МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА САЙТУ З ПРОДАЖУ НЕРУХОМОСТІ ЗАСОБАМИ REACT ТА NODE.JS»**

Виконав: студент 2 курсу, групи 8.1211-1іпз
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми інженерія програмного забезпечення
(назва освітньої програми)
В.С. Дмитрієв
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, к.ф.-м.н. Мильцев О.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент завідувач кафедри фундаментальної та прикладної математики, професор, д.т.н. Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2022

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент
_____ Лісняк А.О.
(підпис)

« ____ » _____ 2022 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Дмитрієву Валерію Сергійовичу
(прізвище, ім'я та по–батькові)

1. Тема роботи Розробка сайту з продажу нерухомості засобами React та Node.js

керівник роботи Мильцев Олександр Михайлович, к.ф.-м.н.
(прізвище, ім'я та по–батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 04 » травня 2022 року № 500-с

2. Строк подання студентом роботи 01.12.2022

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі, аналіз предметної області.
2. Проєктування програмного доповнення.
3. Реалізація та тестування програмного доповнення.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 04.05.2022

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану виконання кваліфікаційної роботи магістра.	01.06.2022	
2.	Збір вихідних даних та аналіз предметної області.	08.07.2022	
3.	Обробка методичних та теоретичних джерел.	12.08.2022	
4.	Специфікація вимог до системи. Робота над першим розділом.	21.09.2022	
5.	Проєктування системи. Робота над другим розділом.	09.10.2022	
6.	Реалізація та тестування системи. Робота над третім розділом.	13.11.2022	
7.	Оформлення та нормоконтроль кваліфікаційної роботи магістра.	01.12.2022	
8.	Захист кваліфікаційної роботи магістра.	15.12.2022	

Студент _____ В.С. Дмитрієв
(підпис) (ініціали та прізвище)

Керівник роботи _____ О.М. Мильцев
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка сайту з продажу нерухомості засобами React та Node.js»: 65 с., 36 рис., 9 джерел, 1 додаток.

БІБЛІОТЕКА, НЕРУХОМІСТЬ, EXPRESS.JS, MYSQL, NODE.JS, REACT, UML.

Об’єкт дослідження – інструменти для створення веб додатків з продажу нерухомості за допомогою React та Node.js.

Мета роботи – розробити веб додаток для продажу нерухомості.

Методи дослідження – моделювання, проєктування, програмний, аналітичний.

У роботі проаналізовано сучасний фрейморк та бібліотеку на основі JavaScript для створення веб застосунків, більш детально описано React та Express.js.

Обґрунтовано використання Express.js для написання веб застосунків, а саме його гнучкість, завдяки котрій можливо виконувати широкий спектр задач без зайвого витрачення часу. Оскільки Express.js був створений на базі Node.js, він також отримав і такі переваги як продуктивність, масштабованість. До переваг React можна віднести такі технічні можливості як використання віртуальної об’єктної моделі документу, повторне застосування компонентів, швидка навігація всередині веб застосунку, наявність сховищ даних, таких як Redux.

В результаті спроектовано та створено веб застосунок для зручного продажу нерухомості на території всієї України за допомогою React та фрейморку Express.js.

SUMMARY

Master's qualifying paper «Development of the Website for the Sale of Real Estate using React and Node.js»: 65 pages, 36 figures, 9 references, 1 supplement.

LIBRARY, REAL ESTATE, EXPRESS.JS, MYSQL, NODE.JS, REACT, UML.

The object of the study is tools for creating real estate web applications using React and Node.js.

The purpose of the work is to develop a web application for real estate sales.

The methods of research are modeling, design, software, analytical.

The paper analyzes a modern JavaScript-based framework and library for creating web applications, React and Express.js are described in more detail.

The rationale for using Express.js for writing its web applications is its flexibility, which makes it possible to perform a wide range of tasks without spending too much time. Since Express.js was created on the basis of Node.js, it also received such advantages as performance, scalability. The advantages of React include such technical capabilities as the use of a virtual object model document, component reusability, fast navigation in a central web application, and the availability of a data store such as Redux.

As a result, a website for the sale of convenient real estate throughout Ukraine was designed and created using React and the Express.js framework.

ЗМІСТ

ВСТУП

Актуальність сайтів з продажу нерухомості завжди була досить високою, адже кожній людині потрібне житло. Зайшовши на таку платформу людина одразу бачить пропозиції всіх рієлторських агенцій міста. Для агенств, в свою чергу, знаходження на такій платформі є більш актуальне, ніж розміщати пропозиції на власному сайті, адже розміщуючи пропозиції вони зможуть отримати більше потенційних клієнтів, ніж на власному сайті. Власник такої платформи має можливість брати певний відсоток від вартості пропозиції, за розміщення оголошення.

Актуальність дослідження: дослідження даної теми стало актуальним завдяки масовості використання інтернету для пошуку житла, а також, на жаль, через бойові дії на території України, багато людей потребують нового житла.

Беручи до уваги все перераховане вище, ми можемо визначити наступні цілі та задачі для нашого дослідження:

Мета: спроєктувати та розробити веб додаток для продажу нерухомості від ім’я агенств засобами React та Node.js.

Задачі:

1) сформувати вимоги до вебдодатку;
2) спроектувати архітектуру вебдодатку;
3) реалізувати проєкт вебдодатку;
4) виконати тестування вебдодатку.

Об’єкт дослідження – інструменти для написання веб додатків з використанням React та Node.js.

Предмет дослідження: створення веб додатку, для розміщення пропозицій з продажу нерухомості на території України.

Методи дослідження – моделювання, проєктування, програмний, аналітичний.

Перший розділ кваліфікаційної роботи містить формування вимог до веб додатку, опис та можливості схожих вебдодатків для продажу нерухомості, а також опис інструментів розробки проєкту.

У другому розділі показано етапи проєктування системи продажу нерухомості, а також, приведено опис прецедентів системи, діаграму бази даних системи та діаграми діяльності основної бізнес логіки користувачів системи.

Третій розділ містить інформацію про реалізацію та тестування вебдодатку з продажу нерухомості.

1 ТЕХНІЧНЕ ЗАВДАННЯ

1.1 Терміни та визначення

1.1.1 Загальні терміни

Система – вебдодаток створений за допомогою React та Express.js.

Sequelize – це ORM (Object-Relational Mapping – об'єктно-реляційне відображення або перетворення) для роботи з СУБД.

Геокодування – це процес призначення географічних ідентифікаторів (таких як географічні координати, виражені у вигляді широти та довготи) об'єктам карти.

Користувач – людина, зареєстрована в системі з певними правами доступу до функціональності веб додатку.

Клієнт – користувач системи, з обмеженою можливістю взаємодії з системою (перегляд оголошень, відгук на оголошення).

Адміністратор – користувач системи, з можливістю редагування системи (додавання агенцій/рієлторів/оголошень, видалення агенцій/рієлторів/оголошень, редагування агенцій/рієлторів/оголошень, створення агенцій/рієлторів/оголошень, перегляд агенцій/рієлторів/оголошень).

1.1.2 Технічні терміни

ІС – інформаційна система.

БД – база даних проекту, місце збереження інформації ІС.

API – прикладниий програмний інтерфейс, набір чітко визначених методів для взаємодії різних компонентів.

Компонент – структурна одиниця React, з реалізацією функціоналу та дизайну системи.

Модель – надає дані бази даних та реагує на команди від контролера.

Контролер – реалізація певного функціоналу для взаємодією з моделлю даних.

Сховище – містись стан даних всього веб додатку.

1.2 Функціональні вимоги

1.2.1 Призначення створеної системи

Функціональні вимоги до системи – спроєктувати та реалізувати функції платформи з продажу нерухомості, а саме реєстрація агенств та рієлторів, додавання оголошень, перегляд оголошень, редагування оголошень, видалення оголошень.

Експлуатаційне призначення системи – система призначена для використання агенціями, рієлторами, клієнтами та адміністраторами.

Мета створення системи – реалізація платформи для продажу нерухомості на всій території України.

1.2.2 Функціональні можливості системи

Адміністратор інформаційної системи повинен мати наступні функціональні можливості:

– додавання/редагування/видалення/перегляд/пошук агенцій;
– додавання/редагування/видалення/перегляд/пошук рієлторів;
– додавання/редагування/видалення/перегляд/пошук оголошень;

– видалення/створення/редагування/перегляд/пошук облікового запису користувача;
– створення/видалення токену авторизації користувача.

Клієнт інформаційної системи повинен мати наступні функціональні можливості:

– перегляд/пошук агенцій;
– перегляд/пошук рієлторів;
– перегляд/пошук оголошень;
– фільтрування оголошень за крітеріями;
– видалення/створення/редагування облікового запису користувача;
– оцінити рієлтора;
– перегляд місця знаходження нерухомості на мапі.

Обліковий запис агенції інформаційної системи повинен мати наступні функціональні можливості:

– реєстрація/перегляд/видалення/редагування облікового запису рієлтора;
– реєстрація/перегляд/видалення/редагування власного облікового запису в системі;
– перегляд оголошень своїх рієлторів;
– відправка повідомлень своїм рієлторам.

Обліковий запис рієлтора інформаційної системи повинен мати наступні функціональні можливості:

– перегляд/редагування власного облікового запису в системі;
– додавання/редагування/видалення оголошень з продажу нерухомості;
– відправка повідомлень своїй агенції.

1.2.3 Нефункціональні вимоги

Інтерфейс користувача.

Інформаційна система повинна коректно відображати інтерфейс користувача на будь-якому пристрої.

Інтерфейс повинен бути інтуїтивно зрозумілим для користувача.

Підтримка браузерів.

Система повинна працювати для наступних браузерів останніх версій:

– MS Internet Explorer;
– MS Edge;
– Mozilla Firefox;
– Google Chrome;
– Safari;
– Opera.

Вимоги до безпеки.

Інформаційна система повинна надавати доступ до інтерфейсу адміністративної панелі тільки користувачу з правами адміністратора.

Інформаційна система повинна надавати доступ редагування та перегляд повних даних користувачів тільки адміністратору, або самому користувачу.

1.3 Опис області кваліфікаційної роботи

Областю кваліфікаційної роботи є створення веб додатку для продажу нерухомості по всій території України. Ця інформаційна система має працювати наступним чином.

Агенція реєструється в системі, вносячи свої дані, а саме назву, місто та місце розташування, скільки агенція існує, контактний номер та додаткову інформацію. Адміністратор перевіряє цю агенцію та активує її аккаунт. Після активації акаунту агенція додає свої рієлторів до БД системи, заповнюючи їх

дані, такі як ім’я, прізвище, стаж роботи, контактний номер, та іншу додаткову інформацію. Після створення акаунту рієлтора, він має змогу додавати оголошення до БД інформаційної системи, заповнивши таку інформацію як, розташування будівлі (місто, район, вулиця, будинок), додавши фото пропозиції, вказавши ціну, площу, поверх, поверховість та іншу додаткову інформацію.

Клієнт, в свою чергу, має змогу обирати пропозиції за вказаними фільтрами, такими як наприклад місто, район, ціна, поверх та кількість кімнат. Знайшовши необхідну пропозицію клієнт переходить до сторінки з детальною інформацією пропозиції, переглянувши інформацію (опис, деталі, місце розташування на мапі), клієнт маж змогу зв’язатися з рієлтором за допомогою соціальних мереж, або набравши його за вказаним номером.

1.4 Огляд можливостей інструменту розробки React

React вже зарекомендував себе як потужний інструмент для створення інтерфейсів користувача. Інтерфейс користувача (UI) – одна з найважливіших частин веб-програми, адже це те, що користувач бачить взаємодіє [1].

Об'єктна модель документа (DOM) визначає деревоподібну структуру HTML-документа, що надсилається клієнту сервером після відповідного запиту. DOM представляє веб-сторінку в об'єктно-орієнтованому форматі, щоб мови програмування могли взаємодіяти з нею [1].

Браузер регулярно перевіряє будь-які зміни в DOM, оновлюючи її належним чином. Зміна об'єктної моделі документа провокується безліччю факторів. Браузер оновлює DOM щоразу, коли сторінка змінюється. Оскільки DOM сучасних сайтів є величезними, оновлення займає багато часу, уповільнюючи загальну продуктивність веб-додатку [1].

Замість повільних та незручних взаємодій безпосередньо з реальною об'єктною моделлю документа, React взаємодіє з її полегшеною копією –

віртуальної DOM, тому реальна DOM оновлюється лише після взаємодії з віртуальною DOM [1].

Отже, віртуальний DOM забезпечує розробникам дві серйозні переваги. Про них зараз і поговоримо [1].

Ефективність.

Оновлення всього DOM, щоб зробити веб-сторінку “реактивною” – вкрай неефективно, оскільки споживає надто багато ресурсів. Тому, власне, при зміні веб-сторінки (наприклад, через запит або дію користувача) React оновлює спеціальний віртуальний DOM [1].

React зберігає в пам'яті дві версії віртуального DOM – оновлений віртуальний DOM та його резервну копію, створену до оновлення. Після оновлення React порівнює обидві версії між собою, щоб знайти змінені елементи, а потім – оновлює частину реального DOM, що виключно змінилася [1].

Висока продуктивність.

Одна з життєво важливих цілей будь-якого стартапу – написати веб-додаток швидким і чуйним, забезпечити найкраще обслуговування клієнтів. Віртуальна DOM, на відміну від реального DOM, займає мало місця та швидко оновлюється, тим самим підвищуючи продуктивність програми [1].

Віртуальна DOM дозволяє сторінці негайно отримувати відповіді від сервера та відображати оновлення. Наприклад, Facebook застосовує технологію віртуального DOM для оновлення чатів та стрічок користувачів без перезавантаження сторінки [1].

При роботі з ReactJS створюються багаторазові компоненти: найчастіше, компонент інтерфейсу користувача можна використовувати в інших частинах коду або навіть у різних проектах практично без змін [1].

Більш того, розробникам React-програм доступні бібліотеки готових компонентів з відкритим вихідним кодом. Завдяки їм час розробки скорочується, що дуже важливо для стартапів, яким необхідно заощаджувати не лише гроші, а й час [1].

