Міністерство освіти і науки України
Криворізький національний університет
Кафедра моделювання та програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти магістра

зі спеціальності 121 – Інженерія програмного забезпечення

На тему: Розробка веб-застосунку маркетплейсу з продажу нерухомості

Засвідчую, що в цій кваліфікаційній роботі немає запозичень із праць інших авторів без відповідних посилань.

Студент гр. ІПЗ-23-2м

_____ /О. О. Мальченко/

Керівник
кваліфікаційної роботи _____ / А. М. Стрюк /

Завідувач кафедри _____ / А. М. Стрюк /

Кривий Ріг

2024

Криворізький національний університет

Факультет: Інформаційних технологій

Кафедра: Моделювання та програмного забезпечення

Ступінь вищої освіти: магістр

Спеціальність: 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

Зав. кафедри

_____ А. М. Стрюк

«___» _____ 20__ р

ЗАВДАННЯ

на кваліфікаційну роботу

студенту групи ІПЗ-23-2м Мальченку Олександру Олександровичу

1. Тема: Розробка веб-застосунку маркетплейсу з продажу нерухомості затверджено наказом по КНУ N278с від «15» квітня 2024 р
2. Термін подання студентом закінченої роботи: «6» грудня 2024р.
3. Вихідні дані по роботі: розроблюваний веб-застосунок маркетплейс для ринку нерухомості повинен відповідати сучасним вимогам та тенденціям розробки веб-застосунку, підтримувати та реалізовувати повний цикл продажу, оренди та розміщення оголошень про нерухомість.
4. Зміст пояснювальної записки (перелік питань, що їх треба розробити): виконати аналіз предметної області та існуючих аналогів, розробити структуру та архітектуру майбутнього застосунку, провести вдосконалення застарілої версії застосунку, здійснити програмну реалізацію розробленого застосунку, виконати аналіз економічної ефективності розроблювального застосунку, провести тестування.
5. Перелік ілюстративного матеріалу: діаграма використання, блок-схеми розроблених алгоритмів, діаграма послідовності, зображення екранів застосунку, схема взаємодії модулів системи.

Календарний план:

№	Найменування етапів кваліфікаційної роботи	Термін виконання етапів роботи
1	Огляд літератури за тематикою та збір даних	25.12.2023 – 11.02.2024
2	Проведення аналізу предметної області та існуючих аналогів	12.02.2024 – 15.04.2024
3	Формування гіпотези, задач та мети роботи	16.04.2024– 25.04.2024
4	Оформлення матеріалів першого розділу роботи	26.04.2024 – 06.05.2024
5	Створення блок-схем алгоритмів, діаграм використання та послідовності, аналіз вимог	07.05.2024 – 29.06.2024
6	Оформлення матеріалів другого розділу роботи	30.06.2024– 09.07.2024
7	Розробка веб-застосунку	10.07.2024– 30.09.2024
8	Оформлення матеріалів третього розділу роботи	01.10.2024– 09.10.2024
9	Економічний аналіз та оцінка ефективності	10.10.2024 – 14.11.2024
10	Оформлення матеріалів четвертого розділу роботи	15.11.2024 – 29.11.2024
11	Остаточне оформлення пояснювальної записки	30.11.2024– 03.12.2024

Дата видачі завдання «_» _____ 20__ р.
Студент _____ / О. О. Мальченко /
Керівник роботи _____ / А. М. Стрюк /

РЕФЕРАТ

ВЕБ-ЗАСТОСУНОК, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, МАРКЕТПЛЕЙС, ПРОДАЖ НЕРУХОМОСТІ, ЕКОНОМІЧНА ЕФЕКТИВНІСТЬ, TYPESCRIPT, NODEJS, NEXTJS, REACT, EXPRESSJS, FULL-STACK РОЗРОБКА.

Пояснювальна записка: 129 с., 4 табл., 20 рис., 1 дод., 15 джерел.

Метою кваліфікаційної роботи є концептуалізація, розробка та оптимізація сучасного веб-застосунку маркетплейсу для купівлі та продажу нерухомості. Дослідження спрямоване на створення інноваційної платформи, що забезпечить користувачам розширений набір функцій для ефективної участі в операціях на ринку нерухомості. Зокрема, планується модернізація попередньої версії веб-додатка та аналіз економічної доцільності впровадження таких рішень. Об’єктом дослідження є сучасні веб-застосунки, призначені для функціонування як цифрові маркетплейси у сфері купівлі та продажу нерухомості.

У роботі проведено аналіз сучасних тенденцій розробки веб-застосунків у сфері купівлі та продажу нерухомості, визначено ключові функціональні можливості, які сприяють підвищенню ефективності користування маркетплейсами.

Розроблено повнофункціональний веб-застосунок із використанням сучасного технологічного стеку. На фронтенді застосовано TypeScript, React, Next.js та Material-UI (MUI), що забезпечило адаптивність інтерфейсу, зручність користувацького досвіду та високу продуктивність. Для бекенд-частини використано Express.js, Node.js, TypeScript, а також інтегровано Redis для кешування та MongoDB для роботи з базами даних.

З метою підвищення масштабованості та ефективності, серверна частина була контейнеризована за допомогою Docker, а для оптимізації запитів застосовано tRPC. Процес компіляції та транспіляції коду здійснено за допомогою SWC, що дозволило досягти високої швидкості обробки.

ABSTRACT

WEB APPLICATION, SOFTWARE, MARKETPLACE, REAL ESTATE SALES, COST-EFFECTIVENESS, TYPESCRIPT, NODEJS, NEXTJS, REACT, EXPRESSJS, FULL-STACK DEVELOPMENT.

Explanatory note: 129 p., 4 tables, 20 figure, 1 pp., 15 sources.

The purpose of the qualification work is to conceptualize, develop and optimize a modern web-based marketplace application for buying and selling real estate. The research is aimed at creating an innovative platform that will provide users with an expanded set of functions for effective participation in real estate transactions. In particular, it is planned to modernize the previous version of the web application and analyze the economic feasibility of implementing such solutions. The object of the study is modern web applications designed to function as digital marketplaces in the field of real estate purchase and sale.

The paper analyzes the current trends in the development of web applications in the field of real estate purchase and sale, identifies the key functionalities that contribute to increasing the efficiency of using marketplaces.

A fully functional web application has been developed using a modern technology stack. TypeScript, React, Next.js, and Material-UI (MUI) were used on the frontend, which ensured the adaptability of the interface, user experience, and high performance. For the backend part, we used Express.js, Node.js, TypeScript, and integrated Redis for caching and MongoDB for working with databases.

To increase scalability and efficiency, the server side was containerized using Docker, and tRPC was used to optimize requests. The process of compiling and transpiling the code was carried out using SWC, which allowed it to achieve high processing speed.

ЗМІСТ

ВСТУП

В умовах стрімкого розвитку цифрових технологій веб-додатки виявилися незамінними для покращення взаємодії між покупцями та продавцями, особливо в секторі нерухомості, з акцентом на вторинному ринку житла. З часом ці додатки перетворилися на фундаментальні інструменти для автоматизації бізнес-процесів, оптимізації пошуку нерухомості та підвищення якості обслуговування клієнтів. Зростаюча залежність від таких інструментів була ще більше посилена зовнішнім тиском, таким як війна, що триває в Україні, яка суттєво вплинула на ринок нерухомості та вимоги користувачів.

На тлі потрясінь, спричинених конфліктом, значна частина населення зіткнулася з проблемою пошуку альтернативного житла через втрату своїх домівок. Це спричинило підвищений попит на швидкі, зручні та ефективні платформи для полегшення операцій з нерухомістю. Водночас багато власників нерухомості прагнуть прискорити продаж своїх активів, щоб профінансувати переїзд або задовольнити нагальні потреби. Ці тенденції підкреслюють необхідність модернізації існуючих рішень та адаптації до мінливих потреб користувачів шляхом розробки інноваційних веб-додатків.

Метою цього дослідження є розробка та впровадження конкурентоспроможного, зручного веб-додатку, призначеного для купівлі та продажу нерухомості. Запропоноване рішення поєднує інтуїтивно зрозумілий інтерфейс з розширеним набором функціональних можливостей для задоволення різноманітних потреб кінцевих користувачів, від приватних осіб до агентств нерухомості. Основна увага в дослідженні приділяється оптимізації користувацького досвіду (UX) як наріжному каменю для підтримки актуальності та популярності додатку в конкурентному цифровому середовищі.

У дослідженні застосовано комплексний підхід до розробки веб-додатків з використанням сучасного технологічного стеку. Фронтенд використовує TypeScript, React, Next.js та Material-UI (MUI), тоді як бекенд

використовує Express.js, Node.js, Redis, MongoDB, Docker та tRPC, забезпечуючи масштабованість та надійність. Окрім технічних міркувань, дослідження заглиблюється в аналіз поточних тенденцій, визначення основних характеристик для конкурентоспроможності та оцінку впливу технологічного вибору на ефективність UX.

Науковий внесок цієї роботи полягає в визначення її економічної ефективності та поглибленні знань про розробку веб-додатків для сектору нерухомості. Вона пропонує розуміння того, як інтегрувати передові технології та інструменти, балансуючи між технічною точністю та орієнтованим на користувача дизайном. Практичні результати включають функціональне, готове до ринку рішення, здатне підвищити якість послуг, оптимізувати транзакційні процеси та сприяти інноваціям у сфері веб-розробки.

Вирішуючи нагальні та довгострокові проблеми ринку нерухомості, це дослідження не лише забезпечує надійну основу для вдосконалення існуючих платформ, але й закладає фундамент для створення нових, конкурентоспроможних додатків. Його цілісна методологія та зосередженість на практичній реалізації забезпечують його застосовність у різних контекстах, значно збагачуючи як спільноту розробників програмного забезпечення, так і індустрію нерухомості.

1. ТЕОРЕТИЧНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

При проведенні теоретичного аналізу предметної області було встановлено кілька критичних вимог. Найголовнішою з них було всебічне вивчення сучасних тенденцій в індустрії нерухомості, з особливим акцентом на впровадженні та впливі веб-застосунків у цьому секторі. Це вимагало вивчення ролі, яку ці додатки відіграють у полегшенні ключових операційних процесів, таких як розміщення об'єктів нерухомості, взаємодія між клієнтом і агентом та управління транзакціями. Крім того, важливо було визначити переваги та виклики, пов'язані з впровадженням повнофункціональних веб-додатків на ринку нерухомості та визначення економічної ефективності такого рішення. Цей аналіз мав на меті виявити фактори, які найбільше впливають на ефективність і прибутковість цих рішень, а також ті, що мають менший вплив. Розуміння цієї динаміки дозволило закласти надійну основу для розробки додатку, здатного задовольнити потреби ринку, підвищити рівень задоволеності користувачів та покращити операційні результати.

1.1. Огляд сучасних тенденцій у веб-застосунках з продажу нерухомості.

Цифрова трансформація є ключовим чинником розвитку сучасного бізнесу. Веб-застосунки у сфері продажу нерухомості відіграють важливу роль у забезпеченні зручності користувачів та операційної ефективності компаній. У сучасних умовах технологічний розвиток зосереджується на персоналізації, інтерактивності та автоматизації процесів, що особливо важливо для ринку нерухомості, який вимагає точності та швидкості у наданні інформації клієнтам.

Персоналізація через штучний інтелект та рекомендаційні системизасновані на штучному інтелекті (AI) [1] та машинному навчанні (ML) [2], дозволяють покращити користувацький досвід шляхом надання

персоналізованих пропозицій. Системи аналізують вподобання клієнтів, історію пошуку, демографічні дані та інші фактори для підбору релевантних об’єктів нерухомості.

AI-аналітика дозволяє прогнозувати тенденції ринку та надавати цінову аналітику на основі великих даних (Big Data). Чат-боти на основі нейромережевих технологій забезпечують миттєве консультування та відповіді на запитання клієнтів у реальному часі, що знижує потребу у великій кількості персоналу [3].

Технології доповненої реальності (AR) та віртуальної реальності (VR) відіграють значну роль у візуалізації об’єктів нерухомості [4]. Потенційні покупці можуть здійснювати віртуальні тури по житлових або комерційних приміщеннях без фізичної присутності.

AR-моделі дозволяють переглядати тривимірні плани кімнат у реальному середовищі через мобільні пристрої. VR-тури мінімізують потребу у фізичних показах об’єктів, економлячи час агентів та клієнтів. Ці технології підвищують ефективність пошуку нерухомості та забезпечують більш високий рівень задоволеності клієнтів.

Впровадження хмарних технологій забезпечує масштабованість та доступність веб-застосунків для ринку нерухомості: AWS та Google Cloud дозволяють обробляти та зберігати великі обсяги даних, включаючи зображення, відео та документи [5]. Автоматичне масштабування дозволяє системам ефективно обробляти піковий трафік під час підвищеного інтересу клієнтів. Хмарні рішення сприяють надійності роботи веб-застосунків та знижують витрати на утримання серверів.

Застосування Big Data дозволяє агентствам нерухомості збирати та аналізувати інформацію про поведінку клієнтів, ринкові тренди та ефективність рекламних кампаній. Аналітичні інструменти допомагають створювати детальні звіти та адаптувати стратегії бізнесу. Збір даних у режимі реального часу дозволяє швидко реагувати на зміни у вподобаннях клієнтів та підвищувати конверсію.

Сучасні тенденції [6] у веб-застосунках для продажу нерухомості базуються на використанні передових технологій, таких як штучний інтелект, доповнена реальність, хмарні рішення та інтернет речей. Ці інновації забезпечують персоналізований досвід для клієнтів, автоматизацію процесів та оперативний аналіз ринкових даних.

1.2. Цілі та задачі кваліфікаційної роботи.

Ціль роботи полягає в концептуалізації, розробці та оптимізації сучасного веб-застосунку маркетплейсу для купівлі та продажу нерухомості з урахуванням сучасних технологічних тенденцій та потреб користувачів ринку. Для досягнення поставленої мети були визначені наступні завдання:

- Провести аналіз предметної області та існуючих аналогів веб-застосунків на ринку нерухомості, зокрема їх функціональних можливостей, переваг та недоліків.
- Визначити ключові технологічні вимоги для розробки сучасного маркетплейсу з акцентом на продуктивність, масштабованість та зручність користувацького досвіду.
- Розробити архітектуру веб-застосунку, що включає як фронтенд-, так і бекенд-частини, із використанням сучасного технологічного стеку (TypeScript, React, Next.js, MUI, Node.js, Express.js, MongoDB, Redis та Docker).
- Реалізувати функціональний веб-застосунок, що забезпечує: Адаптивний інтерфейс для зручності користувачів; Швидку обробку даних завдяки використанню Redis для кешування; Оптимізацію запитів за допомогою tRPC; Швидке транспілювання та компіляцію коду з використанням SWC.

- Дослідити економічну доцільність впровадження розробленого рішення шляхом аналізу потенційної ефективності для кінцевих користувачів та бізнес-агентств.
- Провести тестування веб-застосунку для виявлення можливих помилок та визначення рівня продуктивності, надійності та користувацького задоволення.
- Розробити рекомендації для масштабування платформи, включно з подальшою оптимізацією функціональності та адаптацією до потреб мінливого ринку.

Результатом роботи є створення конкурентоспроможного веб-застосунку для купівлі та продажу нерухомості, який поєднує сучасні технології, інтуїтивно зрозумілий інтерфейс та економічну ефективність. Запропоноване рішення сприятиме покращенню якості обслуговування, оптимізації транзакційних процесів та впровадженню інновацій у галузі нерухомості.

1.3. Переваги та виклики розробки веб-застосунку для ринку нерухомості.

Індустрія нерухомості протягом останнього десятиліття переживає значну цифрову трансформацію. Поява платформ для купівлі, продажу та оренди нерухомості, таких як Zillow, Realtor.com чи Airbnb, показала, що ефективний веб-застосунок може забезпечити взаємодію між продавцями, покупцями та орендарями у реальному часі. Головною особливістю є об'єднання зручного фронтенду для користувачів із потужним серверним бекендом для обробки даних та аналітики.

Веб-застосунок дозволяє користувачам мати швидкий і зручний доступ до актуальної інформації щодо нерухомості. Завдяки інтеграції баз даних та

картографічних інструментів, користувачі можуть переглядати дані про об'єкти, оцінювати їх стан, місце розташування та порівнювати ціни в режимі реального часу. Zillow [7] використовує технологію REST API для інтеграції даних із зовнішніх джерел, надаючи користувачам точну інформацію про ринок.

Веб-застосунки, побудовані на сучасних хмарних рішеннях, таких як AWS (Amazon Web Services) [8] чи Microsoft Azure, мають гнучку архітектуру, що дозволяє масштабувати ресурс в залежності від навантаження. Це забезпечує стабільну роботу системи навіть при великій кількості користувачів.

Завдяки автоматизованим алгоритмам, веб-застосунок спрощує багато процесів:

- Обробка заявок на купівлю/оренду.
- Управління угодами через інтеграцію систем CRM (Customer Relationship Management).
- Платіжні операції завдяки використанню Stripe або PayPal API.

Розробка full-stack застосунку передбачає інтеграцію фронтенду та бекенду:

- Frontend: React.js, Vue.js або Angular для побудови інтерактивних інтерфейсів.
- Backend: Node.js, Django (Python), або Spring Boot (Java) для серверної логіки та обробки запитів.
- База даних: MySQL, PostgreSQL або MongoDB для збереження структурованих даних.

Розробка веб-застосунку для ринку нерухомості є перспективним напрямом, що сприяє автоматизації, покращенню користувацького досвіду та ефективному обміну даними. Проте виклики, пов'язані з безпекою, складністю архітектури та оптимізацією, потребують застосування сучасних технологічних рішень і високої компетентності розробників.

1.4. Аналіз існуючих аналогів.

Airbnb та Zillow є глобальними лідерами на ринку веб-додатків для оренди та продажу нерухомості.

Airbnb [9] фокусується на короткостроковій оренді житла для туристів та подорожуючих. Платформа приваблює своєю інтуїтивно зрозумілою навігацією та акцентом на візуальну складову пропозицій (див. рис. 1.1).

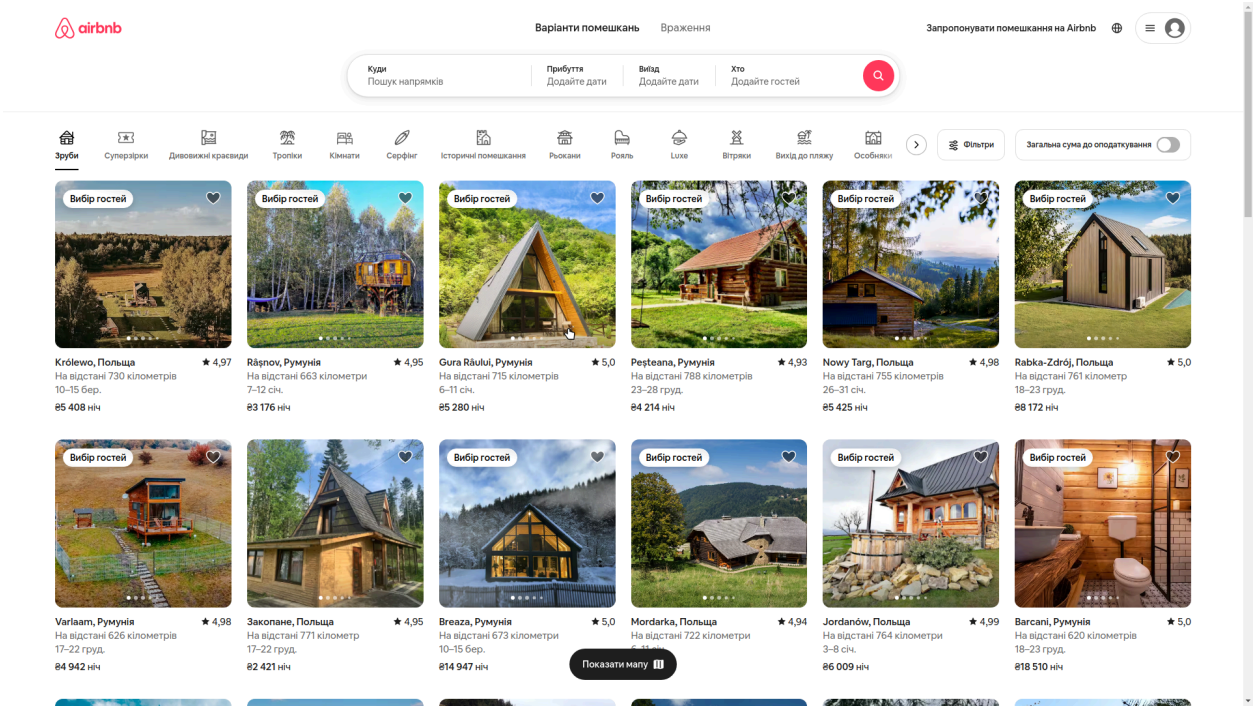

Рисунок 1.1 – Головна сторінка сайту Airbnb

Головні особливості UI/UX дизайну Airbnb:

- Візуальний акцент на зображеннях: Відразу на головній сторінці представлено великий вибір варіантів житла з високоякісними фото. Це сприяє швидкому залученню уваги користувача.
- Фільтри та персоналізація: У верхній частині сторінки знаходиться панель фільтрів (дата, місце, кількість гостей), що дозволяє швидко звузити пошук.
- Категоризація об'єктів: Є іконки "Зруби", "Суперрезиденції", "Тропіки" та інші категорії, що роблять навігацію простою.
- Мінімалістичний дизайн: Велика кількість білого простору підкреслює елементи інтерфейсу та зменшує когнітивне навантаження користувача.
- Заклики до дії (CTA): "Вибір гостей" під кожним варіантом житла мотивує до негайного перегляду.

Висновок для Airbnb: платформа використовує емоційний дизайн, орієнтований на візуальне сприйняття. Користувачі отримують естетичний досвід, що стимулює їх до оренди житла. Сильна сторона – легка навігація та персоналізація пошуку.

Zillow спеціалізується на довгостроковому продажу та купівлі житла з додатковими сервісами для агентів та позичальників. Головна сторінка зосереджена на наданні ключової інформації про доступні об'єкти та фінансові інструменти (див. рис. 1.2).

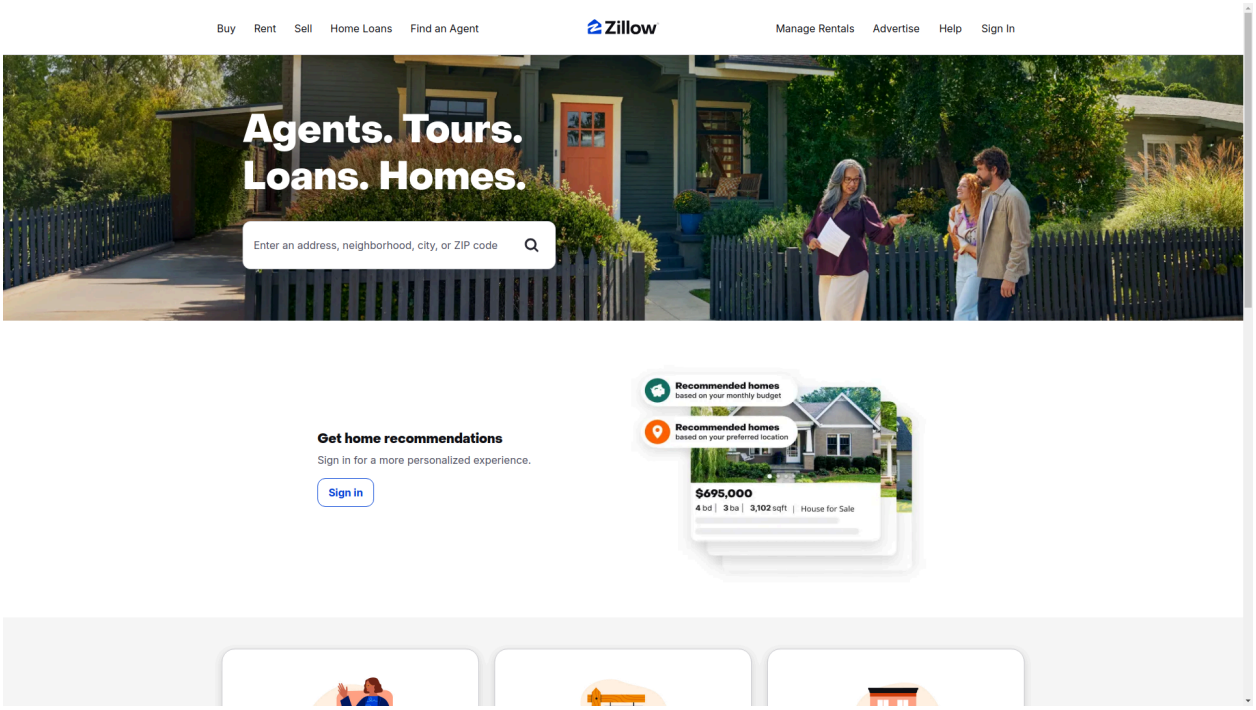

Рисунок 1.2 – Головна сторінка сайту Zillow

Головні особливості UI/UX дизайну Zillow:

- Фокус на функціональність: Zillow пропонує головну сторінку з акцентом на пошук та інформаційну доступність. Панель пошуку розташована у центрі й домінує в інтерфейсі.
- Інтерактивна аналітика: Платформа надає прогнозовані пропозиції житла на основі бюджету користувача та локації.
- Структуровані CTA: Кнопки «Sign In», «Find an Agent» організовані логічно та розташовані вгорі сторінки.

- Інформаційний блок: Під пошуком присутні рекламні картки про фінансові послуги (кредити) та агенти, що підкреслює Zillow як "екосистему для нерухомості".
- Фонові зображення: Головний банер із житлом виглядає професійно, створюючи відчуття довіри.

Висновок для Zillow: Платформа орієнтована на функціональність та інформативність. Користувачі отримують конкретні інструменти для пошуку нерухомості та фінансових рішень.

Rieltor.ua та Flatfy є популярними платформами на українському ринку для пошуку, оренди та купівлі нерухомості.

Rieltor.ua [10] орієнтована на класичний функціональний підхід до пошуку нерухомості, пропонуючи чітку структуру та акцент на текстові фільтри (див. рис. 1.3).

