });
  }
}
```

Додаток Б

Код сервісу account.service клієнтської частини

```
import { Injectable } from '@angular/core';
import { AuthService } from './auth.service';
import { BehaviorSubject, Observable, catchError, map, tap, throwError } from
'rxjs';
import moment from 'moment';
import { APIEndpoints } from '@shared/api.endpoints';
import { HttpClient } from '@angular/common/http';

@Injectable({
  providedIn: 'root'
})
export class AccountService {
  private userSubject = new BehaviorSubject<any | null>(null);
  public isLoggedIn$ = this.userSubject.asObservable().pipe(
    map(user => !!user)
  );
  private lastProfileFetch: Date | null = null;
  private timer: any;
  refreshInterval = 5 * 60 * 1000;

  constructor(private http: HttpClient) {
    const user = localStorage.getItem('currentUser');

    if (user) {
      console.log(typeof(user));
      let parsedUser;
      try {
        parsedUser = JSON.parse(user);
      } catch (error) {
        console.error('Error parsing user data:', error);
        return;
      }
      this.userSubject.next(parsedUser);
    }

    this.isLoggedIn$.subscribe(isLoggedIn => {
      if (isLoggedIn)
        this.startBackgroundFetch();
    });
  }

  updateProfile(data: any): Observable<any> {
    return this.http.put(APIEndpoints.profile, data)
      .pipe(tap(() => { this.fetchProfile().subscribe()}));
  }

  getUser(): Observable<any> {
      return this.userSubject.asObservable();
  }

  fetchProfile(): Observable<any> {
    return this.http.get<any>(APIEndpoints.profile).pipe(
      tap(data => {
        console.log(data)
        this.userSubject.next(data);
        this.lastProfileFetch = new Date();
```

```
        localStorage.setItem('currentUser', JSON.stringify(data));
        console.log('User data fetched and stored: ' + JSON.stringify(data));
      }),
      catchError(error => {
        return throwError(error);
      })
    );
  }

  logout() {
    this.userSubject.next(null);
    localStorage.removeItem('currentUser');
    this.stopBackgroundFetch();
  }

  startBackgroundFetch() {
    this.timer = setInterval(() => {
      if (!this.userSubject.getValue() ||
        (this.lastProfileFetch && moment().diff(this.lastProfileFetch, 'minutes')
>= 5))
        console.log("startBackgroundFetch",);

        this.fetchProfile().subscribe()
    }, this.refreshInterval);
  }

  stopBackgroundFetch() {
    clearInterval(this.timer);
  }

  isLoggedIn(): boolean {
    return !!this.userSubject.getValue();
  }

  isAdmin(): boolean {
    if (this.isLoggedIn() === false) {
      return false;
    }
    let role = "";
    let claims: any[] = [];
    claims = this.userSubject.getValue().claims;
    claims.forEach(c => {
      if (c.type ===
"http://schemas.microsoft.com/ws/2008/06/identity/claims/role") {
        role = c.value;
      }
    });
    if (role === "Admin" || role === "SuperAdmin") {
      return true;
    }
    return false;
  }

  changePassword(oldPassword: string, newPassword: string, confirmPassword:
string): Observable<any> {
    return this.http.post(APIEndpoints.changePassword, { oldPassword, newPassword,
confirmPassword })
  }
}
```

Додаток В

Код сервісу property.service клієнтської частини

```
import { Injectable } from '@angular/core';
import { HttpClient, HttpParams } from '@angular/common/http';
import { Observable } from 'rxjs';
import { APIEndpoints } from '@shared/api.endpoints';
import { Property } from '@models/property';

@Injectable({ providedIn: 'root' })
export class PropertyService {
  constructor(private http: HttpClient) {}

  getAllProperties(): Observable<Property[]> {
    return this.http.get<Property[]>(APIEndpoints.property);
  }

  getMyProperties(): Observable<Property[]> {
    return this.http.get<Property[]>(APIEndpoints.userProperties);
  }

  getUserProperties(userId: string): Observable<Property[]> {
    return this.http.get<Property[]>(
      `${APIEndpoints.userProperties}/${userId}`
    );
  }

  getPropertyById(id: number): Observable<Property> {
    return this.http.get<Property>(`${APIEndpoints.property}/${id}`);
  }

  createProperty(property: Property): Observable<Property> {
    return this.http.post<Property>(APIEndpoints.property, property);
  }

  updateProperty(property: Property): Observable<Property> {
    return this.http.put<Property>(`${APIEndpoints.property}`, property);
  }

  deleteProperty(id: number): Observable<void> {
    return this.http.delete<void>(`${APIEndpoints.property}/${id}`);
  }

  addPropertyToUser(userId: string, propertyId: number): Observable<void> {
    const params = new HttpParams()
      .set('userId', userId)
      .set('propertyId', propertyId.toString());
    return this.http.post<void>(
      `${APIEndpoints.userAddProperty}`,
      {},
      { params }
    );
  }

  removePropertyFromUser(userId: string, propertyId: number): Observable<void> {
    const params = new HttpParams()
      .set('userId', userId)
      .set('propertyId', propertyId.toString());
    return this.http.delete<void>(`${APIEndpoints.userRemoveProperty}`, {
```

