МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)
09 червня 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавр

зі спеціальності 122 – Комп'ютерних наук,
освітньо- професійної програми «Інформатика»
на тему: «Веб-орієнтована система інформаційної підтримки діяльності рієлтерської агенції»
здобувача групи ІН-93 Фоміна Микити Костянтиновича

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

_____ Микита ФОМІН
(підпис)

Керівник
доцент,
кандидат технічних наук Ігор ШЕЛЕХОВ _____
(підпис)

Суми – 2023

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавра

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІН-93 Фоміна Микити Костянтиновича

1. Тема роботи: «Веб-орієнтована система інформаційної підтримки діяльності рієлтерської агенції.»
затверджую наказом по СумДУ від *«01» червня 2023 р. № 0475-VI*
2. Термін здачі здобувачем кваліфікаційної роботи *до 09 червня 2023 року*
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1) Аналіз основних аспектів проблеми, аналіз конкурентів та потреб аудиторії, постановка й формування завдань дослідження. 2) Огляд існуючих інструментів, що використовуються для розробки систем та вибір серед них оптимальних. 3) Розроблення веб-орієнтованої системи інформаційної підтримки. 4) Аналіз та виклад результатів
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
6. Консультанти до проєкту (роботи), із значенням розділів проєкту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «____» _____ 20 ___ р.

Завдання прийняв до виконання _____ (підпис) Керівник _____ (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз основних аспектів проблеми, аналіз конкурентів та потреб аудиторії, постановка й формування завдань дослідження*		
2	*Огляд існуючих інструментів, що використовуються для розробки систем та вибір серед них оптимальних*		
3	*Розроблення веб-орієнтованої системи інформаційної підтримки.*		
4	*Аналіз та виклад результатів*		
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*		

Здобувач вищої освіти _____ (підпис) Керівник _____ (підпис)

АНОТАЦІЯ

Записка: 49 стр., 24 рис., 3 додатки, 32 використаних джерел.

Обґрунтування актуальності теми роботи – Тема кваліфікаційної роботи є актуальною, так як присвячена автоматизації бізнес процесів діяльності рієлтерської агенції шляхом застосування інформаційних технологій.

Об'єкт дослідження – процес автоматизації інформаційної підтримки.

Мета роботи – розробка веб-системи інформаційної підтримки діяльності рієлтерської агенції з використанням сучасних мов програмувань та технологій.

Методи дослідження – моделі та методи електронної комерції у галузі нерухомості та методи й алгоритми побудови веб-систем.

Результати – розроблено веб-систему інформаційної підтримки діяльності рієлтерської агенції, яка надає користувачам можливість працювати з послугами у галузі нерухомості, а менеджерам управлінню бізнес процесами агенції.

ВЕБ-СИСТЕМА, РІЄЛТЕРСЬКА АГЕНЦІЯ, ІНФОРМАЦІЙНА ПІДТРИМКА, NODEJS, REACTJS

ЗМІСТ

ВСТУП

Актуальність. Тема кваліфікаційної роботи є актуальною, так як присвячена автоматизації бізнес процесів діяльності рієлтерської агенції шляхом застосування інформаційних технологій.

Об’єкт дослідження. Процес автоматизації інформаційної підтримки.

Предмет дослідження. Методологія розробки веб-систем для управління інформаційною підтримкою в межах організації.

Гіпотеза. Автоматизації надання послуг з нерухомості можна досягнути завдяки використанню інформаційних технологій , що реалізують систему управління інформаційною підтримкою рієлтерської агенції.

Наукова новизна. Описане у даній роботі програмне рішення дозволить ефективно автоматизувати системи надання послуг з нерухомості, дозволяючи автоматизувати внутрішні процеси та спростити аналіз та управління статистичними даними.

Структура. Дана робота складається зі вступу, аналітичного огляду, постановки задачі, вибору технологій та інструментів розв’язання поставленої задачі, опису програмного забезпечення веб системи, висновків, списку використаних джерел та додатків.

1 ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Аналіз основних аспектів проблеми

Наразі сучасний світ важко уявити без інформаційних технологій. Всесвітня інформаційна павутина поєднує мільярди користувачів по всьому світу, надає можливості розвитку різних сфер суспільства, економіки, освіти, комунікацій, промисловості, розвитку бізнес комерції. На сьогоднішній день Інтернет полегшує процеси взаємодії з електронною комерцією такі як замовлення квитків в кінотеатр, авіаквитків, автомобілів та підбір нерухомості не виходячи з дому [1,2].

З появою та розвитку інформаційних технологій електронна комерція почала розширюватись, змушуючи власників компаній підпорядковуватися актуальним нововведенням та розвиватися в напрямку онлайн продаж. На сьогоднішній день інтерес до електронної комерції стає дедалі більшим, адже купувати, продавати та рекламувати продукцію в інтернеті набагато зручніше, простіше, ніж робити це офлайн. Електронна бізнес комерція (або e-Commerce) – сфера економіки, в якій реклама, просування, торгові та фінансові угоди здійснюються безпосередньо в Інтернеті. З боку власників бізнесу, виробників та постачальників е-комерція більше підходить для просування та надання своїх товарів чи послуг через мережу Інтернет [3].

Електронна комерція у сфері постачання послуг стала одним із найпривабливіших сегментів для залучення інвестицій. За статистикою, кожен другий українець купує нині товари та послуги онлайн. Крім цього з появою Інтернет маркетингу електронна комерція увійшла у список нововведень Індустрії 4.0 – провідний тренд «Четвертої революцій». Характерними рисами стали повністю автоматизовані системи виробництва та постачання послуг, на яких керівництво всіма процесами здійснюється в режимі реального часу і з урахуванням мінливим зовнішніх умов. Фізичні процеси заміняються

автоматизованими складними інформаційними системами, які відкриті для використання клієнтами та партнерами[4,5] .

Інформаційна система – це організаційно впорядкована сукупність засобів, інформаційних ресурсів та інформаційних технологій, зокрема з використанням засобів обчислювальної техніки і зв'язку, що реалізовують такі інформаційні процеси, як отримання вхідних даних; обробка цих даних, видача результату для досягнення поставленої мети. [6] Інформаційні системи можна класифікувати за різними ознаками (рис. 1.1):

Рисунок 1.1 – Класифікація інформаційних систем

Інформаційна система має декілька підвидів. Одним із таких підвидів є автоматизована інформаційна система. Вона характеризується низкою компонентів, які є обов’язковими в розробці структури такої системи:

- Апаратне забезпечення. До цього компоненту відноситься сам комп’ютер і вся його допоміжна апаратура. Іншими обладнаннями можуть бути пристрої введення, виведення та зберігання даних.
- Програмне забезпечення. Цей компонент включає в себе програми, за допомогою яких буде впроваджена основна логіка роботи

інформаційної системи.

- Дані. Такий компонент описує основні моделі систем управління базами даних користувачів.
- Люди. Кожна система потребує людей для коректного управління отриманої інформації. Цей компонент найбільше впливає на коректну роботу інформаційної системи та успіх бізнесу на ринку комерції.
- Зворотній зв’язок. Компонент включає в себе можливість взаємозв’язку клієнтів та системи та системи та менеджерів (наприклад рієлтерська агенція).

Завдяки цим компонентам підпорядковуються бізнес процеси компанії та задовольняються потреби користувачів, які можуть бути потенційними клієнтами в подальшому розвитку бізнесу [7].

Саме така інформаційна система буде використовуватися при створені веб-ресурсу рієлтерської агенції. В такий спосіб можливо залучити нових клієнтів, утримати існуючих та надати їм найкращі умови, які будуть задовольняти їх потреби, при цьому автоматизувати систему постачання послуг нерухомості та взаємодіяти з клієнтами.

1.2 Аналіз конкурентів

В наш час аналіз конкурентів є найважливішим аспектом перед створенням онлайн бізнесу та просування його на комерційний ринок. Таким чином можна проаналізувати основні проблеми конкурентів в напрямку розробки та дизайну, зрозуміти основну бізнес логіку конкурента та знайти креативні ідеї для створення своєї автоматизованої інформаційної системи.

Зазвичай рієлтерські агенції побудовані по принципу повної офлайн взаємодії з клієнтами. Проте на сьогодні економія часу є невід'ємною складовою людського життя. Тому основною задачею при розробці власної системи є зекономити час клієнта при підборі або надання своєї нерухомості для агенції та

взаємодії з потенційним клієнтом в онлайн режимі. Взявши це до уваги було проведено аналіз конкурентів у сфері надання послуг рієлтерської агенції (табл. 1.1):

Таблиця 1.1 – Аналіз конкурентів

Назва компанії	URL	Онлайн форма зв'язку з користувачем	Кабінет менеджера	Дизайн	Надання послуг підбору нерухомості	Внесення власної нерухомості
Агентство «САН»	https://san.sumy.ua/	+	–	+	+	–
Сумська нерухомість	https://sn.sumy.ua/ua/	+	–	+–	+	–
НоваЛад	https://novalad.com.ua/catalog	–	–	+	+	–
EAV	https://eav.estate/	+	–	+	–	–

Проаналізувавши аспекти ведення онлайн постачання послуг було виокремлено основні переваги та недоліки, які потрібно врахувати в створені власної веб системи діяльності рієлтерської агенції.

До переваг можна віднести: онлайн форма консультації з клієнтами, клієнтоорієнтований дизайн, детальна інформація про агентство, інформація про послуги, надання карток з наявною нерухомістю.

До недоліків можна віднести наступні: відсутність кабінету менеджера, відсутність кабінета користувача, відсутня можливість ввести в реєстр власну нерухомість.

Проаналізувавши недоліки та переваги можливо зробити висновок, яка логіка та функції будуть використані в майбутній розробці.

1.3 Аналіз потреб цільової аудиторії

Також одним із важливих аспектів перед створенням власного бізнесу надання нерухомості є аналіз потреб цільової аудиторії. Зазвичай за допомогою такого аналізу можливо виокремити основні проблеми конкурентів та проаналізувати актуальність та важливість розробки.