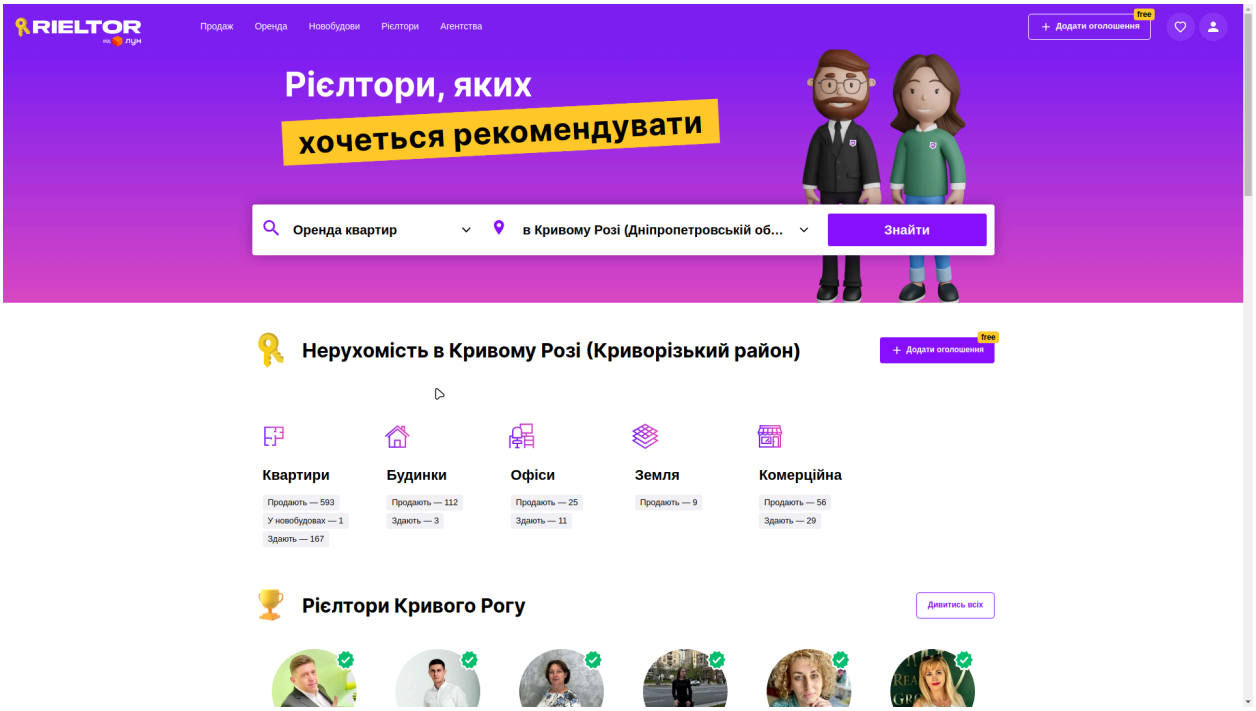

Рисунок 1.3 – Головна сторінка сайту Rieltor.ua

Головні особливості UI/UX дизайну Rieltor.ua:

- Функціональний мінімалізм: Головна сторінка зосереджена на текстовому полі для пошуку нерухомості, що дозволяє швидко розпочати користування платформою.
- Фільтри та структурованість: Панель фільтрів розташована у верхній частині сторінки.
- Доступні параметри, такі як локація, ціна, площа, тип нерухомості, є чіткими та зрозумілими.
- Інформаційні блоки: Додаткові секції для популярних міст, пропозицій для оренди/продажу, а також партнерські інструменти.
- Відсутність візуальної складової: Платформа мінімізує використання фото на головній сторінці, фокусуючись на функціональності.
- CTA-елементи: Основний заклик до дії зосереджено навколо кнопок "Знайти" для пошуку об'єктів.

Висновок для Rieltor.ua: Платформа обирає класичний підхід UX-дизайну із фокусом на текстову інформацію та структуровані фільтри. Це забезпечує ефективність для досвідчених користувачів, але може відлякати новачків через недостатню візуальну привабливість.

Flatfy [11] фокусується на сучасному дизайні з перевагою візуальної привабливості та інтерактивного досвіду користувача (див. рис. 1.4).

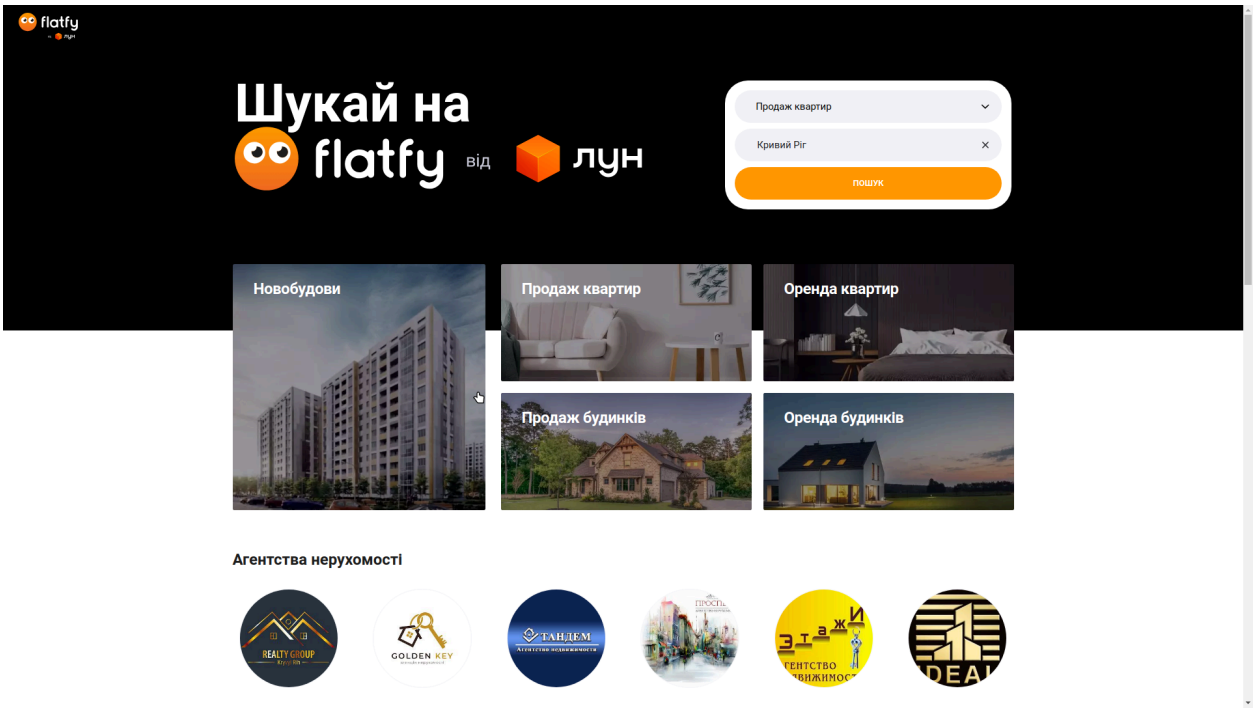

Рисунок 1.4 – Головна сторінка сайту Flatfy

Головні особливості UI/UX дизайну Flatfy:

- Візуальний акцент: Flatfy використовує великі зображення та інтерактивні банери для демонстрації популярних об'єктів та регіонів. Це допомагає користувачеві зануритись у візуальний контент.
- Інтерактивна пошукова панель: Головна сторінка містить велику панель пошуку з контекстними підказками. Ця функція полегшує навігацію для нових користувачів.
- Розділи категорій: Flatfy пропонує розділи "Оренда", "Продаж", а також фільтри за категоріями (новобудови, вторинний ринок, комерційна нерухомість).
- Візуалізація даних: Платформа надає інтерактивні картки з фото, ціною та основними характеристиками об'єктів, що сприяє швидкому скануванню інформації.

- Сучасний дизайн: Використання білого простору, м'яких кольорів та зручного шрифту створює естетично приємний досвід.

Висновок для Flatfy: Flatfy орієнтується на візуальну привабливість і зручність для нових користувачів. Платформа пропонує емоційний досвід, інтегруючи сучасні UI/UX тренди для швидкого перегляду та вибору житла.

Усі успішні платформи на ринку нерухомості, такі як Airbnb, Zillow, Flatfy та Rieltor.ua, мають спільну мету — спростити процес пошуку нерухомості для кінцевих користувачів і підвищити ефективність бізнес-процесів. Вони забезпечують інтуїтивно зрозумілий інтерфейс та персоналізований досвід, що зменшує когнітивне навантаження на користувача. Основною точкою взаємодії є зручна пошукова система з динамічними фільтрами, що дозволяє швидко налаштувати критерії пошуку, такі як тип нерухомості, ціна, локація та інші важливі параметри.

Важливим фактором є візуальна привабливість — високоякісні фотографії, інтерактивні галереї та відеопрезентації об'єктів створюють емоційний зв'язок і підвищують довіру користувачів до платформи. Багато сервісів успішно поєднують естетику з функціональністю, забезпечуючи баланс між красивим дизайном і інформативністю контенту. Наприклад, Airbnb робить акцент на емоційному дизайні через візуальні елементи, тоді як Zillow зосереджується на аналітичних інструментах і глибокій інтеграції даних.

Окрім базового функціоналу, платформи активно впроваджують інструменти аналітики та інтерактивні карти. Ці рішення дозволяють користувачам оцінювати ринок, вивчати ціноутворення, аналізувати історію змін вартості об'єктів та прогнозувати можливі тенденції. Інтерактивні мапи, які прив'язують нерухомість до геолокації та інфраструктурних особливостей, стають стандартом для ефективної оцінки привабливості локацій.

Попри численні переваги, аналізовані платформи мають і спільні проблеми. Зокрема, відсутність відкритої аналітики для власників бізнесу та

агентів створює обмеження для ефективного використання даних. Багато сервісів також є замкнутими екосистемами, що підвищує витрати на розробку та підтримку.

Таким чином, для створення конкурентоспроможного веб-додатку для ринку нерухомості необхідно поєднати візуальну привабливість у стилі Airbnb з аналітичними можливостями та функціональністю Zillow. Це дозволить розробити рішення, орієнтоване як на кінцевих користувачів, так і на професіоналів галузі. Додаток має включати інтерактивну мапу, інструменти аналітики для прогнозування ринку, динамічний пошук із фільтрами та прозору взаємодію між покупцями, продавцями та агентами.

1.5. Елементи економічного аналізу при впровадженні веб-застосунку.

Економічний аналіз є невіддільною складовою впровадження веб-застосунків у бізнес-середовищі, оскільки він дозволяє обґрунтувати фінансову доцільність, оцінити витрати та спрогнозувати потенційні прибутки. Основними елементами економічного аналізу при розробці та впровадженні веб-застосунку є прямі витрати, непрямі витрати, потенційні доходи та окупність інвестицій. До прямих витрат відноситься розробка функціоналу застосунку, створення дизайну та інфраструктурних компонентів, таких як сервери, бази даних і хмарні сервіси. Витрати на інфраструктуру включають не лише початкову розробку, але й подальшу підтримку, масштабування та забезпечення безпеки даних. Регулярні оновлення веб-застосунку, що відповідають змінам у вимогах користувачів чи технологічному середовищі, є невід'ємною частиною експлуатаційних витрат.

Непрямі витрати пов'язані з інтеграцією веб-застосунку у поточні бізнес-процеси та навчанням персоналу для його ефективного використання. Наприклад, впровадження нового застосунку в організації, де вже використовуються ERP або CRM-системи, потребує додаткових ресурсів на забезпечення інтеграції та перевірку сумісності. Крім того, слід враховувати

витрати на маркетинг і просування веб-застосунку, особливо якщо він орієнтований на зовнішніх клієнтів. Ефективна маркетингова стратегія, спрямована на залучення нових користувачів, є критично важливою для досягнення швидкого економічного ефекту.

Потенційні доходи від впровадження веб-застосунку досягаються завдяки автоматизації бізнес-процесів, підвищенню продуктивності працівників і покращенню клієнтського досвіду. Автоматизовані рішення дозволяють мінімізувати ручну працю та час на обробку операцій, що позитивно впливає на загальну ефективність компанії. Наприклад, веб-застосунок для ринку нерухомості може оптимізувати взаємодію агентів з клієнтами, надаючи зручні інструменти для пошуку, оцінки та обговорення об'єктів. Покращена комунікація зі споживачами також сприяє підвищенню лояльності та утриманню клієнтів, що напряму впливає на прибуток.

Важливим етапом економічного аналізу є розрахунок окупності інвестицій. Окупність інвестицій дозволяє оцінити часовий період, протягом якого витрати на розробку та впровадження веб-застосунку компенсуються доходами від його використання. За допомогою сценарного моделювання можна передбачити різні варіанти розвитку бізнесу залежно від масштабів підприємства та інтенсивності використання веб-застосунку. Наприклад, для малого бізнесу ефективність впровадження може бути швидшою завдяки гнучким процесам та меншій кількості користувачів, тоді як для великих організацій цей процес може бути більш тривалим через необхідність глибшої інтеграції та масштабування.

Серед потенційних ризиків слід враховувати конкурентне середовище, можливість низької активності користувачів або технічні обмеження, що можуть зменшити очікувану економічну вигоду. Водночас, ретельний аналіз витрат і доходів, а також оцінка ключових показників ефективності (KPI), таких як час обробки запитів, рівень задоволеності користувачів і кількість транзакцій, дозволяє мінімізувати ці ризики. Впровадження веб-застосунку, що відповідає сучасним вимогам технологічності та функціональності,

забезпечує не лише оперативну ефективність, але й стратегічний розвиток бізнесу, роблячи його конкурентоспроможним на ринку.

1.6. Висновки з теоретичного аналізу та реалізації веб-застосунку для ринку нерухомості.

За результатами проведеного дослідження встановлено, що веб-застосунки відіграють ключову роль у цифровій трансформації ринку нерухомості, зокрема на вторинному ринку житла. Вони забезпечують ефективну взаємодію між продавцями та покупцями, оптимізують процеси пошуку та продажу об’єктів і сприяють підвищенню якості обслуговування. Особливо актуальною є їхня роль у контексті сучасних викликів, зокрема зростання попиту на швидкі та надійні платформи через значні соціально-економічні потрясіння.

Ефективність реалізації веб-застосунку значною мірою залежить від вибору технологічного стеку та архітектури системи. Використання сучасних інструментів, таких як React, NextJS, NodeJS, ExpressJS, MongoDB, Redis та Docker забезпечує високу продуктивність, масштабованість та зручність використання. При цьому особлива увага приділяється користувацькому досвіду (UX), оскільки він є вирішальним фактором для залучення і утримання клієнтів.

Одним із ключових викликів у процесі аналізу стала закритість даних про ефективність існуючих платформ та їх вплив на ринкові показники. У такому середовищі необхідно орієнтуватися на інструменти аналітики, які фокусуються на ключових метриках: кількість користувачів, активність у додатку, конверсія пошуку та ефективність рекламних кампаній.

Впровадження економічного аналізу є важливим етапом розробки веб-додатку. Він дозволяє оцінити прямі й непрямі витрати на розробку, а також спрогнозувати дохідність проєкту. Інструменти аналітики допомагають визначити найбільш ефективні аспекти використання веб-застосунку для підвищення прибутковості бізнесу.

Таким чином, розробка конкурентоспроможного веб-застосунку для ринку нерухомості вимагає комплексного підходу, що поєднує інноваційні технології, продуманий UX-дизайн та економічну ефективність. Запропоноване рішення здатне забезпечити значне підвищення операційної ефективності, задовольнити зростаючі потреби ринку та сприяти подальшій цифровій трансформації індустрії нерухомості.

2. МОДЕЛЮВАННЯ СИСТЕМИ ТА АЛГОРИТМІВ, СХЕМ ЗАСТОСУНКУ ТА РЕДИЗАЙН МАКЕТІВ ІНТЕРФЕЙСУ

2.1. Створення математичної моделі для аналізу економічної вигоди.

Для оцінки економічної ефективності впровадження веб-застосунку в ринку нерухомості необхідно розробити математичну модель, що ґрунтується на зібраних даних. Основною метою моделі є визначення рівня економічної вигоди, отриманої від використання системи, порівнюючи доходи та витрати, пов’язані з її розробкою та експлуатацією.

Економічна ефективність впровадження веб-застосунку може бути обчислена за допомогою відношення чистої вигоди до загальних витрат. Загальна математична формула для оцінки ефективності має вигляд:

$$E = \frac{D-C}{C} \times 100\%, \tag{2.1}$$

де D – сукупний дохід, отриманий завдяки використанню веб-застосунку за певний період, C – загальні витрати на розробку, впровадження та обслуговування системи у тому ж періоді;

$$D = \sum_{i=1}^{n} (P_i \times T_i), \tag{2.2}$$

де P_i – середній дохід від однієї транзакції чи оголошення, T_i – кількість транзакцій або розміщених оголошень за аналізований період, n – кількість категорій доходів;

$$C = C_f + C_v, \tag{2.3}$$

де C_f – витрати на початкову розробку системи, C_v – операційні витрати на утримання та розвиток веб-застосунку протягом аналізованого періоду.

2.2. Моделювання взаємодій між користувачем та системою.

Щоб забезпечити структурований та деталізований аналіз взаємодії користувачів із веб-застосунком для купівлі та продажу нерухомості, було застосовано діаграму IDEF0. Ця діаграма є ефективним інструментом для моделювання функціональних взаємодій системи з її користувачами, що дозволяє чітко візуалізувати основні сценарії використання. Вона не лише відображає потоки даних і управління між компонентами системи, але й забезпечує їх декомпозицію на рівні окремих функцій і підфункцій. Завдяки цьому підходу вдається провести глибокий аналіз ключових операцій, визначити оптимальні шляхи їх виконання та створити основу для подальшої розробки інтерфейсу користувача, орієнтованого на конкретні потреби (див. рис. 2.1).

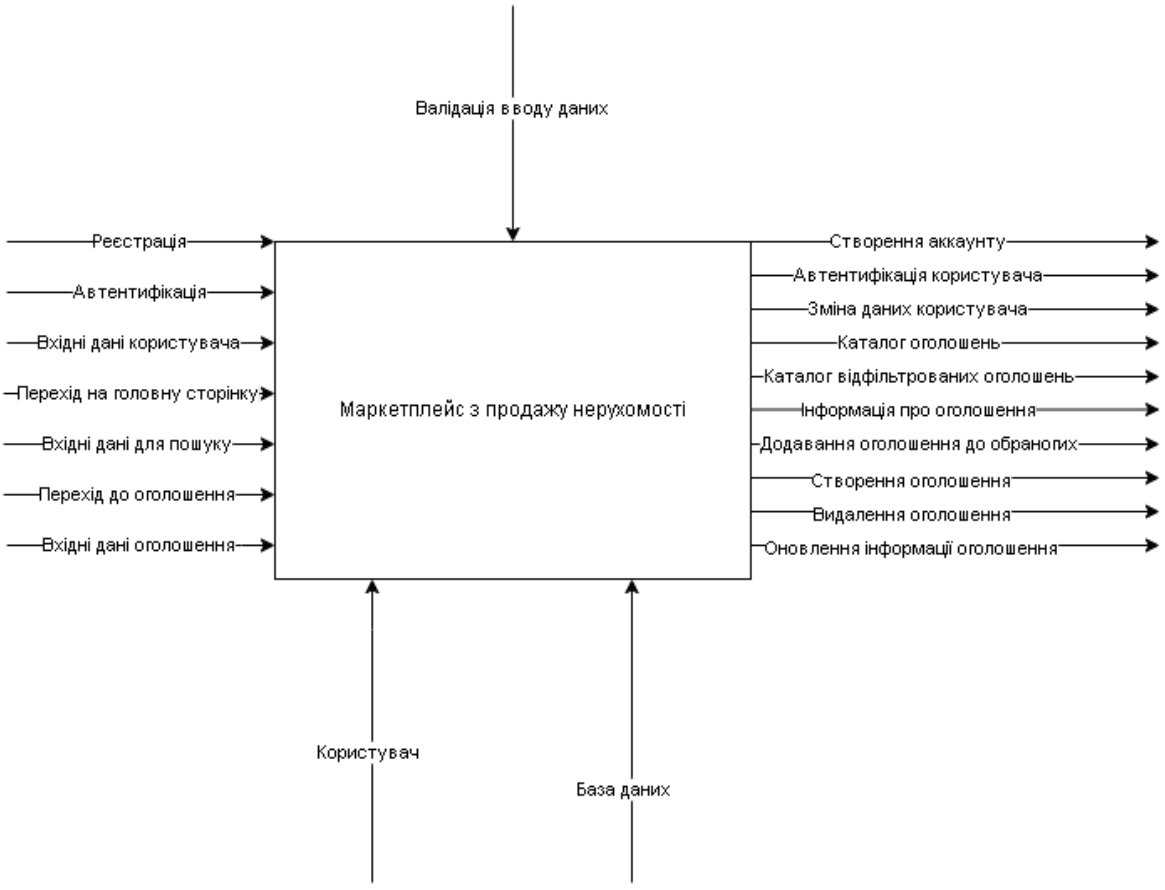

Рисунок 2.1 – Функціональна IDEF0-діаграма веб-застосунку

Завдяки цьому стало можливим відстежити ключові функції, які забезпечують користувацькі операції, та розробити інтуїтивно зрозумілий інтерфейс користувача.

2.3. Розробка структури бази даних.

Розробка структури бази даних є одним із ключових етапів, що забезпечують ефективне управління та зберігання даних у веб-застосунку. Комплексний аналіз функціональних вимог дозволив визначити основні сутності, їхні атрибути та зв'язки, що забезпечують оптимальну організацію інформації у системі. У контексті веб-додатку для ринку нерухомості, архітектура бази даних передбачає створення кількох взаємопов’язаних таблиць, орієнтованих на управління даними користувачів, оголошень про нерухомість та додаткової інформації, необхідної для ефективного функціонування платформи (див. рис. 2.2).

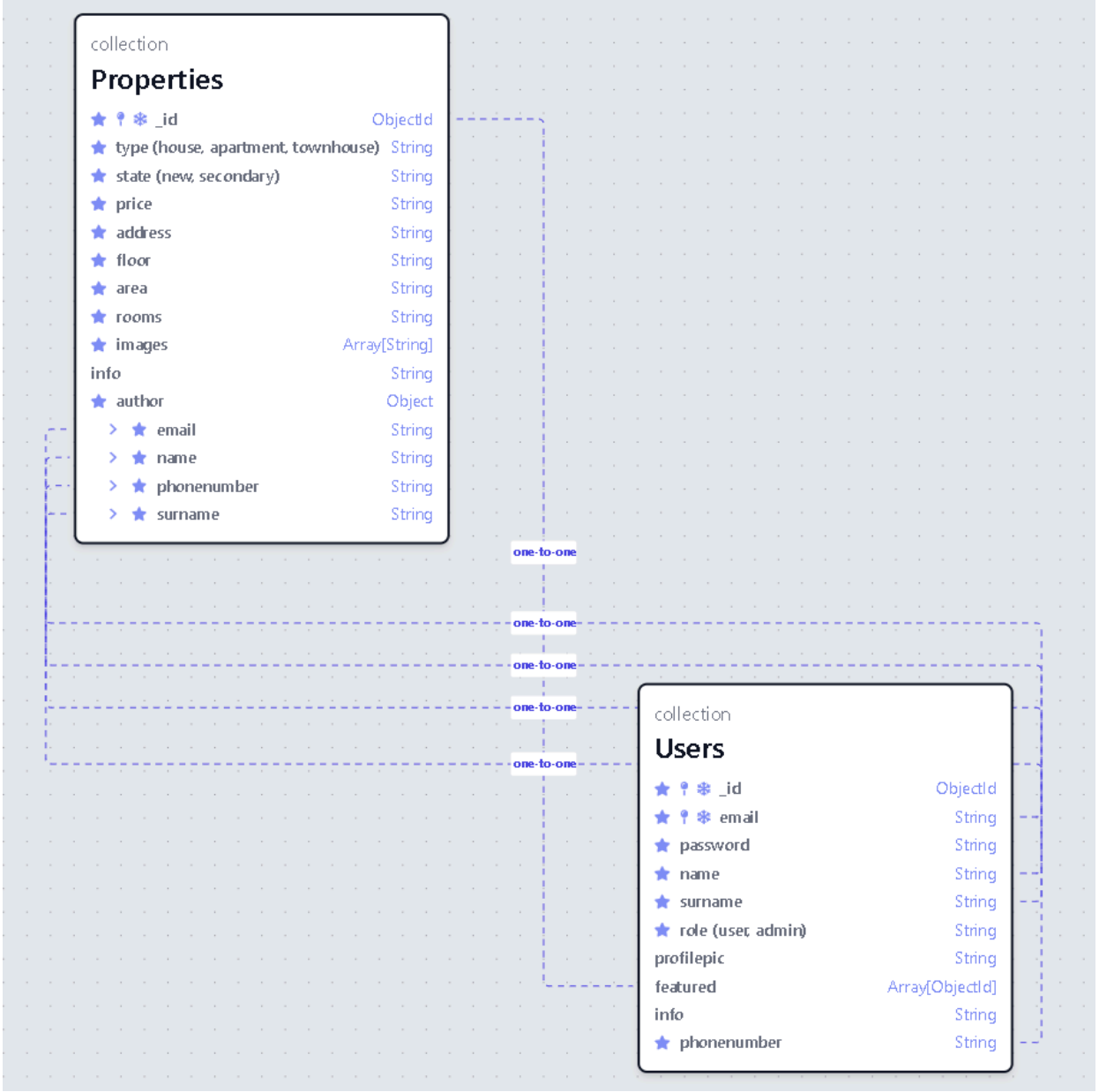

Рисунок 2.2 – ER-діаграма структури бази даних

Таблиця Users зберігає дані користувачів, необхідні для автентифікації та подальшої взаємодії із системою.

Основні поля включають:

- Email та password – облікові дані для входу до системи, забезпечуючи безпечний доступ.
- Name та surname – персональна інформація користувача.

- Phonenumber – контактний номер для відображення в оголошеннях.
- Profilepic – зображення профілю користувача.
- Roles – визначає рівень доступу та дозволи.
- Featured – позначає вибрані оголошення користувача.
- Info – додаткове поле для зберігання необхідних деталей.

Таблиця Properties слугує сховищем даних для оголошень з продажу нерухомості.

Основні поля:

- Type – категорія об'єкта нерухомості.
- State – поточний стан нерухомості. price – вартість об'єкта.
- Address – адреса, що сприяє геопросторовому аналізу.
- Floor – поверх розташування об'єкта.
- Area – площа нерухомості, важлива для оцінки її просторових характеристик.
- Rooms – кількість кімнат.
- Images – масив фотографій об'єкта.
- Author – посилання на користувача з таблиці Users, що створив оголошення.
- Info – додаткові дані про нерухомість.

2.4. Створення блок-схем алгоритмів веб-застосунку.

Розробка блок-схем для алгоритмів веб-застосунку є критичним етапом проектування системи. Блок-схеми забезпечують візуальне представлення логічного потоку операцій і дозволяють проаналізувати кожен етап функціонування. У контексті веб-застосунку для ринку нерухомості було розроблено кілька ключових блок-схем, що охоплюють різні аспекти користувацької взаємодії, такі як реєстрація, автентифікація, пошук об'єктів та створення оголошень.

Процес реєстрації користувача включає перевірку введених даних, їх валідацію, шифрування та збереження у базі даних. У разі помилок користувачеві пропонується їх виправити, а успішне проходження перевірки завершується створенням облікового запису у системі разом з зашифрованими чутливими даними користувача (див. рис. 2.3).