```
      params,
    });
  }

  getPropertyImages(propertyId: number): Observable<any[]> {
    return this.http.get<any[]>(
      `${APIEndpoints.property}/${propertyId}/images`
    );
  }

  updatePropertyImage(
    imageId: number | string,
    formData: FormData
  ): Observable<any> {
    return this.http.put(`${APIEndpoints.propertyImages}/${imageId}`, formData);
  }

  uploadPropertyImages(
    propertyId: number,
    formData: FormData,
    clearExisting: boolean = true
  ): Observable<any> {
    const url =
`${APIEndpoints.propertyImages}/${propertyId}?clearExisting=${clearExisting}`;
    return this.http.post(url, formData);
  }

  deletePropertyImage(imageId: number | string): Observable<any> {
    return this.http.delete(`${APIEndpoints.propertyImages}/${imageId}`);
  }

  reorderPropertyImages(
    propertyId: number,
    orderedIds: (number | string)[]
  ): Observable<any> {
    return this.http.put(`${APIEndpoints.propertyImages}/reorder`, {
      propertyId,
      orderedImageIds: orderedIds,
    });
  }
}
```

Додаток Г

Код сервісу applications.service клієнтської частини

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
import { APIEndpoints } from '@shared/api.endpoints';
import Application from '@models/application';


@Injectable({ providedIn: 'root' })
export class ApplicationService {
  constructor(private http: HttpClient) { }

  getAllApplications(): Observable<Application[]> {
    return this.http.get<Application[]>(APIEndpoints.applications);
  }

  getApplicationById(id: number): Observable<Application> {
    return this.http.get<Application>(`${APIEndpoints.application}/${id}`);
  }

  createApplication(propertyId: number): Observable<Application> {
    return
this.http.post<Application>(`${APIEndpoints.application}/${propertyId}`, {});
  }

  getMyApplications(): Observable<Application[]> {
    return this.http.get<Application[]>(APIEndpoints.myApplications);
  }

  getUserApplications(id: string): Observable<Application[]> {
    return this.http.get<Application[]>(`${APIEndpoints.userApplication}/${id}`);
  }

  approveApplication(id: number): Observable<Application> {
    return
this.http.post<Application>(`${APIEndpoints.applicationStatusApprove}/${id}`, {});
  }

  rejectApplication(id: number): Observable<Application> {
    return
this.http.post<Application>(`${APIEndpoints.applicationStatusReject}/${id}`, {});
  }

  deleteApplication(id: number): Observable<void> {
    return this.http.delete<void>(`${APIEndpoints.application}/${id}`);
  }
}
```

Додаток Д

Код модуля ApplicationContext, де визначено структуру БД

```
using Urbix.Models.AdminModels;
using Urbix.Models.ApplymentModels;
using Urbix.Models.ClientModels;
using Microsoft.EntityFrameworkCore;

namespace Urbix.Models
{
	public class ApplicationContext(DbContextOptions<ApplicationContext>
options) : DbContext(options)
	{
		public DbSet<Applyment> Applyments { get; set; }
		public DbSet<User> Users { get; set; }
		public DbSet<Client> Clients { get; set; }
		public DbSet<Admin> Admins { get; set; }
		public DbSet<Property> Properties { get; set; }
		public DbSet<Claim> Claims { get; set; }
		public DbSet<PropertyImage> PropertyImages { get; set; }

		protected override void OnModelCreating(ModelBuilder modelBuilder)
		{
			base.OnModelCreating(modelBuilder);
			modelBuilder.Entity<Property>()
				.Property(p => p.Coords)
				.HasConversion(new CoordinatesConverter());

			// Configure Table-per-Hierarchy (TPH) inheritance
			modelBuilder.Entity<User>()
				.HasDiscriminator<string>("Discriminator")
				.HasValue<Client>("Client")
				.HasValue<Admin>("Admin");

			modelBuilder.Entity<User>()
				.Property(e => e.Id)
				.ValueGeneratedOnAdd();

			modelBuilder.Entity<User>()
			.HasMany(u => u.Claims)
			.WithMany(u => u.Users)
			.UsingEntity(j => j.ToTable("UserCaims"));

			modelBuilder.Entity<Applyment>()
				.HasOne(a => a.Client)
				.WithMany(c => c.Applyments)
				.HasForeignKey(a => a.ClientId);

			modelBuilder.Entity<Applyment>()
				.HasOne(a => a.Property)
				.WithMany()
				.HasForeignKey(a => a.PropertyId);

			modelBuilder.Entity<PropertyImage>()
				.HasOne(pi => pi.Property)
				.WithMany(p => p.Images)
				.HasForeignKey(pi => pi.PropertyId);
		}
	}
}
```