Односторонній потік даних в React – ще одна дуже корисна функція. Такий потік даних також називають "згори донизу" або "від батька до дитини" [1].

Пояснення досить просте – у React-додатку дані між елементами передаються тільки одним способом. Східний потік даних запобігає помилкам, полегшує налагодження [1].

Але ReactJS не можна назвати повноцінним фреймворком, хоч таке його уявлення і поширене. React – це не інструмент “все в одному” для створення цілої програми. Отже, якщо ви слідуєте патерну проектування "Модель-Подання-Контролер" (Model-View-Controller або MVC), то React буде відповідати лише за подання. Дві інші частини – модель і контролер – створюються за допомогою додаткових інструментів. Якщо ви застосовуєте React у проекті, вам потрібно буде інтегрувати додаткові інструменти для маршрутизації, написання інтерфейсів прикладного програмування (Application Programming Interface або API) та інших частин повноцінного веб-додатку [1].

1.5 Огляд можливостей інструменту розробки Node.js

Node.js – платформа з відкритим кодом для виконання високопродуктивних мережевих застосунків, написаних мовою JavaScript. Засновником платформи є Раян Дал (Ryan Dahl). Якщо раніше JavaScript застосовувався для обробки даних в браузері користувача, то node.js надав можливість виконувати JavaScript-скрипти на сервері та відправляти користувачеві результат їхнього виконання. Платформа Node.js перетворила JavaScript на мову загального використання з великою спільнотою розробників [2].

Node.js має наступні властивості [2]:

– асинхронна одно-нитева модель виконання запитів;

– неблокуючий ввід/вивід;
– система модулів CommonJS;
– рушій JavaScript Google V8.

Для керування модулями використовується пакетний менеджер npm (node package manager) [2].

Платформа Node.js призначена для виконання високопродуктивних мережевих застосунків, написаних мовою програмування JavaScript. Платформа окрім роботи із серверними скриптами для веб-запитів, також використовується для створення клієнтських та серверних програм. Платформа використовує розроблений компанією Google рушій V8 [2].

Для забезпечення обробки великої кількості паралельних запитів у Node.js використовується асинхронна модель запуску коду, заснована на обробці подій в неблокуючому режимі та визначенні обробників зворотніх викликів (callback). Як способи мультиплексування з'єднань підтримується epoll, kqueue, /dev/poll і select. Для мультиплексування з'єднань використовується бібліотека libuv, для створення пулу нитей (thread pool) задіяна бібліотека libeio, для виконання DNS-запитів у неблокуючому режимі інтегрований c-ares. Всі системні виклики, що спричиняють блокування, виконуються всередині пулу потоків і потім, як і обробники сигналів, передають результат своєї роботи назад через неіменовані канали (pipe) [2].

За своєю суттю Node.js схожий на фреймворки Perl AnyEvent, Ruby Event Machine і Python Twisted, але цикл обробки подій (event loop) у Node.js прихований від розробника і нагадує обробку подій у веб застосунку, що працює в браузері. При написанні програм для Node.js необхідно враховувати специфіку подієво-орієнтованого програмування, з очікуванням завершення роботи і наступною обробкою результатів, Node.js використовує принцип асинхронного виконання [2].

При цьому управління миттєво перейде до коду який слідує після виклику функції db.query, а результат запиту буде оброблений як тільки будуть оброблені дані. Жодна функція в Node.js не повинна безпосередньо

виконувати (блокуючі) операції вводу/виводу – для отримання даних з диска, від іншого процесу або з мережі потрібна установка callback-обробника [2].

1.6 Використання фреймворку Express на базі Node.js для реалізації серверної частини додатку

Express.js, або просто Express – програмний каркас розробки серверної частини веб застосунків для Node.js, реалізований як вільне і відкрите програмне забезпечення під ліцензією MIT. Він спроєктований для створення веб застосунків і API. Де-факто є стандартним каркасом для Node.js. Автор фреймворка, TJ Holowaychuk, описує його як створений на основі написаного на мові Ruby каркаса Sinatra, маючи на увазі, що він мінімалістичний, але має велику кількість плаґінів, що підключаються [3].

Більшість веб додатків, серверна частина яких написана за допомогою Node.js, використовують фреймворк Express, адже він сильно скорочує та спрощує розробку за рахунок можливостей яких не має Node.js. Для цього лише потрібно встановити пакет Express за допомогою node package manager, та підключити його до проєкту. В свою чергу Express дає можливість використовувати такі основні можливості як:

- проміжні обробники, які дають нам змогу розбити виконання функціоналу на окремі функції, для більш чіткого розуміння коду;
- маршрути, для чіткої структуризації контролерів за їх приналежністю до певної моделі;
- функції обробки HTTP запитів, що прибирає потрібність їх створювати;
- роздача статичних файлів клієнтській частині веб додатків.

Окрім основних можливостей, Express дозволяє використовувати вже існуючі модулі, для вирішення майже всіх поставлених задач перед розробником.

2 ПРОЄКТУВАННЯ ПЛАТФОРМИ З ПРОДАЖУ НЕРУХОМОСТІ

2.1 Використання UML для моделювання розробки платформи з продажу нерухомості

Першим етапом створення будь-якої інформаційної системи є отримання вимог від замовника, та подальше їх обговорення. На цьому етапі відбувається підбір технологій розробки та моделювання, оцінка схожих проєктів. Результатом першого етапу є складене технічне завдання до проєкту, згідно з отриманою та проаналізованою інформацією.

Другим етапом для розробників є моделювання поведінки, сценаріїв використання інформаційної системи. Моделі компонентів системи дають змогу продумати функціональні можливості інформаційної системи, до початку реалізації, що в свою чергу прибирає велику кількість майбутніх помилок та мінімізує витрати часу на переробку певної функціональності системи. Для досягнення такого результату потрібно дотримуватись ієрархічного принципу побудови моделей, застосування графічної нотації, а також моделювання з використанням різного рівня деталізації.

Графічне представлення моделей дозволяє розуміти життєвий цикл, та інші аспекти системи людям займаючи різні рівні роботи над проєктом. Саме для досягнення такого ефекту, графічне представлення системи повинне бути уніфікованим, тобто необхідно використовувати мову, що визначає дану нотацію та її семантику. Одним з найбільш поширених представником уніфікованих мов моделювання є UML.

Unified Modeling Language (UML) – уніфікована мова моделювання, підходить для широкого класу проєктованих програмних систем, різних областей додатків, типів організацій, рівнів компетентності, розмірів проєктів. UML описує об'єкт в єдиному заданому синтаксисі, тому де б ви не

намалювали діаграму, її правила будуть зрозумілими для всіх, хто знайомий з цією графічною мовою – навіть в іншій країні [4].

Одне із завдань UML – служити засобом комунікації всередині команди та при спілкуванні з замовником. Давайте розгляднемо можливі варіанти використання дігарам [4]:

1) проектування – UML – діаграми стануть у пригоді при моделюванні архітектури великих проектів, в якій можна зібрати як великі, так і дрібніші деталі і намалювати каркас (схему) програми;
2) реверс–інжиніринг – створення UML-моделі з існуючого коду додатку, зворотна побудова;
3) з моделей можна витягувати текстову інформацію і генерувати відносно читабельні тексти – документувати.

Беручи до уваги все вище написане, задля уникнення великої кількості помилок, а також для розуміння основних функціональних можливостей проєкту, було вирішено використовувати UML для моделювання компонентів системи.

2.2 Діаграма варіантів використання системи

Діаграма прецедентів є графом, що складається з множини акторів, прецедентів (варіантів використання) обмежених межею системи (прямокутник), асоціацій між акторами та прецедентами, відношень серед прецедентів, та відношень узагальнення між акторами. Діаграми прецедентів відображають елементи моделі варіантів використання [5].

Суть діаграми прецедентів полягає в тому, що проєктована система подається у вигляді множини сутностей чи акторів, що взаємодіють із системою за допомогою так званих варіантів використання. Варіант використання використовують для описання послуг, які система надає актору.

При цьому нічого не говориться про те, яким чином буде реалізовано взаємодію акторів із системою [5].

У мові UML є кілька стандартних видів відношень між акторами і варіантами використання [5]:

– асоціації (англ. association relationship);
– включення (англ. include relationship);
– розширення (англ. extend relationship);
– узагальнення (англ. generalization relationship).

При цьому загальні властивості варіантів використання можна подати трьома різними способами, а саме – за допомогою відношень включення, розширення і узагальнення [5].

Відношення асоціації – одне з фундаментальних понять у мові UML і в тій чи іншій мірі використовується під час побудови всіх графічних моделей систем у формі канонічних діаграм [5].

Включення (англ. include) у мові UML – це різновид відношення залежності між базовим варіантом використання і його окремим випадком. При цьому відношенням залежності (англ. dependency) є таке відношення між двома елементами моделі, за якого зміна одного елемента (незалежного) спричиняє зміну іншого елемента (залежного) [5].

Відношення розширення (англ. extend) визначає взаємозв'язок базового варіанту використання з іншим варіантом використання, функціональна поведінка якого залучається базовим не завжди, а тільки за виконання додаткових умов [5].

Діаграма варіантів використання (рис. 2.1) відображає основні функціональні можливості та користувачів системи. Згідно цієї діаграми ми бачимо, що кожен актор є користувачем системи, тому частина функціональних можливостей будуть однакові, в той час як інша частина унікальна для кожного актора.

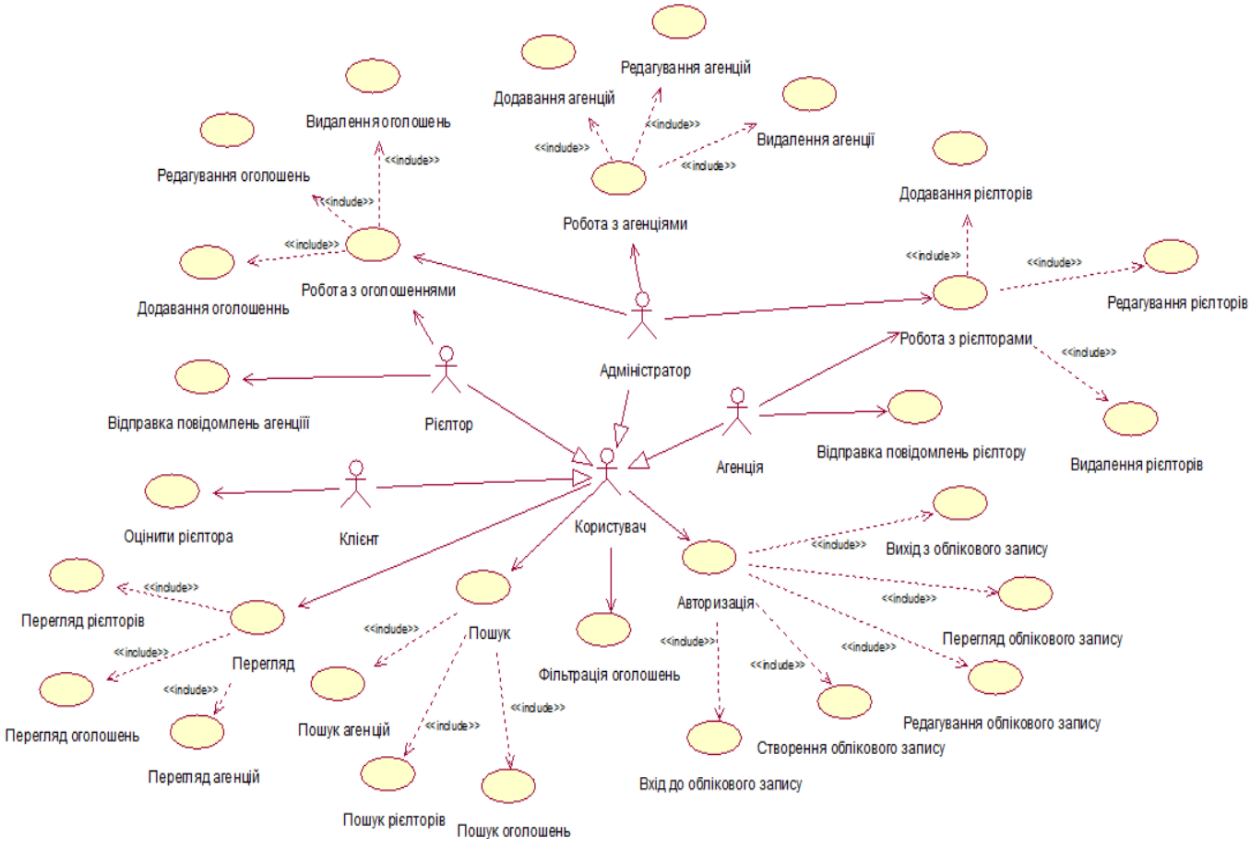

Рисунок 2.1 – Діаграма прецедентів використання системи

Більш детальний розбір діаграми розпочнемо з адміністратора та його можливостей (рис. 2.2). Адміністратор окрім загальних можливостей користувача з авторизації та пошуку даних, має змогу переглядати повну інформацію агенції, рієлтора, чи оголошення, адже всім іншим користувачам системи не відображається вся наявна інформація щодо певного запису в системі. Також адміністратор має змогу редагувати будь-який обліковий запис, чи оголошення в системі, такі можливості надані, на випадок, якщо користувач системи забуде власні данні для входу, чи матиме якісь проблеми з редагуванням власних даних. Адміністратор системи має змогу створювати облікові записи агенцій, рієлторів та клієнтів, а також видаляти будь-який запис з системи.