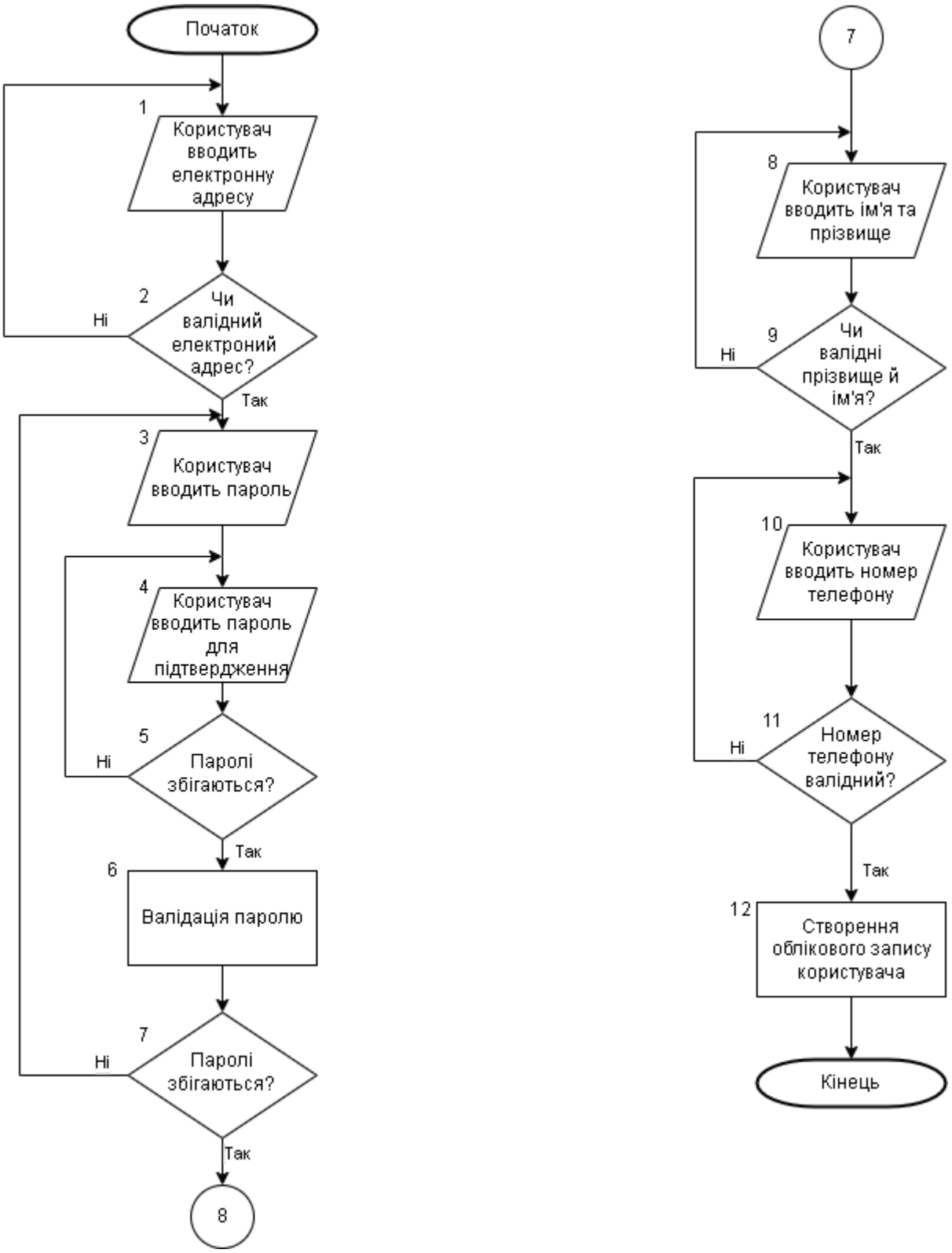

Рисунок 2.3 – Блок-схема алгоритму реєстрації

Розробка блок-схеми алгоритму є фундаментальним етапом проєктування програмних систем, оскільки вона дозволяє чітко візуалізувати

логічну послідовність операцій, що забезпечують роботу веб-застосунку. У контексті розробки платформи для купівлі та продажу нерухомості блок-схема відображає повний цикл використання системи користувачем. Це включає початковий доступ до веб-додатку, автентифікацію, взаємодію з основними функціональними модулями, обробку даних та завершення сеансу роботи. Блок-схема відіграє важливу роль у моделюванні процесів, що забезпечують плавну та інтуїтивну взаємодію користувача із системою, дозволяючи ідентифікувати можливі вузькі місця та оптимізувати структуру логіки додатку.

Алгоритм використання веб-застосунку розпочинається з ініціалізації процесу, коли користувач відкриває головну сторінку платформи. На цьому етапі відображаються основні функціональні компоненти системи, які надають доступ до ключових можливостей, таких як пошук нерухомості, перегляд оголошень, авторизація та реєстрація, а також інші розділи додатку. Якщо користувач уже має обліковий запис, система пропонує виконати автентифікацію шляхом введення облікових даних. У разі успішної перевірки введених даних система надає користувачу доступ до функціональності платформи відповідно до його ролі та рівня дозволів. Нові користувачі, які не мають облікового запису, проходять процес реєстрації, під час якого надають необхідну інформацію, таку як ім’я, прізвище, електронна пошта, пароль та номер телефону. Дані проходять обов’язкову валідацію перед збереженням у базі даних системи. Після реєстрації користувачу надається можливість увійти до свого профілю для подальшої роботи з додатком.

Після автентифікації користувач отримує доступ до функціоналу головної сторінки, де він може скористатися можливостями пошуку нерухомості, переглядати рекомендовані оголошення, додавати власні об’єкти для продажу чи оренди або керувати особистим профілем. Процес пошуку реалізується шляхом введення або вибору фільтрів, що уточнюють параметри пошуку, такі як категорія нерухомості, її стан, площа, кількість кімнат, діапазон цін та розташування. Система обробляє запит і повертає результати

на основі даних, збережених у таблиці Properties. При виборі конкретного оголошення користувач переходить на сторінку детального перегляду, де відображається інформація про об’єкт, включно з описом, зображеннями, ціною, адресою та контактними даними продавця.

Для користувачів, які бажають розмістити оголошення, система пропонує інтерфейс створення нового запису про нерухомість. У процесі створення оголошення користувач вводить ключові параметри об’єкта, включаючи тип нерухомості, площу, ціну, адресу, стан, кількість кімнат, а також завантажує фотографії. Дані проходять перевірку на коректність і повноту, після чого зберігаються у базі даних із прив’язкою до профілю користувача. Надалі користувач отримує можливість редагувати, видаляти або переглядати створені оголошення через особистий кабінет.

Крім цього, блок-схема передбачає реалізацію функції управління особистим профілем користувача, що дозволяє змінювати персональні дані, оновлювати зображення профілю та переглядати збережені оголошення. Для забезпечення безперебійної навігації система реалізує логічний перехід між різними розділами додатку, такими як список нерухомості, обране або сторінка управління власними оголошеннями. У випадку завершення роботи з платформою користувач має можливість вийти з облікового запису, що завершує поточний сеанс і забезпечує безпеку даних.

Представлена блок-схема наочно демонструє послідовність кроків, які проходить користувач від моменту першого входу до завершення роботи із системою. Вона охоплює ключові етапи, такі як реєстрація та автентифікація, пошук і перегляд об'єктів, створення нових оголошень та управління ними, а також налаштування профілю користувача. Завдяки такій структурі забезпечується логічна узгодженість процесів, що є критично важливим для оптимізації користувацького досвіду та ефективної реалізації функціональних можливостей веб-додатку. Блок-схема служить важливим інструментом для розробників системи, дозволяючи виявити потенційні вузькі місця та

забезпечити ефективну інтеграцію усіх компонентів системи на етапах її проєктування та впровадження (див. рис. 2.4).

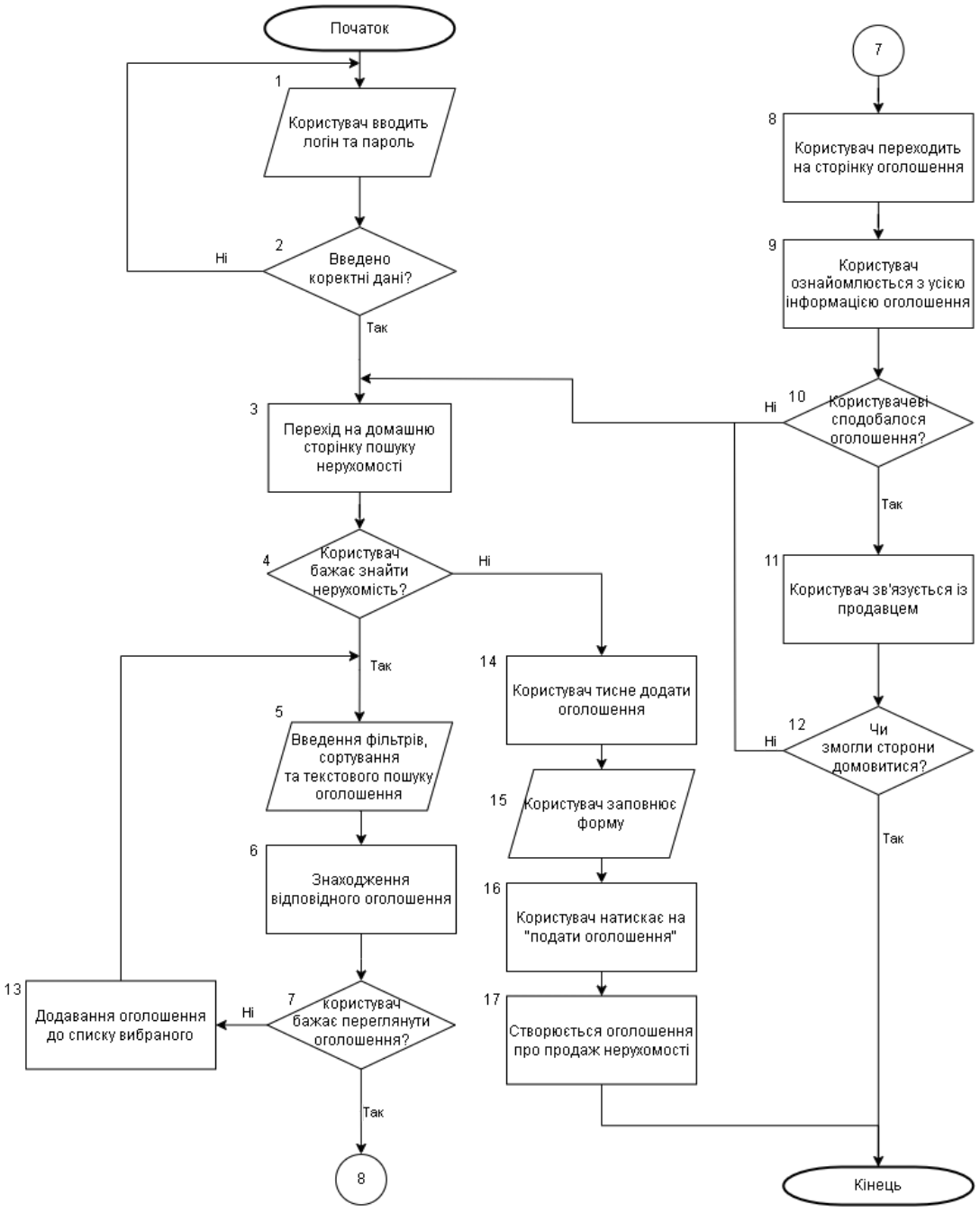

Рисунок 2.4 – Блок-схема алгоритму повного циклу використання веб-застосунку

2.5. Розробка інтерфейсу веб-застосунку.

Розробка графічного інтерфейсу (GUI) веб-застосунку для купівлі та продажу нерухомості здійснювалася із застосуванням принципів Google Material Design, що забезпечило інтуїтивно зрозумілий і адаптивний користувацький досвід. Усі компоненти, які раніше були створені за допомогою кастомного React-коду, були мігровані на сучасну компонентну базу Material-UI (MUI) версії 6.2.0. Цей перехід дозволив стандартизувати дизайн і покращити його масштабованість, забезпечивши відповідність найсучаснішим UX/UI-практикам. Попри збереження загальної структури інтерфейсу, кожен елемент було оновлено для гармонійної інтеграції з бібліотекою MUI та покращення візуальної узгодженості.

Головна сторінка залишилася центральним вузлом взаємодії користувачів із платформою, але отримала нові візуальні й функціональні вдосконалення. Панель пошуку була реалізована через компонент Autocomplete, що підтримує динамічні підказки для полегшення навігації серед великої кількості даних (див. рис. 2.5).

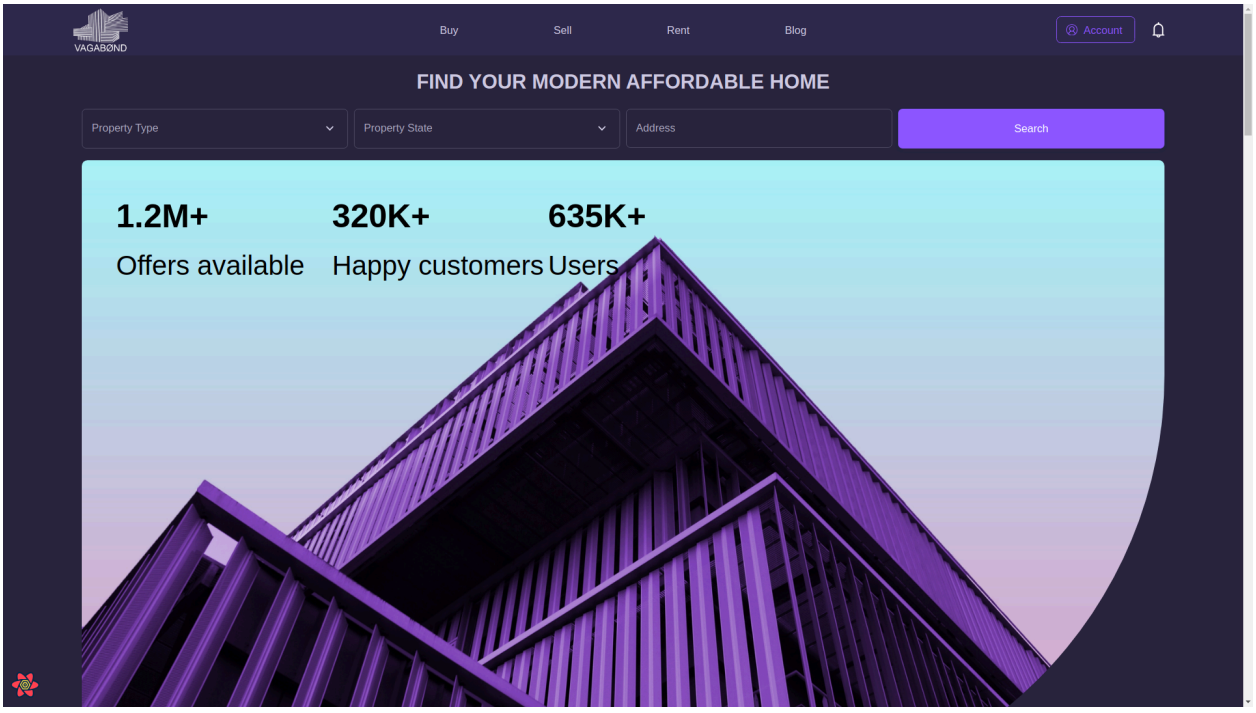

Рисунок 2.5 – Інтерфейс головної сторінки

Секція рекомендованих пропозицій, побудована за допомогою компонента Grid, адаптується до різних розмірів екранів, забезпечуючи зручність перегляду незалежно від пристрою (див. рис. 2.6).

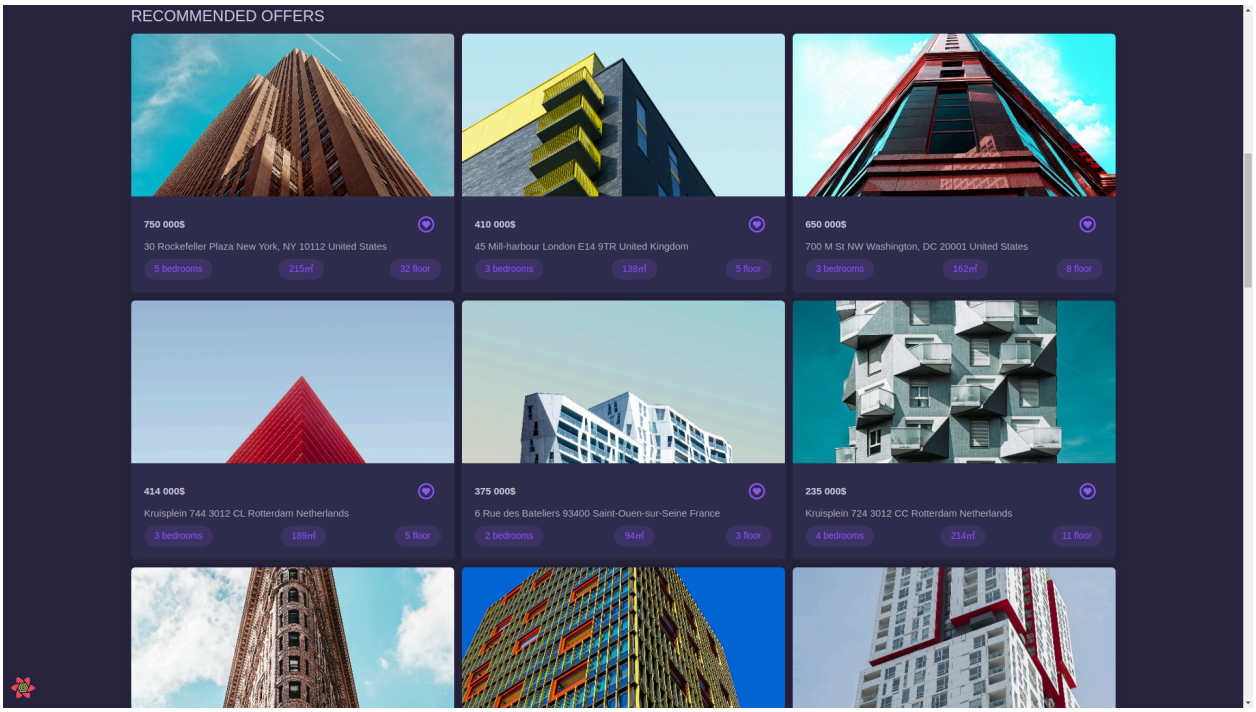

Рисунок 2.6 – Інтерфейс секції рекомендованих пропозицій

Розділ міні-блогу також було оновлено із застосуванням компонентів Card і Typography, що покращило структуру вмісту і зробило його більш читабельним для користувачів (див. рис. 2.7).

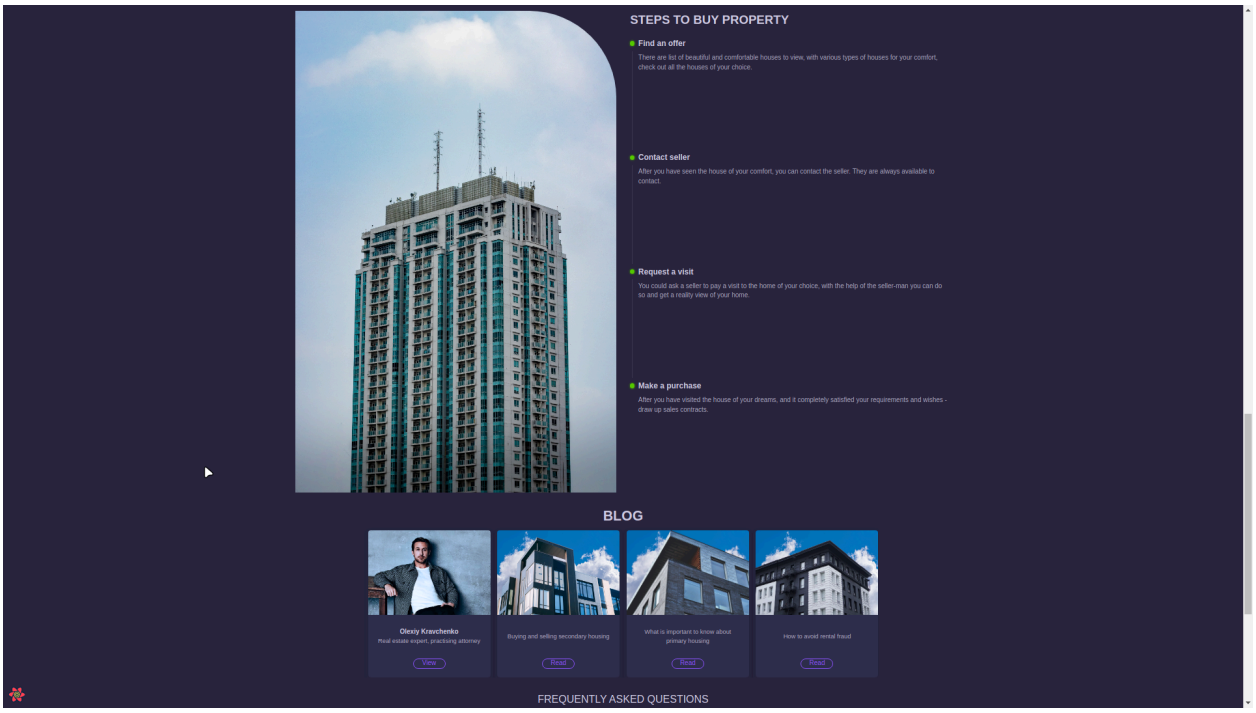

Рисунок 2.7 – Інтерфейс секції міні-блогу

Реалізація розділу FAQ здійснена за допомогою компонента Accordion, який дозволяє згрупувати запитання та відповіді в компактному та інтуїтивно зрозумілому форматі. Кожен пункт FAQ представлений у вигляді розкривного елементу, що забезпечує швидкий доступ до відповідної інформації без перевантаження інтерфейсу зайвими деталями. Завдяки адаптивності компонента, розділ коректно відображається на різних пристроях, гарантуючи зручність для користувачів як на мобільних, так і на десктопних платформах.

У кожному пункті FAQ застосовано такі підходи: запитання відображається великим шрифтом із високою контрастністю, що відповідає вимогам доступності, а відповідь структурована з використанням компонента Typography, забезпечуючи читабельність тексту. Для додаткового візуального розділення застосовано відступи між елементами, що створює чітку ієрархію контенту.

Цей розділ не лише покращує навігацію платформою, але й слугує важливим інструментом навчання нових користувачів, надаючи їм базову

інформацію про роботу системи, зокрема про створення оголошень, використання пошуку, налаштування профілю та інші аспекти. Завдяки цьому розділ FAQ сприяє підвищенню рівня задоволеності користувачів, допомагаючи їм швидко вирішувати проблеми та знаходити потрібну інформацію без зайвих зусиль (див. рис. 2.8).

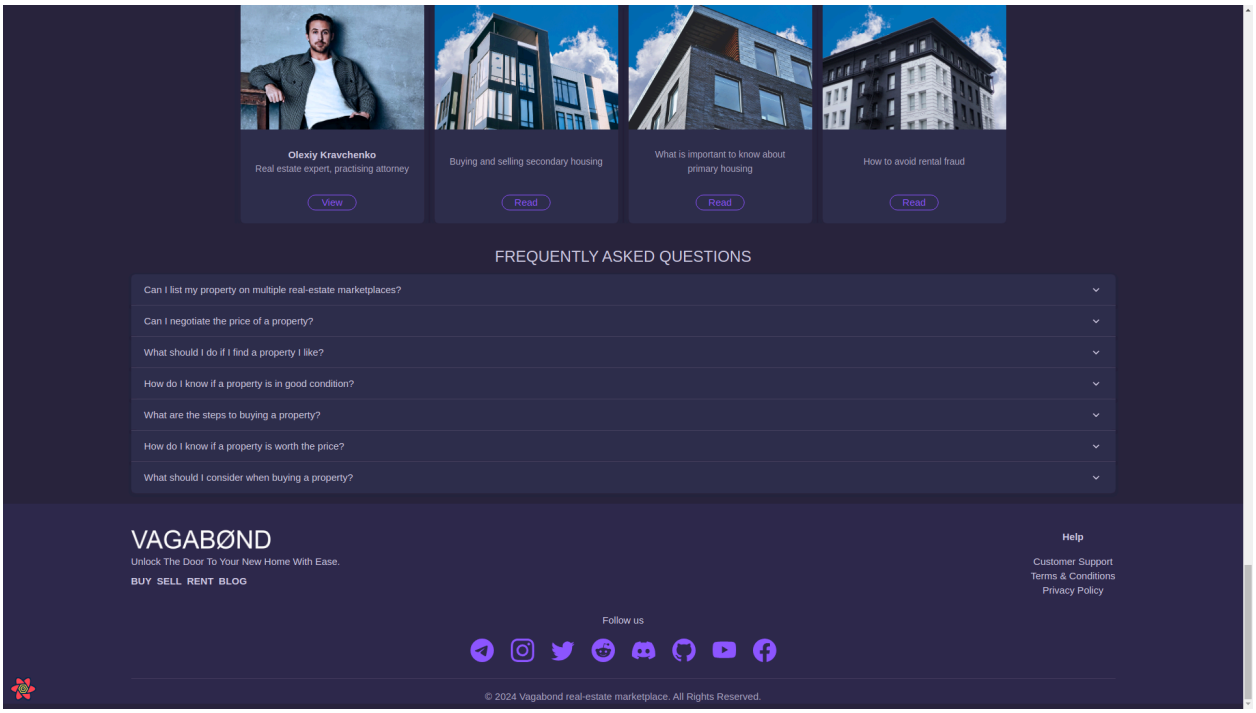

Рисунок 2.8 – Інтерфейс секції поширених запитань

Сторінки реєстрації та авторизації було значно вдосконалено за рахунок адаптації полів введення та кнопок до компонентів TextField і Button. Це забезпечило не лише візуальну узгодженість із загальним дизайном, але й підтримку динамічної валідації введених даних (див. рис. 2.9).