За проведеним дослідженням було винесено наступні висновки:

– Серед респондентів переважають чоловіки (55.8%), жінки займаються отримання послуг з агентств нерухомості менше (44.2%)(рис. 1.2);

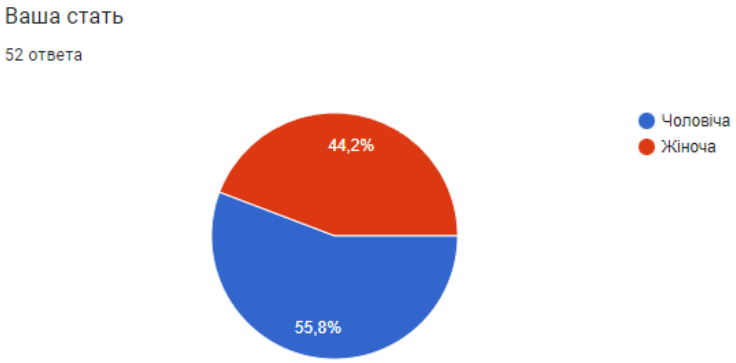

Рисунок 1.2 – Стать респондентів

– За віком респондентів переважають люди середнього та старшого віку – 59.6% (25-40 років) середнього та 26.9% (40-60 років) старшого віку, інші 13.5% - молоді люди (рис. 1.3);

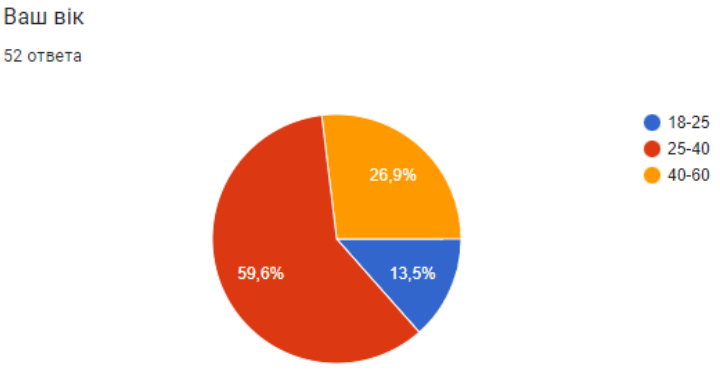

Рисунок 1.3 – Вік респондентів

– За дослідженням більшість використовують веб-ресурси при підборі

нерухомості. Серед таких 59.6% проголосували, що використовують та 40.4% проголосували, що не використовують веб-ресурс та переважно використовують лише оголошеннями в газетах (рис. 1.4);

Рисунок 1.4 – Використання веб-ресурсів

– Серед відповідей на питання «на що звертали (б) увагу при використані веб-систем агентств нерухомості» переважають відповіді за можливість внесення власної нерухомості та заявки на розміщення в базі нерухомості агентства – 73.1%. Також респонденти звернули увагу на актуальність даних на сайті (67.3%) та форму зворотного зв’язку (59.6%). Менше всього респонденти проголосували за зрозумілий інтерфейс (48.1%) та інформацію про агентство (53.8%)(рис. 1.5).

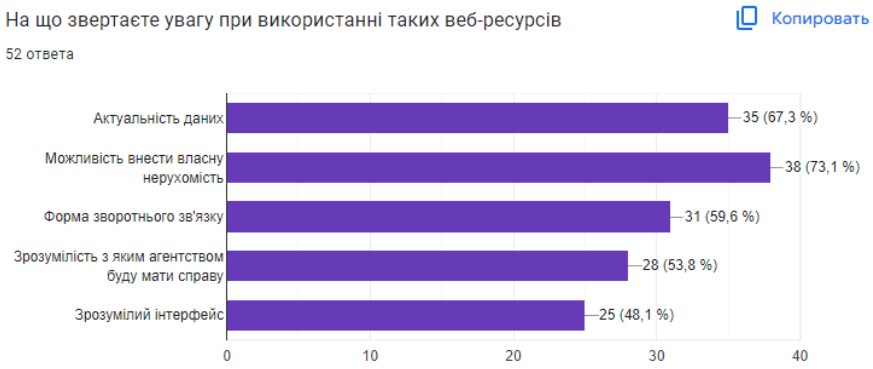

Рисунок 1.5 – Основні аспекти потреб

1.4 Постановка задачі

Метою роботи є розроблення веб-орієнтованої системи інформаційної підтримки діяльності рієлтерської агенції, використовуючи сучасні мови програмування та їх фреймворки. Система має забезпечувати потреби клієнтів у сфері надання нерухомості у Всесвітній мережі Інтернет та передбачає наступні задачі:

- Огляд існуючих інструментів для розробки та вибір оптимальних;
- Розробка серверної частини веб-системи рієлтерської агенції;
- Розробка інтерфейсу веб-системи інформаційної підтримки.
- Розробка кабінету користувача та кабінету менеджера та їх логіки роботи;
- Тестування веб-системи з метою перевірки функціональності.

При успішній реалізації вище поставлених задач буде забезпечена низка можливостей:

- Для користувача:
  1) Переглядати актуальну нерухомість в реєстрі агентства;
  2) Замовляти консультацію з менеджером;
  3) Замовляти будинок через форму зворотнього зв’язку;
  4) Надавати відомості про власну нерухомість та відправляти менеджеру на перевірку;
  5) Переглядати надані нерухомості.
- Для менеджера:
  1) Переглядати список всіх клієнтів;
  2) Переглядати базу зареєстрованої нерухомості;
  3) Переглядати заявки користувачів на консультацію та замовлення.

2 ВИБІР МЕТОДІВ РІШЕННЯ ЗАДАЧІ

2.1 Огляд існуючих інструментів для розробки систем

На сьогоднішній день існує різноманіття готових конструкторів сайтів для створення клієнтської і серверної частини систем електронної комерції.

Конструктор сайтів – онлайн платформа, за допомогою якої користувач може створити власний веб-сайт самостійно. Для цього не потрібно володіти навичками з веб-дизайну та програмування, лише зареєструватися на подібному ресурсі, вибрати шаблон, внести свої дані, відредагувати дизайн шаблону, налаштувати додаткові параметри (форми, аналітику, мета-теги, опис тощо) та опублікувати сайт. До таких сайтів можливо віднести наступні: Weblium, Shopify, Хорошоп, Squarespace, Webflow та Framer. Основними перевагами таких сайтів є функціональність, можливість налаштувати дизайн та верстку в одному місці, налаштувати параметри без знань програмування, проте до мінусів можна віднести тарифи на опублікування власних сайтів та систем (ціни можуть змінюватися в залежності вибраних параметрів та вмісту сайту), шаблонний дизайн, обмежений функціонал (залежить від конструктора), швидкість загрузки сторінки та безпека даних [8,9].

Таким чином використання конструкторів сайту не завжди вирішальне правило для створення якісного продукту, особливо для студентів. Тому існує безліч безкоштовних ресурсів та програмних забезпечень для розробки веб-систем. До них входять: Figma, Bootstrap, Sublime Text, Visual Studio Code, GitHub, React. Такі ресурси пропонують різноманітні функціональні рішення, включаючи запуск локальних серверів, створення дизайну для веб-ресурсу, а також створення клієнтської та серверної частини веб-ресурсу. Деякі з програм надають вбудовані плагіни та розширення для створення різних систем та тестування їх [10].

2.2 Вибір оптимальних інструментів та мов програмування

Перед програмною реалізацією будь-якого продукту потрібно вибрати оптимальні інструменти та мови програмування для ефективного забезпечення зазначений функцій.

Як середовище розробки вибрано програму Visual Studio Code. Програма надає безліч розширень та плагінів за допомогою яких можливо налаштувати локальний сервер та підказки при створенні функцій та компонентів [11].

Мовою програмування для клієнтської частини було обрано ReactJS. ReactJS – це JavaScript бібліотека, яка застосовується для розробки UI (інтерфейсу користувача), та відповідає за те, як кінцевий веб-додаток буде виглядати. Основними перевагами використання ReactJS є швидкість та продуктивність, так як такі сайти є не тільки красиві, а й мають велику швидкість створення сторінки в Інтернеті. Також гнучка структура папок та компонентів дає змогу створювати сторінки з певних блоків та за запитом змінювати лише певні компоненти, а не всю сторінку. Для створення таких блоків використовується HTML – стандартизована мова розмітки документів, а для стилізації CSS – каскадні таблиці стилів [12].

Для керування базами даними було обрано MongoDB – крос-платформна, потужна, гнучка та документно-орієнтована база даних. Її різниця з іншими системами керування даними в тому, що це NoSQL система, тобто збережена інформація не зберігається в пов’язаних та структурованих таблицях, а зберігається в спеціальних документах, подібних до JSON об’єктів. Такий формат для зберігання даних створений на базі мови JavaScript, яка є основою в нашій роботі і має структуру «ключ» - «значення» [13,14].

Для взаємодії програмним компонентам один з одним обрано програмний інтерфейс API (REST). Така взаємодія відбувається завдяки використання набори визначень і протоколів. Загальний принцип роботу API можливо побачити на рисунку 2.1 [15].

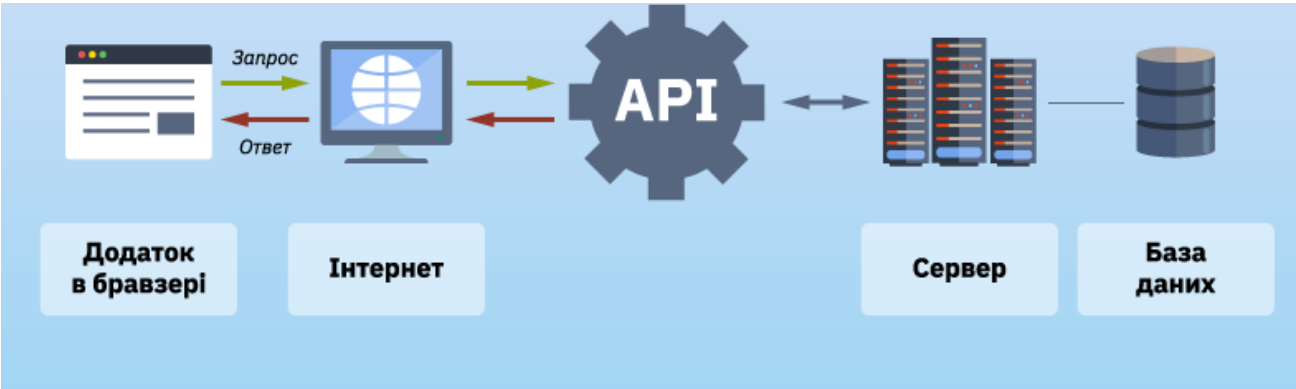

Рисунок 2.1 – Принцип роботи API

Інтерфейси REST хоч і мають менше вимог до безпеки, проте покращують сумісність з клієнтом браузера та використовують методи GET, POST, PUT та DELETE для отримання, передачі, редагування та видалення даних за допомогою HTTP [16].