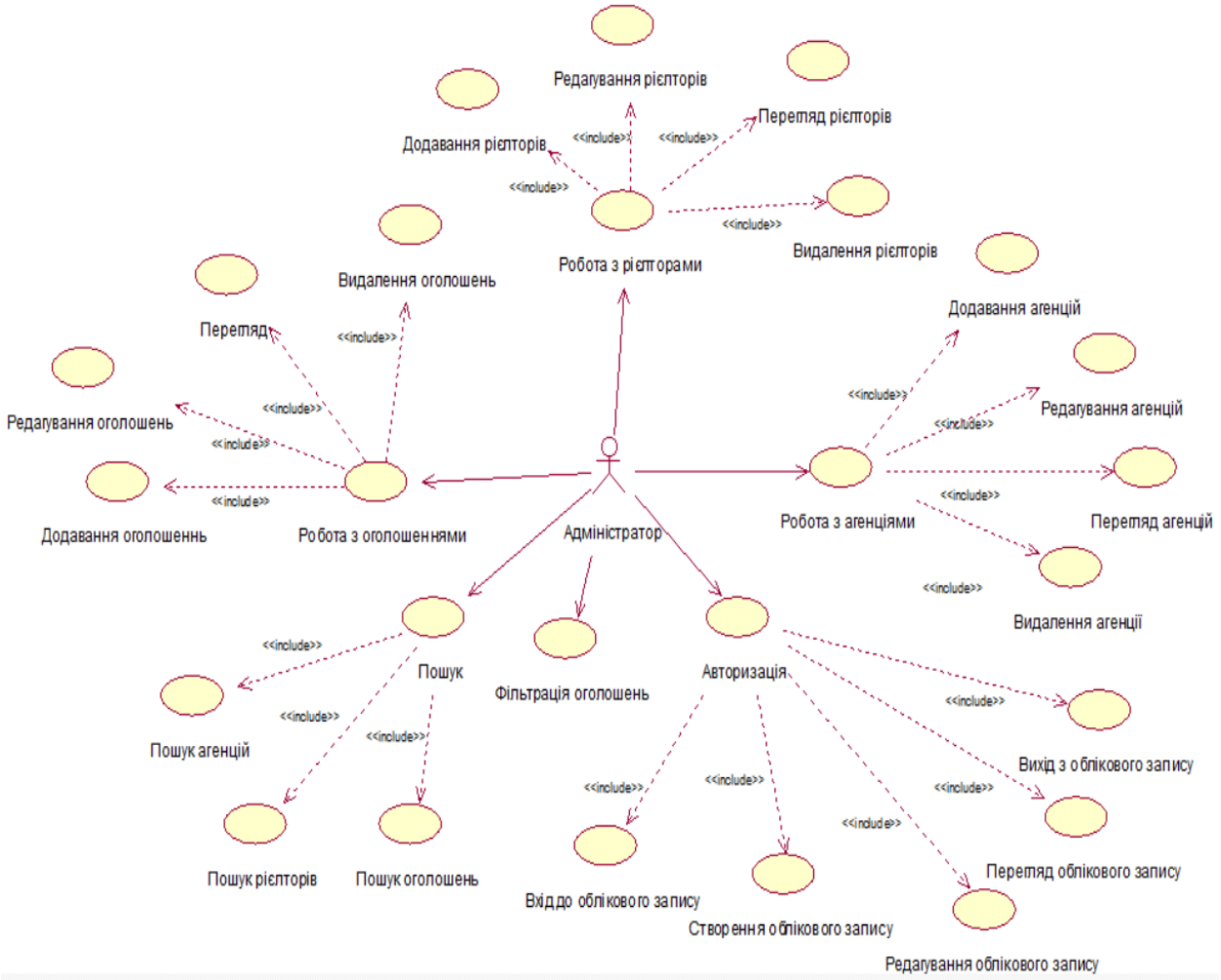

Рисунок 2.2 – Діаграма варіантів використання адміністратора

Основними функціональними можливостями агенції є додавання, редагування та видалення власних рієлторів. Після реєстрації в системі, агенція створює облікові записи своїх рієлторів, які в свою чергу мають змогу викладати оголошення з продажу нерухомості. Агенції окрім можливості переглядати інформацію щодо інших агенцій, рієлторів та оголошень, після авторизації в системі мають змогу переглядати всі оголошення своїх рієлторів, відправляти повідомлення власним рієлторам, а також переглядати та редагувати повну інформацію кожного свого рієлтора. Більш детальна діаграма варіантів використання агенції зображена на рисунку (рис. 2.3).

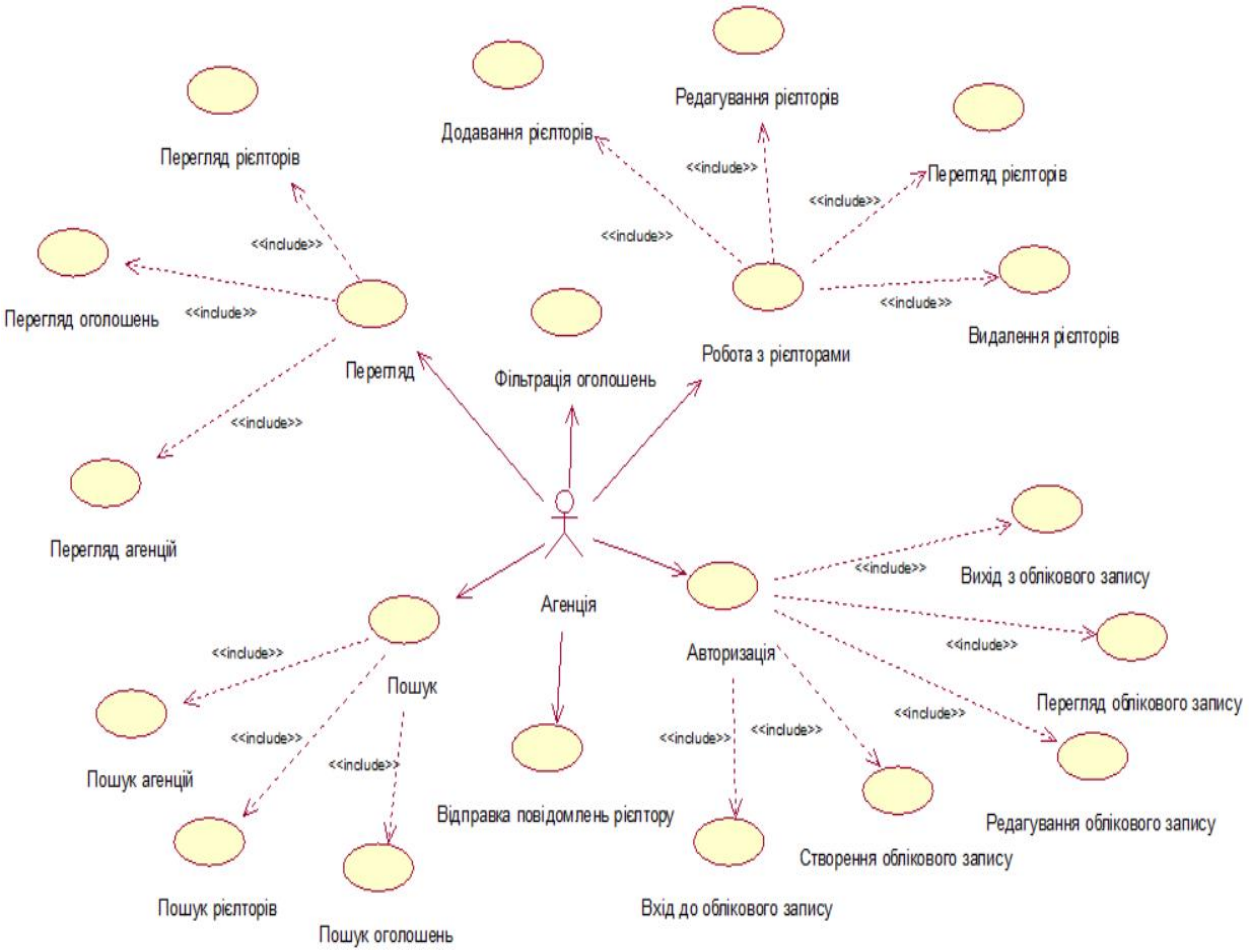

Рисунок 2.3 – Діаграма варіантів використання агенції

Функціональні можливості рієлтора дозволяють йому переглядати інших рієлторів, агенції, та оголошення, а також після авторизації в системі рієлтор має змогу додавати, редагувати, видаляти оголошення, переглядати повну інформацію всіх власних оголошень, надсилати повідомлення власній агенції. Навідміну від можливостей агенції, рієлтор має змогу лише переглядати повну інформацію власного облікового запису, та редагувати його, але не видаляти.

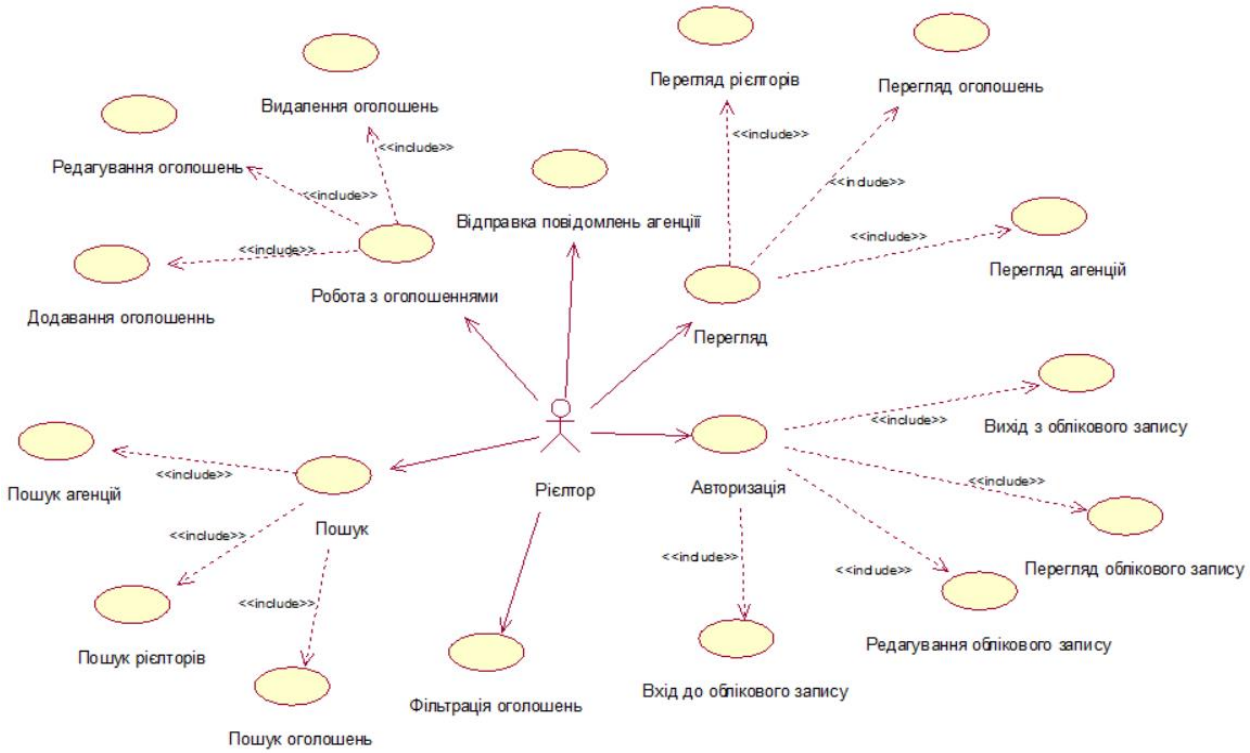

Рисунок 2.4 – Діаграма варіантів використання рієлтора

Основними функціональними можливостями системи для клієнта – це перегляд, пошук, фільтрація оголошень. Клієнт починає пошук з головної сторінки сайту, якщо клієнт потрапляє на сайт вперше, йому автоматично показує оголошення в місті Київ, клієнт вказує потрібне йому місто, а також інші параметри пошуку, та отримує всі оголошення від агенцій цього міста. Якщо клієнта зацікавила пропозиція, він переходить до її сторінки з більш детальною інформацією даної пропозиції (фотографії, опис, деталі, місце знаходження на мапі). На сторінці пропозиції клієнт також бачить короткі дані рієлтора, який виклав це оголошення, якщо клієнт й далі зацікавлений цією пропозицією, він дзвонить, або пише повідомлення в соціальних мережах, які вказав рієлтор. Також клієнт має змогу оцінювати рієлтора, перейшовши на сторінку потрібного рієлтора, здійснювати пошук потрібного рієлтора, чи агенції.

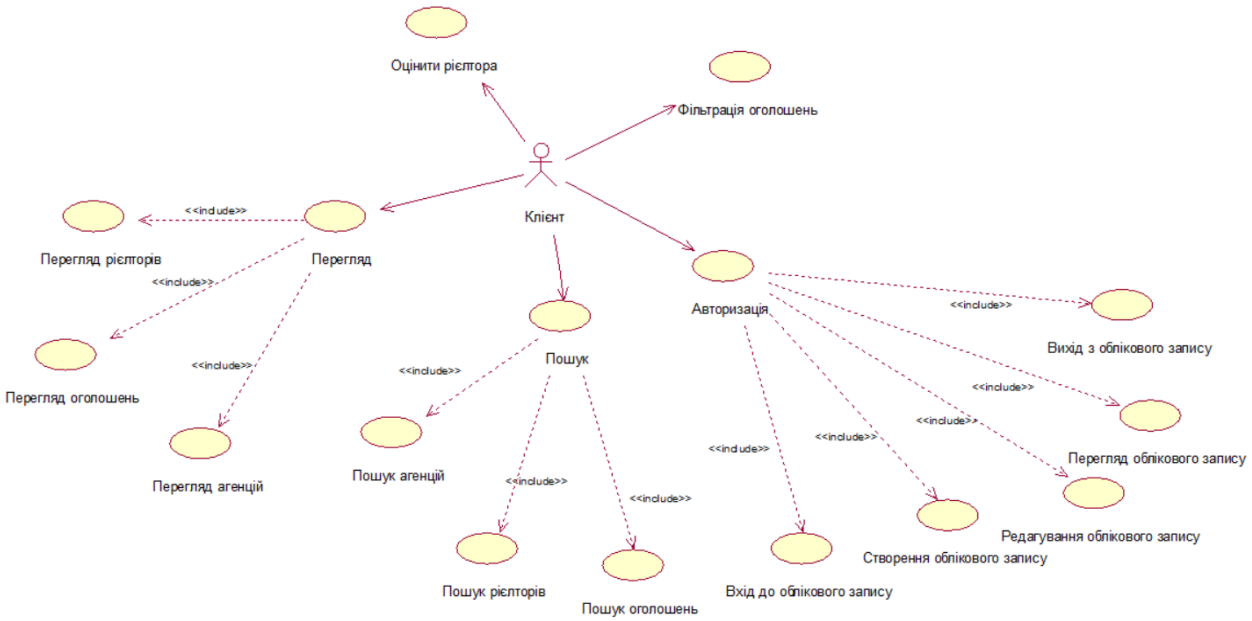

Рисунок 2.5 – Діаграма варіантів використання клієнта

2.2.1 Опис прецедентів системи

Прецедент «Додавання агенцій».