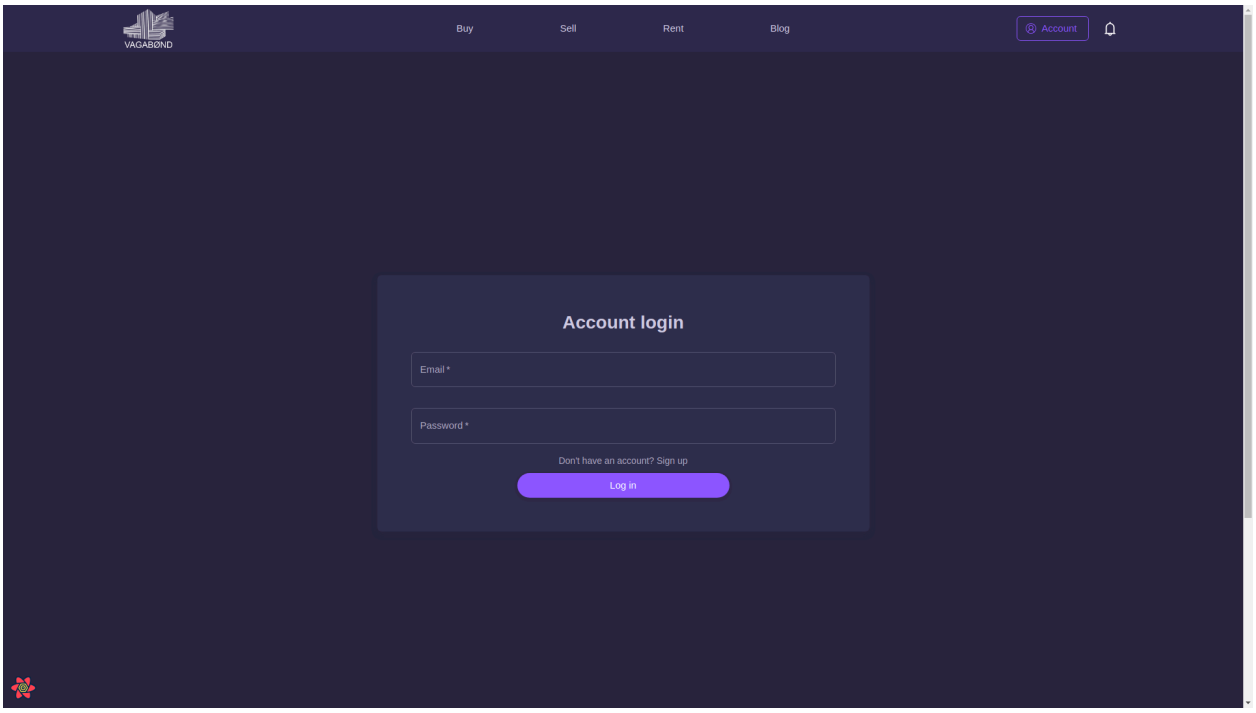

Рисунок 2.9 – Інтерфейс сторінки авторизації

Для перевірки коректності інформації використовувалася бібліотека Zod у поєднанні з React Hook Form, що дозволило спростити управління станом форм і зменшити навантаження на сервер. Повідомлення про помилки було реалізовано через компонент react-toastify, який дотримується рекомендацій Google Material Design (див. рис. 2.10).

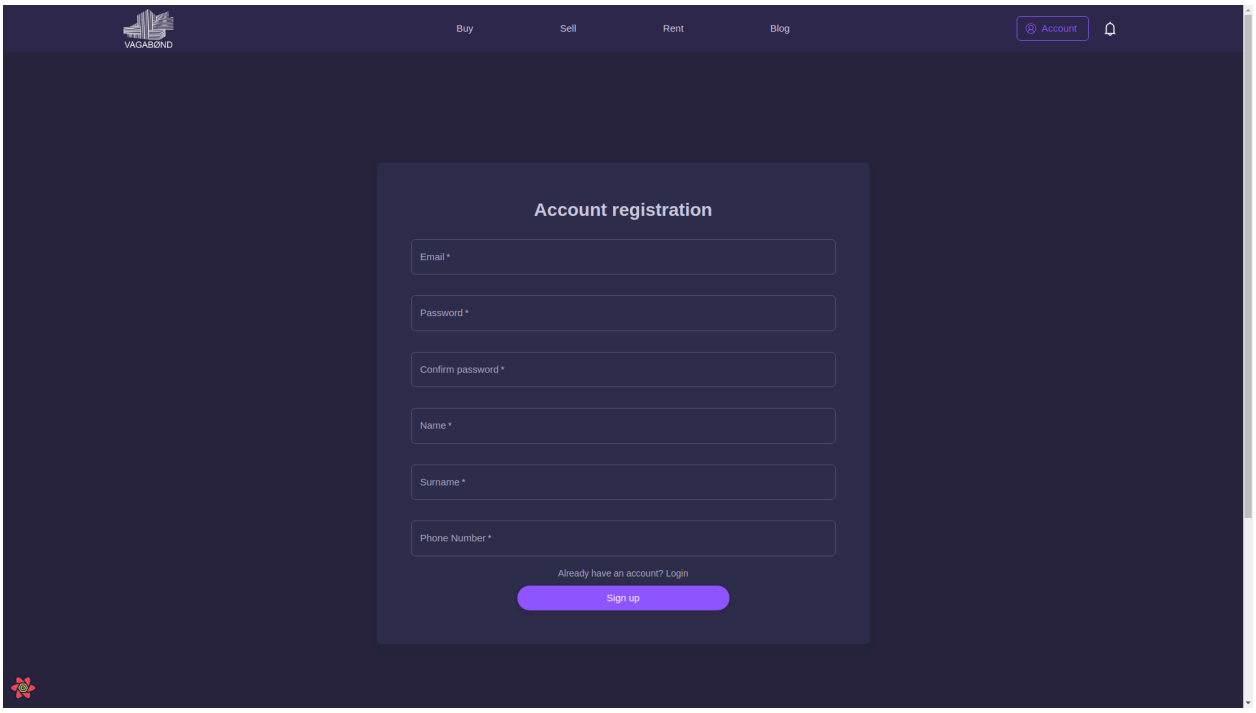

Рисунок 2.10 – Інтерфейс сторінки реєстрації

Інтерфейс профілю користувача було оновлено, щоб підвищити його зручність і відповідність сучасним вимогам. Основні секції профілю тепер представлені у вигляді структурованих блоків на базі компонента Paper, що підкреслює візуальний поділ між різними функціональними елементами. Зміна особистих даних, таких як ім'я, прізвище, контактний номер або фото профілю, здійснюється через компоненти FormControl і Avatar (див. рис. 2.11).

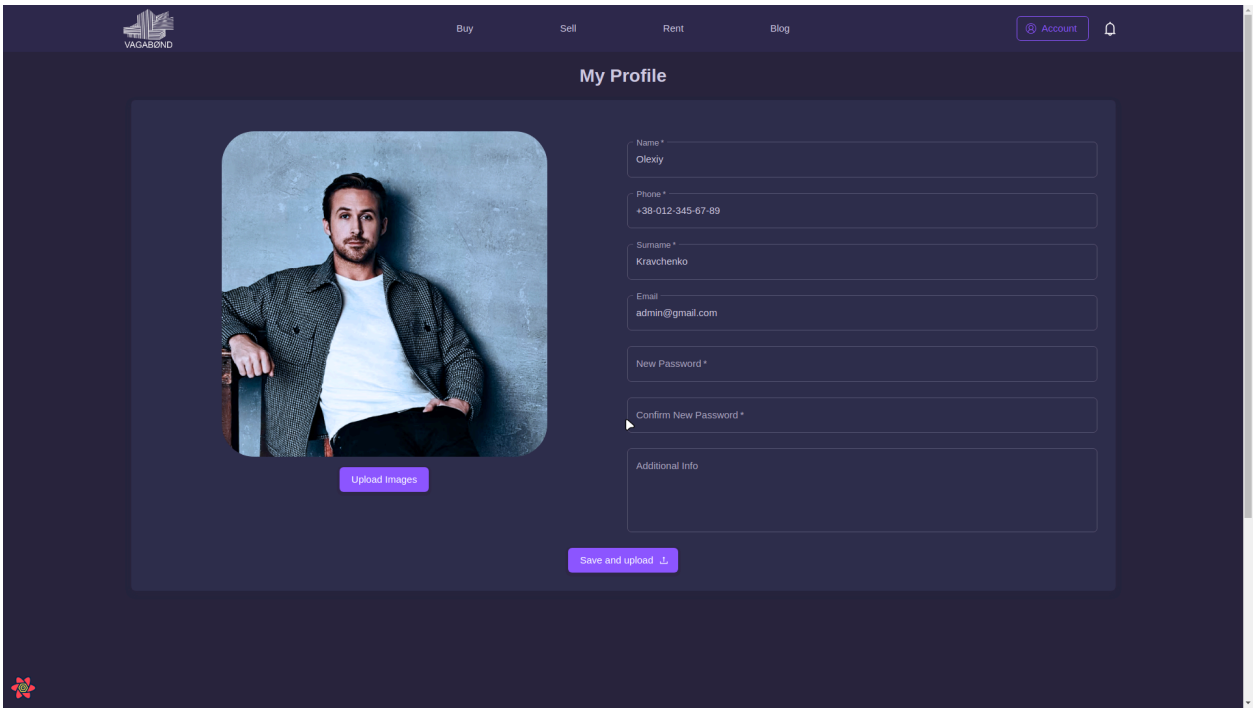

Рисунок 2.11 – Інтерфейс сторінки профілю користувача

Сторінка пошуку нерухомості була адаптована до нових вимог. Фільтри пошуку реалізовано за допомогою компонента Accordion, що зменшило когнітивне навантаження на користувачів, дозволивши згрупувати критерії за категоріями. Результати пошуку, представлені у вигляді адаптивних карток, створених за допомогою компонента Card, містять ключову інформацію, таку як ціна, площа, адреса й кількість кімнат. Усі ці компоненти повністю відповідають рекомендаціям Google Material Design і забезпечують узгоджений стиль платформи (див. рис. 2.12).

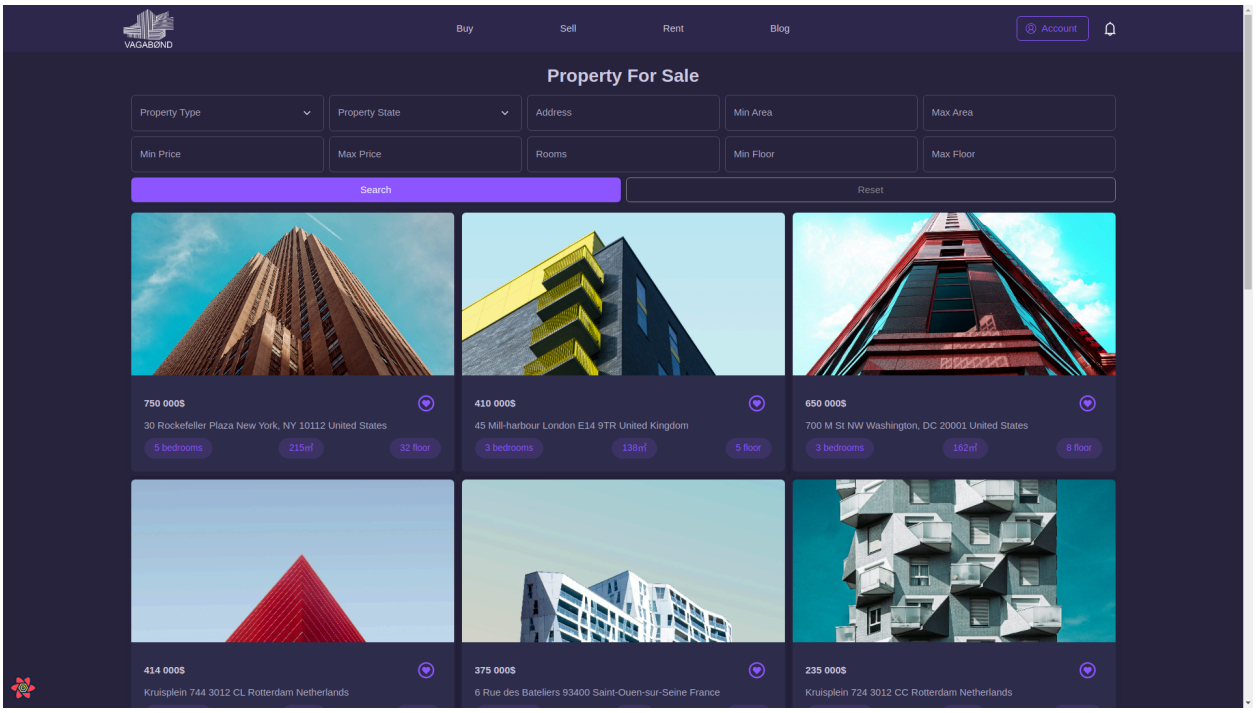

Рисунок 2.12 – Інтерфейс сторінки пошуку нерухомості

Сторінка детального перегляду нерухомості була покращена за рахунок інтеграції компонентів Table для відображення характеристик об'єкта та ImageList для візуалізації фотографій (див. рис. 2.13).

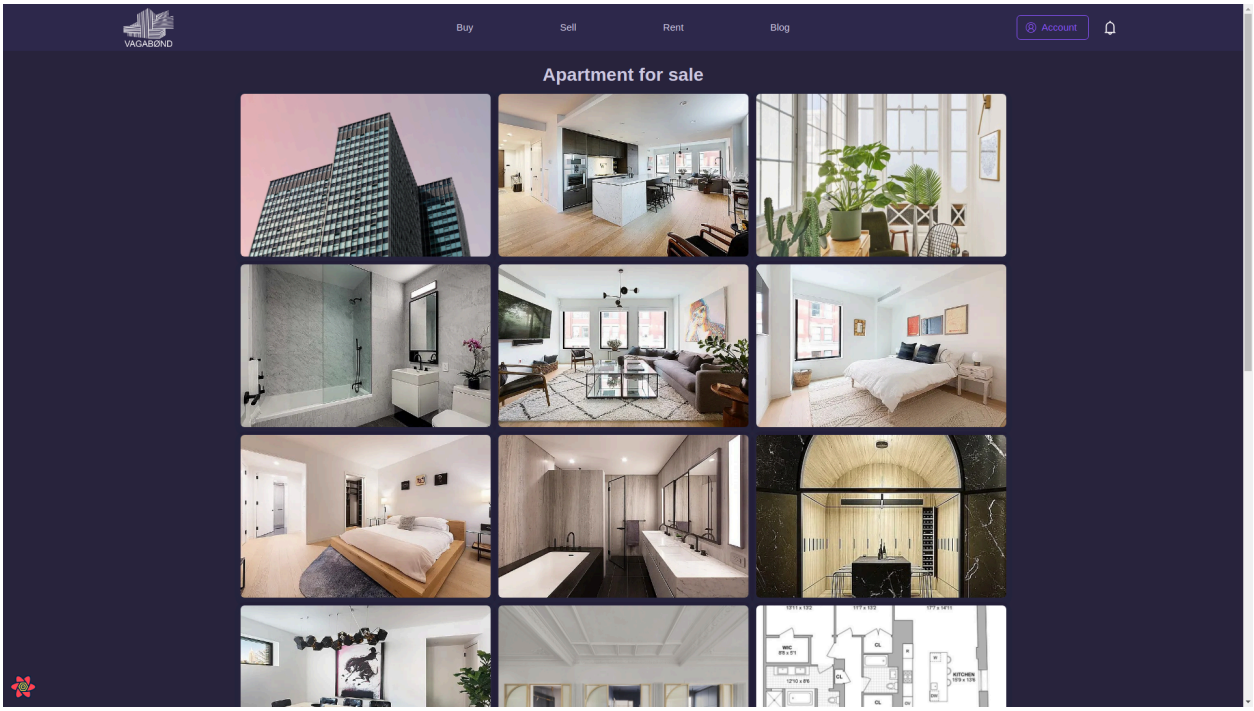

Рисунок 2.13 – Інтерфейс сторінки з інформацією про об'єкт

Уся інформація про об'єкт представлена у структурованій формі, що дозволяє користувачам швидко оцінювати об'єкти й приймати рішення (див. рис. 2.14).

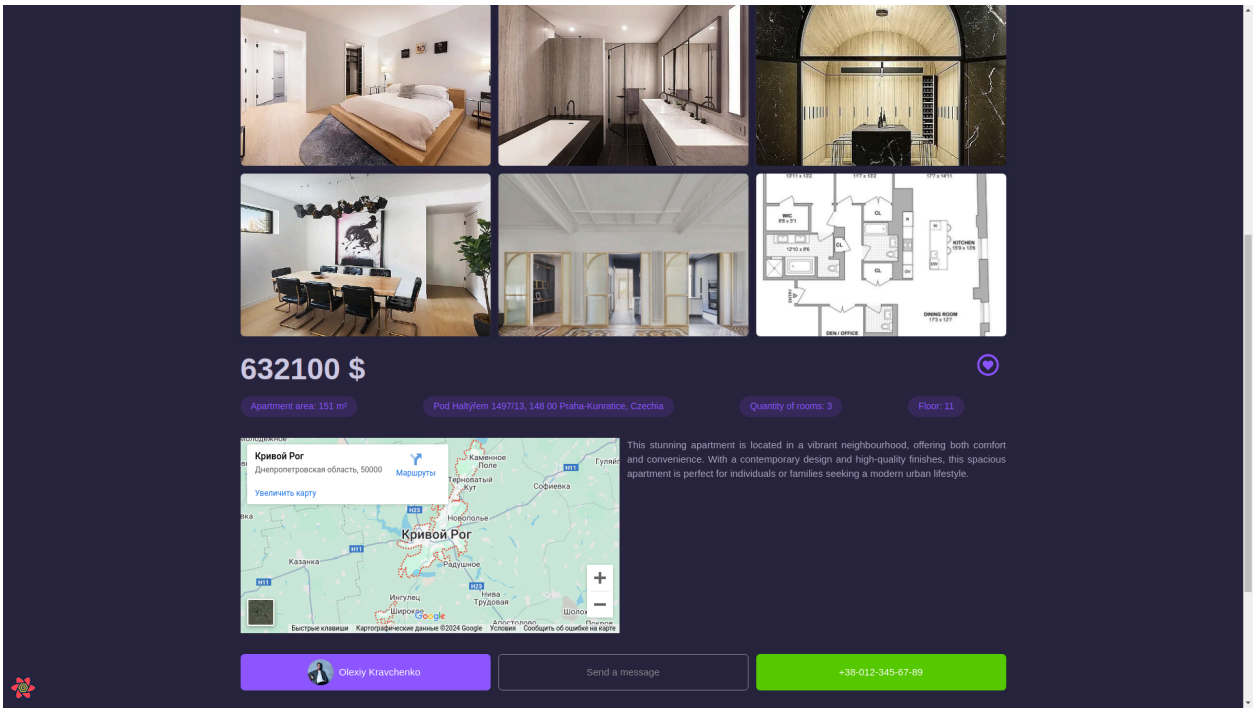

Рисунок 2.14 – Інтерфейс сторінки з інформацією про об'єкт

Процес створення оголошення був значно спрощений. Введення даних здійснюється через компонент FormGroup, який структуровано організовує поля для заповнення. Завантаження фотографій об'єктів реалізовано через інтеграцію Dropzone, що забезпечило зручність і гнучкість для користувачів. Сторінка управління оголошеннями користувача, яка дозволяє редагувати або видаляти записи, тепер використовує компонент DataGrid, що підтримує функції сортування й фільтрації для швидкого доступу до потрібної інформації (див. рис. 2.15).

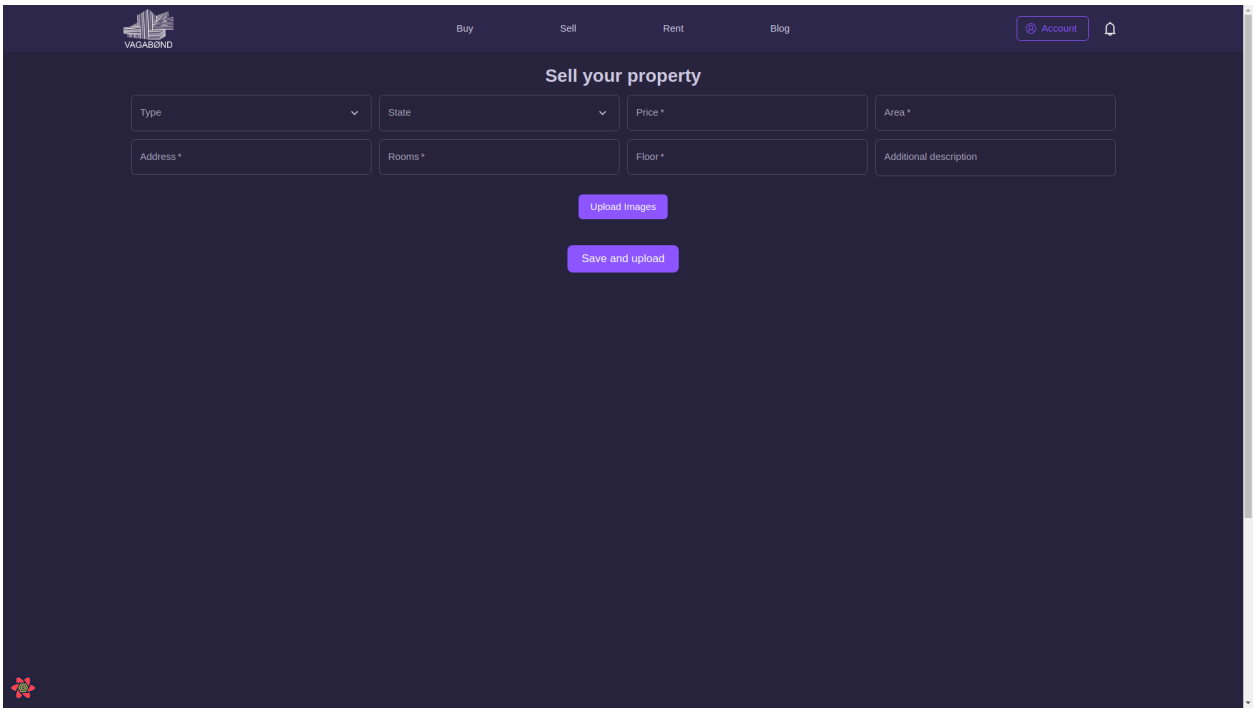

Рисунок 2.15 – Інтерфейс сторінки створення оголошення

Інші ключові елементи, такі як відображення улюблених оголошень та список активних оголошень користувача, також зазнали змін. Вони були адаптовані для використання сучасних компонентів MUI, що значно покращило зручність навігації та управління контентом. Усі ці оновлення зробили інтерфейс не лише привабливішим, але й ефективнішим для користувачів (див. рис. 2.16).

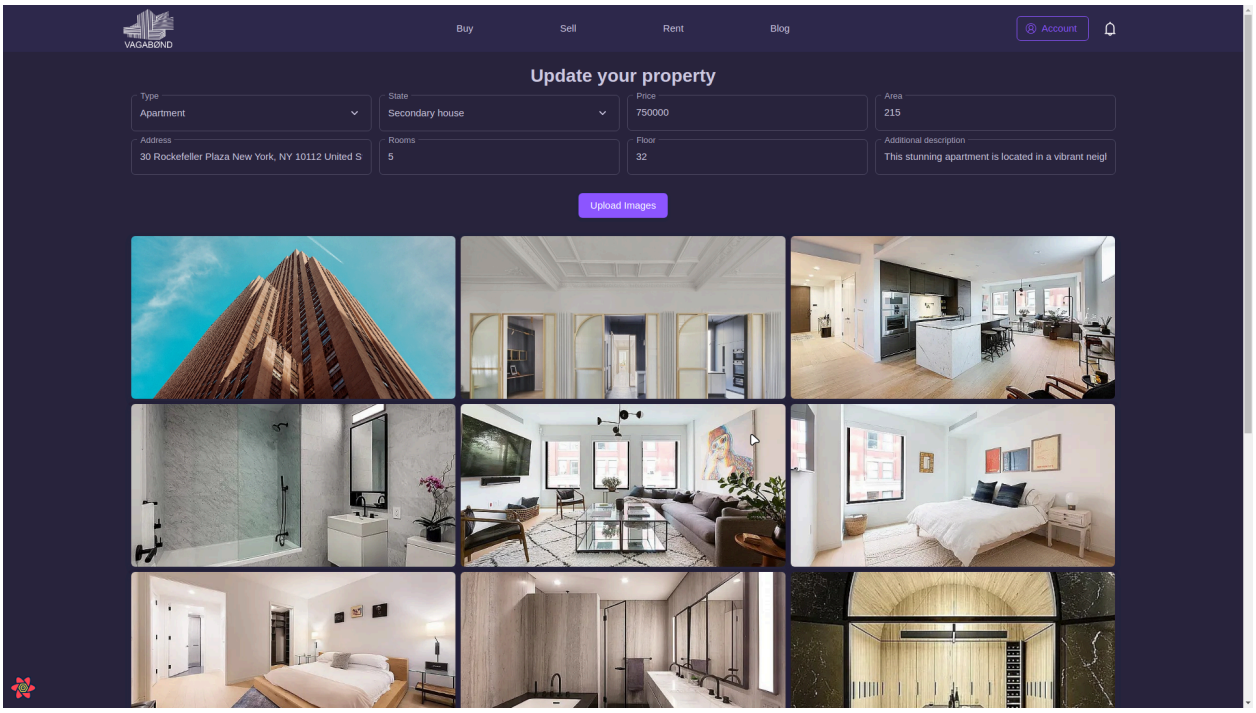

Рисунок 2.15 – Інтерфейс сторінки редагування оголошення

Загалом, структура інтерфейсу залишилася незмінною, однак усі елементи були вдосконалені відповідно до принципів Google Material Design і перенесені на компонентну базу MUI v6.2.0. Це дозволило досягти візуальної узгодженості, підвищити адаптивність системи та значно полегшити подальше обслуговування та розвиток платформи. Усі зміни були реалізовані з урахуванням потреб кінцевих користувачів, що забезпечило створення сучасного, функціонального та естетично привабливого веб-застосунку. Представлені результати оновлення можна побачити на зображеннях, які демонструють новий дизайн інтерфейсу для кожного ключового розділу системи.

3. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, БАЗИ ДАНИХ ТА МІГРАЦІЯ ЗАСТАРІЛИХ ЧАСТИН

3.1. Аналіз обраної середи програмування

Для реалізації даного проєкту було обрано середовище програмування Visual Studio Code (VS Code), розроблене Microsoft. VS Code є одним із найпопулярніших редакторів вихідного коду завдяки своїй універсальності, гнучкості та підтримці великої кількості мов програмування. Його багатий набір функцій та зручна екосистема забезпечують розробникам комфортні умови для написання, налагодження та тестування коду на різних платформах.

Однією з ключових особливостей VS Code є широка підтримка мов програмування та технологій [12], включаючи TypeScript, JavaScript, Node.js, Express.js, React і Next.js, які активно використовуються у цьому проєкті. Редактор забезпечує підсвічування синтаксису, інтелектуальні пропозиції та автодоповнення, що значно покращує продуктивність розробника. Крім того, VS Code підтримує роботу з безліччю інших мов та форматів, що робить його універсальним інструментом для командної розробки.

VS Code дозволяє персоналізувати середовище розробки за допомогою широкого набору тем, розширень та налаштувань. Наприклад, можна вибрати тему, що відповідає вподобанням розробника, або інтегрувати інструменти для перевірки якості коду, такі як ESLint чи Prettier. Це дає змогу адаптувати редактор під конкретні завдання, забезпечуючи ефективну організацію робочого процесу.

Одна з найсильніших сторін VS Code — це його ринок розширень. Завдяки цьому розробники можуть інтегрувати додаткові функціональні можливості, включаючи підтримку специфічних мов, інтеграцію з Docker, інструменти тестування, візуалізацію даних та багато іншого. Зокрема, для

проєкту активно використовувалися розширення для роботи з TypeScript, Next.js і Docker.