Для тестування API було обрано HTTP клієнт Postman. Таким способом можливо:

– Складати та відправляти HTTP-запити, перевірячи їх статус та отримані дані
– Створювати колекції даних для скорочення часу на тестування
– Перевіряти роботу токенів авторизації

Тестування API є невід'ємною частиною розробки програми. Таким чином можливо передбачити як реагує система на реального користувача [17].

Мовою програмування для створення серверної частини було обрано NodeJS та її фреймворк ExpressJS. NodeJS – це середовище на базі JavaScript, яке виконує код поза браузером. Тут використовується асинхронний характер, таким чином сервер не чекає, поки дані повернуться, а виконує процеси паралельно, не блокуючи один одного, що робить продукт швидким. Також середовище є крос-платформним і може використовуватись на різних системах від Windows, Linux, MacOS. А завдяки движку JavaScript V8, який використовується у браузері Chrome робота прискорюється, що забезпечує швидкий потік передачі даних [18].

Як вже було сказано раніше також використовується фреймворк NodeJs – ExpressJS. Використання такої платформи значно пришвидшує процедуру створення сайтів, усуваючи необхідність створення кількох модулів NodeJS [19]. Цей компонент можливо дістати з менеджеру компонентів npm, завдяки якого також взято ресурси для клієнтської частини. Менеджер дозволяє нам легко використовувати код, написаний іншими, без необхідності писати його самостійно під час розробки. Основними перевагами використання менеджеру компонентів є: встановлення бібліотек, інструментів для проекту, прискорення процесу розробки за рахунок використання створених залежностей, широкий вибір інструментів [20].

Дизайн є невід'ємною частиною розробки. Від дизайну залежить майбутнє представлення клієнтської частини. Для створення прототипів дизайну та майбутнього дизайну було обрано безкоштовну програму Figma. В програмі можливо створювати екрани та компоненти, які потім можливо відтворити в клієнтській частині. Основними перевагами використання даної проблеми є легкість у використанні, робота на різних операційних системах, низький рівень знань для роботи в програмі [21].

2.3 Створення веб-дизайну веб-системи

Дизайн у бізнесі має вирішальне значення, адже це те, що люди бачать насамперед і що формує їхнє сприйняття та очікування. Перед початком розроблення дизайну майбутнього ресурсу потрібно продумати макет системи. Основними аспектами успішного макету та майбутнього дизайну є:

- Привабливий контент. Текст є мабуть основною частиною у дизайні сайту, так як він може передати ті сенси, які сам дизайн не в змозі це зробити. Використання закликів до дії у тексті може дати користувачам більше довіри до замовлення послуг або залишання на сайті
- Зручна навігація. Доступним для сприйняття має бути не лише контент, а й навігація на сайті, тобто кнопки, картки, робочі посилання.

– Кількість інформації. Занадто багато інформації або фотографій лише сповільнюють роботу сайту і роблять сайт важким для читання. Якщо на сторінках буде багато інформації користувачу буде складно знайти важливу інформацію.
– Якісні фотографії. Погане зображення може зашкодити довірі до компанії.

Таким чином спираючись на ці аспекти [22] спроектовано майбутній інтерфейс та структуру веб-орієнтованої інформаційної системи. Як приклад гарного дизайну та структури було взято дизайн сайту Dim.Ria [23].

Спираючись на мету розробки наша система має мати наступні сторінки:

– Головна сторінка з блоками:
  1) Головний блок з закликом до дії (рис. 2.2)
  2) Блок про агенцію (рис. 2.3)
  3) Блок переваг агенції (рис. 2.4)
  4) Блок контактів агенції та форма консультації (рис. 2.5)
  5) Блок питань та відповідей (рис. 2.6)
– Сторінка перегляду нерухомості (рис. 2.7)
– Кабінет менеджера (рис. 2.8)
– Кабінет користувача (рис. 2.9)