Виконавець: адміністратор.

Призначення: цей прецедент надає можливість адміністратору платформи з продажу нерухомості створювати облікові записи агенцій.

Основний потік подій: виконується після натискання «Додати агенцію». Відкривається інтерфейс реєстрації агенції.

Передумова: перед виконанням прецеденту, адміністратор повинен бути авторизований в системі.

Прецедент «Редагування агенцій».

Виконавець: адміністратор.

Призначення: цей прецедент надає можливість адміністратору платформи з продажу нерухомості редагувати облікові записи агенцій.

Основний потік подій: виконується після натискання «Редагувати запис». Відкривається інтерфейс редагування облікового запису.

Передумова: перед виконанням прецеденту, адміністратор повинен бути авторизований в системі.

Прецедент «Видалення агенцій».

Виконавець: адміністратор.

Призначення: цей прецедент надає можливість адміністратору платформи з продажу нерухомості видаляти облікові записи агенцій.

Основний потік подій: виконується після натискання «Видалити запис». Обліковий запис видаляється після підтвердження.

Передумова: перед виконанням прецеденту, адміністратор повинен бути авторизований в системі.

Прецедент «Перегляд агенцій».

Виконавець: адміністратор, агенція, рієлтор, клієнт.

Призначення: цей прецедент надає можливість адміністратору платформи з продажу нерухомості переглядати всі наявні дані облікового запису агенції, всі інші користувачі системи мають можливість переглядати часткову інформацію облікового запису агенцій.

Основний потік подій: виконується після натискання на потрібну агенцію. Відкривається інтерфейс перегляду облікового запису.

Передумова: перед виконанням прецеденту, адміністратор повинен бути авторизований в системі.

Прецедент «Додавання рієлторів».

Виконавець: адміністратор, агенція.

Призначення: цей прецедент надає можливість адміністратору платформи з продажу нерухомості та агенції створювати обліковий запис рієлтора.

Основний потік подій: виконується після натискання «Додати рієлтора». Відкривається інтерфейс реєстрації облікового запису рієлтора.

Передумова: перед виконанням прецеденту, адміністратор або агенція повинні бути авторизовані в системі.

Прецедент «Редагування рієлторів».

Виконавець: адміністратор, агенція.

Призначення: цей прецедент надає можливість адміністратору платформи з продажу нерухомості та агенції редагувати обліковий запис рієлтора.

Основний потік подій: виконується після натискання «Редагувати запис». Відкривається інтерфейс редагування облікового запису рієлтора.

Передумова: перед виконанням прецеденту, адміністратор або агенція повинні бути авторизовані в системі.

Прецедент «Перегляд рієлторів».

Виконавець: адміністратор, агенція, рієлтор, клієнт.

Призначення: цей прецедент надає можливість адміністратору платформи з продажу нерухомості та агенції переглядати повну інформацію облікового запису рієлтора, всі інші користувачі системи мають можливість переглядати часткову інформацію облікового запису рієлтора.

Основний потік подій: виконується після натискання на потрібний обліковий запис. Відкривається інтерфейс перегляду облікового запису рієлтора.

Передумова: перед виконанням прецеденту, адміністратор або агенція повинні бути авторизовані в системі.

Прецедент «Видалення рієлторів».

Виконавець: адміністратор, агенція.

Призначення: цей прецедент надає можливість адміністратору платформи з продажу нерухомості та агенції видаляти обліковий запис рієлтора.

Основний потік подій: виконується після натискання на «Видалити запис». Обліковий запис видаляється після підтвердження.

Передумова: перед виконанням прецеденту, адміністратор або агенція повинні бути авторизовані в системі.

Прецедент «Додавання оголошень».

Виконавець: адміністратор, рієлтор.

Призначення: цей прецедент надає можливість адміністратору платформи з продажу нерухомості або рієлтору додавати оголошення.

Основний потік подій: виконується після натискання на «Додати оголошення». Відкривається інтерфейс додавання оголошення.

Передумова: перед виконанням прецеденту, адміністратор або рієлтор повинні бути авторизовані в системі.

Прецедент «Редагування оголошень».

Виконавець: адміністратор, рієлтор.

Призначення: цей прецедент надає можливість адміністратору платформи з продажу нерухомості або рієлтору редагувати оголошення.

Основний потік подій: виконується після натискання на «Редагувати запис». Відкривається інтерфейс редагування оголошення.

Передумова: перед виконанням прецеденту, адміністратор або рієлтор повинні бути авторизовані в системі.

Прецедент «Видалення оголошень».

Виконавець: адміністратор, рієлтор.

Призначення: цей прецедент надає можливість адміністратору платформи з продажу нерухомості або рієлтору видаляти оголошення.

Основний потік подій: виконується після натискання на «Видалити запис». Оголошення видаляється після підтвердження.

Передумова: перед виконанням прецеденту, адміністратор або рієлтор повинні бути авторизовані в системі.

Прецедент «Перегляд оголошень».

Виконавець: адміністратор, агенція, рієлтор, клієнт.

Призначення: цей прецедент надає можливість адміністратору платформи з продажу нерухомості або рієлтору переглядати повну інформацію оголошення, всі інші користувачі системи мають можливість переглядати часткову інформацію стосовно оголошення.

Основний потік подій: виконується після натискання на потрібне оголошення. Відкривається інтерфейс перегляду оголошення.

Передумова: перед виконанням прецеденту, адміністратор або рієлтор повинні бути авторизовані в системі.

Прецедент «Пошук оголошень».

Виконавець: адміністратор, агенція, рієлтор, клієнт.

Призначення: цей прецедент надає можливість всім користувачам здійснювати пошук необхідних оголошень за певними критеріями.

Основний потік подій: виконується після вводу потрібних критеріїв. З’являються необхідні оголошення.

Передумова: перед виконанням прецеденту, адміністратор повинен бути авторизований в системі.

Прецедент «Пошук рієлторів».

Виконавець: адміністратор, агенція, рієлтор, клієнт.

Призначення: цей прецедент надає можливість всім користувачам здійснювати пошук необхідних рієлторів за певними критеріями.

Основний потік подій: виконується після вводу потрібних критеріїв. З’являються необхідні рієлтори.

Передумова: перед виконанням прецеденту, адміністратор повинен бути авторизований в системі.

Прецедент «Пошук агенцій».

Виконавець: адміністратор, агенція, рієлтор, клієнт.

Призначення: цей прецедент надає можливість всім користувачам здійснювати пошук необхідних агенцій за певними критеріями.

Основний потік подій: виконується після вводу потрібних критеріїв. З’являються необхідні агенції.

Передумова: перед виконанням прецеденту, адміністратор повинен бути авторизований в системі.

Прецедент «Фільтрація оголошень».

Виконавець: адміністратор, агенція, рієлтор, клієнт.

Призначення: цей прецедент надає можливість всім користувачам здійснювати фільтрацію всіх оголошень за такими критеріями як місто, кількість кімнат, поверх, поверховість та інші.

Основний потік подій: виконується після вводу потрібних критеріїв. З’являються необхідні оголошення.

Передумова: перед виконанням прецеденту, адміністратор повинен бути авторизований в системі.

Прецедент «Відправка повідомлень агенції».

Виконавець: рієлтор.

Призначення: цей прецедент надає можливість рієлтору відправляти повідомлення власній агенції.

Основний потік подій: виконується після натискання «Написати». Відкривається інтерфейс переписки з агенцією.

Передумова: перед виконанням прецеденту, рієлтор повинен бути авторизований в системі.

Прецедент «Відправка повідомлень рієлтору».

Виконавець: агенція.

Призначення: цей прецедент надає можливість агенції відправляти повідомлення власній агенції.

Основний потік подій: виконується після натискання «Написати». Відкривається інтерфейс переписки з рієлтором.

Передумова: перед виконанням прецеденту, агенція повинена бути авторизована в системі.

Прецедент «Оцінити рієлтора».

Виконавець: клієнт.

Призначення: цей прецедент надає можливість клієнтові залишити відгук рієлтору

Основний потік подій: виконується після натискання «Рекомендувати». Відкривається інтерфейс переписки з рієлтором.

Передумова: перед виконанням прецеденту, клієнт повинен бути авторизований в системі.

Прецедент «Створення облікового запису».

Виконавець: адміністратор, клієнт, агенція.

Призначення: даний прецедент дозволяє адміністратору, агенції та клієнту створювати облікові записи в системі. Клієнт має можливість створити тільки власний обліковий запис. Адміністратор може створювати необмежену кількість облікових записів всіх типів користувачів системи. Агенція має можливість створювати власний обліковий запис, та обліковий запис свої рієлторів.

Основний потік подій: виконується після натискання на кнопку «Зареєструватися» для клієнта та агенції, та «Додати рієлтора/агенцію/користувача/адміністратора» для адміністратора, а також «Додати рієлтора» для агенції. Система відображає форму для реєстрації облікового запису користувача.

Альтернативний потік подій: користувач ввів невірні дані, система повідомить користувача про це, та дозволить ввести дані повторно.

Передумова: перед виконанням прецеденту, адміністратор повинен бути авторизований в системі, а також агенція повинна бути авторизована в системі перед додаванням рієлторів.

Виняткова ситуація 1: користувач ввів невірні дані – система надає можливість ввести дані повторно.

Виняткова ситуація 2: користувач з такими даними вже зареєстрований в системі – система повідомляє про це користувача та надає можливість ввести дані повторно.

Прецедент «Редагування облікового запису».

Виконавець: адміністратор, клієнт, рієлтор, агенція.

Призначення: даний прецедент дозволяє всім користувачам системи редагувати власний обліковий запис. Але адміністратор має можливість

редагувати будь-який обліковий запис в системі. Агенція, окрім власного облікового запису, може редагувати облікові записи своїх рієлторів.

Основний потік подій: виконується після натискання на кнопку «Редагувати запис». Система відображає форму для редагування облікового запису користувача.

Альтернативний потік подій: користувач ввів невірні дані, система повідомить користувача про це, та дозволить ввести дані повторно.

Передумова: перед виконанням прецеденту, всі групи користувачів системи повинні бути авторизовані в системі.

Виняткова ситуація 1: користувач ввів невірні дані – система надає можливість ввести дані повторно.

Прецедент «Видалення облікового запису».

Виконавець: адміністратор, клієнт, агенція.

Призначення: даний прецедент дозволяє адміністратору, агенції та клієнту системи видаляти облікові записи. Клієнт має можливість видалити тільки власний обліковий запис. Адміністратор може видалити будь-який обліковий запис користувача. Агенція може видалити власний обліковий запис, а також облікові записи своїх рієлторів.

Основний потік подій: виконується після натискання на кнопку «Видалити запис». Система просить підтвердження на видалення облікового запису користувача.

Передумова: перед виконанням прецеденту, адміністратор, агенція та клієнт повинні бути авторизовані в системі.

Прецедент «Авторизація в системі».

Виконавець: адміністратор, клієнт, рієлтор, агенція.

Призначення: даний прецедент дозволяє адміністратору, агенції, рієлтору та клієнту авторизуватись в системі.

Основний потік подій: виконується після натискання на кнопку «Увійти». Система просить ввести дані користувача для авторизації.

Передумова: перед виконанням прецеденту, адміністратор, рієлтор, агенція та клієнт повинні зайти на будь-яку сторінку системи.

Прецедент «Вихід з системи».

Виконавець: адміністратор, рієлтор, агенція, клієнт.

Призначення: даний прецедент дозволяє адміністратору, рієлтору, агенції та клієнту вийти з облікового запису в системі.

Основний потік подій: виконується після натискання на кнопку «Вийти». Система просить підтвердити дію.

Передумова: перед виконанням прецеденту, адміністратор, рієлтор, агенція та клієнт повинні бути авторизовані у системі.

2.3 Діаграма діяльності системи

В Уніфікованій мові моделювання (UML) діаграма діяльності – це графічне зображення виконаного набору дій процедурної системи та розглядається варіація діаграми стану стану. Діаграми діяльності описують паралельні та умовні дії, використовують випадки та системні функції на детальному рівні, моделють послідовність робочого потоку великої діяльності, орієнтуючись на послідовності дій та відповідні умови ініціювання дії [6].

Нижче представлені діаграми діяльності для основного потоку діяльності таких користувачів системи як агенція, рієлтор та клієнт (рис. 2.6 – 2.8).

Діаграма діяльності агентсва (див. рис. 2.6) відображає основний потік подій для агенції. Вперше потрапивши на сайт, представнику агенції потрібно її зареєструвати, для цього йому потрібно натиснути на конку авторизації в хедері, далі система відображає сторінку авторизації, та пропонує увійти, або зареєструватися в системі. Потрапивши на сторінку реєстрації, та ввівши всі необхідні дані, система перевіряє їх на валідність (чи існує така агенція в

системі, чи відповідає пароль вимогам) та в разі успіху додає агенцію до системи, або повідомляє про помилку. Далі представник агенції потрапляє в особистий кабінет, де натиснувши «Додати рієлтора» система відображає інтерфейс реєстрації облікових акаунтів рієлторів. Після вводу необхідних даних, система перевіряє, дані на валідність (email, пароль, інші поля, а також чи існує вже такий рієлтор в системі), якщо валідація була пройдена, створюється обліковий запис рієлтора в системі. Далі представник агенції повинен передати облікові дані для входу своєму рієлторові.