Редактор забезпечує бездоганну інтеграцію з системами контролю версій, такими як Git і GitHub, що є критично важливим для командної роботи. Можливість перегляду змін, об’єднання гілок і вирішення конфліктів прямо у редакторі дозволяє зекономити час та уникнути помилок під час розробки.

Вибір VS Code для реалізації проєкту обумовлений його універсальністю, багатим функціоналом і гнучкістю. Він став основою продуктивної роботи команди, забезпечивши стабільне середовище для інтеграції всіх необхідних інструментів і технологій. Його підтримка основних фреймворків і мов, що використовуються у проєкті, зробила розробку максимально ефективною, а інтеграція з іншими інструментами спростила командну взаємодію.

3.2. Підбір технологічних рішень для реалізації поставлених задач.

Реалізація даного проєкту вимагає використання різноманітних сучасних технологій, кожна з яких має свої сильні сторони і забезпечує виконання специфічних завдань. У цьому розділі розглядається набір технологій, обраних для побудови ефективної та масштабованої системи.

Турбопак, вбудований у Next.js, є сучасним інструментом для швидкої збірки та розробки додатків. На відміну від традиційних інструментів збірки, таких як Webpack або Vite, Турбопак забезпечує оптимізовану продуктивність завдяки використанню Rust. Основні переваги цього інструмента:

- Швидкість збірки: Турбопак виконує операції набагато швидше завдяки багатопоточності й низькорівневій обробці даних.

- Інтеграція з Next.js: Турбопак забезпечує безшовну роботу із серверними та клієнтськими компонентами в проєкті.
- Підтримка модулів ES: Інструмент працює безпосередньо з сучасними стандартами JavaScript, що дозволяє уникнути зайвих операцій.

Ці характеристики роблять Турбопак ідеальним вибором для проєктів, які потребують високої швидкості розробки та стабільної роботи у великих командах.

React забезпечує створення динамічних і модульних інтерфейсів користувача. Його компонентний підхід дозволяє створювати повторно використовувані елементи, що спрощує розробку і підтримку коду. Next.js, як фреймворк, розширює можливості React, додаючи такі функції:

- Рендеринг на сервері (SSR): Прискорює завантаження сторінок, покращуючи SEO та досвід користувачів.
- Статичний експорт: Дозволяє створювати статичні сторінки для високонавантажених додатків.
- Автоматичне розбиття коду: Оптимізує завантаження ресурсів.
- Підтримка API-роутів: Забезпечує створення серверних функцій без потреби у додаткових бекенд-інструментах.

Ці можливості дозволяють створювати високоефективні веб-додатки з інтерактивними функціями.

TypeScript є надбудовою над JavaScript, яка додає статичну типізацію. Його основні переваги:

- Підвищення надійності коду: Типізація дозволяє уникнути багатьох помилок на етапі компіляції.
- Інтеграція з редакторами: Завдяки підтримці IntelliSense розробники отримують підказки й автозаповнення.
- Зручність рефакторингу: Типи спрощують зміну структури коду без ризику його порушення.

Це робить TypeScript стандартом для сучасної веб-розробки, особливо у великих проєктах.

MUI забезпечує створення сучасних і адаптивних інтерфейсів користувача. Він надає готові компоненти, що відповідають вимогам Google Material Design guidelines [13]. Основні можливості:

- Гнучка кастомізація: Дозволяє змінювати стилі компонентів під специфіку проєкту.
- Швидка розробка: Завдяки багатому набору попередньо створених компонентів.
- Підтримка TypeScript: Забезпечує типобезпеку при роботі з компонентами.

MUI оптимально підходить для створення як корпоративних додатків, так і додатків для кінцевих користувачів.

Zustand є бібліотекою для управління станом у додатках React. Її основні особливості:

- Простота використання: Легкий синтаксис для створення та використання сховищ стану.
- Легковаговість: Мінімальний вплив на продуктивність додатка.

- Підтримка похідного стану: Автоматичне обчислення складних значень на основі базового стану.

Це робить Zustand зручним вибором для управління станом у сучасних додатках.

Zod є інструментом для валідації даних і забезпечення їхньої цілісності. Його основні переваги:

- Інтуїтивність: Простий API для створення схем перевірки.
- Інтеграція з TypeScript: Забезпечує перевірку типів на рівні коду.
- Гнучкість: Підтримка складних структур даних, таких як об'єкти та масиви.

Zod забезпечує надійний захист від помилок, пов'язаних із некоректними даними.

Express.js є фреймворком для створення серверної частини додатків, що забезпечує:

- Обробку HTTP-запитів: Простота у створенні API та маршрутизації.
- Інтеграцію з базами даних: Зручна робота з MongoDB.
- Гнучкість і легкість: Дозволяє створювати як прості, так і складні серверні архітектури.

TRPC інтегрується з TypeScript, забезпечуючи:

- Безпечну передачу даних: Гарантує узгодженість типів між клієнтом і сервером.

- Зручність у розробці: Використання одного інструмента для визначення типів і процедур.

Ця комбінація забезпечує масштабованість і безпеку серверної частини.

Для баз даних, використано MongoDB та Redis.

MongoDB:

- Гнучке зберігання даних у JSON-подібному форматі.
- Масштабованість для роботи з великими обсягами даних.
- Висока продуктивність для читання та запису.

Redis:

- Швидке зберігання в пам’яті для кешування. Застосування для обробки сесій і високонавантажених операцій.

Ці бази даних забезпечують ефективне управління даними та їхню доступність.

Docker використовується для контейнеризації додатка, що дозволяє:

- Створення ізольованих середовищ: Легке перенесення додатка між серверами.
- Масштабованість: Можливість швидкого розгортання нових інстанцій.
- Підтримка CI/CD: Автоматизація процесу розробки та розгортання.

Ретельно підібраний стек технологій забезпечує високу продуктивність, масштабованість та гнучкість розробки. Комбінація Turbopack, Next.js,

Zustand, Zod, Express.js, TypeScript, і MongoDB дозволяє створити сучасну систему, що відповідає всім вимогам проєкту.

3.3. Розробка бази даних.

Розробка бази даних є невід'ємною частиною створення програмного забезпечення, що вимагає збереження значних обсягів даних та забезпечення ефективного доступу до них. У цьому проєкті для управління даними було обрано MongoDB та Redis, що гарантують гнучкість і високу продуктивність роботи з даними.

MongoDB — це NoSQL база даних, яка зберігає дані в форматі BSON (двійковий JSON) [14]. Така структура дозволяє ефективно працювати зі складними ієрархіями даних.

Основні переваги MongoDB:

- Гнучкість: Відсутність жорсткої схеми дозволяє легко адаптувати базу даних до змін вимог.
- Масштабованість: Підтримка горизонтального масштабування дозволяє обробляти великі обсяги даних.
- Інтеграція: Зручна взаємодія з Node.js через офіційний драйвер MongoDB.
- Індексація: Можливість створення складних індексів для пришвидшення запитів.

У цьому проєкті MongoDB використовується для зберігання структурованих даних, таких як профілі користувачів, оголошення та їх властивості.

Для забезпечення надійності та зручності обслуговування база даних розгорнута на платформі MongoDB Atlas. Це дозволяє:

- Використовувати хмарні ресурси для зберігання даних.
- Налаштовувати автоматичне резервне копіювання.
- Забезпечити високий рівень безпеки за рахунок конфігурації доступу через IP-фільтри та аутентифікацію користувачів.

Після налаштування бази даних проєкт підключається до неї через URI, який зберігається у файлі .env. Це забезпечує захист конфіденційної інформації.

Для локальної розробки база даних MongoDB розгортається за допомогою Docker, що забезпечує ізоляцію та узгодженість середовищ. Використовуваний образ Docker:

docker run --name mongodb-container -d -p 27017:27017 mongo

Цей підхід дозволяє легко масштабувати та тестувати додаток у середовищі, яке відповідає реальним умовам.

Структура бази даних спроєктована на основі діаграми сутність-зв’язок (ERD). У MongoDB кожна сутність представлена як окрема колекція:

Користувачі (Users):

- Поля: email, ім’я, прізвище, пароль, номер телефону, роль.
- Індекси: email (унікальний).

Оголошення (Properties):

- Поля: тип, стан, ціна, адреса, площа, кімнати, поверх, автор, зображення, інформація.
- Індекси: тип, адреса. Використання посилань між колекціями забезпечує зв’язки між сутностями, наприклад, між оголошенням та автором.

Redis [15] використовується як сховище ключ-значення для забезпечення швидкого доступу до часто використовуваних даних. Його основні переваги:

- Швидкість: Зберігання даних у пам’яті гарантує низький час відгуку.
- Підтримка складних структур: Крім рядків, Redis дозволяє зберігати списки, множини, хеші та відсортовані множини.
- Використання TTL: Можливість встановлення терміну життя (Time-To-Live) для кешованих даних.

Redis в основному застосовується для:

- Кешування результатів складних запитів.
- Зберігання сесій користувачів.
- Реалізації черг завдань для фонової обробки.

Redis також розгортається у середовищі Docker, що забезпечує простоту налаштування і тестування. Використовуваний образ Docker:

docker run --name redis-container -d -p 6379:6379 redis

Це забезпечує ізольоване середовище для обробки кешу та швидкий доступ до даних.

Інтеграція MongoDB і Redis здійснюється через відповідні драйвери Node.js:

- Mongoose: Для взаємодії з MongoDB. Забезпечує зручний інтерфейс для визначення схем та виконання запитів.

- Ioredis: Для підключення до Redis та управління кешем.

Підключення до баз даних реалізується у вигляді окремих модулів, які забезпечують повторне використання коду та спрощують обслуговування.

Приклад частини коду з реалізацією TypeScript классу User в MongoDB за допомогою бібліотеки Typegoose:

```
import {
  prop,
  getModelForClass,
  index,
  modelOptions,
  pre,
} from "@typegoose/typegoose";
import bcrypt from "bcrypt";
export enum Roles {
  ADMIN = "admin",
  USER = "user",
}
@index({ email: 1 })
@pre<User>("save", async function () {
  // Only hash password if it's modified and exists
  if (this.password && this.isModified("password")) {
    this.password = await bcrypt.hash(this.password, 12);
  }
})
@modelOptions({
  schemaOptions: {
    timestamps: true,
  },
})
export class User {
  @prop({ required: true, unique: true })
  public email!: string;
```

```
        @prop({ minlength: 8, maxlength: 32, select: false })
        public password?: string;
        @prop({ required: true })
        public name!: string;
        @prop({ required: true })
        public surname!: string;
        @prop({ required: true, enum: Roles, default: "user" })
        public role!: Roles;
        @prop({ required: true, unique: true })
        public phonenumber!: string;
        @prop({ default: "", required: false })
        public info?: string;
        @prop({ default: "", required: false })
        public profilepic?: string;
        @prop({ type: () => [String], default: [], required: false
})
        public featured?: string[];
        async comparePasswords(hashedPassword: string,
comparingPassword: string) {
          return await bcrypt.compare(comparingPassword,
hashedPassword);
        }
      }
      const UserModel = getModelForClass<typeof User>(User);
      export default UserModel;
```

Цей приклад демонструє, як за допомогою Typegoose можна визначити схему користувача, що забезпечує зручність роботи з MongoDB.

Використання MongoDB для основного зберігання даних і Redis для кешування забезпечує баланс між гнучкістю, масштабованістю та продуктивністю. Інтеграція цих технологій з бекендом через сучасні драйвери гарантує надійність і ефективність роботи всієї системи.

3.4. Програмна реалізація необхідних та основних функцій застосунку.

Основні функції програми поділяються на клієнтські та серверні, що забезпечує чітке розмежування обов’язків між інтерфейсом користувача і серверною логікою. Такий підхід сприяє кращій модульності, масштабованості та зручності підтримки системи.

Функціонал клієнтської частини охоплює:

- Пошук і фільтрація: Користувачі можуть здійснювати пошук нерухомості за різними критеріями, такими як тип, стан, ціна, адреса, площа тощо. Для реалізації пошуку використовується компонент Autocomplete з бібліотеки Material-UI.
- Взаємодія з оголошеннями: Перегляд детальної інформації про оголошення. Додавання оголошень до списку обраних (потрібна автентифікація).
- Реєстрація та авторизація: Реалізовано форми реєстрації та входу, що перевіряють введені дані за допомогою бібліотеки Zod. Дані передаються на сервер через API-запити.
- Створення оголошень: Після автентифікації користувачі можуть створювати нові оголошення, додаючи необхідні поля та зображення.
- Інтерактивність: Використання Zustand для централізованого управління станом. Динамічні оновлення сторінок без перезавантаження завдяки React.

Серверна частина забезпечує такі функції:

- API для оголошень: Створення, читання, оновлення та видалення (CRUD) оголошень. Пошук оголошень за фільтрами. Реалізовано за допомогою Express.js.
- Управління користувачами: Реєстрація нових користувачів із хешуванням паролів за допомогою bcrypt. Вхід і вихід із системи з генерацією JWT для авторизації.
- Збереження та кешування даних: Збереження даних у MongoDB. Кешування часто використовуваних запитів у Redis для підвищення продуктивності.
- Автентифікація та авторизація: Захист API-ендпоінтів за допомогою middleware для перевірки токенів.Використання tRPC для безпечної типізованої взаємодії між клієнтом і сервером.
- Обробка зображень: Зберігання зображень у локальній файловій системі чи на хмарному сховищі (наприклад, AWS S3 як зазначено в мембер панелі MongoDB Atlas). Перевірка форматів і розмірів файлів.

Архітектура API побудована на основі REST-принципів з додатковою інтеграцією tRPC для забезпечення типобезпеки. Приклад структури маршруту для роботи з оголошеннями:

- GET /api/properties: Повертає список оголошень.
- POST /api/properties: Створює нове оголошення (потребує access_token).

- PUT /api/properties/:id: Оновлює оголошення за ID (потребує access_token).
- DELETE /api/properties/:id: Видаляє оголошення за ID (потребує access_token).

Поділ функціоналу на клієнтський і серверний забезпечує гнучкість, масштабованість і ефективність системи. Завдяки використанню сучасних бібліотек і фреймворків, таких як NextJS, Express.js, tRPC, MongoDB і Redis, реалізовано надійний і продуктивний веб-застосунок.

3.5. Забезпечення безпеки системи.

Розробка веб-застосунку вимагає впровадження механізмів безпеки для захисту даних користувачів, серверів та загальної інфраструктури. У цьому розділі описано основні підходи, застосовані для забезпечення захисту системи.

Захист даних користувачів:

- Хешування паролів: Паролі користувачів зберігаються у хешованому вигляді за допомогою алгоритму bcrypt. Це забезпечує захист навіть у разі компрометації бази даних.
- Шифрування переданих даних: Вся передача даних між клієнтом і сервером здійснюється через HTTPS, що запобігає перехопленню даних.
- Валідація вхідних даних: Дані користувачів перевіряються за допомогою бібліотеки Zod на клієнтській стороні і на серверній стороні для забезпечення їхньої відповідності очікуваним форматам.

Захист серверної інфраструктури:

- JSON Web Tokens (JWT): Для автентифікації використовується JWT. Кожен токен має термін дії та цифровий підпис, який перевіряється сервером.
- Middleware для авторизації: Серверні маршрути захищені middleware, який перевіряє наявність і дійсність токену.
- Обмеження запитів: Використовується rate-limiting для запобігання атакам типу brute force та DDoS.
- Захист від CORS: Налаштовано політики Cross-Origin Resource Sharing (CORS), що обмежують доступ до API лише з дозволених доменів.

Захист від поширених загроз:

- Запобігання SQL-ін’єкціям: Оскільки система використовує MongoDB, всі запити до бази даних обробляються через безпечні драйвери та фільтруються ще на клієнтській частині.
- Захист від XSS: Дані, введені користувачами, очищаються перед відображенням у веб-інтерфейсі.
- CSRF-захист: Використовується CSRF-токени для захисту від міжсайтових запитів підробок.

Безпека файлового сховища:

- Перевірка файлів: Завантажені файли перевіряються на тип і розмір. Недозволені формати або великі файли відхиляються.

- Зберігання файлів: Зображення та інші файли зберігаються на сервері з використанням політик доступу, що обмежують права на запис і читання.

Впровадження вищезазначених методів безпеки забезпечує захист системи від поширених загроз і зберігає конфіденційність даних користувачів. Постійний моніторинг і регулярне оновлення інфраструктури додатково гарантують її стабільність та захищеність.

3.6. Тестування застосунку.

Забезпечення високої якості та стабільності програмного забезпечення є ключовим етапом у сучасній розробці веб-застосунків. У процесі розробки було використано комбінований підхід до тестування, що включає юніт-тести, статичний аналіз коду за допомогою ESLint, а також ручне тестування на реальній системі розробника. Вказаний підхід дозволив мінімізувати кількість помилок на ранніх етапах та забезпечити коректну роботу додатка в різних сценаріях.

Юніт-тести (unit tests) стали основою для автоматизованого тестування, оскільки вони дозволяють перевіряти окремі функції, компоненти та логічні блоки програми у відокремленому середовищі. Під час розробки юніт-тести охоплювали такі аспекти:

- Локалізація помилок на ранніх етапах: Тести писалися для ключових функцій та компонентів на фронтенді, таких як валідація форм, обробка подій введення, стан компонентів та взаємодія з API. Застосування Jest та React Testing Library забезпечувало швидке виконання тестів та підтримку мокування зовнішніх залежностей.

- Охоплення коду тестами: Основний фокус був на покритті критичних частин застосунку, зокрема:Валідація форм: перевірка на некоректні дані, наприклад, короткий пароль чи помилковий формат електронної пошти. Стан компонентів: тестування компонентів Material-UI, їхньої взаємодії зі станом React та відображення на основі переданих пропсів. Асинхронні операції: обробка відповідей від бекенду у випадках успіху, помилок та порожніх результатів.

Приклад тестового сценарію:

```
import { render, screen } from '@testing-library/react';
import userEvent from '@testing-library/user-event';
import LoginForm from '../components/LoginForm';
test('Валідація неправильної електронної пошти', async () =>
{
  render(<LoginForm />);
  const emailInput = screen.getByPlaceholderText(/email/i);
  userEvent.type(emailInput, 'wrongemail');
  userEvent.click(screen.getByRole('button', { name:
/submit/i }));
expect(await screen.findByText(/невірний
формат/i)).toBeInTheDocument();
});
```

Для збереження якості коду та запобігання синтаксичним помилкам було використано ESLint. Цей інструмент дозволяє виконувати статичний аналіз коду, виявляючи потенційні проблеми ще до його запуску. Основні завдання ESLint у процесі розробки:

- Виявлення помилок: Синтаксичні помилки, некоректне використання функцій та неправильний порядок викликів методів. Пошук невикористаних змінних та функцій для оптимізації коду.

- Дотримання кодстайлу: Налаштовані правила дозволили забезпечити єдиний стиль написання коду (відповідно до стандартів Airbnb та рекомендацій для TypeScript).
- Інтеграція у редактор: ESLint був інтегрований у VSCode, що дозволяло розробнику миттєво отримувати підказки щодо проблем у коді.

Використання статичного аналізу дозволило скоротити час на ручну перевірку помилок та знизити ризик їхнього виникнення на пізніх етапах.

Конфігурація ESLint:

```
/* eslint-disable import-x/no-named-as-default-member */
import comments from
'@eslint-community/eslint-plugin-eslint-comments/configs';
import react from '@eslint-react/eslint-plugin';
import { FlatCompat } from '@eslint/eslintrc';
import js from '@eslint/js';
import prettierConfig from 'eslint-config-prettier';
import eslintPluginImportX from 'eslint-plugin-import-x';
import regexPlugin from 'eslint-plugin-regexp';
import security from 'eslint-plugin-security';
import globals from 'globals';
import tseslint from 'typescript-eslint';
import tailwind from 'eslint-plugin-tailwindcss';
const compat = new FlatCompat({
  baseDirectory: import.meta.dirname,
});
const config = tseslint.config(
  {
    ignores: ['.next', 'node_modules'],
  },
  // Base
  js.configs.recommended,
  ...tseslint.configs.strictTypeChecked,
```

```
...tseslint.configs.stylisticTypeChecked,
eslintPluginImportX.flatConfigs.recommended,
eslintPluginImportX.flatConfigs.typescript,
comments.recommended,
regexPlugin.configs['flat/recommended'],
security.configs.recommended,

// Next.js / React
...compat.extends('plugin:@next/next/recommended'),
...compat.extends('plugin:react-hooks/recommended'),
...compat.plugins('react-compiler'),
react.configs['recommended-type-checked'],

// Tailwind
...tailwind.configs['flat/recommended'],

// Custom Rules and Settings
{
  linterOptions: {
    reportUnusedDisableDirectives: true,
  },
  languageOptions: {
    ecmaVersion: 'latest',
    sourceType: 'module',
    parserOptions: {
      projectService: true,
      tsconfigRootDir: import.meta.dirname,
      ecmaFeatures: {
        jsx: true,
      },
    },
    globals: {
      ...globals.browser,
      ...globals.node,
    },
```

```
        },
        settings: {
          tailwindcss: {
            callees: ['classnames', 'clsx', 'ctl', 'cn', 'cva'],
          },
          react: {
            version: 'detect',
          },
          'import-x/parsers': {
            '@typescript-eslint/parser': ['.ts', '.tsx'],
          },
          'import-x/resolver': {
            node: {},
            typescript: {
              project: './tsconfig.json',
            },
          },
        },
        rules: {
          // Existing Rules
          '@typescript-eslint/no-unused-vars': [
            'error',
            { argsIgnorePattern: '^_', varsIgnorePattern: '^_'
},
          ],
          '@typescript-eslint/consistent-type-imports': [
            'warn',
            { prefer: 'type-imports', fixStyle:
'separate-type-imports' },
          ],
          '@typescript-eslint/no-misused-promises': [
            'error',
            { checksVoidReturn: { attributes: false } },
          ],
          '@typescript-eslint/no-unnecessary-condition': [
```

```
            'error',
            { allowConstantLoopConditions: true },
          ],
          '@typescript-eslint/consistent-type-exports': [
            'error',
            { fixMixedExportsWithInlineTypeSpecifier: true },
          ],
          'import-x/no-unresolved': ['error', { ignore:
['geist'] }],
          'react-compiler/react-compiler': 'error',

          // Additional Rules
          'jsx-a11y/alt-text': 'off',
          'react/display-name': 'off',
          'react/no-children-prop': 'off',
          '@next/next/no-img-element': 'off',
          '@next/next/no-page-custom-font': 'off',
          '@typescript-eslint/ban-ts-comment': 'off',
          '@typescript-eslint/no-explicit-any': 'off',
          '@typescript-eslint/no-non-null-assertion': 'off',
          'lines-around-comment': [
            'error',
            {
              beforeBlockComment: true,
              beforeLineComment: true,
              allowBlockStart: true,
              allowObjectStart: true,
              allowArrayStart: true,
            },
          ],
          'padding-line-between-statements': [
            'error',
            {
              blankLine: 'any',
              prev: 'export',
```

```
              next: 'export',
            },
            {
              blankLine: 'always',
              prev: ['const', 'let', 'var'],
              next: '*',
            },
            {
              blankLine: 'any',
              prev: ['const', 'let', 'var'],
              next: ['const', 'let', 'var'],
            },
            {
              blankLine: 'always',
              prev: '*',
              next: ['function', 'multiline-const',
'multiline-block-like'],
            },
            {
              blankLine: 'always',
              prev: ['function', 'multiline-const',
'multiline-block-like'],
              next: '*',
            },
          ],
          'newline-before-return': 'error',
          'import-x/newline-after-import': ['error', { count: 1
}],
          'import-x/order': [
            'error',
            {
              groups: ['builtin', 'external', ['internal',
'parent', 'sibling', 'index'], ['object', 'unknown']],
              pathGroups: [
```

```
                { pattern: 'react', group: 'external', position:
'before' },
                { pattern: 'next/**', group: 'external',
position: 'before' },
                { pattern: '~/**', group: 'external', position:
'before' },
                { pattern: '@/**', group: 'internal' },
              ],
              pathGroupsExcludedImportTypes: ['react', 'type'],
              'newlines-between': 'always-and-inside-groups',
            },
          ],
          '@typescript-eslint/no-restricted-types': [
            'error',
            {
              types: {
                Function: {
                  message: 'Use a more specific function type
instead of `Function`.',
                  fixWith: '(...args: any[]) => any',
                },
                Object: {
                  message: 'Avoid `Object` type. Use
`Record<string, unknown>` or `{}` instead.',
                  fixWith: 'Record<string, unknown>',
                },
                Boolean: {
                  message: 'Use `boolean` (lowercase) instead of
`Boolean`.',
                  fixWith: 'boolean',
                },
                Number: {
                  message: 'Use `number` (lowercase) instead of
`Number`.',
                  fixWith: 'number',
```

```
                },
                String: {
                  message: 'Use `string` (lowercase) instead of
`String`.',
                  fixWith: 'string',
                },
                Symbol: {
                  message: 'Use `symbol` (lowercase) instead of
`Symbol`.',
                  fixWith: 'symbol',
                },
                // If you still want to disable certain bans,
use `false`
                any: 'false',
                '{}': 'false',
              },
            },
          ],
        },
      },
      // File Overrides
      {
        files: ['**/*.cjs', '**/*.cts'],
        languageOptions: {
          sourceType: 'commonjs',
        },
      },
      {
        files: ['*.ts', '*.tsx', 'src/iconify-bundle/*'],
        rules: {
          '@typescript-eslint/explicit-module-boundary-types':
'off',
          '@typescript-eslint/no-var-requires': 'off',
        },
      },
```

```
  prettierConfig,
);
export default config;
/* eslint-enable import-x/no-named-as-default-member */
```

4. ЕКОНОМІЧНИЙ АНАЛІЗ ТА ОЦІНКА ЕФЕКТИВНОСТІ ВПРОВАДЖЕННЯ

У сучасних умовах стрімкого розвитку цифрових технологій, ринок нерухомості зазнає значних змін, особливо у секторі вторинного ринку житла. Впровадження веб-додатків стає невід'ємною частиною стратегії багатьох компаній, спрямованих на оптимізацію бізнес-процесів, покращення взаємодії з клієнтами та збільшення ефективності операцій. В умовах, коли зовнішні фактори, такі як військовий конфлікт в Україні, значно впливають на ринок нерухомості, потреба у швидких та ефективних рішеннях стає ще більш актуальною.

Економічний аналіз є критично важливим етапом у процесі впровадження нових технологічних рішень, оскільки дозволяє визначити фінансову доцільність проекту, прогнозувати його економічні результати та мінімізувати потенційні ризики. Враховуючи специфіку ринку нерухомості та особливості вторинного ринку житла, економічний аналіз допоможе виявити ключові чинники, що впливають на успішність проекту, а також розробити стратегії для підвищення його ефективності.

У цьому розділі буде проведено детальний аналіз витрат на розробку та впровадження веб-застосунку, оцінено потенційні доходи та визначено терміни окупності інвестицій. Крім того, будуть розглянуті переваги та виклики, що виникають при інтеграції цифрових рішень у ринок нерухомості, а також запропоновані рекомендації щодо оптимізації економічної ефективності проекту. Завдяки цьому аналізу, можна буде отримати всебічне уявлення про фінансові аспекти впровадження веб-застосунку та його вплив на бізнес-процеси в секторі нерухомості.