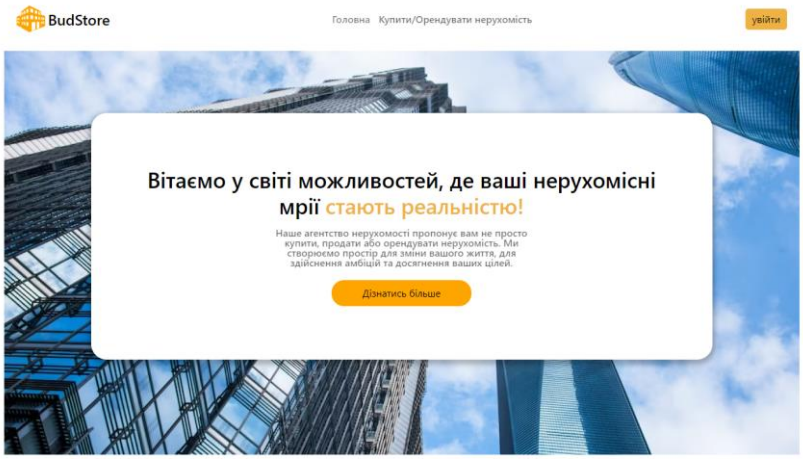

Рисунок 2.2 – Блок головної сторінки

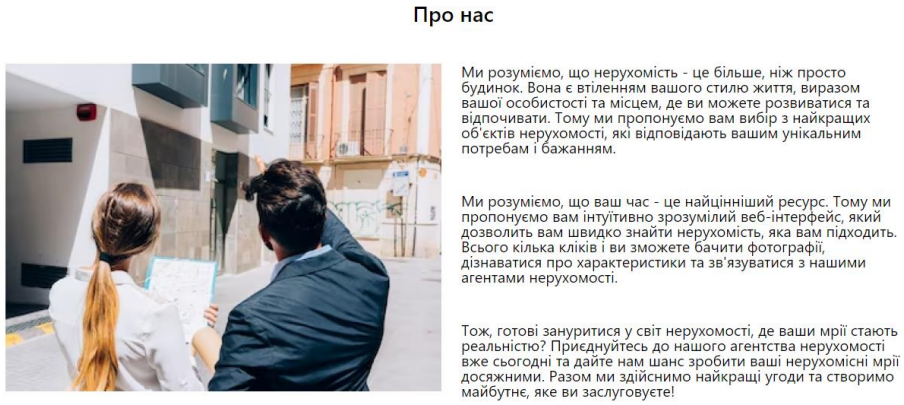

Рисунок 2.3 – Блок про рієлторську агенцію

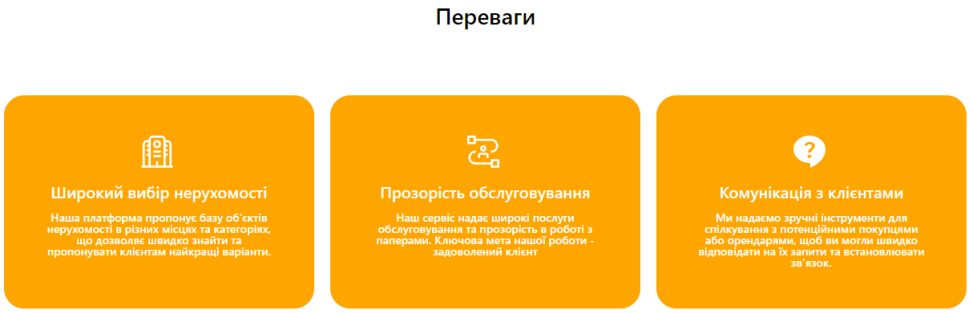

Рисунок 2.4 – Блок переваг

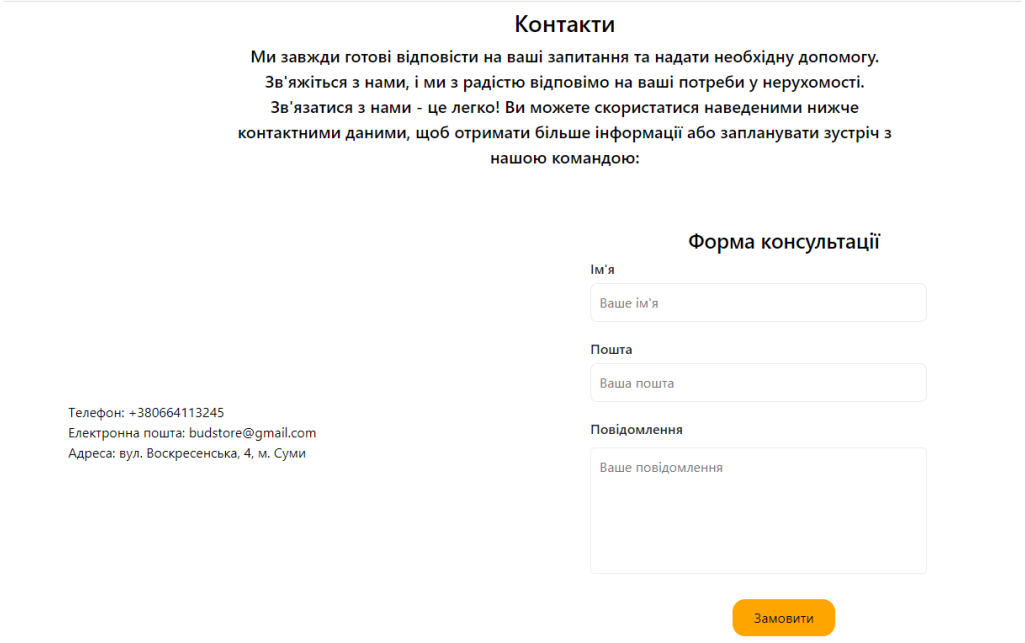

Рисунок 2.5 – Блок консультації та контактів

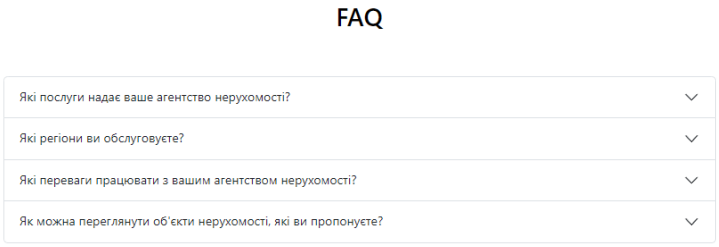

Рисунок 2.6 – Блок питання та відповіді

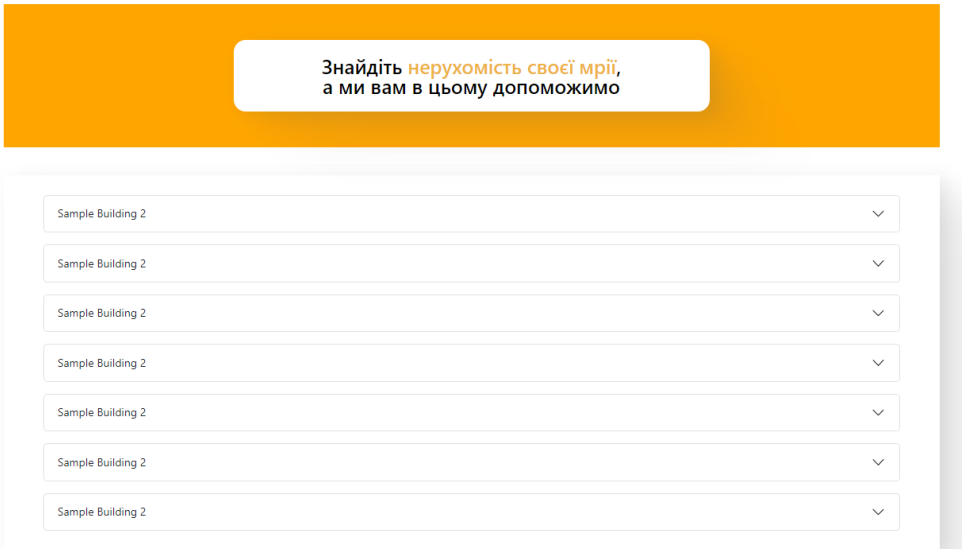

Рисунок 2.7 – Сторінка перегляду нерухомості

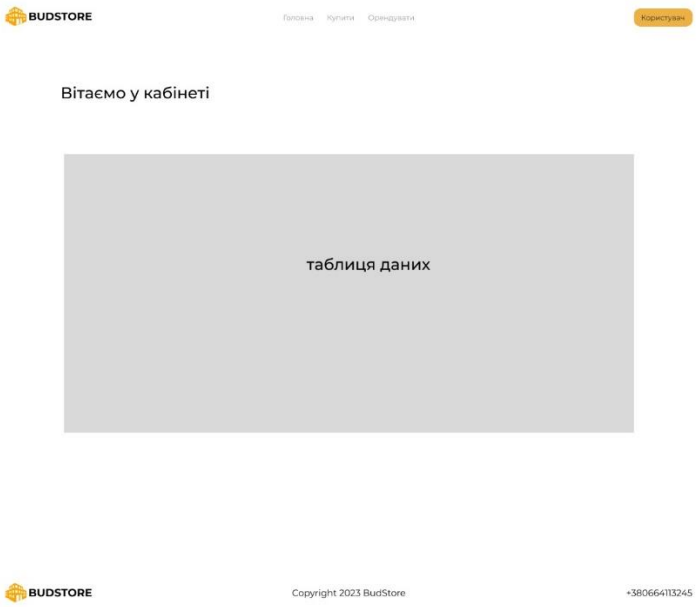

Рисунок 2.8 – Кабінет менеджера

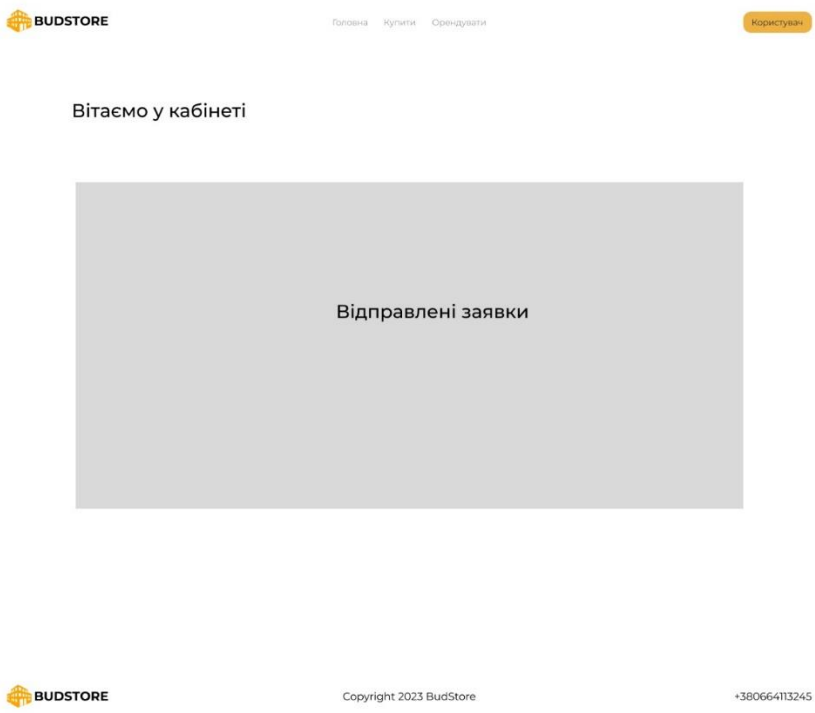

Рисунок 2.9 – Кабінет клієнта

Проглянувши структуру та дизайн веб системи можливо побачити, що в більшості використовується помаранчевий колір та його відтінки. Даний колір більше підходить для надання послуг, так як надає користувачеві відчуття надійності та оптимізму. Використання даного кольору допоможе ефективно постачати послуги та заохочувати багато клієнтів [24].

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Налаштування бібліотек та структури додатку

Щоб забезпечити швидкодію та надійність роботи веб системи важливо використовувати додаткові бібліотеки та залежності та гарно структуру додатку. В нашій системі використано наступні бібліотеки та залежності:

Для серверної частини:

- BcryptJS – для шифрування паролів користувачів у базі даних
- Nodemon – для автоматичного запуску серверу після внесення нових даних у програмній реалізації
- Mongoose – для коректної роботи з базою даних MongoDB, створення моделей, відправлення та отримання даних
- Express – для застосування саме фреймворку ExpressJS.
- Jsonwebtoken – для застосування перевірки авторизованого користувача завдяки токену сесії.
- Cors – для можливості роботи клієнтної та серверної частини одночасно
- Dotenv – для зберігання системних змін (наприклад секретного значення токену, його сроку придатності, даних від порту серверу тощо)

Для клієнтської частини:

- ReduxJS – для застосування Redux у клієнтській частині. Redux – контейнер станів для додатків, який вирішує проблеми роботи клієнтської частини та серверної частини (отримання даних, відслідковування станів помилок тощо) [25].
- Axios – для з’єднання з серверною частиною та роботи з запитами.
- Bootstrap та react-bootstrap – для стилізації сторінок з використанням фреймворку Bootstrap та його компонентів[26].
- React та компоненти react-dom react-router-dom – для використання

фреймворку ReactJS для верстки сторінок клієнтської частини та мінімізації коду.

- React-hook-form – для використання створених хуків для валідації даних у формах та передачі у функції даних з форми.
- Sass – для використання препроцесору Scss при стилізації сторінок [27].

Завдяки цим компонентам та бібліотекам робота веб-системи відбувається швидко та забезпечується конфіденційність та безпека даних.

Структура проекту має 2 основні папки, які відповідають за основну роботу системи: папка клієнтської частини та папка серверної частини. До **клієнтської частини** відносяться наступні папки:

- Public – вміст: папка з картинками, логотипи, фавікон сайту;
- Src – є основною папкою структуру клієнтської частини і включає в себе:
  1) Components – компоненти сайту, які повторюються, або мають спеціальний контент (шапка сайту, футер сайту, таблиці);
  2) Pages – різні сторінки сайту, включаючи сторінки кабінетів, реєстрації та авторизації;
  3) Redux – компоненти для роботи з даними серверної частини;
  4) Utility – містить файл для налаштування компонента axios.

До **серверної частини** відносяться наступні папки:

- Controllers – сюди відносяться файли роботи з моделями бази даних (відправка даних, отримання даних);
- Models – включає в собі всі моделі, які використані в системі та зберігаються у вигляді колекцій у системі керування даними MongoDB;
- Routers – включає в собі всі напрямки запитів контролерів;
- Utils – включає в собі файл з перевіркою авторизованого користувача.

3.2 Налаштування взаємодії з базою даних

Для роботи з базою даних використовується вже згадана бібліотека Mongoose. Для підключення до бази даних використовується наступна функція:

```
//Connect to Database
mongoose
    .connect(process.env.DBURL, {
        useNewUrlParser: true,
    })
    .then(() => console.log("Підключено до БД"))
    .catch((error) => console.log("Щось пішло не так" + error));
```

Змінна посилання для підключення до бази даних зберігається у конфігураційному файлі і на данному етапі веб-система представлено в локальному середовищі і має наступний вигляд:

```
DBURL = 'mongodb://127.0.0.1:27017/estate'
```

Для коректної роботи з даними потрібно створити моделі колекцій. В нашій системі використовується низка моделей:

- Модель BuildModel – описує поля заголовку нерухомості, опис, кількість кімнат, поверх, тип (будівля чи квартира), наявність гаражу, послугу (продаж або оренда), адресу нерухомості, ціну та загальну площу нерухомості;
- Модель ClientModel – описує поля ім’я, прізвище клієнта, його пошту, пароль та його роль (клієнт або менеджер);
- Модель requestModel – описує поля передані з форми консультації або замовлення якоїсь нерухомості для перегляду: ім’я користувача, пошта, повідомлення або назву будинку;
- Модель reqBuildModel – описує поля моделі BuildModel та ім’я користувача та його ідентифікаційний номер. Використовується в системі при передачі користувачем власної нерухомості для розміщення в базі агенції.