Діаграма діяльності рієлтора (див. рис. 2.7) відображає основний потік діяльності рієлтора. Отримавши дані облікового запису від свого представника агенції, рієлтор переходить на будь-яку сторінку сайту, натискає на іконку авторизації в хедері, система відображає йому сторінку авторизації. Рієлтор вводить дані для авторизації, далі система перевіряє дані на валідність (чи існує такий рієлтор, та чи вірний пароль), якщо валідація пройдена, відображається особистий кабінет рієлтора. Рієлтор натискає «Додати оголошення», система відображає інтерфейс додавання оголошення. Рієлтор вводить дані оголошення, та відправляє форму, система перевіряє дані на валідність, та в разі проходження валідації додає оголошення до системи.

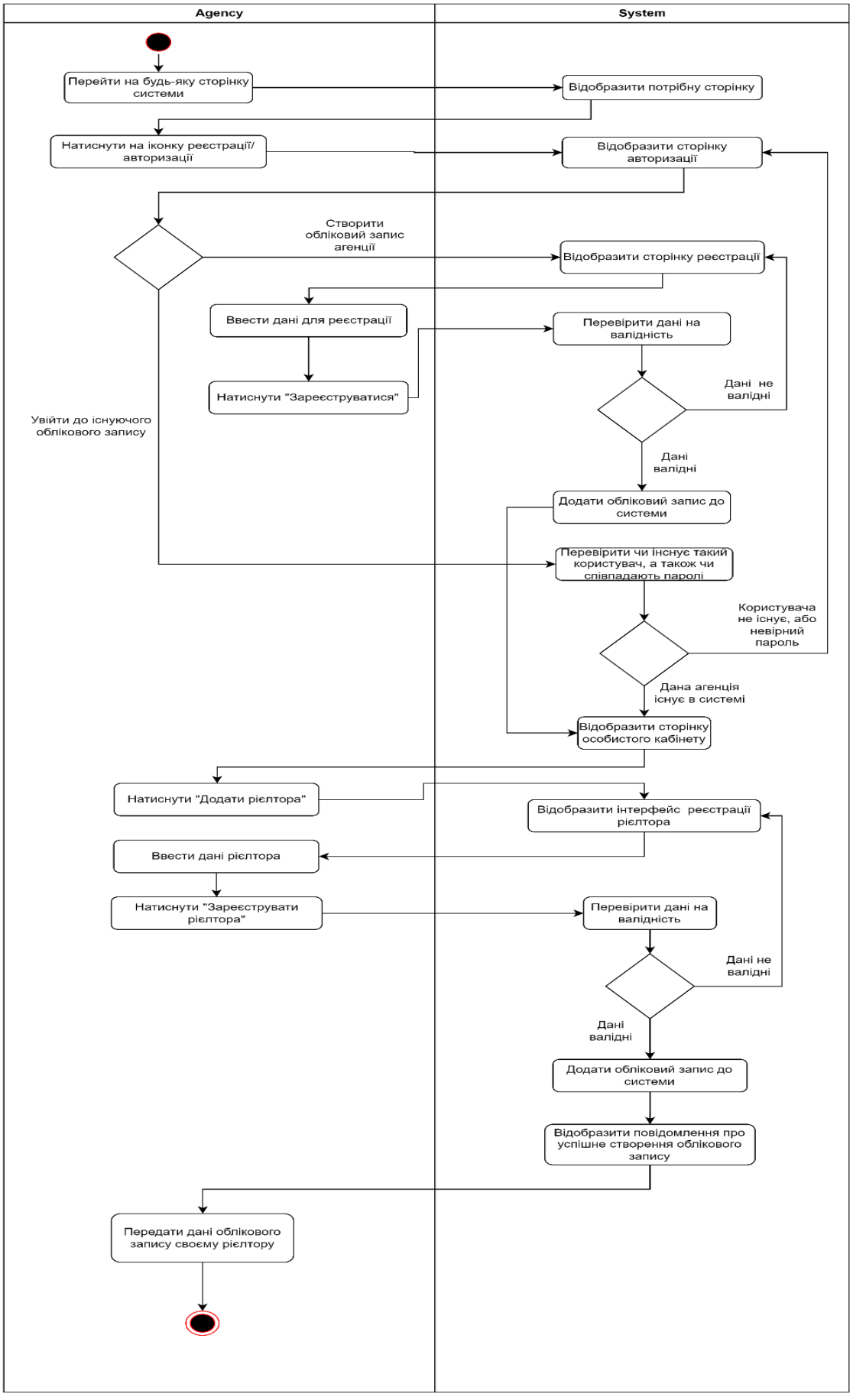

Рисунок 2.6 – Діаграма діяльності агенства

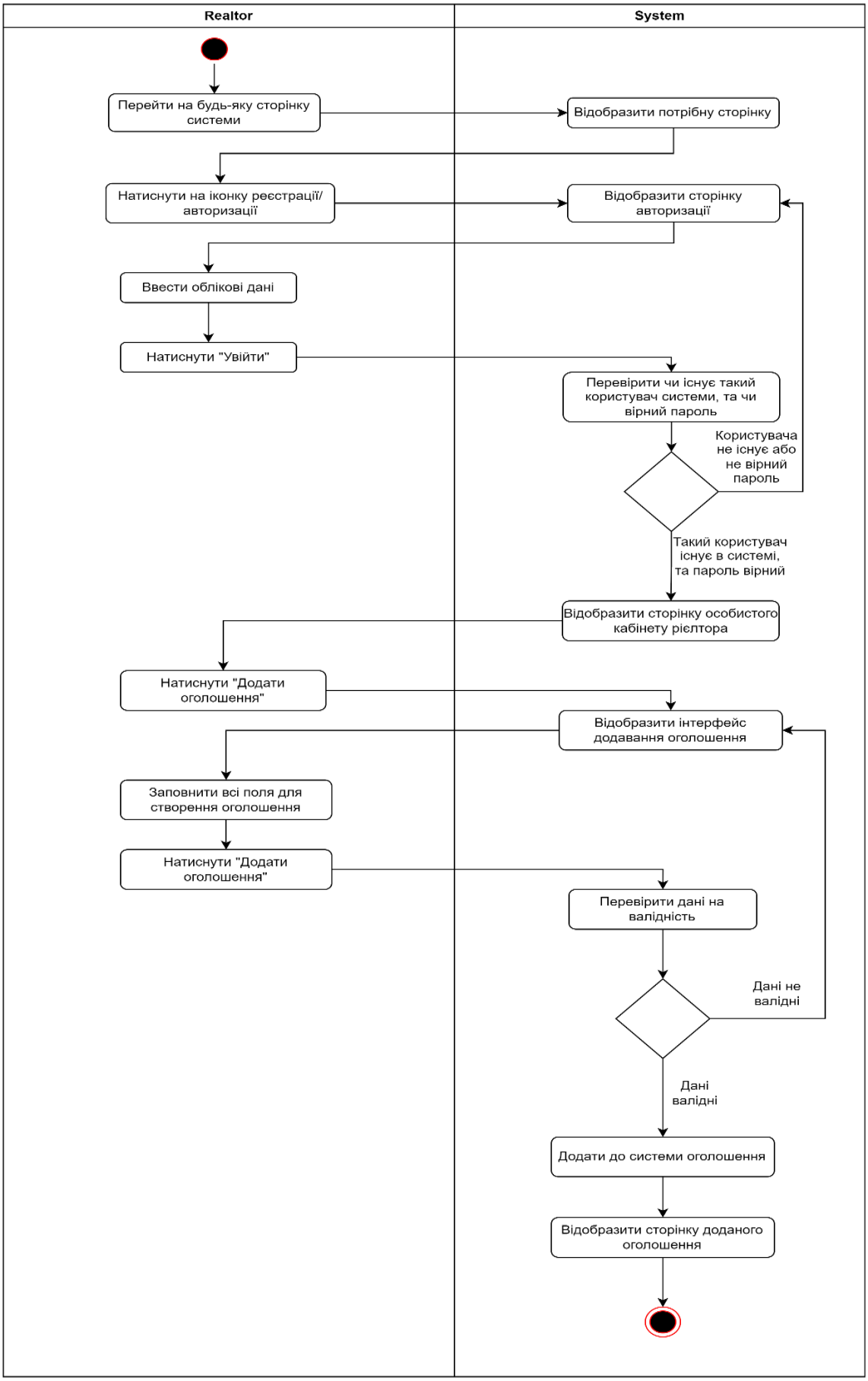

Рисунок 2.7 – Діаграма діяльності рієлтора

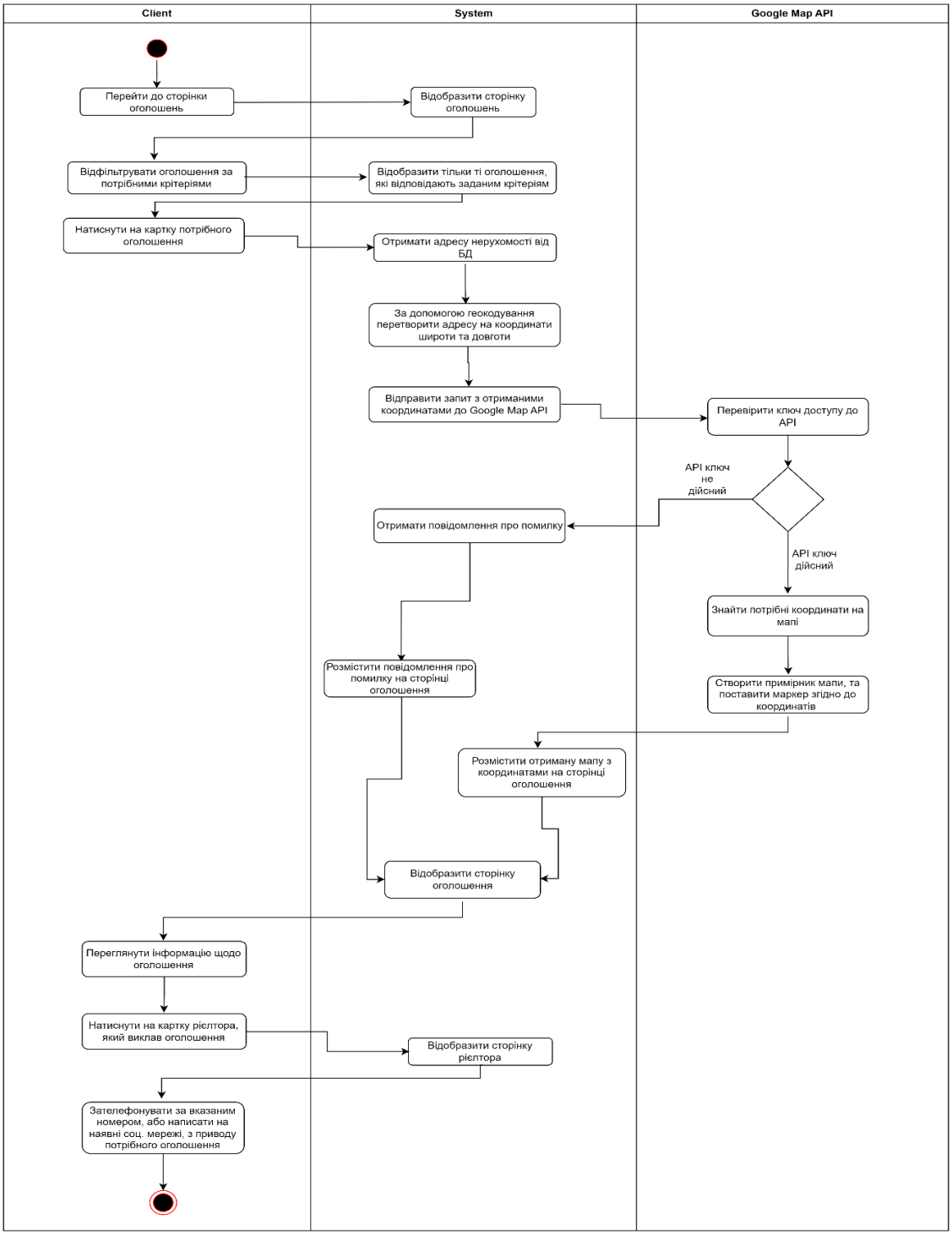

Рисунок 2.8 – Діаграма діяльності клієнта

Діаграма діяльності клієнта (див. рис. 2.8) відображає основний потік діяльності клієнта. Потрапивши на сторінку оголошень вперше, клієнт обирає потрібне місто та вводить додаткові фільтри. Система відображає відповідні оголошення клієнту. Наступним кроком клієнт натискає на потрібну пропозицію. В свою чергу система перед відображенням сторінки з повною

інформацією оголошення, отримує адресу нерухомості та за допомогою геокодування перетворює її на координати широти та довготи, після чого виконує запит, з отриманими координатами, до Google Maps API. Перш за все перевіряється ключ доступу до API, якщо ключ валідний, створюється примірник мапи з маркером за вказаними координатами, після чого примірник передається системі, яка додає його до сторінки оголошення. Переглянувши повну інформацію оголошення, клієнт натискає на рієлтора, який виклав дане оголошення, та потрапляє на його сторінку, де в він має змогу зв’язатися з ним подзвонивши на вказаний номер, або написавши на наявні соціальні мережі.

2.4 Використання ER-діаграми для проектування бази даних систем

Схема «сутність-зв'язок» (також ERD або ER-діаграма) – це різновид блок-схеми, де показано, як різні «сутності» (люди, об'єкти, концепції і так далі) пов'язані між собою всередині системи. ER-діаграми найчастіше застосовуються для проектування і налагодження реляційних баз даних в сфері освіти, дослідження та розробки програмного забезпечення та інформаційних систем для бізнесу. ER-діаграми (або ER-моделі) покладаються на стандартний набір символів, включаючи прямокутники, ромби, овали і сполучні лінії, для відображення сутностей, їх атрибутів і зв'язків [7].