4.1. Особливості впровадження застосунків у складних умовах.

Сучасний ринок нерухомості зазнає значного впливу від зовнішніх факторів, таких як економічні кризи та військові конфлікти. В Україні, на тлі

війни, ситуація ускладнилася, що суттєво вплинуло на попит на цифрові рішення. Багато людей втратили житло і шукають альтернативні варіанти, а власники нерухомості прагнуть швидше продати свої активи для забезпечення фінансової стабільності. У таких умовах веб-застосунки відіграють критичну роль, забезпечуючи швидкий, ефективний та доступний спосіб управління процесами купівлі-продажу. Однак впровадження таких технологій у складних економічних та соціальних умовах супроводжується низкою викликів, які вимагають особливого підходу до розробки та реалізації проекту.

Ключові виклики впровадження:

- Обмежені фінансові ресурси, кризові умови часто супроводжуються зниженням платоспроможності компаній і споживачів. Це обмежує можливість інвестування у розробку та впровадження веб-застосунків. Замість масштабних проєктів компанії змушені зосереджуватись на оптимізації витрат, обираючи дешевші або готові рішення.
- Нестабільність попиту та змінні потреби користувачів. Під час кризи попит на певні види нерухомості може змінюватися, як і уподобання клієнтів. Наприклад, покупці стають більш чутливими до цін і надають перевагу платформам, які забезпечують прозорість інформації, швидкість обробки запитів і доступність.
- Складність інтеграції. У кризових умовах більшість компаній не можуть собі дозволити суттєвих змін у бізнес-моделі. Інтеграція веб-застосунку має бути максимально швидкою, без значних витрат часу чи коштів, а також не створювати додаткового навантаження на існуючі системи.

- Технічні обмеження. Використання сучасних технологій, таких як хмарні рішення, аналітика великих даних (Big Data), доповнена реальність (AR) чи віртуальна реальність (VR), вимагає значних обчислювальних ресурсів і інвестицій у серверну інфраструктуру. У складних умовах такі витрати можуть бути надмірними.
- Ризик низької прийнятності серед користувачів. Під час економічної нестабільності клієнти можуть бути менш схильними освоювати нові технології, особливо якщо вони здаються складними або вимагають значного навчання. Це вимагає від розробників створення інтуїтивно зрозумілого інтерфейсу і простих у використанні функцій.

Щоб забезпечити успішне впровадження веб-застосунку в умовах кризи, слід дотримуватися наступних стратегій:

- Оптимізація витрат. Використання готових рішень або хмарних сервісів для мінімізації початкових витрат. Наприклад, оренда серверів замість їх купівлі дозволяє заощадити на інфраструктурі.
- Інтуїтивний дизайн. Створення інтерфейсу, який є максимально простим для користувачів різного рівня технічної підготовки. Інтуїтивність і зручність зменшують бар’єри для прийняття нових технологій.
- Модульна архітектура. Розробка застосунку з можливістю поступового розширення функціоналу, що дозволяє впроваджувати інновації поетапно, залежно від доступних ресурсів і змін ринкових умов.

- Акцент на економічну ефективність. Включення в бізнес-модель джерел доходів, таких як преміум-функції, рекламні інструменти або партнерські програми, що дозволяють компенсувати витрати на розробку.
- Залучення користувачів через маркетинг. Просування веб-застосунку з використанням стратегій цифрового маркетингу (SEO, контекстна реклама, соціальні мережі), які забезпечують широке охоплення аудиторії при відносно низьких витратах.

Особливості впровадження веб-застосунків у складних умовах полягають у необхідності балансу між витратами, технічними можливостями та очікуваннями користувачів. Важливими факторами успіху є гнучкість, адаптивність та фокус на потребах клієнтів. Незважаючи на виклики, правильна стратегія впровадження дозволяє компаніям не лише подолати кризу, але й забезпечити собі конкурентну перевагу у довгостроковій перспективі.

4.2. Аналіз та огляд витрат на розробку та впровадження веб-застосунку.

Розробка веб-застосунку для ринку нерухомості може бути реалізована через три основні моделі, кожна з яких має свої переваги, недоліки та економічну доцільність:

- Найм працівників та відкриття власного IT-відділу.
- Аутсорсинг розробки застосунку сторонній компанії.
- Використання готових рішень за підпискою.

Для більш глибокого аналізу цих моделей були створені таблиці із орієнтовними витратами та визначено ключові фактори, які впливають на економічну ефективність кожного підходу.

4.2.1. Аналіз вартості розробки працівниками та відкриття власного ІТ-відділу.

Цей підхід передбачає найм команди розробників, дизайнерів і тестувальників, а також створення необхідної інфраструктури для роботи. Це найбільш ресурсозатратний варіант, але він забезпечує максимальний контроль над процесом розробки.

Таблиця 4.1 – витрати на працівників

Посада	Кількість осіб	Зарплата, грн	Обладнання, грн	Загальні витрати, грн	Разові витрати, грн
Frontend-розробник	2	72000–93000	51000-73000	142000-186000	102000-146000
Backend розробник	2	74000-111000	55500-85500	148000-222000	111000-161000
UI/UX дизайнер	1	52000-71000	49000-71000	52000-71000	49000-71000
QA-інженер	1	58000-79000	55500-70000	58000-79000	55500-70000
Проєктний мереджер	1	117000-151000	50500-70000	117000-151000	50500-70000
Загалом	7			**517000-709000**	**367500-518000**

Було створено таблицю з додатковими витратами на впровадження ІТ-відділу.

Таблиця 4.2 – додаткові витрати

Категорія витрат	Разові витрати, грн	Щомісячні витрати, грн
Ліцензії на програми	106000-195000	5370-9660
Хостинг та сервери	0	16300-19700
Додаткові витрати	52000-77000	8500-13600
Загалом	158000-272000	30170-42960

У підсумку маємо такі суми загальних витрат:

- мінімальні – 525500 грн разових витрат та 547170 грн/міс;
- максимальні – 790000 грн разових витрат та 751960 грн/міс.

Мінімальна вартість формується, коли під час розробки веб-застосунку використовуються ресурси із низьким рівнем витрат. Це включає найм молодших спеціалістів із нижчими зарплатами, а також використання базового обладнання, наприклад, ноутбуків початкового рівня замість потужних пристроїв для складних обчислень.

В якості програмного забезпечення застосовуються безкоштовні інструменти, такі як Visual Studio Code для кодування. Сервери орендуються у бюджетних хостинг-провайдерів із мінімальними тарифами. Цей підхід дозволяє значно зменшити початкові та операційні витрати, проте може призвести до повільнішої розробки та обмежень у продуктивності застосунку під час високого навантаження. Наприклад, обмеження серверного обладнання може негативно вплинути на обробку великої кількості одночасних запитів.

Максимальна вартість передбачає залучення досвідчених фахівців з високими зарплатами, що забезпечує швидке й якісне виконання завдань. Для розробки застосовуються преміальні інструменти, такі як JetBrains WebStorm для фронтенду або платні CI/CD-системи для автоматизації тестування.

Обладнання включає потужні ноутбуки з високою обчислювальною потужністю, що дозволяє ефективно працювати із складними завданнями.

Для хостингу використовуються високопродуктивні сервери з мінімальними затримками, що гарантує стабільну роботу системи навіть під час пікового навантаження. В результаті, максимальні інвестиції дозволяють забезпечити гнучкість у розробці, швидке внесення змін та високу продуктивність кінцевого продукту. Основними перевагами такого підходу є повний контроль над процесами розробки, можливість інтеграції нових функцій на різних етапах проєкту та висока якість застосунку. Однак головним недоліком є значні витрати на оплату праці фахівців, ліцензування програмного забезпечення та підтримку інфраструктури.

4.2.2. Аналіз вартості аутсорсингу розробки застосунку сторонній компанії.

Аутсорсинг у сфері розробки веб-застосунків передбачає передачу всього циклу створення продукту сторонній компанії або команді. Замовник зосереджується на формулюванні технічного завдання та фінансуванні ключових етапів, включно з хостингом, інфраструктурою та інтеграцією інструментів аналітики.

Таблиця 4.3 – вартість аутсорс рішення

Категорія	Опис	Витрати, грн
Дизайн	UI/UX-дизайн інтерфейсу	71000–179000
Розробка	Створення фронтенду та бекенду	571000-917000
Тестування	Забезпечення якості (QA)	41000-123000
Хостинг	Вартість серверних ресурсів на 1 рік	83200-227000
Загальні витрати		766200-1446000

Мінімальні витрати спостерігаються, коли замовник обирає локальні аутсорсингові компанії чи фрілансерів із нижчими ставками. У таких

випадках реалізація продукту передбачає лише базовий функціонал, стандартний дизайн і мінімальні налаштування аналітичних систем.

Максимальна вартість формується при співпраці з висококваліфікованими командами з міжнародним досвідом. Проєкт включає індивідуальний UI/UX-дизайн, розширений набір функцій, інтеграцію з внутрішніми системами підприємства та детальну аналітику. Цей підхід забезпечує високий рівень персоналізації продукту та його технологічну досконалість.

Таким чином, вибір між мінімальними та максимальними витратами залежить від потреб замовника, фінансових можливостей і стратегічних пріоритетів розвитку продукту.

4.2.3. Аналіз вартості розробки при використанні готових рішень.

Готові рішення забезпечують низку можливостей для ефективного впровадження застосунків. Наприклад, існують шаблони програмних продуктів, які дозволяють інтегрувати систему без необхідності починати розробку з нуля. Крім того, однією з ефективних практик є використання вже готових застосунків з можливістю підключення до їх сервісів, що значно знижує загальні витрати на розробку. У межах дослідження було складено таблицю витрат на використання готових рішень.

Таблиця 4.4 – вартість стороннього рішення

Категорія	Опис	Витрати, грн
Стартова інтеграція	Перше налаштування	41000
Щомісячна ставка	Ліцензія (включно з підтримкою)	19100
Додаткові функції	Різний преміум функціонал, як персоналізація, доступ до розширених баз даних	77000
Загальні витрати		137100

Згідно з таблицею, одноразові витрати на інтеграцію готового рішення складають 137100 грн. Протягом першого року загальні витрати на роботу із застосунком сягатимуть 270200 грн, а за 3 роки – 728600 грн. Отже, підписка є найвигіднішим варіантом у короткостроковій перспективі. Проте її довгострокова ефективність зберігається лише за умови відсутності необхідності значної кастомізації функціоналу.

Переваги використання готових рішень:

- Низькі початкові витрати – впровадження не вимагає значних інвестицій на старті.
- Швидке впровадження – процес інтеграції значно пришвидшується без залучення внутрішніх ресурсів компанії.
- Постійна підтримка та оновлення – технічний супровід здійснюється постачальником послуг.

Недоліки використання готових рішень:

- Обмежена гнучкість – можливості персоналізації дизайну та функціоналу є суттєво обмеженими.
- Високі довгострокові витрати – сукупна вартість підписки може перевищувати витрати на аутсорсинг.
- Залежність від постачальника – у разі виникнення проблем у компанії-постачальника користувачі не можуть вплинути на їх вирішення.

Таким чином, мінімальні витрати дозволяють бізнесу суттєво заощадити, проте часто призводять до обмеженого функціоналу або компромісів у якості продукту. Натомість максимальні витрати забезпечують

унікальність та високу якість рішення, однак можуть бути фінансово невигідними для невеликих підприємств, таких як кафе чи локальні магазини.

Вибір між готовими рішеннями та іншими підходами залежить від розміру бізнесу, бюджету та стратегічних цілей підприємства. Готові рішення є доцільними для бізнесів, які прагнуть швидкого впровадження за мінімальних витрат і не потребують глибокої кастомізації.

4.3. Розрахунок економічної ефективності веб-застосунку для ринку нерухомості з урахуванням витрат.

Розглянемо приклад застосування формули наведених раніше у розрахунку на тестових даних, що симулюють реальні бізнес-кейси. Для прикладу візьмемо дані двох бізнес-кейсів – із позитивними показниками факторів та негативними.

Дані для розрахунків позитивного сценарію:

- Кількість активних користувачів: 5000.
- Платні оголошення (20% користувачів): $5000 \times 20\% = 1000$.
- Розміщення в топі (30% користувачів):$5000 \times 30\% = 1500$.
- Кількість агентств реклами: 50.
- Витрати на розробку: 1000000 грн.
- Операційні витрати: 1757647 грн/рік.

Розрахунок доходів:

Платні оголошення: $1000 \times 120 = 120000$ грн.

Розміщення в топі: $1500 \times 360 \times 12 = 6480000$ грн.

Реклама агентств: $50 \times 720 \times 12 = 432000$ грн.

Загальний дохід: $D = 120000 + 6480000 + 432000 = 7032000$ грн.

Економічна ефективність впровадження веб-застосунку, в даному випадку дорівнює: $E = \frac{7032000-2757647}{2757647} \times 100\% \approx 155\%$

Отже, підприємство з такими показниками отримає річну вигоду в 155%, що свідчить про високу економічну ефективність веб-застосунку.

Дані для розрахунків негативного сценарію:

- Кількість активних користувачів: 1000.
- Платні оголошення (10% користувачів): $1000 \times 10\% = 100$.
- Розміщення в топі (10% користувачів):! $000 \times 10\% = 100$.
- Кількість агентств реклами: 10.
- Витрати на розробку: 1000000 грн.
- Операційні витрати: 300000 грн/рік.

Розрахунок доходів:

Платні оголошення: $100 \times 120 = 12000$ грн.

Розміщення в топі: $100 \times 360 \times 12 = 432000$ грн.

Реклама агентств: $10 \times 720 \times 12 = 86400$ грн.

Загальний дохід: $D = 12000 + 432000 + 86400 = 530400$ грн.

Економічна ефективність впровадження веб-застосунку, в даному випадку дорівнює: $E = \frac{530400-1300000}{1300000} \times 100\% \approx -55.2\%$

У такому випадку підприємство зазнає значних збитків, а розробка застосунку є економічно недоцільною.

Таким чином, залежно від бізнес-параметрів та кількості користувачів, ефективність веб-застосунку може суттєво змінюватися. Для досягнення позитивного Е необхідно збільшувати базу активних користувачів та

рекламодавців, а також оптимізувати витрати на розробку та підтримку платформи.

4.4. Аналіз стратегій мінімізації збитків та витрат на впровадження веб-застосунків у секторі нерухомості.

Розробка та впровадження веб-застосунків для ринку нерухомості вимагають детального аналізу стратегій мінімізації витрат, оскільки вибір відповідної моделі впливає на економічну ефективність проєкту. Різні підходи до впровадження застосунків мають свої переваги та недоліки, які визначають доцільність їх застосування залежно від масштабу бізнесу, стратегічних цілей і доступних фінансових ресурсів. Цей розділ дослідження фокусується на оцінці оптимальних моделей інтеграції в коротко- та довгостроковій перспективі.

У короткостроковій перспективі (<3 років) основною метою бізнесу є мінімізація початкових витрат при швидкому впровадженні функціонального рішення. Оптимальним підходом для малих і середніх підприємств, таких як агентства нерухомості чи невеликі забудовники, є підписка на готові рішення. Ця модель передбачає мінімальні разові витрати на інтеграцію платформи, а також помірну щомісячну оплату за користування. Наприклад, використання готових платформ на кшталт SaaS-рішень дозволяє отримати повнофункціональний веб-додаток для управління оголошеннями та базою клієнтів без значних фінансових зобов’язань.

Крім того, аутсорсинг розробки може бути ефективним варіантом для середніх бізнесів, які мають доступ до більших бюджетів, але прагнуть уникнути складнощів, пов’язаних із наймом власного ІТ-персоналу. У такому випадку підприємства отримують кастомізовані рішення з фіксованими витратами. Цей підхід дозволяє адаптувати функціонал платформи до специфіки ринку нерухомості, зокрема до автоматизації процесів пошуку та продажу об’єктів.

Для довгострокових проєктів (>3 років) вибір стратегії залежить від масштабу бізнесу, його амбіцій та планів щодо розширення. Великі забудовники та мережі агентств нерухомості можуть інвестувати у створення власного IT-відділу. Хоча цей підхід передбачає значні початкові витрати, він надає повний контроль над розробкою та дозволяє впроваджувати унікальні функції, такі як інтеграція з CRM-системами чи аналітичними інструментами. Власний IT-відділ є найкращим рішенням для бізнесів, які планують безперервно вдосконалювати свої веб-застосунки та розширювати їхній функціонал, забезпечуючи високий рівень кастомізації та автономності.

Середні підприємства, такі як місцеві ріелторські агентства чи регіональні забудовники, у довгостроковій перспективі можуть обрати аутсорсинг. Хоча витрати на початкову розробку у цій моделі вищі, відсутність постійних витрат на утримання внутрішньої команди робить її привабливою для компаній зі стабільним, але обмеженим бюджетом. Аутсорсинг дозволяє створити веб-застосунок, який враховує унікальні потреби бізнесу, зокрема інтеграцію з локальними платформами для маркетингу або оптимізацію UX/UI-дизайну.

Невеликі підприємства, такі як окремі ріелтори або невеликі агенції, часто стикаються з обмеженнями бюджету. Для них ідеальним рішенням залишається підписка на готові сервіси. Це забезпечує доступ до базового функціоналу без значних початкових інвестицій і дозволяє сфокусуватися на обслуговуванні клієнтів. Швидкість впровадження та низька вартість підтримки роблять цю модель оптимальною для невеликих бізнесів.

Вибір стратегії мінімізації витрат залежить від специфіки бізнесу та його цілей. У короткостроковій перспективі підписка є найекономічнішим рішенням для малих і середніх підприємств. У довгостроковій перспективі великі компанії отримують найбільшу вигоду від створення власного IT-відділу, тоді як середні бізнеси можуть використовувати аутсорсинг для зниження витрат на підтримку. Ретельний аналіз доступних моделей дозволяє

ефективно планувати інвестиції у веб-застосунки, оптимізуючи витрати та підвищуючи конкурентоспроможність.

4.5. Прогнозування майбутніх тенденцій у використанні веб-застосунків у сфері нерухомості.

Інтеграція веб-застосунків у сфері нерухомості стає ключовим драйвером економічного розвитку та оптимізації бізнес-процесів. У контексті швидкого впровадження сучасних технологій, веб-застосунки відкривають значний потенціал для підвищення ефективності операцій, зниження витрат та покращення взаємодії з користувачами. Аналізуючи майбутні тенденції, можна виділити кілька ключових напрямків, які визначатимуть економічний вплив таких рішень у найближчі роки.

За статистичними даними, впровадження веб-застосунків здатне підвищити доходи компаній на 10-15% щорічно. Інноваційні функції, такі як динамічний пошук, інтерактивні картографічні сервіси та персоналізовані рекомендації для користувачів, значно покращують клієнтський досвід. Зокрема, автоматизація операцій дозволяє зменшити витрати на традиційні канали взаємодії, такі як фізичні офіси чи телефонні консультації. За прогнозами, у 2025 році близько 60-70% транзакцій у сфері нерухомості здійснюватимуться через веб-застосунки, що забезпечить додаткове зростання прибутків.

Впровадження веб-застосунків сприяє значному зниженню операційних витрат, зокрема завдяки автоматизації процесів пошуку, реєстрації та управління оголошеннями. Наприклад, системи автоматизації дозволяють зменшити потребу в адміністративному персоналі, що скорочує витрати на 10-15% протягом перших двох років після впровадження.

Веб-застосунки, що використовують персоналізовані рекомендації на основі аналізу даних користувачів, мають суттєвий вплив на середній чек клієнта. За статистикою, платформи, які впроваджують персоналізацію, можуть збільшити середній чек на 12-15%, що зумовлено більш

релевантними пропозиціями для користувачів. У найближчі роки удосконалення алгоритмів машинного навчання та рекомендаційних систем дозволить бізнесам підвищити рівень повторних замовлень та збільшити загальні доходи на 15-18%.

Використання аналітики даних через веб-застосунки змінює підходи до маркетингу. Таргетовані рекламні кампанії дозволяють значно знизити витрати на маркетинг, зберігаючи при цьому високу ефективність. Прогнозується, що витрати на цифровий маркетинг у сфері нерухомості зростатимуть на 10-12% щорічно, проте загальна ефективність рекламних кампаній підвищиться завдяки використанню точних інструментів аналітики та оптимізації.

Прогнозовані тенденції свідчать про те, що веб-застосунки продовжуватимуть залишатися ключовим інструментом для підвищення економічної ефективності та конкурентоспроможності компаній у сфері нерухомості. Інвестиції в інноваційні технології, такі як персоналізація, автоматизація та мобільні платежі, є виправданими у світлі зростаючих потреб клієнтів і нових бізнес-викликів. Зосередженість на інноваціях дозволить компаніям забезпечити довгострокове зростання та адаптацію до змінюваних умов ринку.

4.6. Стратегії масштабування веб-застосунків у секторі нерухомості.

Масштабування веб-застосунків є важливим кроком для досягнення ефективності, підвищення продуктивності та адаптації до мінливих умов ринку. Завдяки інтеграції інноваційних рішень можна забезпечити стабільний ріст і створення конкурентних переваг. У цьому розділі розглянуто основні стратегії масштабування веб-застосунків, включаючи використання хмарних технологій, штучного інтелекту, модульної архітектури та інших підходів.

Хмарні платформи, такі як Amazon Web Services (AWS) і Google Cloud Platform, дозволяють обробляти великі обсяги даних, забезпечувати

надійність і масштабованість системи. Ці технології особливо важливі для ринку нерухомості, оскільки вони допомагають ефективно управляти піковими навантаженнями, такими як сезонне збільшення попиту або промо-кампанії. Автоматичне масштабування хмарної інфраструктури знижує витрати на підтримку серверів і підвищує стабільність роботи системи.

Модульний підхід до побудови веб-застосунків забезпечує їхню гнучкість і можливість швидкого впровадження нових функцій. Наприклад, окремі модулі можуть відповідати за управління оголошеннями, аналітику чи платіжні системи. Така архітектура дозволяє адаптувати систему до нових вимог ринку без значних змін у базовій кодовій базі .

Технології штучного інтелекту (ШІ) та машинного навчання є ключовими для персоналізації та автоматизації процесів. Впровадження рекомендаційних систем, заснованих на аналізі даних користувачів, дозволяє підвищити релевантність контенту та залученість клієнтів. Крім того, чат-боти на основі ШІ знижують навантаження на персонал, забезпечуючи швидку обробку запитів клієнтів у реальному часі.

Масштабування через адаптацію веб-застосунків до специфіки різних регіонів передбачає інтеграцію локальних платіжних систем, підтримку кількох мов і врахування культурних особливостей. Наприклад, для китайського ринку доцільно використовувати AliPay або WeChat Pay, а для європейського — PayPal або SEPA. Такий підхід сприяє розширенню клієнтської бази та забезпеченню відповідності місцевим вимогам.

Масштабування веб-застосунків потребує ретельного планування та впровадження передових технологій. Використання хмарних платформ, модульної архітектури, штучного інтелекту та географічної адаптації дозволяє значно підвищити ефективність роботи системи, забезпечити її стабільність і відповідність ринковим викликам. Ці стратегії створюють умови для сталого розвитку бізнесу та підвищення його конкурентоспроможності на цифровому ринку нерухомості.

ВИСНОВКИ

У сучасних умовах стрімкого розвитку цифрових технологій і соціально-економічних викликів, таких як війна в Україні, веб-додатки відіграють критичну роль у забезпеченні швидкого, зручного та ефективного обслуговування в секторі нерухомості. Це дослідження продемонструвало важливість розробки конкурентоспроможного веб-додатку, який враховує поточні тенденції ринку, технічні інновації та зростаючі потреби користувачів.

Результати роботи підтвердили, що інтуїтивно зрозумілий інтерфейс і передовий технологічний стек є фундаментальними для створення високоякісних продуктів. Використання TypeScript, React, Next.js та Material-UI на фронтенді, а також Express.js, Node.js, Redis, MongoDB, Docker і tRPC на бекенді забезпечує не лише надійність і масштабованість, але й адаптивність до мінливих потреб користувачів. Запропонований підхід дозволяє оптимізувати транзакційні процеси, покращити якість взаємодії з клієнтами та підвищити економічну ефективність рішень.

Ця робота робить значний внесок у розвиток індустрії веб-розробки та ринку нерухомості. Зокрема, дослідження надає цінні рекомендації щодо інтеграції передових технологій, створення функціональних платформ та покращення користувацького досвіду. Практичний результат у вигляді готового до ринку веб-додатку демонструє можливість вирішення як нагальних, так і довгострокових проблем ринку нерухомості.

Таким чином, проведене дослідження не лише розширює теоретичну базу знань про розробку веб-додатків для сектору нерухомості, але й пропонує інноваційний інструмент для вирішення актуальних проблем. Застосування комплексного підходу до проектування, орієнтація на потреби користувачів та впровадження сучасних технологій закладають основу для подальшого розвитку галузі та стимулювання інновацій у сфері цифрових рішень.

ПЕРЕЛІК ПОСИЛАНЬ

1 Wilson G. Exploring the Use of Artificial Intelligence in Personalizing Marketing Campaigns // G. Wilson, O. Johnson, W. Brown. – 2024. – С. 6-11. – Режим доступу: https://www.preprints.org/manuscript/202408.0007/download/final_file

2 Bykovskii G., Dutot V. Machine learning and artificial intelligence in a real estate marketing: a systematic review // Bykovskii G., Dutot V. – 2022. – № 3. – С. 10–15. – Режим доступу: https://www.researchgate.net/publication/365781052_Machine_learning_and_artificial_intelligence_in_a_real_estate_marketing_a_systematic_review

3 The influence of virtual tour on urban visitor using a network approach [Електронний ресурс] / M.Chang, G. Lee, L. J. Hyun, L. Marvin. – 2023. – Режим доступу до ресурсу: https://www.sciencedirect.com/science/article/abs/pii/S147403462300153

4 Patel D., Gupta R., Kumar N. Cloud Solutions in Real Estate Platforms // Journal of Cloud Computing and Data Storage. – 2020. – № 9(2). – С. 23–34.

5 Liu Y., Zhang X., Wang T. Big Data Analytics in the Real Estate Market // Journal of Business Intelligence and Analytics. – 2021. – № 11(5). – С. 34–49.