Створивши моделі, також потрібно налаштувати контролери (Додаток А) та напрямки запитів (Додаток Б) для взаємодії клієнтської та серверної частини.

Також потрібно перевірити чи авторизований користувач в системі та чи має власний токен сесії. Для цього використано бібліотеку jsonwebtoken та наступну функцію:

```
const jwt = require('jsonwebtoken');
const Client = require('../models/clientModel');
require('dotenv').config();
module.exports = async (req, res, next) => {
    const token = (req.headers.authorization ||
'').replace(/Bearer\s?/, '');

    if (token) {
        try {
            const decoded = jwt.verify(token,
process.env.JWT_SECRET);
            const client = await
Client.findById(decoded.clientId);
            req.client = client;
            next();
        } catch (error) {
            return res.status(403).json({
                error: "Немає доступу",
            });
        }
    } else {
        return res.status(403).json({
            error: "Немає доступу",
        });
    }
}
```

Таким чином відбувається взаємодія з базою даних та API в серверному середовищі.

3.3 Створення та налаштування процесу авторизації та реєстрації

Під час створення веб-системи для рієлтерської агенції важливим фактором є забезпечення авторизації та реєстрації користувачів та менеджерів у системі. В процесі реєстрації передані дані, особливо паролі, шифруються за допомогою бібліотеки BcryptJs (Додаток А clientController.js) та зберігаються в зашифрованому вигляді (рис. 3.1).

```
_id: ObjectId('646fe485cdbc80b1ce73f78b')
name: "Petr"
surname: "Petrik"
email: "petrpetrik@gmail.com"
password: "$2a$10$uNLiFlwKNlAyg.9vCtFtseN4O7W9M928Kn..H1KSNSGcqG0/bryNS"
clientRole: "client"
```

Рисунок 3.1 – Зберігання шифрованих паролів

На стороні клієнта дані передаються за допомогою форми (рис. 3.2).

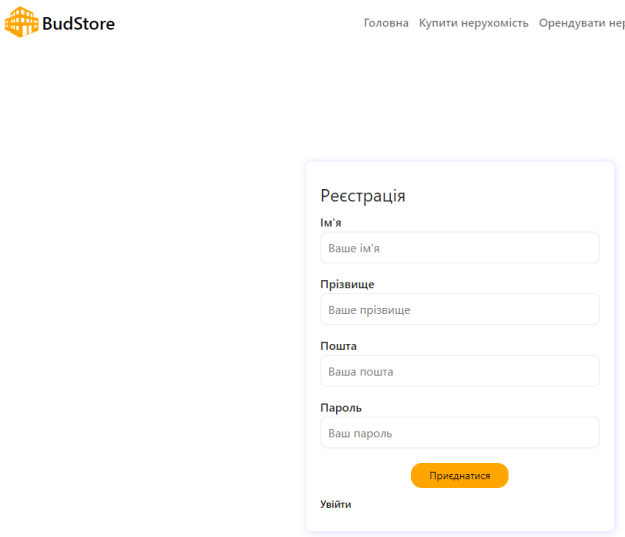

Рисунок 3.2 – Форма реєстрації

В процесі авторизації, паролі перевіряються за тим же алгоритмом bcrypt (Додаток А clientController.js), а значення беруться також з форми (рис. 3.3):

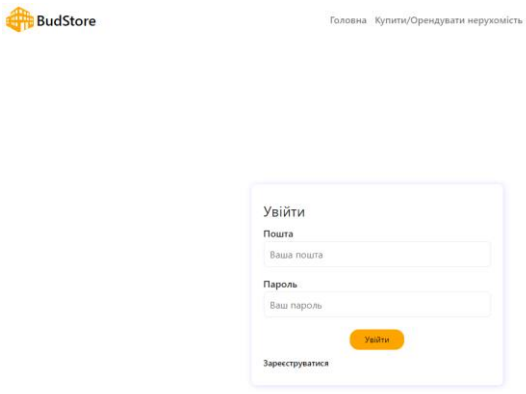

Рисунок 3.3 – Форма авторизації

З клієнтської сторони дані взаємодіють з сервером за допомогою функцій створених в фрагментах Redux – набори логіки та дій Redux для функцій у

додатку, які визначаються разом в одному файлі [28]. Так взаємодія з моделлю користувача описані у одному фрагменті authSlice (Додаток Г), а дані використовуються у хуках (useNavigate, useSelector, useDispatch, useEffect)[29–31] на певних сторінках.

Для сторінки реєстрації:

```
const navigate = useNavigate();
const Registered = useSelector(selectisRegistered);
const { error } = useSelector((state) => state.auth)
const dispatch = useDispatch();
const onSubmit = (values) => {
        dispatch(registerClient(values));
    }
    useEffect(() => {
        if (Registered) {
            navigate('/login');
        }
    }, [navigate, Registered]);
```

Для сторінки авторизації:

```
const isAuth = useSelector(selectisAuth);
const onSubmit = (values) => {
        dispatch(loginClient(values));
    }
    console.log(error);
    useEffect(() => {
        if (isAuth) {
            navigate('/');
        }
    }, [isAuth, navigate]
    )
```

Змінна Registered – перевіряє чи пройшла реєстрація користувача, в разі помилок в написанні чи вже зайнятої пошти користувач отримує повідомлення про це.

Змінна isAuth – перевіряє чи пройшла форма авторизації користувача, в разі помилок в написанні чи неправильного паролю або пошти користувач отримує відповідне повідомлення.

Values – це змінні, які ми передаємо приймаючи форму реєстрації або авторизації, ці значення змінних потім використовуються у відповідних функціях фрагментів. Таким чином відбувається взаємодія клієнтської та серверної частини при процесі авторизації або реєстрації.

3.4 Налаштування форм для передачі даних

Застосування форм є основною складовою при розробці будь-якої системи. В нашому випадку форми використовуються майже на всіх сторінках системи: форма консультації з питанням та передачею бажаючої нерухомості, авторизація, реєстрація, форма подачі будівлі в реєстр нерухомості агенції. Кожна форма проходить перевірку на коректність введених даних. Для цього застосовується хук useForm [32], який реєструє дані у формі та передає на сервер та перевіряє ці дані на валідність. Наприклад реєстрація та перевірка на валідність пошти на формах авторизація та реєстрації:

```
<label>Пошта</label>
<input
    type="text"
    placeholder={`Ваша пошта`}
    className={`${errors.email ? login['error-input'] : ""}`}
    {...register("email", { required: `Поле пошта є обов'язковим`,
pattern: { value: /[a-z0-9._%+-]+@[a-z0-9.-]+\.[a-z]{2,}$/,
message: "Некоректний формат пошти" } })}
/>
```

Хук реєструє значення отримане з поля у змінну, в цьому випадку «email», а подалі ця змінна буде використовуватися в функції контроллерів в серверній частині системи. Якщо сталась помилка, то значення помилки записується у відповідну змінну і виводиться на екран.

Також за допомогою регулярного виразу система перевіряє чи в полі введено символи англійською, присутній знак «@» та текст з крапкою після нього. В разі, якщо користувач вводить невірну пошту (рис. 3.4) або пошту не в правильному форматі (рис. 3.5) з’являється відповідне повідомлення.

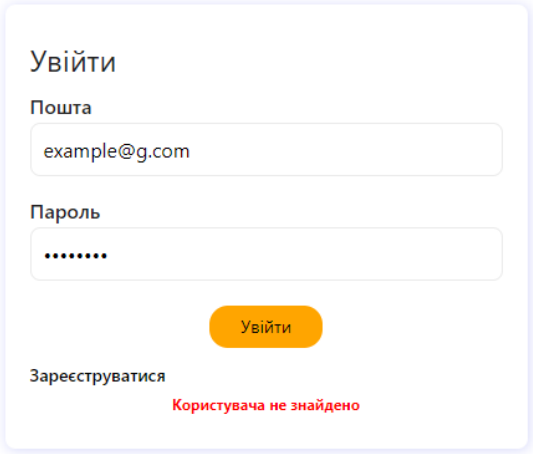

Рисунок 3.4 – Авторизація з невідомою поштою

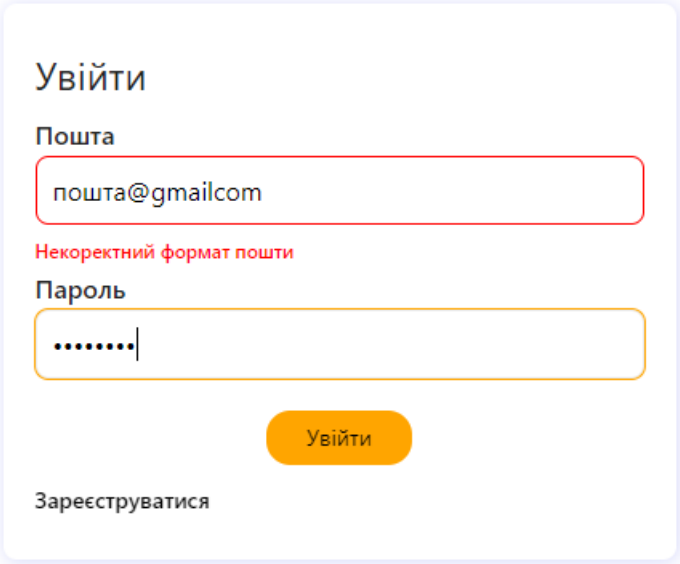

Рисунок 3.5 – Авторизація з невалідною поштою

Валідація даних та повідомлення на сторінці реєстрації відбуваються по такому ж сценарію.

Наступною формою для перевірки на правильність є форма консультації або передачі бажаючої нерухомості менеджеру. Вони мають трошки різний дизайн проте функціонал їх однаковий. Тут перевіряються наступні стани:

– Поля пусті та нажата кнопка «Замовити» (рис. 3.6);
– Поля заповнені невалідними даними та натиснута кнопка «замовити» (рис. 3.7)

– Поля заповнені валідними даними та натиснута кнопка «замовити» (рис. 3.8).