Розбір Er-діаграми розташованої нижче (рис. 2.9) розпочнемо з сутності «користувач», вона містить дані клієнта, та адміністратора, а також зв’язок «один до одного» до сутності «токени авторизації користувачів», яка в свою чергу містить токен та дату створення токена для кожного адміністратора чи клієнта в системі. Початковою сутністю є «місто», яка містить назву міста та має зв’язки «один до багатьох» до таких сутностей як «агенція», «рієлтор», «оголошення», адже в одному місті може бути велика кількість агенцій,

рієлторів, та оголошень. Наступною сутністю є «агенція», яка містить інформацію агенції та має зв’язок «один до багатьох» до сутності «рієлтор», адже одна агенція може мати безліч рієлторів. Сутність «рієлтор» містить інформацію про рієлтора, а також має зв’язок «один до багатьох» до сутності «нерухомість», адже один рієлтор може мати безліч оголошень. Сутність «нерухомість» містить інформацію про нерухомість, а також зв’язок «один до багатьох» до сутності «фотографії», адже рієлтор може додати велику кількість фотографій однієї пропозиції, та зв’язок «один до одного» до сутності «стіни», адже у нерухомості повинен бути один ти стін.

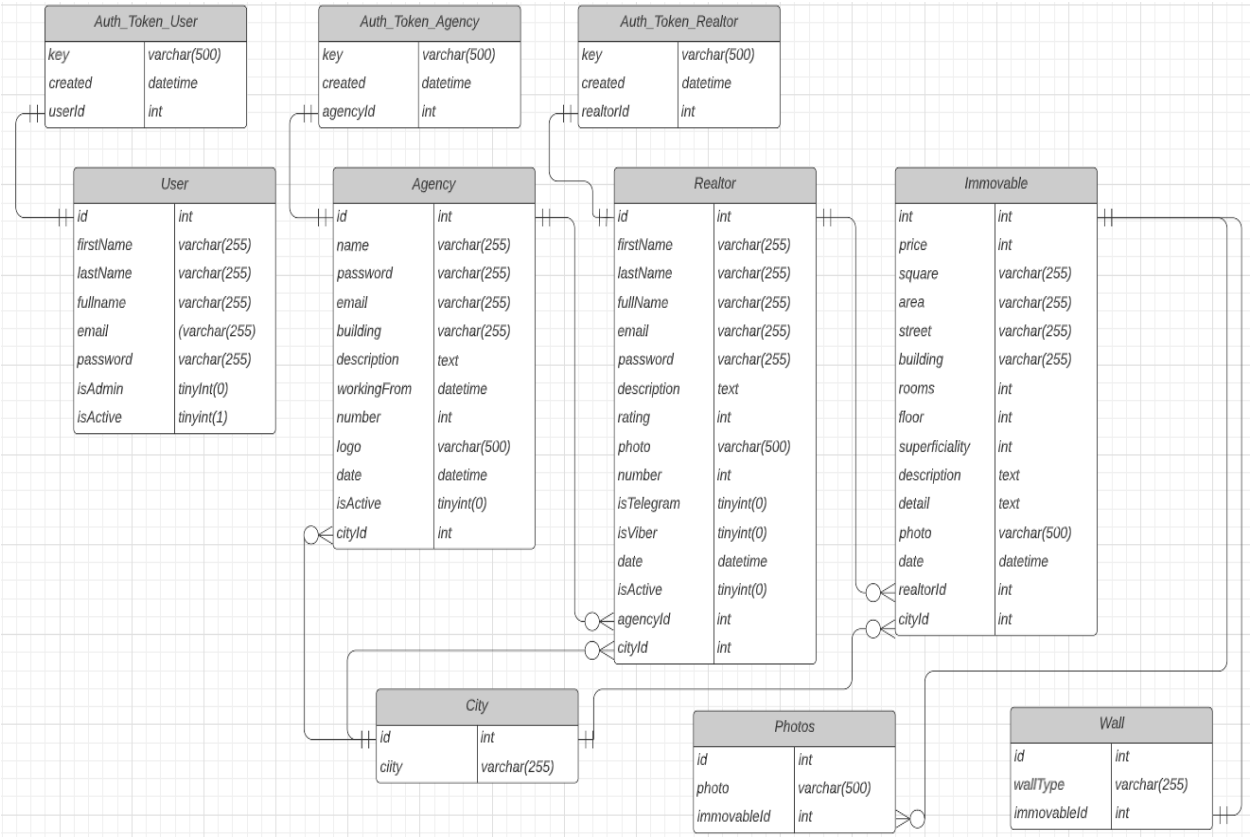

Рисунок 2.9 – Er-діаграма системи

3 РЕАЛІЗАЦІЯ ПЛАТФОРМИ З ПРОДАЖУ НЕРУХОМОТСІ

3.1 Створення та налагодження серверної та клієнтської частини за допомогою Node.js та React

3.1.1 Створення та налагодження проєкту Node.js

Спочатку створюємо теку «back» після чого за допомогою консолі переходимо до цієї теки та виконує команду «npm init», яка створює файл «package.json» з налаштуваннями, залежностями, скриптами та іншими параметрами. Далі інсталюємо всі необхідні пакети, а саме ORM бібліотеку Sequelie та її інтерфейс командного рядка Sequelize CLI для взаємодії з базою даних, MySQL2 для створення з'єднання з базою даних, Dotenv для використання змінних середовища, фреймворк Express для реалізації логіки серверу, проміжне пз Cors для отримання доступу до ендпоінтів з іншого домену, Jsonwebtoken для реалізації авторизації в проєкті, Bcrypt для хешування паролів користувачів, Multer для збереження файлів в теки проєкту, та Jest для тестування застосунку. Всі потрібні пакети встановлюємо за допомогою команди «npm install --save 'назва пакету'».

```
"dependencies": {
  "bcrypt": "^5.0.1",
  "cors": "^2.8.5",
  "dotenv": "^16.0.2",
  "express": "^4.18.1",
  "jest": "^29.2.2",
  "jsonwebtoken": "^8.5.1",
  "multer": "^1.4.5-lts.1",
  "mysql2": "^2.3.3",
  "sequelize": "^6.21.4",
  "sequelize-cli": "^6.4.1"
},
```

Рис.3.1 – Залежності проєкту

Після встановлення пакетів, створюємо теки sequelize та server, спочатку переходимо до теки sequelize та виконуємо команду «npx sequelize-cli init», яка ініціює sequelize та створює наступні теки:

– config/config.json – в цьому файлі створюється конфігурації для баз даних (див. рис. 3.2);
– migrations – ця тека містить в собі всі файли міграцій для бази даних;
– models – в папці models зберігаються файли моделей проекту;
– seeders – ця тека містить в собі файли сидів бази даних.

Тепер потрібно вказати налаштування потрібних баз даних в файлі config.json та створити за допомогою команд sequelize cli потрібні моделі, та міграції бази даних.

Рисунок 3.2 – Налаштування підключення до баз даних проєкту

Закінчивши з налаштуванням, та створенням моделей, а також міграцій для бази даних, переходимо до теки server та починаємо створювати структуру проєкту:

- server.js – файл серверу, в якому підключені налаштування cors, static для роздачі статичних файлів, а також router для розгалуження логіки проєкту;
- .env – файл змінних середовища;
- routers – ця тека містить всі наявні маршрути для обробки ендпоінтів;
- public – в цій теці розташовані інші теки з статичними файлами проєкту;
- multerConfig – ця тека містить в собі налаштування для пакету multer, а саме вказання з яким найменуванням та до яких тек завантажувати ті, чи інші файли;
- middlewares – всередині цієї теки знаходяться файли проміжного пз, такі як валідатори електроної пошти, паролю, а також функції авторизації, перевірки прав доступу до ендоінтів, та обробка помилок;
- controllers – тека з класами-контролерами моделей бази даних;
- __test__ – тека містить в собі файли з юніт тестами контролерів, та проміжного пз.

3.1.2 Створення та налагодження проєкту React

Для розгортання проєкту за допомогою React спочатку створюємо теку «front», після переходим до цієї теки завдяки консолі, та виконуємо команду «npx create-react-app front-end». Дана команда розгорне проєкт з теками за замовчанням. Після цього потрібно встановити такі пакети як React Router Dom для реалізації навігації між сторінок проєкту, Redux Toolkit та Redux для

створення сховища даних проєкту, та інші менш важливі пакети. Після рефакторингу теки проєкту, структура буде мати наступний вигляд:

- public – тека, яка містить index.html та інші файли, такі як favicon.ico, manifest.json та інші;
- src – в цьому каталозі знаходяться вихідні файли проекту;
- src/components – каталог містить компоненти проєкту;
- src/hooks – ця тека містить користувацькі хуки проєкту;
- src/pages – в цій теці зберігаються забрані з компонентів сторінки платформи з продажу нерухомості;
- src/services – в цій теці знаходяться сервіси для взаємодії з даними проєкту;
- src/store – тека, яка містить в собі сховище даних, та функції взаємодії з ним;
- src/utils – тека з допоміжними функціями (валідатори, перетворення дати та інш.), а також з константами проєкту;
- src/App.js – головний компонент проєкту;
- src/index.js – головний файл проєкту;
- src/routes.js – файл з маршрутами сторінок проєкту.

Для виконання умовного відтворення сторінок, було створено файл routes з списками сторінок, які повинні відображатись тому, чи іншому користувачу. Цей список містить в собі об’єкти з властивостями path та component. Властивість path вказує за яким саме URL відображати компонент, в той чай як component вказує на потрібну для відображення сторінку. Ці списки імпортуються до компоненту маршрутизації. В компоненті маршрутизації ми також отримуємо стан користувача з сховища, та відтворюємо ті маршрути, до яких він має доступ (рис. 3.3).

```
export default function AppRouter () {
    const {user} = useSelector(state=>state.login)
    const {path, component} = notFound
    return(
        <>
        <ScrollToTop/>
        <Routes>
            {user.isAdmin && authRoutes.map(({path, component})=>{
                return <Route key={path} path={path} element={component}/>
            })}
            {publickRoutes.map(({path, component})=>{
                return <Route key={path} path={path} element={component}/>
            })}
            {user.isRealtor && realtorOnlyPages.map(({path, component})=>{
                return <Route key={path} path={path} element={component} />
            })}
            {user.isAgency && agencyOnlyPages.map(({path, component})=>{
                return <Route key={path} path={path} element={component} />
            })}
            {user.isClient && clientOnlyPages.map(({path, component})=>{
                return <Route key={path} path={path} element={component} />
            })}
            <Route path={path} element={component}/>
        </Routes>
        </>
    )
}
```

Рисунок 3.3 – Маршрутизація проєкту

```
function App() {

  const dispatch = useDispatch()

  useEffect(()=>{
    if(localStorage.getItem('token')){
      dispatch(CheckTokenAction())
    }
  },[])

  return (
    <BrowserRouter>
      <AppRouter/>
    </BrowserRouter>
  );
}

export default App;
```

Рисунок 3.4 – Підключення компоненту з маршрутами до головного компоненту

Для взаємодії з сховищем даних, потрібно імпортувати компонент Provider з пакета React Redux, а також створений store з усіма ред'юсерами проєкту, після чого відбувається підключення сховища як на наступному рисунку (рис. 3.5.).

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';
import {Provider} from "react-redux"
import {store} from './store/index'

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(
    <Provider store={store}>
      <App />
    </Provider>
);
```

Рисунок 3.5 – Підключення сховища до проєкту

Виконавши ці дії, завершуються основні налаштування проєкту, далі буде розглянуто реалізацію основної логіки платформи з продажу нерухомості.

3.2 Реалізація основних функціональних можливостей системи

3.2.1 Реалізація основних функціональних можливостей адміністратора

Для реалізації функціональних можливостей було обрано спосіб написання API за допомогою Node.js та Express, а також створення інтерфейсу взаємодії за допомогою React.

Тож розпочнемо огляд з реалізації адміністративної панелі сайту. Перед виконанням будь-якого запиту до кінцевої точки з адміністративної панелі, спочатку виконується перевірка, який саме користувач зробив запит, якщо це був адміністратор, запит виконується, в іншому разі відображається помилка авторизації. Наступні рисунки відображають проміжне пз для перевірки ролі користувача, а також його підключення перед виконанням кінцевих точок (рис. 3.6 – рис. 3.9).