Додаток Е

Код контролера ApplymentController серверної частини

```
using Urbix.Models;
using Urbix.Models.ApplymentModels;
using Urbix.Models.ClientModels;
using Urbix.Services;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Swashbuckle.AspNetCore.Annotations;
using System.Security.Claims;

namespace Urbix.Controllers
{
	[Route("api/[controller]")]
	[ApiController]
	public class ApplymentController : ControllerBase
	{
		private readonly ILogger<ApplymentController> _logger;
		private readonly ApplymentService _applymentService;

		public ApplymentController(ILogger<ApplymentController> logger,
ApplymentService applymentService)
		{
			_logger = logger;
			_applymentService = applymentService;
		}

		/// <summary>
		/// Get all applyments.
		/// </summary>
		/// <returns>A list of all applyments.</returns>
		[HttpGet("applications")]
		public async Task<IActionResult> GetApplyments()
		{
			var applyments = await _applymentService.GetApplymentsAsync();
			return Ok(applyments);
		}

		/// <summary>
		/// Get applyments for the current user.
		/// </summary>
		/// <returns>A list of applyments for the current user.</returns>
		[HttpGet("user")]
		[Authorize(Roles = "Client")]
		public async Task<IActionResult> GetUserApplyments()
		{
			var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);
			var client = await _applymentService.GetUserAsync(userId);

			if (client == null)
				return NotFound();

			var applyments = client.Applyments.Select(a => new ApplymentDTO
			{
				ClientId = a.ClientId,
				PropertyId = a.Property.Id,
				ApplymentDate = a.ApplymentDate,
				Status = a.Status,
```

```
                        Id = a.Id,
                        Property = a.Property
                  });

                  return Ok(applyments);
            }

            /// <summary>
            /// Get applyments for a specific user.
            /// </summary>
            /// <param name="id">The ID of the user.</param>
            /// <returns>A list of applyments for the specified user.</returns>
            [HttpGet("user/{id}")]
            [Authorize(Roles = "Admin")]
            public async Task<IActionResult> GetUserApplyments(string id)
            {
                  var client = await _applymentService.GetUserAsync(id);

                  if (client == null)
                        return NotFound();

                  var applyments = client.Applyments.Select(a => new ApplymentDTO
                  {
                        ClientId = a.ClientId,
                        PropertyId = a.Property.Id,
                        ApplymentDate = a.ApplymentDate,
                        Status = a.Status,
                        Id = a.Id,
                        Property = a.Property
                  });

                  return Ok(applyments);
            }

            /// <summary>
            /// Get applyment by ID.
            /// </summary>
            /// <param name="id">The ID of the applyment to retrieve.</param>
            /// <returns>The applyment with the specified ID.</returns>
            [HttpGet("{id}")]
            [SwaggerOperation(Summary = "Get applyment by id", Description = "Get
applyment by id")]
            public async Task<IActionResult> Get(int id)
            {
                  var applyment = await
_applymentService.GetApplymentByIdAsync(id);
                  if (applyment == null)
                        return NotFound();
                  return Ok(applyment);
            }

            /// <summary>
            /// Creates a new applyment.
            /// </summary>
            /// <param name="propertyId">The ID of the property for the
applyment.</param>
            /// <returns>The created applyment.</returns>
            [HttpPost("{propertyId}")]
            [Authorize(Roles = "Client")]
            [SwaggerResponse(200, "Applyment created", typeof(Applyment))]
            public async Task<IActionResult> Create(int propertyId)
            {
                  var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);
```

```
                  if (!ModelState.IsValid)
                        return BadRequest();

                  var applyment = await
_applymentService.CreateApplymentAsync(userId, propertyId);
                  if (applyment == null)
                        return StatusCode(409, "Applyment already exists or
invalid data.");

                  return Ok(applyment);
            }

            /// <summary>
            /// Approve an applyment.
            /// </summary>
            /// <param name="id">The ID of the applyment to approve.</param>
            /// <returns>Ok if the applyment was approved successfully, NotFound
otherwise.</returns>
            [HttpPost("approve/{id}")]
            [Authorize(Roles = "Admin")]
            public async Task<IActionResult> Approve(int id)
            {
                  var success = await
_applymentService.ApproveApplymentAsync(id);
                  if (!success)
                        return NotFound();
                  return Ok();
            }

            /// <summary>
            /// Reject an applyment.
            /// </summary>
            /// <param name="id">The ID of the applyment to reject.</param>
            /// <returns>Ok if the applyment was rejected successfully, NotFound
otherwise.</returns>
            [HttpPost("reject/{id}")]
            [Authorize(Roles = "Admin")]
            public async Task<IActionResult> Reject(int id)
            {
                  var success = await _applymentService.RejectApplymentAsync(id);
                  if (!success)
                        return NotFound();
                  return Ok();
            }

            /// <summary>
            /// Delete an applyment.
            /// </summary>
            /// <param name="id">The ID of the applyment to delete.</param>
            /// <returns>Ok if the applyment was deleted successfully, NotFound
otherwise.</returns>
            [HttpDelete("{id}")]
            [Authorize(Roles = "Admin")]
            public async Task<IActionResult> Delete(int id)
            {
                  var success = await _applymentService.DeleteApplymentAsync(id);
                  if (!success)
                        return NotFound();
                  return Ok();
            }
      }
}
```