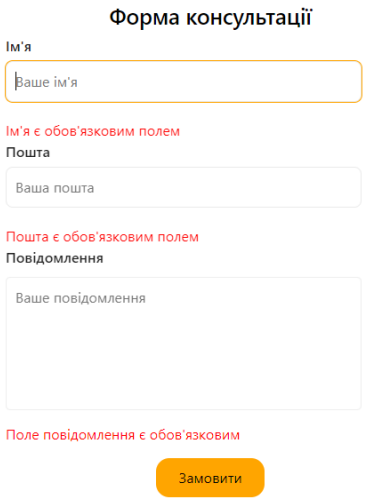

Рисунок 3.6 – Перевірка роботи форми консультації

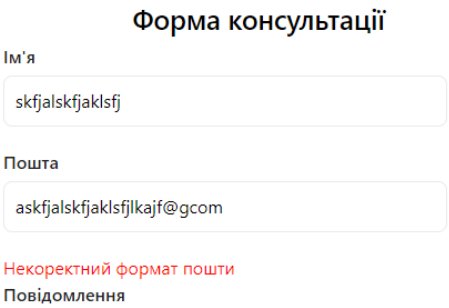

Рисунок 3.7 – Перевірка заповнення некоректної пошти

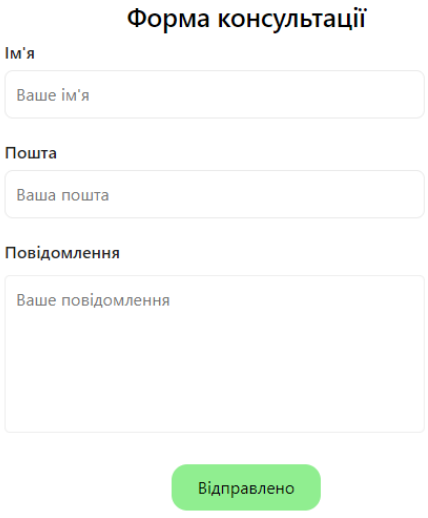

Рисунок 3.8 – Перевірка форми після введення коректної пошти

Наступною формою є форма відправки будівлі для додавання в реєстр будівель агенції. За регулярними виразами ми перевіряємо наступні поля:

– Кількість кімнат – поле приймає значення тільки від 1 до 4, значення не може бути нульовим;
– Кількість поверхів – поле приймає числа від 0 до 25;
– Ціна – поле приймає значення від 10 і до 99999. Поле не приймає значення, що менше нуля, проте приймає ті, що дорівнюють нулю.

Всі поля є обов'язковими і якщо користувач не введе значення у поле або це значення буде некоректне – користувач отримає про це повідомлення (рис. 3.9).

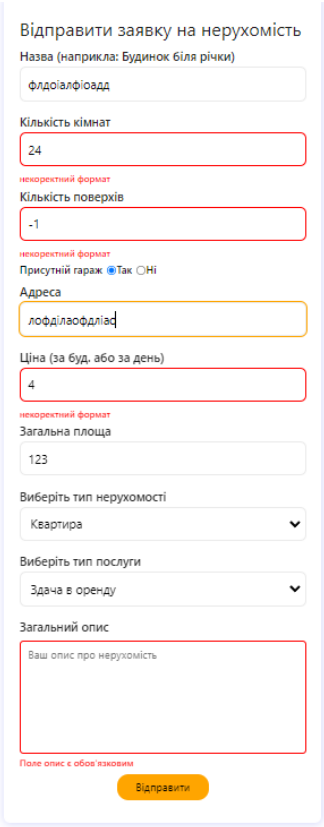

Рисунок 3.9 – Перевірка реакції на невалідне заповнення форми

3.5 Панель користувача

Впровадження панелей для різних ролей є невід'ємною операцією при розробці систем, оскільки це може прибавити користувачів до системи та клієнти зможуть взаємодіяти з системою, що задовольнить їх потреби.

В нашій системі користувач, після авторизації, може переглядати свій профіль з списком своїх відправлених заявок (Додаток А clientController.js, buildReqController.js), відправляти свій будинок за допомогою форми передачі заявки (Додаток В buildSlice.js – registrBuilding, Додаток А buildReqController.js registrBuild).

3.6 Панель адміністратора

Щоб ефективно управляти користувачами та їх замовленнями на будівлі або розташування власних будівель було розроблено панель адміністратора. Зареєструвавшись в системі адміністратору надається широкий спектр можливостей. У власному кабінеті (рис. 3.10) адміністратор може переглядати актуальних користувачів системи (Додаток А clientController.js – getUsers), всі актуальні заявки на консультацію (Додаток А clientReqController.js – fetchRequests), заявки на розташування на сервісі власної нерухомості (Додаток А buildReqController.js fetchAllBuildReq).

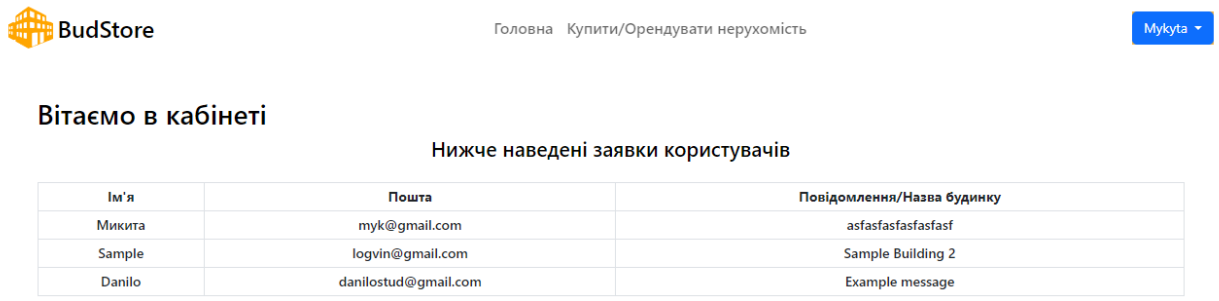

Рисунок 3.10 – Кабінет адміністратора

3.7 Алгоритм роботи за ролями

Неавторизований користувач

Неавторизований користувач має доступ до сторінок: головна, купити/орендувати нерухомість, авторизація та реєстрація. Неавторизований

користувач може заповнювати та надсилати форму для консультації та замовляти на нерухомість на огляд через ту ж форму.

Авторизований користувач

Авторизований користувач – користувач, який має акаунт у системі. Авторизований користувач має доступ до усіх сторінок в межах своєї ролі.

Основною сторінкою авторизованого користувача є сторінка власного кабінету, де клієнт може переглядати у власному профілі свої замовлення, надсилати запит на розташування у реєстрі власної нерухомості та перевіряти каталог нерухомості, розташованої на сайті.

Адміністратор

Адміністратор або менеджер має доступ до свого кабінету. Тільки адміністратор та власник знають свої дані для входу до панелі адміністратора в системі. Це зроблено задля того, щоб звичайний користувач не міг увійти у кабінет адміністратора і зламати бізнес логіку агенції.

У своєму кабінеті адміністратор може переглядати актуальні заявки користувачів та за потреби зв’язуватися з клієнтами, написавши їм на пошту та уточнивши деталі. Адміністратор може перевіряти заявки на внесення нерухомості в реєстр. Дані представлено у таблиці з полями «Назва», «Адреса», «Тип» (тип нерухомості), «Послуга» (тип послуги – оренда/продаж), «Ціна» та «Користувач». За необхідності адміністратор може зв’язатися з користувачем та обговорити деталі його нерухомості. До можливостей адміністратора також входить контроль користувачів. Адміністратор може переглядати актуальних користувачів, їх пошту, ім’я та роль у системі.

Крім цього адміністратор, як і всі користувачі, має доступ до головних сторінок і використовувати функції звичайних користувачів, переглядаючи будівлі.

ВИСНОВКИ

У процесі роботи було створено веб-систему інформаційної підтримки діяльності рієлтерської агенції.

У ході виконання кваліфікаційної роботи були виконані такі завдання:

- Проведено літературний огляд джерел, пов’язаних з електронною комерцією, аналіз конкурентів та аналіз потреб цільової аудиторії, на основі чого було сформовано постановку задачі для подальшої реалізації;
- Здійснено огляд вже існуючих рішень та обрано інструменти для ефективної реалізації поставленої мети;
- Реалізовано доступ до системи з різних ролей;
- Створено ефективний та привабливий дизайн;
- Створено ефективну та безпечну серверну частину з урахуванням алгоритмів шифрування;
- Реалізовано надання каталогу існуючої нерухомості в агенції;
- Визначено та реалізовано основну бізнес логіку веб-системи.

Розроблений додаток здатний до розширення функціоналу. До таких доповнень можна віднести додавання картинок власної нерухомості, розташування сайту на хостингу, онлайн оплата та взаємодія з користувачами, використовуючи штучний інтелект, створення адаптивного виду для коректного відкривання сайту на мобільних пристроях.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Значення інтернету для людини - Dovidka.biz.ua [Electronic resource]. URL: https://dovidka.biz.ua/znachennia-internetu-dlia-liudyny/ (accessed: 28.05.2023).
2. Інтернет у житті людини | Я - журналіст [Electronic resource]. URL: https://ij.ogo.ua/suzh/internet-u-zhitti-lyudini/ (accessed: 28.05.2023).
3. E-Commerce: що це таке та як працює електронна комерція в Інтернеті [Electronic resource]. URL: https://wezom.com.ua/ua/blog/elektronnaya-kommertsiya (accessed: 28.05.2023).
4. Електронна комерція: визнання де-юре [Electronic resource]. URL: https://news.dtkt.ua/state/other/35892 (accessed: 28.05.2023).
5. Industry 4.0 [Electronic resource]. URL: https://www.it.ua/knowledge-base/technology-innovation/industry-4 (accessed: 28.05.2023).
6. Інформаційні системи - Інформаційні системи і технології на підприємствах - Навчальні матеріали онлайн [Electronic resource]. URL: https://pidru4niki.com/1222090547713/informatika/informatsiyni_sistemi (accessed: 28.05.2023).
7. Автоматизовані інформаційні системи: опис, завдання та особливості [Electronic resource]. URL: https://publish.com.ua/it-ta-web/avtomatizovani-informatsijni-sistemi-opis-zavdannya-ta-osoblivosti.html (accessed: 28.05.2023).
8. Найкращі конструктори сайтів у 2023 (ТОП 10) | weblium.com [Electronic resource]. URL: https://ua.weblium.com/blog/najkrashi-konstruktori-sajtiv-2023-top-10 (accessed: 28.05.2023).
9. Переваги та недоліки онлайн конструкторів для створення сайтів [Electronic resource]. URL: https://zcollage.com.ua/perevahy-ta-nedoliky-onlayn-konstruktoriv-dlia-stvorennia-saytiv/ (accessed: 28.05.2023).