```
module.exports = (req, res, next) =>{
    if (req.method === 'OPTIONS'){
        next()
    }
    try {
        const token = req.headers.authorization.split(' ')[1]
        if(!token){
            return res.status(401).send({result:"Не авторізований"})
        }
        const decoded = jwt.verify(token, process.env.SECRET_KEY)
        req.user = decoded
        next()
    } catch (e) {
        res.status(401).send({result:"Не авторізований"})
    }
}
```

Рисунок 3.6 – Перевірка токену на валідність

```
module.exports = (req, res, next) =>{
    req.user.isAdmin ? next() :
        res.status(500).send({result:'У вас немає прав доступу'})
}
```

Рисунок 3.7 – Перевірка, чи є користувач адміністратором

```
router.use(CheckAuth)

router.use('/', AdminRouter)
router.use('/agency', AgencyRouter)
router.use('/realtor', RealtorRouter)
router.use('/offer', OfferRouter)
router.use('/tokens', TokenRouter)
router.use('/city', CityRouter)
router.use('/wall', WallRouter)


module.exports = router
```

Рисунок 3.8 – Підключення проміжного пз для перевірки авторизації

```
const checkIsAdmin = require('../../../middlewares/authMiddleware/checkIsAdminMiddleware')
const OfferController = require('../../../Controllers/OfferControllers/OfferControllers')

router.use(checkIsAdmin)

router.get('/list/:page', jsonParser, OfferController.showForAdminOfferList)
router.get('/find', jsonParser, OfferController.findForAdminOffer)
router.get('/realtors', jsonParser, OfferController.getRealtorList)
router.get('/:id', jsonParser, OfferController.showForAdminOffer)
router.post('/create', offerUpload.single('photo'), OfferController.createOffer)
router.post('/upload-photos/:id', offerDetailUpload.array('photos'), OfferController.uploadOfferPhotos)
router.put('/update/:id', offerUpload.single('photo'), OfferController.updateOffer)
router.delete('/delete/:id', jsonParser, OfferController.deleteOffer)



module.exports = router
```

Рисунок 3.9 – Підключення проміжного пз для перевірки ролі користувача

Після авторизації адміністратор потрапляє до головної сторінки адміністративної панелі, де за допомогою навігації обирає ті дані, для перегляду та редагування, які йому потрібні, наступний рисунок показує приклад інтерфейсу перегляду всіх оголошень в системі (рис. 3.10).

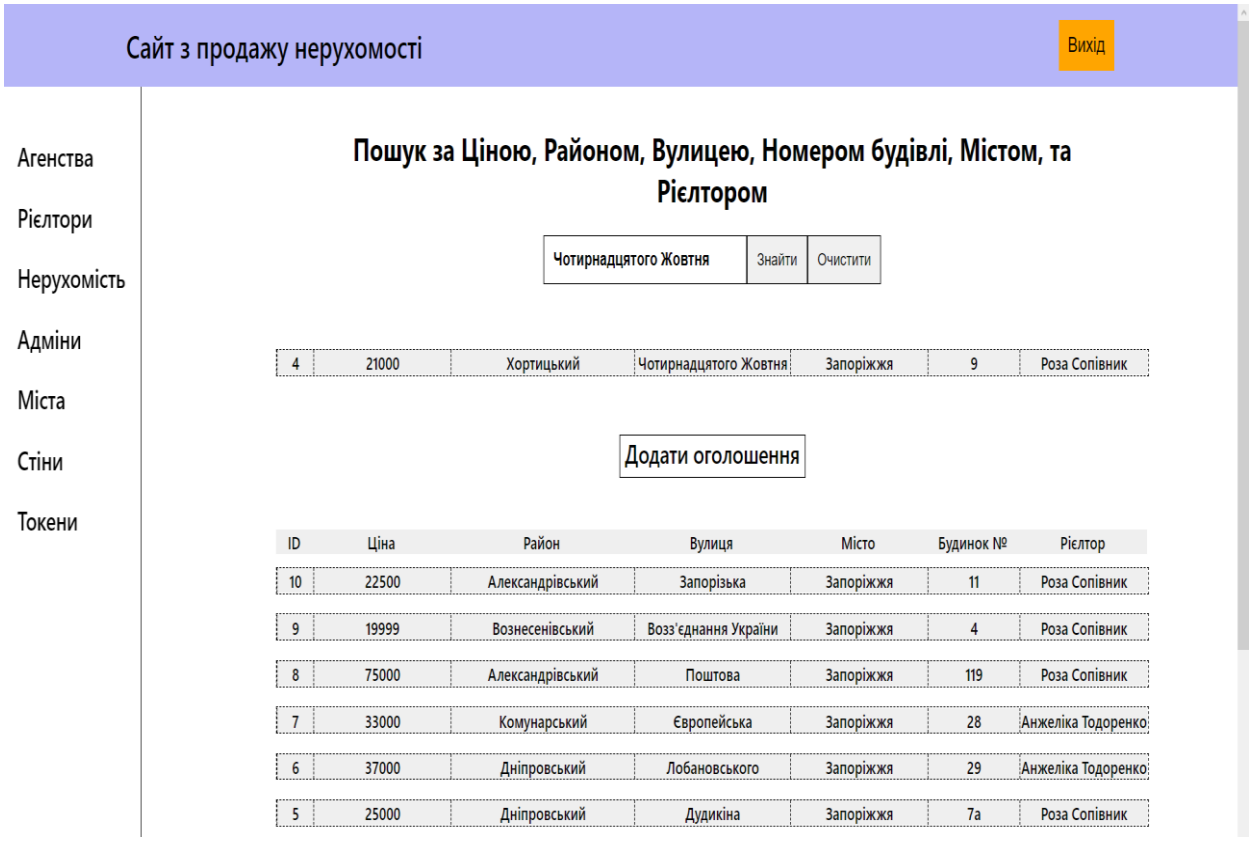

Рисунок 3.10 – Інтерфейс перегляду списку даних

Обравши потрібний запис, адміністратор потрапляє до сторінки перегляду повної інформації цього запису, за цієї ж сторінки надаються можливості видалення, та редагування даних. Нижче представлений рисунок з інтерфейсом перегляду та редагуванням даних оголошень та облікових записів користувачів системи (рис. 3.11).

Рисунок 3.11 – Інтерфейс перегляду та редагування оголошення, або облікових записів

3.2.2 Реалізація основних функціональних можливостей рієлтора

Основними задачами рієлтора є викладення оголошень до системи, які в свою чергу будуть відображатися клієнтові. Після того як рієлтор отримав, від представника агенції, свої облікові та виконав авторизації в системі, він потрапляє до особистого кабінету (рис. 3.12).

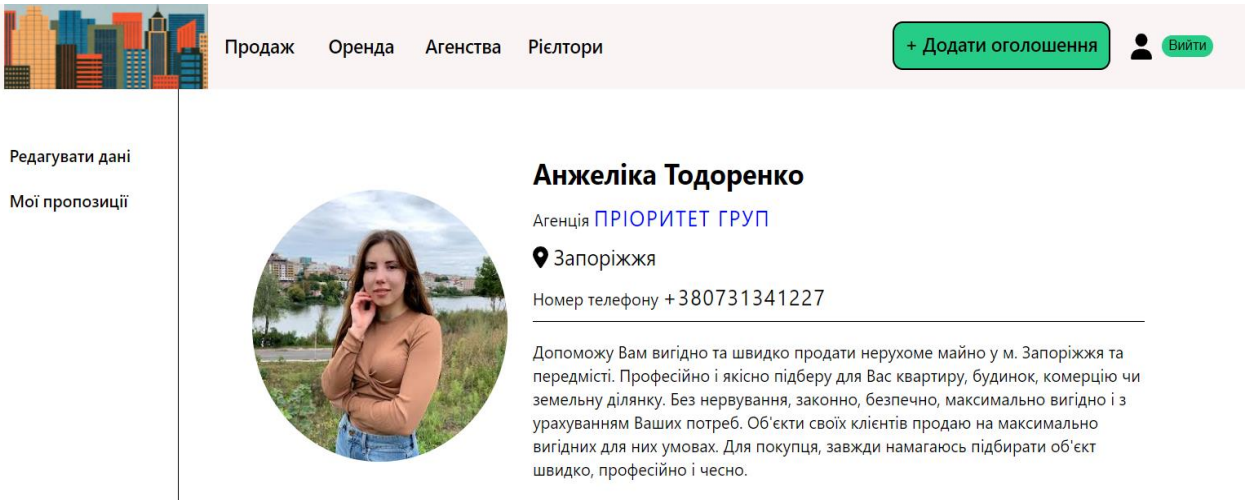

Рисунок 3.12 – Інтерфейс особистого кабінету рієлтора

Наступним кроком рієлтор натискає на кнопку «Додати оголошення», та потрапляє на сторінку додавання оголошень, після заповнення всіх необхідних полів та валідації форми, в системі створюється запис оголошення. Наступні рисунки відображають код та інтерфейс сторінки додавання оголошення (рис. 3.13 – рис. 3.15).

```
async createOffer(req, res) {
    const photo = req.file.filename
    const unlincPath = path.resolve(__dirname, '..', '..', 'public', 'OfferPhotos', photo)
    try {
        const immovable = await Immovables.create({...req.body, photo: photo})
        res.send(JSON.stringify({result:true, id:immovable.id}))
    } catch (e) {
        unlinkSync(unlincPath)
        res.status(500).send({error:"При створенні сталася помилка"})
    }
}
```

Рисунок 3.13 – Функція створення оголошення

```
const request = await fetch(`${HOST}/offer/create`, {
    headers : {
        "Authorization" : `Bearer ${localStorage.getItem("token")}`
    },
    method: 'POST',
    body: formData
});
const response = await request.json()
if(response.result) {
    alert('Запис збереженно')
    navigate(`/realtor-cab/offer/${response.result.id}`)
}
```

Рисунок 3.14 – Функція звернення до кінцевої точки API

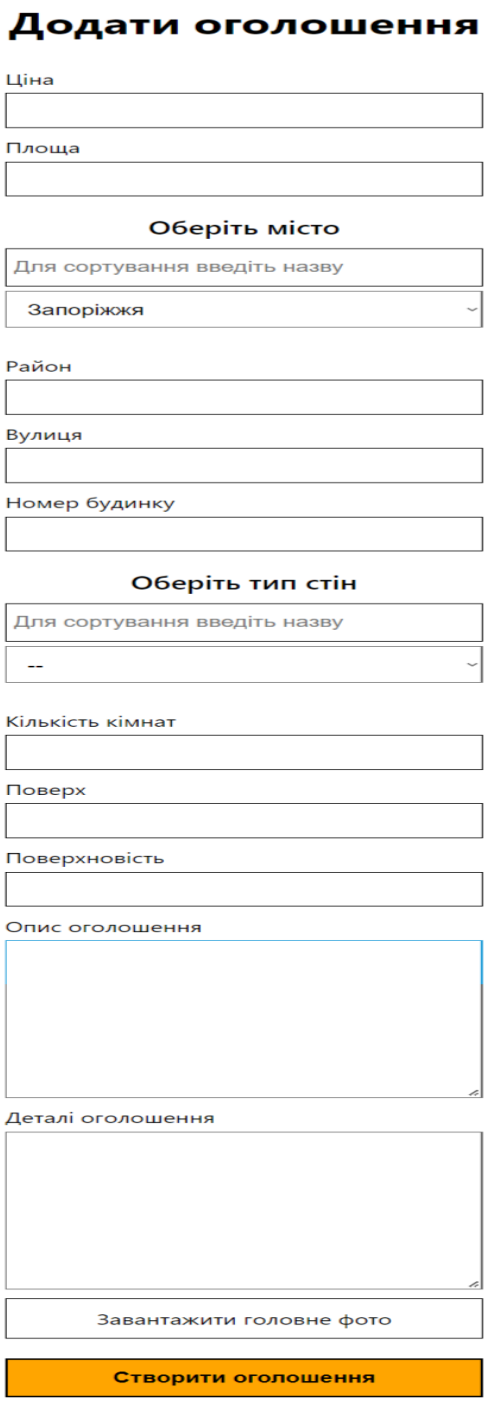

Рисунок 3.15 – Інтерфейс додавання оголошення до системи

3.2.3 Реалізація основних функціональних можливостей клієнта

У цьому розділі розглянуто такі можливості клієнта, як перегляд списку оголошень, а також перегляд сторінки конкретного оголошення. Розпочнемо з огляду можливості відображення списку оголошень. Для відображення сторінки на стороні клієнта, було створено функцію offerSlice, яка приймає такі параметри як:

– name – ім’я функції;
– initial state – базовий стан об’єкту оголошень;
– reducers – об’єкт з функціями взаємодії з станом оголошень;
– extra reducers – об’єкт з варіантами взаємодії зі станом, без створення функцій.

Далі потрібно буде заповнити об’єкт extraReducers (рис. 3.18) трьома варіантами взаємодії з відповідним типом для кожної функції отримання даних, а саме (fulfield, pending, rejected). Наступним кроком ми підключаємо offerSlice до сховища проєкту, а також створюємо функцію отримання даних від серверної частини проєкту. Зміна fetchAgencyList використовує функцію createAsyncThunk (рис. 3.16) для звернення до кінцевої точки API та, відповідно до результату звернення, повертає один зі станів (fulfield, pending, rejected), які обробляються в об’єкті extraReducers. Кінцева точка повертає дані, згідно до параметрів запиту (див. додаток А).

```
export const fetchOfferList = createAsyncThunk(
    "fetchOfferList/getOffers",
    async(data, thunkAPI) => {
        try {
            const offers = await offerServices.getOfferList(data)
            if(offers.error){
                return thunkAPI.rejectWithValue(offers.error)
            }
            return offers.result
        } catch (e) {
            return thunkAPI.rejectWithValue(error)
        }
    }
)
```

Рисунок 3.16 – Функція отримання списку оголошень

```
getOfferList : async (data) => {
    let searchParams = new URLSearchParams(data).toString()
    searchParams = searchParams.split('&').slice(1,searchParams.length).join('&')
    const request = await fetch (backLink + '/list/' + data.page+'?'+searchParams)
    const response = await request.json()
    return response
},
```

Рисунок 3.17 – Функція-сервіс для отримання оголошень

```
extraReducers:{
    [fetchOfferList.fulfilled.type] : (state = initialState, action) => {
        state.error = ''
        state.isLoading = false
        state.offerList = action.payload.offers
        state.pages = action.payload.pages
    },

    [fetchOfferList.pending.type] : (state = initialState, action) => {
        state.isLoading = true
        state.error = ''
    },