6 Шатківська А. С. Аналіз сучасного стану ринку нерухомості в Україні [Електронний ресурс] / А. С. Шатківська // Вісник студентського наукового товариства Донецького національного університету. – 2020. – Режим доступу до ресурсу: https://jvestnik-sss.donnu.edu.ua/article/view/8464

7 Zillow. Real Estate Technology Trends [Електронний ресурс]. – Режим доступу: https://www.zillow.com/research

8 Amazon Web Services. Cloud Architecture for Modern Applications [Електронний ресурс]. – Режим доступу: https://aws.amazon.com/architecture

9 Airbnb [Електронний ресурс]. – Режим доступу: https://www.zillow.com/research

10 Rieltor.ua [Електронний ресурс] // Rieltor.ua – Режим доступу до ресурсу: https://rieltor.ua

11 Flatfy [Електронний ресурс] // Flatfy.ua – Режим доступу до ресурсу: https://flatfy.ua

12 Microsoft Visual Studio Code. VS Code documentation [Електронний ресурс]. – Режим доступу: https://code.visualstudio.com/docs

13 Google Material 3 Design guidelines [Електронний ресурс]. – Режим доступу: https://m3.material.io/

14 MongoDB Atlas database [Електронний ресурс]. – Режим доступу: https://www.mongodb.com/products/platform/atlas-database

15 Redis Documentation [Електронний ресурс]. – Режим доступу: https://redis.io/docs/latest/

Додаток А

Код застосунку

Update.tsx

```
     import { FormProvider, SubmitHandler, useForm } from
"react-hook-form";
     import { useQueryClient } from "@tanstack/react-query";
     import { FC, useEffect, useState } from "react";
     import { object, string, TypeOf, array } from "zod";
     import { zodResolver } from "@hookform/resolvers/zod";
     import { toast } from "react-toastify";
     import { AiOutlineUpload } from "react-icons/ai";
     import FileUpload from "../components/FileUpload";
     import useStore from "../store/store";
     import { trpc } from "../trpc";
     import { useNavigate, useParams } from "react-router-dom";
     import {
       Container,
       Typography,
       Grid,
       TextField,
       Select,
       MenuItem,
       FormControl,
       InputLabel,
       Button,
       Box,
       Card,
       CardMedia,
     } from "@mui/material";

     const updatePropertySchema = object({
       type: string().optional(),
       state: string().optional(),
       price: string().optional(),
       address: string().optional(),
       area: string().optional(),
       rooms: string().optional(),
       floor: string().optional(),
       images: array(string()).optional(),
       info: string().optional(),
     });

     type UpdatePropertyInput = TypeOf<typeof
updatePropertySchema>;

     const Update: FC = () => {
       const store = useStore();
       const [images, setImages] = useState<string[]>([]);
       const { darkMode } = store;
```

```
        const setImagesArray = (newImages: string | string[]) => {
          if (Array.isArray(newImages)) {
            setImages(newImages);
          } else {
            setImages([newImages]);
          }
        };

        const navigate = useNavigate();

        const { propertyId } = useParams<{ propertyId: string
}>();

        if (!propertyId) {
          return null;
        }

        const { data: property } = trpc.getProperty.useQuery(
          { propertyId },
          {
            select: (data) => data.data.property,
            retry: 1,
            onSuccess: (data) => {
              setImages(data.images);
              console.log("Fetched property successfully");
            },
            onError: (error) => {
              toast.error(error.message, {
                position: "top-right",
              });
            },
          }
        );

        const queryClient = useQueryClient();
        const { mutate: updateProperty } =
trpc.updateProperty.useMutation({
          onSuccess: (data) => {
            console.log("Update successful:", data);
            queryClient.refetchQueries(["getProperties"]);
            toast.success("Property updated Successfully", {
              position: "top-right",
            });
            navigate(`/properties/${propertyId}`);
          },
          onError: (error) => {
            console.error("Update failed:", error);
            toast.error(error.message, {
              position: "top-right",
            });
          },
        });
```

```
        const validateInput = useForm<UpdatePropertyInput>({
          resolver: zodResolver(updatePropertySchema),
        });

        const {
          register,
          handleSubmit,
          setValue,
          watch,
          formState: { errors, isSubmitSuccessful },
        } = validateInput;

        useEffect(() => {
          if (property) {
            setValue("type", property.type);
            setValue("state", property.state);
            setValue("price", property.price);
            setValue("address", property.address);
            setValue("area", property.area);
            setValue("rooms", property.rooms);
            setValue("floor", property.floor);
            setValue("info", property.info);
          }
        }, [property, setValue]);

        useEffect(() => {
          if (isSubmitSuccessful) {
            navigate(`/properties/${propertyId}`);
          }
          // eslint-disable-next-line react-hooks/exhaustive-deps
        }, [isSubmitSuccessful]);

        const onSubmitHandler: SubmitHandler<UpdatePropertyInput>
= async (data) => {
          console.log("Submitting data:", data);
          updateProperty({ body: { ...data, images }, params: {
propertyId } });
        };

        const type = watch("type");
        const state = watch("state");
        const price = watch("price");
        const address = watch("address");
        const area = watch("area");
        const rooms = watch("rooms");
        const floor = watch("floor");
        const info = watch("info");

        return (
          <Container
            maxWidth="xl"
```

```
            sx={{
              minHeight: "100vh",
              py: 4,
            }}
          >
            <Typography variant="h3" align="center"
fontWeight="bold" gutterBottom>
              Update your property
            </Typography>
            <FormProvider {...validateInput}>
              <form onSubmit={handleSubmit(onSubmitHandler)}>
                <Grid container spacing={3}>
                  <Grid item xs={12} sm={6} md={3}>
                    <FormControl fullWidth>
                      <InputLabel>Type</InputLabel>
                      <Select
                        {...register("type")}
                        value={type || ""}
                        label="Type"
                        onChange={(e) => setValue("type",
e.target.value)}
                      >
                        <MenuItem value="house">House</MenuItem>
                        <MenuItem
value="apartment">Apartment</MenuItem>
                        <MenuItem
value="townhouse">Townhouse</MenuItem>
                      </Select>
                    </FormControl>
                  </Grid>

                  <Grid item xs={12} sm={6} md={3}>
                    <FormControl fullWidth>
                      <InputLabel>State</InputLabel>
                      <Select
                        {...register("state")}
                        value={state || ""}
                        label="State"
                        onChange={(e) => setValue("state",
e.target.value)}
                      >
                        <MenuItem value="new">New house</MenuItem>
                        <MenuItem value="secondary">Secondary
house</MenuItem>
                      </Select>
                    </FormControl>
                  </Grid>

                  <Grid item xs={12} sm={6} md={3}>
                    <TextField
                      {...register("price")}
                      value={price || ""}
```

```
                    label="Price"
                    placeholder="0"
                    type="number"
                    fullWidth
                    InputLabelProps={{ shrink: true }}
                    onChange={(e) => setValue("price",
e.target.value)}
                  />
                </Grid>

                <Grid item xs={12} sm={6} md={3}>
                  <TextField
                    {...register("area")}
                    value={area || ""}
                    label="Area"
                    placeholder="0"
                    type="number"
                    fullWidth
                    InputLabelProps={{ shrink: true }}
                    onChange={(e) => setValue("area",
e.target.value)}
                  />
                </Grid>

                <Grid item xs={12} sm={6} md={3}>
                  <TextField
                    {...register("address")}
                    value={address || ""}
                    label="Address"
                    placeholder="Enter an address"
                    fullWidth
                    InputLabelProps={{ shrink: true }}
                    onChange={(e) => setValue("address",
e.target.value)}
                  />
                </Grid>

                <Grid item xs={12} sm={6} md={3}>
                  <TextField
                    {...register("rooms")}
                    value={rooms || ""}
                    label="Rooms"
                    placeholder="0"
                    type="number"
                    fullWidth
                    InputLabelProps={{ shrink: true }}
                    onChange={(e) => setValue("rooms",
e.target.value)}
                  />
                </Grid>

                <Grid item xs={12} sm={6} md={3}>
```

```
                    <TextField
                      {...register("floor")}
                      value={floor || ""}
                      label="Floor"
                      placeholder="0"
                      type="number"
                      fullWidth
                      InputLabelProps={{ shrink: true }}
                      onChange={(e) => setValue("floor",
e.target.value)}
                    />
                  </Grid>

                  <Grid item xs={12} sm={6} md={3}>
                    <TextField
                      {...register("info")}
                      value={info || ""}
                      label="Additional description"
                      placeholder="Additional description"
                      fullWidth
                      InputLabelProps={{ shrink: true }}
                      onChange={(e) => setValue("info",
e.target.value)}
                    />
                  </Grid>

                  <Grid item xs={12}>
                    <FileUpload
                      images={images}
                      setImages={setImagesArray}
                      label="Upload Property Images"
                      Icon={AiOutlineUpload}
                    />
                  </Grid>

                  <Grid item xs={12}>
                    <Grid container spacing={2}>
                      {images.map((img, index) => (
                        <Grid item xs={12} sm={6} md={4}
key={index}>
                          <Card>
                            <CardMedia
                              component="img"
                              height="250"
                              image={img}
                              alt={`Property image ${index + 1}`}
                              sx={{
                                objectFit: "cover",
                                width: "100%",
                                height: "250px",
                              }}
                            />
```

```
                        </Card>
                      </Grid>
                    ))}
                  </Grid>
                </Grid>

                <Grid item xs={12}>
                  <Box sx={{ display: "flex", justifyContent:
"center", gap: 2 }}>
                    <Button
                      type="submit"
                      variant="contained"
                      color="primary"
                      size="large"
                    >
                      Update Property
                    </Button>
                    <Button
                      type="button"
                      variant="outlined"
                      color="secondary"
                      size="large"
                      onClick={() =>
navigate(`/properties/${propertyId}`)}
                    >
                      Cancel
                    </Button>
                  </Box>
                </Grid>
              </Grid>
            </form>
          </FormProvider>
        </Container>
      );
    };

    export default Update;

    SingleProperty.tsx
    import { useParams } from "react-router-dom";
    import { trpc } from "../trpc";
    import { toast } from "react-toastify";
    import useStore from "../store/store";
    import PropertyActions from "../components/PropertyAction";
    import {
      Container,
      Typography,
      Grid,
      Box,
      Button,
      Card,
      CardMedia,
```

```
        Chip,
      } from "@mui/material";

      const SingleProperty = () => {
        const { propertyId } = useParams();
        const store = useStore();
        const user = store.authorizedUser;
        const { darkMode } = store;
        const { data: property } = trpc.getProperty.useQuery(
          { propertyId },
          {
            select: (data) => data.data.property,
            retry: 1,
            onSuccess: (data) => {
              console.log("Fetched property successfully");
            },
            onError: (error) => {
              toast(error.message, {
                type: "error",
                position: "top-right",
              });
            },
          }
        );

        if (!property) {
          return null;
        }

        return (
          <Container
            maxWidth="lg"
            sx={{
              minHeight: "100vh",
              py: 4,
            }}
          >
            <Typography variant="h3" align="center"
fontWeight="bold" gutterBottom>
              {property.type.charAt(0).toUpperCase() +
property.type.slice(1)} for
              sale
            </Typography>
            <Grid container spacing={3}>
              {property.images.map((img, index) => (
                <Grid item xs={12} sm={6} md={4} key={index}>
                  <Card>
                    <CardMedia
                      component="img"
                      height="250"
                      image={img}
                      alt="House interior preview"
```

```
                    sx={{ objectFit: "cover", width: "100%",
height: "250px" }}
                  />
                </Card>
              </Grid>
            ))}
          </Grid>
          <Box
            sx={{
              display: "flex",
              justifyContent: "space-between",
              alignItems: "flex-start",
              py: 4,
            }}
          >
            <Box sx={{ flex: 1 }}>
              <Typography fontWeight="bold" variant="h1">
                {property.price} $
              </Typography>
              <Box
                sx={{
                  display: "flex",
                  justifyContent: "space-between",
                  flexWrap: "wrap",
                  gap: 1,
                  mt: 2,
                }}
              >
                <Chip
                  variant="tonal"
                  color="primary"
                  label={`Apartment area: ${property.area} m²`}
                />
                <Chip variant="tonal" color="primary"
label={property.address} />
                <Chip
                  variant="tonal"
                  color="primary"
                  label={`Quantity of rooms: ${property.rooms}`}
                />
                <Chip
                  variant="tonal"
                  color="primary"
                  label={`Floor: ${property.floor}`}
                />
              </Box>
            </Box>
            <Box sx={{ ml: 2 }}>
              <PropertyActions user={user}
propertyId={propertyId} />
            </Box>
          </Box>
```

```
            <Grid container spacing={3} sx={{ py: 4 }}>
              <Grid item xs={12} sm={6}>
                <iframe

src="https://www.google.com/maps/embed?pb=!1m18!1m12!1m3!1d34233
6.0789987482!2d33.03688720238609!3d47.907350174256464!2m3!1f0!2f
0!3f0!3m2!1i1024!2i768!4f13.1!3m3!1m2!1s0x40dadfe03154ab7b%3A0xb
0fa3a177d6b186e!2z0JrRgNC40LLQvtC5INCg0L7Qsywg0JTQvdC10L_RgNC-0L
_QtdGC0YDQvtCy0YHQutCw0Y8g0L7QsdC70LDRgdGC0YwsIDUwMDAw!5e0!3m2!1
sru!2sua!4v1685975068189!5m2!1sru!2sua"
                  style={{ border: 0, width: "100%", height:
"300px" }}
                  loading="lazy"
                ></iframe>
              </Grid>
              <Grid item xs={12} sm={6}>
                <Typography variant="body1" align="justify">
                  {property.info}
                </Typography>
              </Grid>
            </Grid>
            <Grid container spacing={3} sx={{ py: 4 }}>
              <Grid item xs={12} sm={4}>
                <Button
                  variant="contained"
                  color="primary"
                  startIcon={
                    <img
                      src="/images/Agent.png"
                      alt="Agent"
                      style={{ width: 40, height: 40,
borderRadius: "50%" }}
                    />
                  }
                  fullWidth
                  sx={{ height: "100%" }}
                >
                  {`${property.author.name}
${property.author.surname}`}
                </Button>
              </Grid>
              <Grid item xs={12} sm={4}>
                <Button
                  variant="outlined"
                  color="secondary"
                  fullWidth
                  sx={{ height: "100%" }}
                >
                  Send a message
                </Button>
              </Grid>
              <Grid item xs={12} sm={4}>
```

```
                <Button
                  variant="contained"
                  color="success"
                  fullWidth
                  sx={{ height: "100%" }}
                >
                  {property.author.phonenumber}
                </Button>
              </Grid>
            </Grid>
          </Container>
        );
      };

      export default SingleProperty;

      Sell.tsx
      import { FormProvider, SubmitHandler, useForm } from
"react-hook-form";
      import { useQueryClient } from "@tanstack/react-query";
      import { FC, useEffect, useState } from "react";
      import { object, string, TypeOf, array } from "zod";
      import { zodResolver } from "@hookform/resolvers/zod";
      import { toast } from "react-toastify";
      import { AiOutlineUpload } from "react-icons/ai";
      import FileUpload from "../components/FileUpload";
      import useStore from "../store/store";
      import { trpc } from "../trpc";
      import { TRPCError } from "@trpc/server";
      import {
        Container,
        Typography,
        Grid,
        TextField,
        Select,
        MenuItem,
        FormControl,
        InputLabel,
        Button,
        Box,
        Card,
        CardMedia,
      } from "@mui/material";

      const createPropertySchema = object({
        type: string({
          required_error: "Type of property is required",
        }).toLowerCase(),
        state: string({
          required_error: "State of property is required",
        }).toLowerCase(),
        price: string({ required_error: "Price is required" }),
```

```
      address: string({ required_error: "Address is required"
}),
      area: string({ required_error: "Area of property is
required" }),
      rooms: string({ required_error: "Quantity of rooms is
required" }),
      floor: string({ required_error: "Floor is required" }),
      images: array(string()).optional(),
      info: string().optional(),
    });

    type CreatePropertyInput = TypeOf<typeof
createPropertySchema>;

    const Sell: FC = () => {
      const [images, setImages] = useState<string[]>([]);
      const store = useStore();
      const { darkMode } = store;
      const queryClient = useQueryClient();
      const { mutate: createProperty } =
trpc.createProperty.useMutation({
        onSuccess: (data) => {
          queryClient.refetchQueries([["getProperties"]]);
          toast("Property created successfully", {
            type: "success",
            position: "top-right",
          });
        },
        onError: (error) => {
          throw new TRPCError({
            code: "CONFLICT",
            message: error.message,
          });
        },
      });

      const validateInput = useForm<CreatePropertyInput>({
        resolver: zodResolver(createPropertySchema),
      });

      const {
        register,
        reset,
        handleSubmit,
        formState: { errors, isSubmitSuccessful },
      } = validateInput;

      useEffect(() => {
        if (isSubmitSuccessful) {
          reset();
        }
        // eslint-disable-next-line react-hooks/exhaustive-deps
```

```
      }, [isSubmitSuccessful]);

      const onSubmitHandler: SubmitHandler<CreatePropertyInput>
= async (data) => {
        createProperty({ ...data, images });
      };

      return (
        <Container
          maxWidth="xl"
          sx={{
            minHeight: "100vh",
            py: 4,
          }}
        >
          <Typography variant="h3" align="center"
fontWeight="bold" gutterBottom>
            Sell your property
          </Typography>
          <FormProvider {...validateInput}>
            <form onSubmit={handleSubmit(onSubmitHandler)}>
              <Grid container spacing={3}>
                <Grid item xs={12} sm={6} md={3}>
                  <FormControl fullWidth>
                    <InputLabel>Type</InputLabel>
                    <Select {...register("type")} label="Type"
required>
                      <MenuItem value="house">House</MenuItem>
                      <MenuItem
value="apartment">Apartment</MenuItem>
                      <MenuItem
value="townhouse">Townhouse</MenuItem>
                    </Select>
                  </FormControl>
                </Grid>

                <Grid item xs={12} sm={6} md={3}>
                  <FormControl fullWidth>
                    <InputLabel>State</InputLabel>
                    <Select {...register("state")} label="State"
required>
                      <MenuItem value="new">New house</MenuItem>
                      <MenuItem value="secondary">Secondary
house</MenuItem>
                    </Select>
                  </FormControl>
                </Grid>

                <Grid item xs={12} sm={6} md={3}>
                  <TextField
                    {...register("price")}
                    label="Price"
```

```
    placeholder="0"
    type="number"
    fullWidth
    required
  />
</Grid>

<Grid item xs={12} sm={6} md={3}>
  <TextField
    {...register("area")}
    label="Area"
    placeholder="0"
    type="number"
    fullWidth
    required
  />
</Grid>

<Grid item xs={12} sm={6} md={3}>
  <TextField
    {...register("address")}
    label="Address"
    placeholder="Enter an address"
    fullWidth
    required
  />
</Grid>

<Grid item xs={12} sm={6} md={3}>
  <TextField
    {...register("rooms")}
    label="Rooms"
    placeholder="0"
    type="number"
    fullWidth
    required
  />
</Grid>

<Grid item xs={12} sm={6} md={3}>
  <TextField
    {...register("floor")}
    label="Floor"
    placeholder="0"
    type="number"
    fullWidth
    required
  />
</Grid>

<Grid item xs={12} sm={6} md={3}>
  <TextField
```

```
                    {...register("info")}
                    label="Additional description"
                    placeholder="Additional description"
                    fullWidth
                    multiline
                    rows={1}
                  />
                </Grid>

                <Grid item xs={12}>
                  <FileUpload
                    images={images}
                    setImages={setImages}
                    label="Upload Property Images"
                    Icon={AiOutlineUpload}
                  />
                </Grid>

                <Grid item xs={12}>
                  <Grid container spacing={2}>
                    {images.map((img, index) => (
                      <Grid item xs={12} sm={6} md={4}
key={index}>
                        <Card>
                          <CardMedia
                            component="img"
                            height="250"
                            image={img}
                            alt={`Property image ${index + 1}`}
                            sx={{
                              objectFit: "cover",
                              width: "100%",
                              height: "250px",
                            }}
                          />
                        </Card>
                      </Grid>
                    ))}
                  </Grid>
                </Grid>

                <Grid item xs={12}>
                  <Box sx={{ display: "flex", justifyContent:
"center", gap: 2 }}>
                    <Button
                      type="submit"
                      variant="contained"
                      color="primary"
                      size="large"
                    >
                      Save and upload
                    </Button>
```

```
                </Box>
              </Grid>
            </Grid>
          </form>
        </FormProvider>
      </Container>
    );
  };

  export default Sell;

  Search.tsx
  import { useState } from "react";
  import { toast } from "react-toastify";
  import useStore from "../store/store";
  import { trpc } from "../trpc";
  import Property from "../components/Property";
  import {
    Box,
    Button,
    Container,
    FormControl,
    Grid,
    InputLabel,
    MenuItem,
    Select,
    TextField,
    Typography,
  } from "@mui/material";

  const Buy = () => {
    const { data: properties } =
trpc.getProperties.useQuery(undefined, {
      select: (data) => data.data.properties,
      retry: 1,
      onSuccess: (data) => {
        console.log("Successfully fetched properties");
      },
      onError: (error) => {
        toast(error.message, {
          type: "error",
          position: "top-right",
        });
      },
    });

    const store = useStore();
    const { darkMode } = store;

    const [searchParams, setSearchParams] = useState({
      propertyType: store.searchParams?.propertyType || "",
      propertyState: store.searchParams?.propertyState || "",
```

```
        propertyAddress: store.searchParams?.propertyAddress ||
"",
        propertyAreaMin: "",
        propertyAreaMax: "",
        propertyPriceMin: "",
        propertyPriceMax: "",
        propertyRooms: "",
        propertyFloorMin: "",
        propertyFloorMax: "",
      });

      const [filteredProperties, setFilteredProperties] =
useState(properties);

      const handleFilterProperties = () => {
        const filtered = properties.filter((property) => {
          if (
            searchParams.propertyType &&
            searchParams.propertyType !== property.type
          ) {
            return false;
          }
          if (
            searchParams.propertyState &&
            searchParams.propertyState !== property.state
          ) {
            return false;
          }
          if (
            searchParams.propertyAddress &&

!property.address.includes(searchParams.propertyAddress)
          ) {
            return false;
          }
          if (
            searchParams.propertyAreaMin &&
            property.area < Number(searchParams.propertyAreaMin)
          ) {
            return false;
          }
          if (
            searchParams.propertyAreaMax &&
            property.area > Number(searchParams.propertyAreaMax)
          ) {
            return false;
          }
          if (
            searchParams.propertyPriceMin &&
            property.price <
Number(searchParams.propertyPriceMin)
          ) {
```

```
            return false;
          }
          if (
            searchParams.propertyPriceMax &&
            property.price >
Number(searchParams.propertyPriceMax)
          ) {
            return false;
          }
          if (
            searchParams.propertyFloorMin &&
            property.floor <
Number(searchParams.propertyFloorMin)
          ) {
            return false;
          }
          if (
            searchParams.propertyFloorMax &&
            property.floor >
Number(searchParams.propertyFloorMax)
          ) {
            return false;
          }
          if (
            searchParams.propertyRooms &&
            Number(property.rooms) !==
Number(searchParams.propertyRooms)
          ) {
            return false;
          }

          return true;
        });
        setFilteredProperties(filtered);
      };

      const handleReset = () => {
        setSearchParams({
          propertyType: "",
          propertyState: "",
          propertyAddress: "",
          propertyAreaMin: "",
          propertyAreaMax: "",
          propertyPriceMin: "",
          propertyPriceMax: "",
          propertyRooms: "",
          propertyFloorMin: "",
          propertyFloorMax: "",
        });
        setFilteredProperties(properties);
      };
```

```
        return (
          <Container
            maxWidth="xl"
            sx={{
              minHeight: "100vh",
              py: 4,
            }}
          >
            <Typography variant="h3" align="center"
fontWeight="bold" gutterBottom>
              Property For Sale
            </Typography>
            <Grid
              container
              spacing={2}
              sx={{
                mb: 4,
              }}
            >
              <Grid item xs={12} sm={6} md={4} lg={2.4}>
                <FormControl fullWidth>
                  <InputLabel>Property Type</InputLabel>
                  <Select
                    value={searchParams.propertyType}
                    onChange={(e) =>
                      setSearchParams({
                        ...searchParams,
                        propertyType: e.target.value,
                      })
                    }
                    label="Property Type"
                  >
                    <MenuItem value="house">House</MenuItem>
                    <MenuItem
value="apartment">Apartment</MenuItem>
                    <MenuItem
value="townhouse">Townhouse</MenuItem>
                  </Select>
                </FormControl>
              </Grid>

              <Grid item xs={12} sm={6} md={4} lg={2.4}>
                <FormControl fullWidth>
                  <InputLabel>Property State</InputLabel>
                  <Select
                    value={searchParams.propertyState}
                    onChange={(e) =>
                      setSearchParams({
                        ...searchParams,
                        propertyState: e.target.value,
                      })
                    }
```

```
                    label="Property State"
                  >
                    <MenuItem value="new">New Building</MenuItem>
                    <MenuItem value="secondary">Secondary
Building</MenuItem>
                  </Select>
                </FormControl>
              </Grid>

              <Grid item xs={12} sm={6} md={4} lg={2.4}>
                <TextField
                  placeholder="Enter an address"
                  value={searchParams.propertyAddress}
                  onChange={(e) =>
                    setSearchParams({
                      ...searchParams,
                      propertyAddress: e.target.value,
                    })
                  }
                  label="Address"
                  fullWidth
                />
              </Grid>

              <Grid item xs={12} sm={6} md={4} lg={2.4}>
                <TextField
                  placeholder="Min Area"
                  value={searchParams.propertyAreaMin}
                  onChange={(e) =>
                    setSearchParams({
                      ...searchParams,
                      propertyAreaMin: e.target.value,
                    })
                  }
                  label="Min Area"
                  type="number"
                  fullWidth
                />
              </Grid>

              <Grid item xs={12} sm={6} md={4} lg={2.4}>
                <TextField
                  placeholder="Max Area"
                  value={searchParams.propertyAreaMax}
                  onChange={(e) =>
                    setSearchParams({
                      ...searchParams,
                      propertyAreaMax: e.target.value,
                    })
                  }
                  label="Max Area"
                  type="number"
```

```
    fullWidth
  />
</Grid>

<Grid item xs={12} sm={6} md={4} lg={2.4}>
  <TextField
    placeholder="Min Price"
    value={searchParams.propertyPriceMin}
    onChange={(e) =>
      setSearchParams({
        ...searchParams,
        propertyPriceMin: e.target.value,
      })
    }
    label="Min Price"
    type="number"
    fullWidth
  />
</Grid>

<Grid item xs={12} sm={6} md={4} lg={2.4}>
  <TextField
    placeholder="Max Price"
    value={searchParams.propertyPriceMax}
    onChange={(e) =>
      setSearchParams({
        ...searchParams,
        propertyPriceMax: e.target.value,
      })
    }
    label="Max Price"
    type="number"
    fullWidth
  />
</Grid>

<Grid item xs={12} sm={6} md={4} lg={2.4}>
  <TextField
    placeholder="Rooms"
    value={searchParams.propertyRooms}
    onChange={(e) =>
      setSearchParams({
        ...searchParams,
        propertyRooms: e.target.value,
      })
    }
    label="Rooms"
    type="number"
    fullWidth
  />
</Grid>
```

```
<Grid item xs={12} sm={6} md={4} lg={2.4}>
  <TextField
    placeholder="Min Floor"
    value={searchParams.propertyFloorMin}
    onChange={(e) =>
      setSearchParams({
        ...searchParams,
        propertyFloorMin: e.target.value,
      })
    }
    label="Min Floor"
    type="number"
    fullWidth
  />
</Grid>

<Grid item xs={12} sm={6} md={4} lg={2.4}>
  <TextField
    placeholder="Max Floor"
    value={searchParams.propertyFloorMax}
    onChange={(e) =>
      setSearchParams({
        ...searchParams,
        propertyFloorMax: e.target.value,
      })
    }
    label="Max Floor"
    type="number"
    fullWidth
  />
</Grid>

<Grid item xs={12} sm={6} md={4} lg={12}>
  <Box
    sx={{
      display: "flex",
      gap: 2,
      width: "100%",
      justifyContent: "center",
    }}
  >
    <Button
      variant="contained"
      onClick={(e) => {
        e.preventDefault();
        handleFilterProperties();
      }}
      sx={{ flex: 1 }}
    >
      Search
    </Button>
```

```
                <Button
                  variant="outlined"
                  color="secondary"
                  onClick={(e) => {
                    e.preventDefault();
                    handleReset();
                  }}
                  sx={{ flex: 1 }}
                >
                  Reset
                </Button>
              </Box>
            </Grid>
          </Grid>

          <Grid container spacing={3}>
            <Grid item xs={12}>
              {properties?.length === 0 ? (
                <Typography textAlign="center" role="alert">
                  No properties available.
                </Typography>
              ) : (
                <Grid container spacing={3}>
                  {filteredProperties
                    ? filteredProperties.map((property) => (
                        <Grid item xs={12} sm={6} md={4}
key={property._id}>
                          <Property property={property} />
                        </Grid>
                      ))
                    : properties?.map((property) => (
                        <Grid item xs={12} sm={6} md={4}
key={property._id}>
                          <Property property={property} />
                        </Grid>
                      ))}
                </Grid>
              )}
            </Grid>
          </Grid>
        </Container>
      );
    };

    export default Buy;

    Register.tsx
    import { FormProvider, SubmitHandler, useForm } from
"react-hook-form";
    import { object, string, TypeOf } from "zod";
    import { Link, useNavigate } from "react-router-dom";
    import { zodResolver } from "@hookform/resolvers/zod";
```

```
import { useEffect } from "react";
import { toast } from "react-toastify";
import { trpc } from "../trpc";
import useStore from "../store/store";
import {
  Container,
  Typography,
  Button,
  Grid,
  Card,
  CardContent,
  TextField,
} from "@mui/material";

const registerSchema = object({
  email: string({ required_error: "Email is required" })
    .email("Invalid email")
    .trim()
    .toLowerCase(),
  password: string({ required_error: "Password is required"
})
    .trim()
    .min(8, "Password must be more than 8 characters")
    .max(32, "Password must be less than 32 characters"),
  passwordConfirm: string({
    required_error: "Please, confirm your password",
  }).trim(),
  name: string({ required_error: "Name is required"
}).trim(),
  surname: string({ required_error: "Surname is required"
}).trim(),
  phonenumber: string({ required_error: "Phone number is
required" })
    .length(13, "Phone number must be 12 digits only")
    .regex(/^[\d+]+$/, {
      message: "Only plus at the beginning and numbers are
allowed",
    }),
}).refine((data) => data.password === data.passwordConfirm,
{
  path: ["passwordConfirm"],
  message: "Passwords don't match",
});

export type RegisterInput = TypeOf<typeof registerSchema>;

const Register = () => {
  const navigate = useNavigate();
  const { darkMode } = useStore();

  const { mutate: userRegister } =
trpc.userRegister.useMutation({
```

```
      onSuccess: (data) => {
        toast("Successful registration!", {
          type: "success",
          position: "top-right",
        });
        navigate("/login");
      },
      onError: (error) => {
        toast.error(error.message, {
          type: "error",
          position: "top-right",
        });
      },
    });

    const validateInput = useForm<RegisterInput>({
      resolver: zodResolver(registerSchema),
    });

    const {
      reset,
      handleSubmit,
      formState: { isSubmitSuccessful },
    } = validateInput;

    useEffect(() => {
      if (isSubmitSuccessful) {
        reset();
      }
      // eslint-disable-next-line react-hooks/exhaustive-deps
    }, [isSubmitSuccessful]);

    const onSubmitHandler: SubmitHandler<RegisterInput> =
(values) => {
      userRegister(values);
    };

    return (
      <Container
        maxWidth="xl"
        sx={{
          minHeight: "100vh",
          py: 4,
          display: "flex",
          alignItems: "center",
          justifyContent: "center",
        }}
      >
        <Card
          sx={{
            p: 8,
            width: "70vh",
```

```
        }}
      >
        <CardContent>
          <Typography
            variant="h3"
            align="center"
            fontWeight="bold"
            gutterBottom
          >
            Account registration
          </Typography>
          <FormProvider {...validateInput}>
            <form onSubmit={handleSubmit(onSubmitHandler)}>
              <Grid container spacing={2}>
                <Grid item xs={12}>
                  <TextField
                    fullWidth
                    label="Email"
                    {...validateInput.register("email")}
                    placeholder="Emailexample@gmail.com"
                    variant="outlined"
                    margin="normal"
                    required
                  />
                </Grid>
                <Grid item xs={12}>
                  <TextField
                    fullWidth
                    label="Password"
                    {...validateInput.register("password")}
                    type="password"
                    placeholder="input password"
                    variant="outlined"
                    margin="normal"
                    required
                  />
                </Grid>
                <Grid item xs={12}>
                  <TextField
                    fullWidth
                    label="Confirm password"

{...validateInput.register("passwordConfirm")}
                    type="password"
                    placeholder="confirm password"
                    variant="outlined"
                    margin="normal"
                    required
                  />
                </Grid>
                <Grid item xs={12}>
                  <TextField
```

```
                            fullWidth
                            label="Name"
                            {...validateInput.register("name")}
                            placeholder="First name"
                            variant="outlined"
                            margin="normal"
                            required
                          />
                        </Grid>
                        <Grid item xs={12}>
                          <TextField
                            fullWidth
                            label="Surname"
                            {...validateInput.register("surname")}
                            placeholder="Last name"
                            variant="outlined"
                            margin="normal"
                            required
                          />
                        </Grid>
                        <Grid item xs={12}>
                          <TextField
                            fullWidth
                            label="Phone Number"

{...validateInput.register("phonenumber")}
                            placeholder="+380123456789"
                            variant="outlined"
                            margin="normal"
                            required
                          />
                        </Grid>
                        <Grid item xs={12} textAlign="center">
                          <Typography variant="body2">
                            Already have an account? <Link
to="/login">Login</Link>
                          </Typography>
                        </Grid>
                        <Grid item xs={12} textAlign="center">
                          <Button
                            type="submit"
                            variant="contained"
                            color="primary"
                            sx={{
                              width: "50%",
                              borderRadius: "30px",
                            }}
                          >
                            Sign up
                          </Button>
                        </Grid>
                      </Grid>
```

```
                </form>
              </FormProvider>
            </CardContent>
          </Card>
        </Container>
      );
    };

    export default Register;

    Profile.tsx
    import { AiOutlineUpload } from "react-icons/ai";
    import FileUpload from "../components/FileUpload";
    import { FC, useState } from "react";
    import useStore from "../store/store";
    import { FormProvider, SubmitHandler, useForm } from
"react-hook-form";
    import { toast } from "react-toastify";
    import { object, string, TypeOf } from "zod";
    import { useQueryClient } from "@tanstack/react-query";
    import { trpc } from "../trpc";
    import { zodResolver } from "@hookform/resolvers/zod";
    import {
      Container,
      Typography,
      TextField,
      Button,
      Box,
      Grid,
      Card,
      CardContent,
    } from "@mui/material";

    const updateUserSchema = object({
      password: string()
        .trim()
        .min(8, "Password must be more than 8 characters")
        .max(32, "Password must be less than 32 characters")
        .optional(),
      passwordConfirm: string().trim().optional(),
      name: string().trim().optional(),
      surname: string().trim().optional(),
      phonenumber: string()
        .length(13, "Phone number must be 12 digits only")
        .regex(/^[\d+]+$/, {
          message: "Only plus at the beginning and numbers are
allowed",
        })
        .optional(),
      info: string()
        .max(500, "Info must be less than or equal to 500
characters")
```

```
        .optional(),
      profilepic: string().optional(),
      featured: string().array().optional(),
    });

    type UpdateUserInput = TypeOf<typeof updateUserSchema>;

    const Profile: FC = () => {
      const store = useStore();
      const user = store.authorizedUser;
      const queryClient = useQueryClient();
      const [image, setImage] = useState<string[]>([
        user?.profilepic ? user?.profilepic : "",
      ]);
      const [profilepic, setProfilepic] =
useState<string[]>([]);

      const { darkMode } = store;

      const { mutate: updateUser } =
trpc.updateUser.useMutation({
        onSuccess: (data) => {
          // Update store with new user data
          store.setAuthorizedUser(data);

          // Invalidate and refetch queries
          queryClient.invalidateQueries(["getMe"]);

          toast("User updated successfully", {
            type: "success",
            position: "top-right",
          });
        },
        onError(error) {
          toast(error.message, {
            type: "error",
            position: "top-right",
          });
        },
      });

      const validateInput = useForm<UpdateUserInput>({
        resolver: zodResolver(updateUserSchema),
      });

      const {
        register,
        handleSubmit,
        formState: { errors },
      } = validateInput;
```

```
        const onSubmitHandler: SubmitHandler<UpdateUserInput> =
async (data) => {
          // Only send fields that have values
          const filteredData = Object.fromEntries(
            Object.entries(data).filter(
              ([_, value]) => value !== undefined && value !== ""
            )
          );

          updateUser({
            body: { data: filteredData },
            userParams: { email: user?.email },
          });
        };

        return (
          <Container
            maxWidth="xl"
            sx={{
              minHeight: "100vh",
              py: 4,
            }}
          >
            <Typography variant="h3" align="center"
fontWeight="bold">
              My Profile
            </Typography>
            <Card
              sx={{
                mt: 4,
                p: 2,
              }}
            >
              <CardContent>
                <FormProvider {...validateInput}>
                  <form onSubmit={handleSubmit(onSubmitHandler)}>
                    <Grid container spacing={3}
justifyContent="center" mt={2}>
                      <Grid item xs={12} sm={6}
justifyContent="center">
                        <textarea
                          className="hidden"
                          {...register("profilepic")}
                          value={profilepic[0]}
                        />
                        <FileUpload
                          maxUploads={1}
                          imgSize={500}
                          setImagesForm={setProfilepic}
                          presetImage={image[0]}
                          multiple={false}
                        />
```

```
</Grid>
<Grid item xs={12} sm={6}>
  <TextField
    fullWidth
    label="Name"
    {...register("name")}
    defaultValue={user?.name}
    variant="outlined"
    margin="normal"
    required
  />
  <TextField
    fullWidth
    label="Phone"
    {...register("phonenumber")}
    defaultValue={user?.phonenumber}
    variant="outlined"
    margin="normal"
    required
  />
  <TextField
    fullWidth
    label="Surname"
    {...register("surname")}
    defaultValue={user?.surname}
    variant="outlined"
    margin="normal"
    required
  />
  <TextField
    fullWidth
    label="Email"
    value={user?.email}
    variant="outlined"
    margin="normal"
    InputProps={{
      readOnly: true,
    }}
  />
  <TextField
    fullWidth
    label="New Password"
    {...register("password")}
    variant="outlined"
    margin="normal"
    type="password"
    required
  />
  <TextField
    fullWidth
    label="Confirm New Password"
    {...register("passwordConfirm")}
```

```
                        variant="outlined"
                        margin="normal"
                        type="password"
                        required
                      />
                      <TextField
                        fullWidth
                        label="Additional Info"
                        {...register("info")}
                        variant="outlined"
                        margin="normal"
                        multiline
                        rows={4}
                      />
                    </Grid>
                  </Grid>
                  <Box display="flex" justifyContent="center"
mt={4}>
                    <Button
                      type="submit"
                      variant="contained"
                      color="primary"
                      endIcon={<AiOutlineUpload />}
                    >
                      Save and upload
                    </Button>
                  </Box>
                </form>
              </FormProvider>
            </CardContent>
          </Card>
        </Container>
      );
    };

    export default Profile;
```