10. 17 Free Web Design Software Tools to Make a Great Website in 2022 [Electronic resource]. URL: https://blog.hubspot.com/website/free-web-design-tools-2020 (accessed: 28.05.2023).
11. Why Visual Studio Code? [Electronic resource]. URL: https://code.visualstudio.com/docs/editor/whyvscode (accessed: 28.05.2023).
12. Вступ в React, якого нам не вистачало | DevZone [Electronic resource]. URL: https://devzone.org.ua/post/vstup-v-react-yakogo-nam-ne-vistachalo (accessed: 28.05.2023).
13. Що таке JSON | Навіщо потрібен цей формат [Electronic resource]. URL: https://apix-drive.com/ua/blog/useful/scho-take-json (accessed: 28.05.2023).
14. Основи MongoDB [Electronic resource]. URL: https://codeguida.com/post/519 (accessed: 28.05.2023).
15. Що таке API: простими словами про складне | Блог хостера HOSTiQ.ua [Electronic resource]. URL: https://hostiq.ua/blog/ukr/what-is-api/ (accessed: 28.05.2023).
16. Розуміння основ роботи API і REST API – короткий вступ | SebWeo [Electronic resource]. URL: https://sebweo.com/rozuminnya-osnov-roboti-api-i-rest-api-korotkij-vstup/ (accessed: 28.05.2023).
17. Використання Postman в тестуванні | Безкоштовний онлайн-курс від компанії QATestLab [Electronic resource]. URL: https://training.qatestlab.com/blog/technical-articles/use-postman-in-testing/ (accessed: 28.05.2023).
18. Що таке Node JS простими словами - застосування Node JS у програмуванні | DAN-IT [Electronic resource]. URL: https://dan-it.com.ua/uk/blog/chto-jeto-takoe-node-js-prostymi-slovami/ (accessed: 28.05.2023).
19. Express JS Tutorial: What is Express in Node JS? [Electronic resource]. URL: https://www.simplilearn.com/tutorials/nodejs-tutorial/what-is-express-js (accessed: 28.05.2023).

20. What Is npm? An Introduction to Node's Package Manager [Electronic resource]. URL: https://kinsta.com/knowledgebase/what-is-npm/#what-is-npm-node-package-manager (accessed: 28.05.2023).
21. What is FIGMA: Advantages and Disadvantages [Electronic resource]. URL: https://www.fita.in/what-is-figma-advantages-and-disadvantages/ (accessed: 28.05.2023).
22. The Importance Of Design In Business Success - 2023 [Electronic resource]. URL: https://inkbotdesign.com/design-in-business-success/ (accessed: 28.05.2023).
23. DIM.RIA™ — вся нерухомість України. Продаж і оренда будь-якої нерухомості. [Electronic resource]. URL: https://dom.ria.com/uk/ (accessed: 28.05.2023).
24. Color Theory for Designers, Part 1: The Meaning of Color — Smashing Magazine [Electronic resource]. URL: https://www.smashingmagazine.com/2010/01/color-theory-for-designers-part-1-the-meaning-of-color/ (accessed: 28.05.2023).
25. Основи Redux [Electronic resource]. URL: https://codeguida.com/post/624 (accessed: 28.05.2023).
26. Bootstrap · The most popular HTML, CSS, and JS library in the world. [Electronic resource]. URL: https://getbootstrap.com/ (accessed: 28.05.2023).
27. Sass: Syntactically Awesome Style Sheets [Electronic resource]. URL: https://sass-lang.com/ (accessed: 28.05.2023).
28. Redux Essentials, Part 2: Redux Toolkit App Structure | Redux [Electronic resource]. URL: https://redux.js.org/tutorials/essentials/part-2-app-structure (accessed: 28.05.2023).
29. useNavigate v6.11.2 | React Router [Electronic resource]. URL: https://reactrouter.com/en/main/hooks/use-navigate (accessed: 28.05.2023).

30. UseSelector and UseDispatch: A Guide to React-Redux Hooks | Built In [Electronic resource]. URL: https://builtin.com/software-engineering-perspectives/useselector-usedispatch-react-redux (accessed: 28.05.2023).
31. Using the Effect Hook – React [Electronic resource]. URL: https://legacy.reactjs.org/docs/hooks-effect.html (accessed: 28.05.2023).
32. useForm | React Hook Form - Simple React forms validation [Electronic resource]. URL: https://www.react-hook-form.com/api/useform/ (accessed: 28.05.2023).

ДОДАТОК А
ЛІСТИНГ КОНТРОЛЛЕРІВ

A1 clientController.js

```
const jwt = require('jsonwebtoken');
const crypt = require('bcryptjs');
const Client = require('../models/clientModel');
require('dotenv').config();
const register = async (req, res) => {
    try {
        const { name, surname, email, password } = req.body;
        const salt = await crypt.genSalt(10);
        const hash = await crypt.hash(password, salt);

        const existedClient = await Client.findOne({ email });
        if (existedClient) { return res.status(500).json({ msg:
"Користувач з такою поштою вже існує" }) };

        const newClient = await Client.create({
            name,
            surname,
            email,
            password: hash,
        });
        const client = await newClient.save();
        return res.status(200).json({ data: client });
    } catch (error) {
        return res.status(500).json({ msg: "Не вдалось
зареєструватися" })
    }
}
const login = async (req, res) => {
    const { email, password } = req.body;
    const client = await Client.findOne({ email });
    if (!client) { return res.status(404).json({ msg: "Користувача
не знайдено" }) };

    if (await crypt.compare(password, client.password)) {
        const token = jwt.sign(
            {
                clientId: client._id,
                clientrole: client.clientRole,
            },
            process.env.JWT_SECRET,
            {
                expiresIn: process.env.JWT_EXPIRES,
            }
        );
        return res.json({
            data: client,
```

```
            token: token,
        })
    } else {
        return res.status(401).json({ msg: "Неправильний пароль"
});
    }
}
const getMe = async (req, res) => {
    try {
        const client = await Client.findById(req.client._id);
        if (client) {
            const token = jwt.sign(
                {
                    clientId: client._id,
                    clientrole: client.clientRole,
                },
                process.env.JWT_SECRET,
                {
                    expiresIn: process.env.JWT_EXPIRES,
                }
            );
            return res.status(200).json({
                data: client,
                token: token,
            });

        } else {
            return res.status(404).json({ msg: "Користувача не
знайдено" });
        }
    } catch (error) {
        res.status(404).json({ msg: "Сталась помилка пошуку
клієнта" });
    }
}
const getUsers = async (req, res) => {
    const clients = await Client.find({});
    return res.status(200).json({ data: clients });
}
module.exports = {
    login,
    register,
    getMe,
    getUsers,
}
```

A2 buildController.js

```
const Build = require('../models/BuildModel');
const createBuild = async (req, res) => {
    try {
        const newBuilding = await Build.create({ ...req.body });
```

```
        return res.status(200).json({ data: newBuilding });
    } catch (error) {
        return res.status(500).json({ msg: "Не вдалось внести в
реєстр нерухомість" });
    }
}
//Fetch All
const fetchBuilds = async (req, res) => {
    const buildings = await Build.find({});
    return res.status(200).json({ data: buildings });
}
module.exports = {
    fetchTypeBuild,
    createBuild,
    fetchBuilds,
}
```

A3 ClientReqController.js

```
const ClientReq = require('../models/requestModel');
const createReq = async (req, res) => {
    try {
        const { name, email, message } = req.body;
        const newReq = await ClientReq.create({
            name,
            email,
            message,
        });
        const clientRequest = await newReq.save();

        return res.status(200).json({ datareq: clientRequest });
    } catch (error) {
        return res.status(505).json({ msg: "Не вдалось відправити
дані" });
    }
}
const fetchRequests = async (req, res) => {
    try {
        const clientRequests = await ClientReq.find({});
        return res.status(200).json({ requests: clientRequests });
    } catch (error) {
        return res.status(500).json({ msg: "Не вдалось знайти
запити на консультацію" });
    }
}
module.exports = {
    createReq,
    fetchRequests,
}
```

A4 BuildReqController.js

```
const BuildReq = require('../models/reqBuildModel');
const registrBuild = async (req, res) => {
    try {
        const newBuildReq = await BuildReq.create({ ...req.body,
ownerid: req.client.clientId, ownername: req.client.name });
        return res.status(200).json({ data: newBuildReq });
    } catch (error) {
        return res.status(500).json({ msg: "Не вдалось внести в
реєстр нерухомість" });
    }
}
const fetchClientBuild = async (req, res) => {
    const usid = req.client.clientId;
    const gotBuildings = await BuildReq.find({ ownerid: usid });

    return res.status(200).json({ data: gotBuildings });
}
const fetchAllBuildReq = async (req, res) => {
    try {
        const reqbuilds = await BuildReq.find({});
        return res.status(200).json({ reqbuilds: reqbuilds });
    } catch (error) {
        return res.status(404).json({ msg: "Не вдалось знайти
нерухомості" });
    }
}
module.exports = {
    registrBuild,
    fetchClientBuild,
    fetchAllBuildReq,
}
```

ДОДАТОК Б
НАПРЯМКИ ЗАПИТІВ

Б1 clientRouter.js

```
const express = require('express');
const router = express.Router();
const { ClientController } = require('../controllers');
const checkAuthorized = require('../utils/checkAuthorized');
router.post('/login', ClientController.login);
router.post('/register', ClientController.register);
router.get('/getMe', checkAuthorized, ClientController.getMe);
router.get('/fetchusers', ClientController.getUsers);
module.exports = router;
```