Login.tsx

```
    import { Link, useLocation, useNavigate } from
"react-router-dom";
    import { FormProvider, SubmitHandler, useForm } from
"react-hook-form";
    import { object, string, TypeOf } from "zod";
    import { zodResolver } from "@hookform/resolvers/zod";
    import { useEffect } from "react";
    import { toast } from "react-toastify";
    import { trpc } from "../trpc";
    import useStore from "../store/store";
    import {
      Container,
      Typography,
      Box,
```

```
        Button,
        Grid,
        Card,
        CardContent,
        TextField,
      } from "@mui/material";

      const loginSchema = object({
        email: string({ required_error: "Email is required" })
          .email("Invalid email")
          .trim()
          .toLowerCase(),
        password: string({ required_error: "Password is required"
})
          .trim()
          .min(8, "Password must be more than 8 characters")
          .max(32, "Password must be less than 32 characters"),
      });

      export type LoginInput = TypeOf<typeof loginSchema>;

      const Login = () => {
        const navigate = useNavigate();
        const location = useLocation();
        const from = (location.state?.from.pathname as string) ||
"/";
        const { darkMode } = useStore();

        const { mutate: userLogin } = trpc.userLogin.useMutation({
          onSuccess(data) {
            toast("Successful log in", {
              type: "success",
              position: "top-right",
            });
            console.log(data);
          },
          onError(error) {
            toast(error.message, {
              type: "error",
              position: "top-right",
            });
          },
        });

        const validateInput = useForm<LoginInput>({
          resolver: zodResolver(loginSchema),
        });

        const {
          reset,
          handleSubmit,
          formState: { isSubmitSuccessful },
```

```
        } = validateInput;

        useEffect(() => {
          if (isSubmitSuccessful) {
            reset();
            navigate(from);
          }
          // eslint-disable-next-line react-hooks/exhaustive-deps
        }, [isSubmitSuccessful]);

        const onSubmitHandler: SubmitHandler<LoginInput> =
(values) => {
          userLogin(values);
        };

        return (
          <Container
            maxWidth="xl"
            sx={{
              minHeight: "100vh",
              py: 4,
              display: "flex",
              alignItems: "center",
              justifyContent: "center",
            }}
          >
            <Card
              sx={{
                p: 8,
                width: "70vh",
              }}
            >
              <CardContent>
                <Typography
                  variant="h3"
                  align="center"
                  fontWeight="bold"
                  gutterBottom
                >
                  Account login
                </Typography>
                <FormProvider {...validateInput}>
                  <form onSubmit={handleSubmit(onSubmitHandler)}>
                    <Grid container spacing={2}>
                      <Grid item xs={12}>
                        <TextField
                          fullWidth
                          label="Email"
                          {...validateInput.register("email")}
                          placeholder="Emailexample@gmail.com"
                          variant="outlined"
                          margin="normal"
```

```
                        required
                      />
                    </Grid>
                    <Grid item xs={12}>
                      <TextField
                        fullWidth
                        label="Password"
                        {...validateInput.register("password")}
                        type="password"
                        placeholder="input password"
                        variant="outlined"
                        margin="normal"
                        required
                      />
                    </Grid>
                    <Grid item xs={12} textAlign="center">
                      <Typography variant="body2">
                        Don't have an account? <Link
to="/register">Sign up</Link>
                      </Typography>
                    </Grid>
                    <Grid item xs={12} textAlign="center">
                      <Button
                        type="submit"
                        variant="contained"
                        color="primary"
                        sx={{
                          width: "50%",
                          borderRadius: "30px",
                        }}
                      >
                        Log in
                      </Button>
                    </Grid>
                  </Grid>
                </form>
              </FormProvider>
            </CardContent>
          </Card>
        </Container>
      );
    };

    export default Login;

    Listings.tsx
    import { BiTrash } from "react-icons/bi";
    import { FaEdit } from "react-icons/fa";
    import useStore from "../store/store";
    import { trpc } from "../trpc";
    import { toast } from "react-toastify";
```

```
     import PropertyListings from
"../components/PropertyListings";
     import { Container, Typography, Grid, Box, IconButton } from
"@mui/material";

     import Logo from "../components/LogoNotFound";

     const Listings = () => {
       const store = useStore();
       const user = store.authorizedUser;

       const { data: properties } =
trpc.getPropertiesByAuthor.useQuery(
         { email: user?.email },
         {
           select: (data) => data.data.properties,
           retry: 1,
           onSuccess: (data) => {
             console.log("Successfully fetched properties");
           },
           onError: (error) => {
             toast(error.message, {
               type: "error",
               position: "top-right",
             });
           },
         }
       );

       const handleDeleteProperty = async (propertyId: string) =>
{
         await trpc.deleteProperty.mutateAsync({ propertyId });
         toast("Property deleted successfully", {
           type: "success",
           position: "top-right",
         });
       };

       return (
         <Container
           maxWidth="xl"
           sx={{
             minHeight: "100vh",
             py: 4,
           }}
         >
           <Box>
             <Typography variant="h3" align="center"
fontWeight="bold">
               MY LISTINGS
             </Typography>
             {properties?.length === 0 ? (
```

```
                <>
                  <Typography align="center" role="alert">
                    No properties found.
                  </Typography>
                  <Logo className="m-auto" width={600}
height={450} />
                </>
              ) : (
                <Grid
                  container
                  spacing={3}
                  sx={{
                    pt: 4,
                    justifyContent: "space-between",
                  }}
                >
                  {properties?.map((property) => (
                    <Grid item xs={12} sm={6} md={4}
key={property._id}>
                      <PropertyListings property={property} />
                    </Grid>
                  ))}
                </Grid>
              )}
            </Box>
          </Container>
        );
      };

      export default Listings;

      Home.tsx
      import { memo, useState } from "react";
      import { Link } from "react-router-dom";
      import { toast } from "react-toastify";
      import { trpc } from "../trpc";
      import useStore from "../store/store";
      import Property from "../components/Property";
      import {
        Accordion,
        AccordionSummary,
        AccordionDetails,
        Box,
        Button,
        FormControl,
        InputLabel,
        MenuItem,
        Select,
        TextField,
        Typography,
        Container,
        Grid,
```

```
        Card,
        CardContent,
        CardMedia,
      } from "@mui/material";
      import ExpandMoreIcon from "@mui/icons-material/ExpandMore";
      import StepsToBuyProperty from "../components/StepsToBuy";
      import BlogSection from "../components/BlogSections";

      const Home = () => {
        const [propertyType, setPropertyType] =
useState<string>("");
        const [propertyState, setPropertyState] =
useState<string>("");
        const [propertyAddress, setPropertyAddress] =
useState<string>("");

        const store = useStore();
        const { darkMode } = store;

        const handleSetSearchParams = ({
          propertyType,
          propertyState,
          propertyAddress,
        }: {
          propertyType: string;
          propertyState: string;
          propertyAddress: string;
        }) => {
          store.setSearchParams({ propertyType, propertyState,
propertyAddress });
        };

        const { data: properties } =
trpc.getProperties.useQuery(undefined, {
          select: (data) => data.data.properties,
          retry: 1,
          onSuccess: () => {
            console.log("Successfully fetched properties");
          },
          onError: (error) => {
            toast(error.message, {
              type: "error",
              position: "top-right",
            });
          },
        });

        return (
          <Container
            maxWidth="xl"
            component="main"
            sx={{ minHeight: "100vh", py: 4 }}
```

```
        >
          <Box textAlign="center" mb={4}>
            <Typography variant="h3" fontWeight="bold"
gutterBottom>
              FIND YOUR MODERN AFFORDABLE HOME
            </Typography>
          </Box>
          <Box
            component="form"
            sx={{
              display: "grid",
              gridTemplateColumns: {
                xs: "repeat(auto-fit, minmax(200px, 1fr))",
                sm: "repeat(auto-fit, minmax(250px, 1fr))",
              },
              gap: 2,
              mb: 4,
            }}
          >
            <FormControl fullWidth>
              <InputLabel>Property Type</InputLabel>
              <Select
                value={propertyType}
                onChange={(e) =>
setPropertyType(e.target.value)}
                label="Property Type"
              >
                <MenuItem value="house">House</MenuItem>
                <MenuItem value="apartment">Apartment</MenuItem>
                <MenuItem value="townhouse">Townhouse</MenuItem>
              </Select>
            </FormControl>

            <FormControl fullWidth>
              <InputLabel>Property State</InputLabel>
              <Select
                value={propertyState}
                onChange={(e) =>
setPropertyState(e.target.value)}
                label="Property State"
              >
                <MenuItem value="new">New Building</MenuItem>
                <MenuItem value="secondary">Secondary
Building</MenuItem>
              </Select>
            </FormControl>

            <TextField
              placeholder="Enter an address"
              value={propertyAddress}
              onChange={(e) =>
setPropertyAddress(e.target.value)}
```

```
              label="Address"
              fullWidth
            />

            <Button
              variant="contained"
              onClick={() =>
                handleSetSearchParams({
                  propertyType,
                  propertyState,
                  propertyAddress,
                })
              }
              component={Link}
              to="/properties/search"
            >
              Search
            </Button>
          </Box>
          <Box sx={{ position: "relative" }}>
            <Box
              component="ul"
              sx={{
                position: "absolute",
                left: 48, // Adjusts left position (equivalent
to left-12 in Tailwind)
                top: "5%", // Top 20% alignment
                display: "grid",
                gridTemplateColumns: "repeat(3, 1fr)", //
Creates 3 equal columns
                gap: 1.5, // Equivalent to gap-x-6
                fontSize: "1.5rem", // Text-2xl equivalent
                listStyle: "none",
                p: 0,
                m: 0,
              }}
            >
              <Typography
                sx={{
                  color: "#000000",
                  fontWeight: "bold",
                }}
                variant="h1"
              >
                1.2M+
              </Typography>
              <Typography
                sx={{
                  color: "#000000",
                  fontWeight: "bold",
                }}
                variant="h1"
```

```
                >
                  320K+
                </Typography>
                <Typography
                  sx={{
                    color: "#000000",
                    fontWeight: "bold",
                  }}
                  variant="h1"
                >
                  635K+
                </Typography>
                <Typography
                  sx={{
                    color: "#000000",
                  }}
                  variant="h2"
                >
                  Offers available
                </Typography>
                <Typography
                  sx={{
                    color: "#000000",
                  }}
                  variant="h2"
                >
                  Happy customers
                </Typography>
                <Typography
                  sx={{
                    color: "#000000",
                  }}
                  variant="h2"
                >
                  Users
                </Typography>
              </Box>
              <Box
                component="img"
                src="/images/background.jpg"
                alt="Background"
                loading="lazy"
                sx={{
                  pointerEvents: "none", // Disables pointer
events like in Tailwind
                  borderRadius:
"var(--mui-shape-customBorderRadius-lg)", // Rounded-br-[1.5rem]
equivalent
                  borderBottomRightRadius: "600px", //
Rounded-br-[300px] equivalent
                  width: "100%", // Ensures the image spans full
width
```

```
                display: "block",
                height: "900px", // Set the desired height
                objectFit: "cover", // Ensures the image covers
the box
                objectPosition: "bottom", // Cuts the top part
of the image
              }}
            />
          </Box>
          <Box mb={6}>
            <Typography variant="h4" fontWeight="semibold"
gutterBottom>
              RECOMMENDED OFFERS
            </Typography>
            {properties?.length === 0 ? (
              <Typography textAlign="center" role="alert">
                No properties available.
              </Typography>
            ) : (
              <Grid container spacing={3}>
                {properties?.map((property) => (
                  <Grid item xs={12} sm={6} md={4}
key={property._id}>
                    <Property property={property} />
                  </Grid>
                ))}
              </Grid>
            )}
          </Box>
          <StepsToBuyProperty />

          <BlogSection />

          {/* FAQ Section */}
          <Box>
            <Typography
              variant="h4"
              fontWeight="semibold"
              gutterBottom
              textAlign="center"
            >
              FREQUENTLY ASKED QUESTIONS
            </Typography>
            {[
              {
                question:
                  "Can I list my property on multiple
real-estate marketplaces?",
                answer: "Yes, but keep track of listings to
avoid double bookings.",
              },
              {
```

```
                question: "Can I negotiate the price of a
property?",
                answer:
                  "Yes, approach negotiations respectfully with
clear expectations.",
              },
              {
                question: "What should I do if I find a property
I like?",
                answer:
                  "Contact the seller to schedule a visit and
ask any questions.",
              },
              {
                question: "How do I know if a property is in
good condition?",
                answer:
                  "Inspect the property for any damages or
issues before making a purchase.",
              },
              {
                question: "What are the steps to buying a
property?",
                answer:
                  "Find an offer, contact the seller, request a
visit, and make a purchase.",
              },
              {
                question: "How do I know if a property is worth
the price?",
                answer:
                  "Research the market value of similar
properties in the area.",
              },
              {
                question: "What should I consider when buying a
property?",
                answer:
                  "Location, price, condition, and future value
are important factors.",
              },
              // Add other FAQs here
            ].map((faq, index) => (
              <Accordion key={index}>
                <AccordionSummary expandIcon={<ExpandMoreIcon
/>}>
                  {faq.question}
                </AccordionSummary>

<AccordionDetails>{faq.answer}</AccordionDetails>
              </Accordion>
            ))}
```

```
          </Box>
        </Container>
      );
    };

    export default memo(Home);

    Featured.tsx
    import React, { memo } from "react";
    import { Box, Container, Typography, Grid } from
"@mui/material";
    import { toast } from "react-toastify";
    import { trpc } from "../trpc";
    import useStore from "../store/store";
    import Property from "../components/Property";

    import Logo from "../components/LogoNotFound";

    const Featured = () => {
      const store = useStore();
      const user = store.authorizedUser;
      const { darkMode } = store;

      // Fetch properties
      const { data: properties } =
trpc.getProperties.useQuery(undefined, {
        select: (data) => data.data.properties,
        retry: 1,
        onError: (error) => {
          toast(error.message, {
            type: "error",
            position: "top-right",
          });
        },
      });

      // Filter featured properties
      const featuredProperties = properties?.filter((property)
=>
        user?.featured?.includes(property._id)
      );

      return (
        <Container
          maxWidth="xl"
          component="main"
          sx={{
            minHeight: "100vh",
            py: 4,
          }}
        >
          <Box textAlign="center" mb={4}>
```

```
            <Typography variant="h3" fontWeight="bold">
              FEATURED OFFERS
            </Typography>
          </Box>

          {/* Property Grid */}
          <Box mb={6}>
            {featuredProperties?.length === 0 ? (
              <>
                <Typography textAlign="center" role="alert"
variant="h6">
                  No featured properties available.
                </Typography>
                <Logo className="m-auto" width={600}
height={450} />
              </>
            ) : (
              <Grid container spacing={3}>
                {featuredProperties?.map((property) => (
                  <Grid item xs={12} sm={6} md={4}
key={property._id}>
                    <Property property={property} />
                  </Grid>
                ))}
              </Grid>
            )}
          </Box>
        </Container>
      );
    };

    export default memo(Featured);
```