Б2 buildRouter.js

```
const express = require('express');
const router = express.Router();
const {BuildController} = require('../controllers');
router.post('/createbuild', BuildController.createBuild);
router.get('/fetchbuild', BuildController.fetchBuilds);
module.exports = router;
```

Б3 buildReqRouter.js

```
const express = require('express');
const router = express.Router();
const checkAuthorized = require('../utils/checkAuthorized');
const { BuildReqController } = require('../controllers');
router.post('/regbuild', checkAuthorized,
BuildReqController.registrBuild);
router.get('/fetchregbuild', checkAuthorized,
BuildReqController.fetchClientBuild);
router.get('/fetchbuilds', BuildReqController.fetchAllBuildReq);
module.exports = router;
```

Б4 clientReqRouter.js

```
const express = require('express');
const router = express.Router();
const { ClientRequest } = require('../controllers');
router.post('/createReq', ClientRequest.createReq);
router.get('/fetchreqs', ClientRequest.fetchRequests);
module.exports = router;
```

ДОДАТОК В
ФРАГМЕНТИ REDUX

В1 store.js

```
import { configureStore } from '@reduxjs/toolkit';
import { authReducer } from './slices/authSlice';
import { reqsReducer } from './slices/reqSlice';
import { buildReducer } from './slices/buildSlice';
export const store = configureStore({
    reducer: {
        auth: authReducer,
        reqs: reqsReducer,
        builds: buildReducer,
    }
})
```

В2 authSlice.js

```
import { createSlice, createAsyncThunk } from '@reduxjs/toolkit';
import axios from '../../utility/axios';
const initialState = {
    client: null,
    clients: [],
    token: null,
    isAdmin: null,
    loading: false,
    error: null,
    registered: false,
}
export const loginClient = createAsyncThunk('auth/loginClient',
async (params, { rejectWithValue }) => {
    try {
        const { data } = await axios.post('/client/login',
params);
        if (data.token) {
            window.localStorage.setItem('token', data.token);
        }
        return data;
    } catch (error) {
        return rejectWithValue(error.response.data);
    }
})
export const registerClient =
createAsyncThunk('auth/registerClient', async (params, {
rejectWithValue }) => {
    try {
        const { data } = await axios.post('/client/register',
params);
        return data;
    } catch (error) {
        return rejectWithValue(error.response.data);
```

```
    }
})
export const getMe = createAsyncThunk('auth/getMe', async (_, {
rejectWithValue }) => {
    try {
        const { data } = await axios.get('/client/getMe');
        return data;
    } catch (error) {
        return rejectWithValue(error.response.data);
    }
})
export const getUsers = createAsyncThunk('auth/fetchUsers', async
(_, { rejectWithValue }) => {
    const { data } = await axios.get('/client/fetchusers');
    return data;
})
const authSlice = createSlice({
    name: "auth",
    initialState,
    reducers: {
        logout: (state) => {
            state.client = null
            state.isAdmin = null
            state.token = null
            state.error = null
        },
    },
    extraReducers: (builder) => {
        builder
            .addCase(registerClient.pending, (state) => {
                state.loading = false
                state.error = null
            })
            .addCase(registerClient.fulfilled, (state, action) =>
{
                state.loading = true
                state.client = action.payload.data
                state.error = null
            })
            .addCase(registerClient.rejected, (state, action) => {
                state.loading = false
                state.error = action.payload?.msg
            })
            .addCase(loginClient.pending, (state) => {
                state.loading = false
                state.error = null
            })
            .addCase(loginClient.fulfilled, (state, action) => {
                state.loading = true
                state.client = action.payload.data
                state.token = action.payload.token
```

```
                state.isAdmin = Boolean(action.payload.clientRole
=== "manager")
                state.error = null
            })
            .addCase(loginClient.rejected, (state, action) => {
                state.loading = false
                state.error = action.payload.msg
            })
            .addCase(getMe.pending, (state) => {
                state.loading = false
                state.error = null
            })
            .addCase(getMe.fulfilled, (state, action) => {
                state.loading = true
                state.client = action.payload?.data
                state.token = action.payload?.token
                state.isAdmin = action.payload?.clientRole
                state.error = null
            })
            .addCase(getMe.rejected, (state, action) => {
                state.loading = false
                state.error = action.payload.msg
            })
            .addCase(getUsers.pending, (state) => {
                state.loading = false
                state.error = null
            })
            .addCase(getUsers.fulfilled, (state, action) => {
                state.loading = true
                state.clients = action.payload.data
                state.error = null
            })
            .addCase(getUsers.rejected, (state, action) => {
                state.loading = false
                state.error = action.payload.msg
            })
    }
})
export const selectisRegistered = (state) => (state.auth.loading);
export const selectisAuth = (state) => Boolean(state.auth.token)
export const selectIsAdmin = (state) =>
Boolean(state.auth.isAdmin);
export const fetchUser = (state) => (state.auth.client);
export const selectClients = (state) => (state.auth.clients);
export const { logout } = authSlice.actions;
export const authReducer = authSlice.reducer;
```

B3 buildSlice.js

```
import { createAsyncThunk, createSlice } from "@reduxjs/toolkit";
import axios from '../../utility/axios.js';
```

```
const initialState = {
    items: [],
    itemsbyid: [],
    itemsregall: [],
    loading: false,
    error: null,
    success: false,
};
export const registrBuilding = createAsyncThunk('build/registr',
async (params, { _, rejectWithValue }) => {
    try {
        const { data } = await axios.post('/reqbuild/regbuild',
params);
        return data;
    } catch (error) {
        return rejectWithValue(error.response.data);
    }
})
export const getBuildById = createAsyncThunk('build/fetchbyid',
async (_, { rejectWithValues }) => {
    try {
        const { data } = await
axios.get('/reqbuild/fetchregbuild');
        return data;
    } catch (error) {
        return rejectWithValues(error.response.data);
    }
})
export const fetchRegBuilds =
createAsyncThunk('build/fetchregall', async (_, { rejectWithValues
}) => {
    try {
        const { data } = await axios.get('/reqbuild/fetchbuilds');
        return data;
    } catch (error) {
        return rejectWithValues(error.response.data);
    }
})
//Fetch all non registered buildings
export const getBuildings = createAsyncThunk('build/fetchall',
async (_, { rejectWithValues }) => {
    const { data } = await axios.get('/build/fetchbuild');
    return data;
})
const buildSlice = createSlice({
    name: "builds",
    initialState,
    reducers: {
    },
    extraReducers: (builder) => {
        builder
```

```
            .addCase(fetchRegBuilds.pending, (state) => {
                state.loading = false
                state.error = null
            })
            .addCase(fetchRegBuilds.fulfilled, (state, action) =>
{
                state.loading = true
                state.itemsregall = action.payload.reqbuilds
                state.error = null
            })
            .addCase(fetchRegBuilds.rejected, (state, action) => {
                state.loading = false
                state.error = action.payload.msg
            })
            .addCase(getBuildings.pending, (state) => {
                state.loading = false
                state.error = null
            })
            .addCase(getBuildings.fulfilled, (state, action) => {
                state.loading = true
                state.items = action.payload.data
                state.error = null
            })
            .addCase(getBuildings.rejected, (state, action) => {
                state.loading = false
                state.error = action.payload.msg
            })
            .addCase(registrBuilding.pending, (state) => {
                state.loading = false
                state.error = null
            })
            .addCase(registrBuilding.fulfilled, (state, action) =>
{
                state.loading = true
                state.error = null
                state.success = true
            })
            .addCase(registrBuilding.rejected, (state, action) =>
{
                state.loading = false
                state.error = action.error
            })
            .addCase(getBuildById.pending, (state) => {
                state.loading = false
                state.error = null
            })
            .addCase(getBuildById.fulfilled, (state, action) => {
                state.loading = true
                state.itemsbyid = action.payload.data
            })
            .addCase(getBuildById.rejected, (state, action) => {
```

```
                state.loading = false
                state.error = action.payload.msg
            })

    }

})
export const regSubmitted = (state) => (state.builds.success);
export const selectedBuildings = (state) => (state.builds.items);
export const selectedRegBuild = (state) =>
(state.builds.itemsregall);
export const selectClientBuild = (state) =>
(state.builds.itemsbyid);
export const buildReducer = buildSlice.reducer;
```

B4 reqSlice.js

```
import { createAsyncThunk, createSlice } from "@reduxjs/toolkit";
import axios from '../../utility/axios.js';

const initialState = {
    items: [],
    loading: false,
    error: null,
    success: false,
};

export const createReq = createAsyncThunk('reqs/create', async
(params, { rejectWithValues }) => {
    try {
        const { data } = await axios.post('/reqs/createReq',
params);
        return data;
    } catch (error) {
        return rejectWithValues(error.response.data);
    }
})

export const fetchReqs = createAsyncThunk('reqs/fetchAll', async
(_, { rejectWithValues }) => {
    try {
        const { data } = await axios.get('/reqs/fetchreqs');
        return data;
    } catch (error) {
        return rejectWithValues(error.response.data);
    }
})

const reqSlice = createSlice({
    name: "reqs",
    initialState,
```

```
    reducers: {

    },
    extraReducers: (builder) => {
        builder
            .addCase(createReq.pending, (state) => {
                state.loading = false
                state.error = null
            })
            .addCase(createReq.fulfilled, (state, action) => {
                state.loading = true
                state.success = true
                state.error = null
            })
            .addCase(createReq.rejected, (state, action) => {
                state.loading = false
                state.error = action.payload.msg
            })
            .addCase(fetchReqs.pending, (state) => {
                state.loading = false
                state.error = null
            })
            .addCase(fetchReqs.fulfilled, (state, action) => {
                state.loading = true
                state.items = action.payload.requests
                state.error = null
            })
            .addCase(fetchReqs.rejected, (state, action) => {
                state.loading = false
                state.error = action.payload.msg
            })
    }
})
export const isSent = ((state) => state.reqs.success);
export const reqsReducer = reqSlice.reducer;
export const getAllReqs = (state) => (state.reqs.items);
```