    [fetchOfferList.rejected.type] : (state = initialState, action) => {
        state.isLoading = false
        state.offerList = ''
        state.error = action.payload
    },
```

Рисунок 3.18 – Обробка стану отримання оголошень

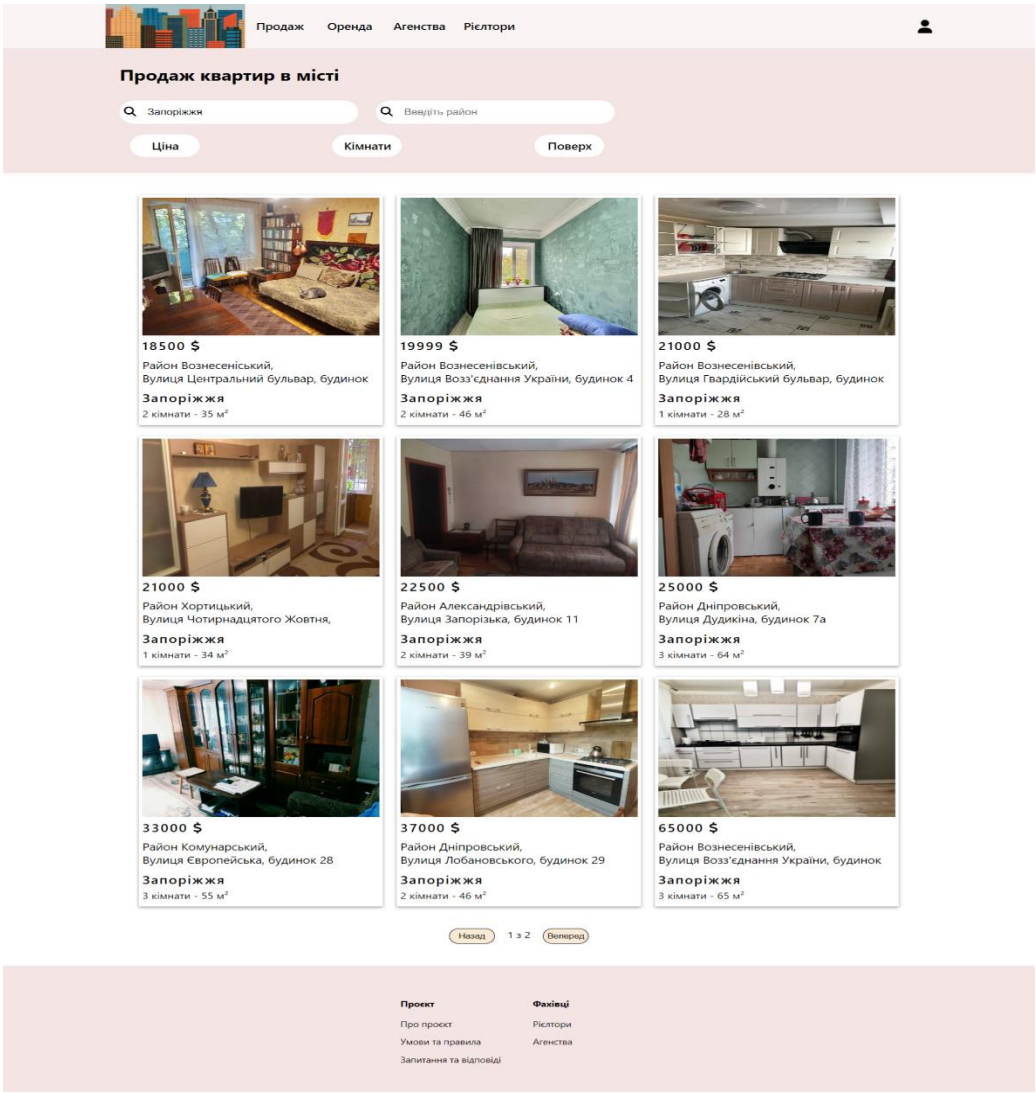

Рисунок 3.19 – Сторінка відображення оголошень

Для відображення сторінки окремого оголошення, потрібно виконати всі ті самі кроки, що виконувалися для сторінки з списком оголошень, для обраного міста (рис. 3.20 – рис. 3.23). Кінцева точка, до якої відбувається звернення, представлена в додатку А.

```
export const fetchOffer = createAsyncThunk(
    "fetchOffer/getOffer",
    async(id, thunkAPI) => {
        try {
            const offer = await offerServices.getOffer(id)
            if( offer.error){
                return thunkAPI.rejectWithValue(offer.error)
            }
            return offer.result
        } catch (e) {
            return thunkAPI.rejectWithValue(error)
        }
    }
)
```

Рисунок 3.20 – Функція отримання оголошення

```
getOffer : async (id) => {
    const request = await fetch (backLink +'/'+ id)
    const response = await request.json()
    return response
},
```

Рисунок 3.21 – Функція-сервіс для отримання оголошення

```
[fetchOffer.fulfilled.type] : (state = initialState, action) => {
    state.error = ''
    state.isLoading = false
    state.offer = action.payload
},

[fetchOffer.pending.type] : (state = initialState, action) => {
    state.isLoading = true
    state.error = ''
},

[fetchOffer.rejected.type] : (state = initialState, action) => {
    state.isLoading = false
    state.error = action.payload
    state.offer = ''
}
```

Рисунок 3.22 – Обробка стану отримання оголошення

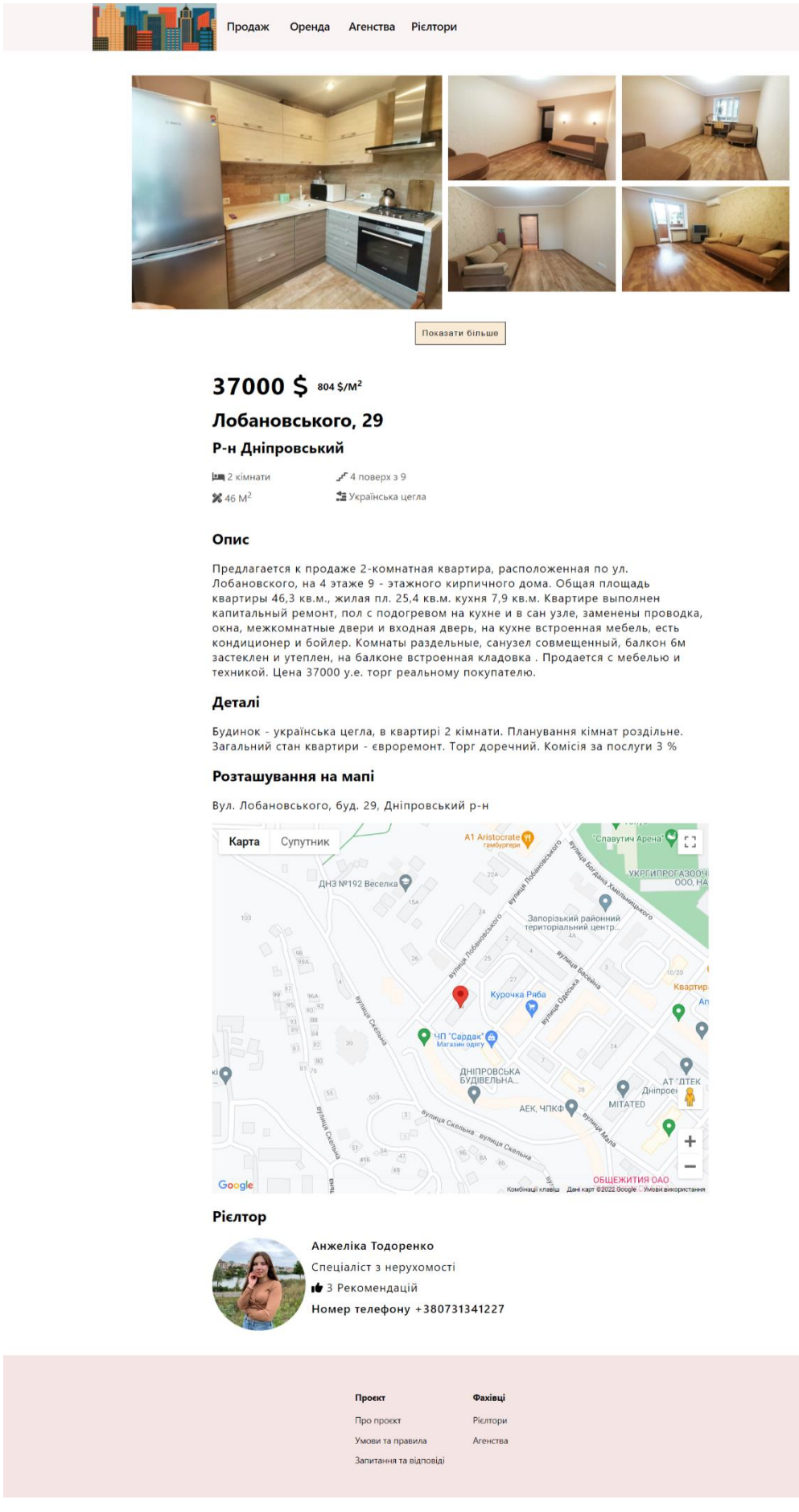

Рисунок 3.23 – Сторінка перегляду оголошення

3.3 Тестування розробленої системи

Тестування програмного забезпечення – це процес технічного дослідження, призначений для виявлення інформації про якість продукту відносно контексту, в якому його мають використовувати. Техніка тестування також включає як процес пошуку помилок або інших дефектів, так і випробування програмних складових із метою оцінки [8].

Jest – це фреймворк для тестування JavaScipt, розроблений для забезпечення правильності будь-якого JavaScript коду. Він дозволяє писати тести з використанням доступного, знайомого і багатофункціонального API, що дає швидкий результат [9].

Jest добре документований, потребує небагато налаштувань і може бути розширений для відповідності вашим вимогам [9].

Для тестування кінцевих точок серверної частини, було обрано саме фреймворк Jest, через його розповсюдженість та легкість у використанні.

Нижче представленні результати автоматичного та ручного тестування системи (рис. 3.24 – рис. 3.27).

3.3.1 Результат тестування класів контролерів системи

```
-------------------------------------|---------|----------|---------|---------|-------------------
File                                 | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
-------------------------------------|---------|----------|---------|---------|-------------------
All files                            |   98.19 |    75.64 |     100 |   98.19 |
 sequelize/models                    |   98.48 |    66.66 |     100 |   98.48 |
  admin.js                           |     100 |      100 |     100 |     100 |
  agency.js                          |     100 |      100 |     100 |     100 |
  city.js                            |     100 |      100 |     100 |     100 |
  immovables.js                      |     100 |      100 |     100 |     100 |
  index.js                           |      95 |    66.66 |     100 |      95 | 13
  photos.js                          |     100 |      100 |     100 |     100 |
  realtor.js                         |     100 |      100 |     100 |     100 |
  token_agency.js                    |     100 |      100 |     100 |     100 |
  token_realtor.js                   |     100 |      100 |     100 |     100 |
  token_user.js                      |     100 |      100 |     100 |     100 |
  wall.js                            |     100 |      100 |     100 |     100 |
 server/Controllers/AgencyControllers|     100 |       75 |     100 |     100 |
  AgencyController.js                |     100 |       75 |     100 |     100 | 198-235
 server/Controllers/OfferControllers |   96.69 |    77.77 |     100 |   96.69 |
  OfferControllers.js                |   96.69 |    77.77 |     100 |   96.69 | 143,248,267-269
 server/Controllers/RealtorsControllers|  97.79 |    77.27 |     100 |   97.79 |
  RealtorController.js               |   97.79 |    77.27 |     100 |   97.79 | 194,332-334
-------------------------------------|---------|----------|---------|---------|-------------------

Test Suites: 3 passed, 3 total
Tests:       72 passed, 72 total
Snapshots:   0 total
Time:        4.89 s, estimated 5 s
Ran all test suites.
```

Рисунок 3.24 – Тестування моделей та головних контролерів системи

3.3.2 Чек-лист тестування системи

	Gchrome	FF	Opera	Safari	Notes
Функіональні можливості адміністратора					
Додавання агенція	Passed	Passed	Passed	Passed	
Редагування агенцій	Passed	Passed	Passed	Passed	
Перегляд всіх агенцій	Passed	Passed	Passed	Passed	
Видалення агенцій	Passed	Passed	Passed	Passed	
Додавання рієлторів	Passed	Passed	Passed	Passed	
Редагування рієлторів	Passed	Passed	Passed	Passed	
Перегляд всіх рієлторів	Passed	Passed	Passed	Passed	
Видалення рієлторів	Passed	Passed	Passed	Passed	
Додавання оголошень	Passed	Passed	Passed	Passed	
Редагування оголошень	Passed	Passed	Passed	Passed	
Перегляд оголошень	Passed	Passed	Passed	Passed	
Видалення оголошень	Passed	Passed	Passed	Passed	
Пошук агенцій	Passed	Passed	Passed	Passed	
Пошук рієлторів	Passed	Passed	Passed	Passed	
Пошук оголошень	Passed	Passed	Passed	Passed	
Створення облікового запису	Passed	Passed	Passed	Passed	
Вхід до облікового запису	Passed	Passed	Passed	Passed	
Редагування облікового запису	Passed	Passed	Passed	Passed	
Перегляд облікового звпису	Passed	Passed	Passed	Passed	
Вихід з облікового запису	Passed	Passed	Passed	Passed	
Функціональні можливості агенції					
Додавання рієлторів	Passed	Passed	Passed	Passed	
Редагування рієлторів	Passed	Passed	Passed	Passed	
Перегляд рієлторів	Passed	Passed	Passed	Passed	
Видалення рієлторів	Passed	Passed	Passed	Passed	
Перегляд агенцій	Passed	Passed	Passed	Passed	
Перегляд оголошень	Passed	Passed	Passed	Passed	
Пошук агенцій	Passed	Passed	Passed	Passed	
Пошук рієлторів	Passed	Passed	Passed	Passed	

Рисунок 3.25 – Чек-лист функціональних можливостей

	Gchrome	FF	Opera	Safari	Notes
Наявність головних елементів сторінки					
Коректне відображення сторінок згідно файлів дизайну	Passed	Passed	Passed	Passed	
Наявність логотипу	Passed	Passed	Passed	Passed	
Наявність головного меню	Passed	Passed	Passed	Passed	
Коректне відображення головного контенту сайту	Passed	Passed	Passed	Passed	
Наявність футера сайту	Passed	Passed	Passed	Passed	
Відображення елементів сайту					
Коректне відображення шрифту елементів	Passed	Passed	Passed	Passed	
Коректне відображення кнопок, блоків меню і т.д.	Passed	Passed	Passed	Passed	
Перевірка всіх пунктів не зареєстрованим / зареєстрованим користувачем	Passed	Passed	Passed	Passed	
Масштабування елементів сайту до розміру	Passed	Passed	Passed	Passed	
Коректне відображення банерів	Passed	Passed	Passed	Passed	
Активні елементи					
Змінення курсору при наведенні на елемент	Passed	Passed	Passed	Passed	
Реагування активних елементів на наведення	Passed	Passed	Passed	Passed	
Наявність підсказок для активних елементів	Passed	Passed	Passed	Passed	
Відображення тексту					
Перевірка орфографії тексту	Passed	Passed	Passed	Passed	
Коректне відображення тексту	Passed	Passed	Passed	Passed	
Відображення зображень					
Коректне центрування зображень в контейнерах	Passed	Passed	Passed	Passed	
Коректні розміри зображень	Passed	Passed	Passed	Passed	
Форми сайту					
Підпис полей вводу	Passed	Passed	Passed	Passed	
Валідація полей вводу	Passed	Passed	Passed	Passed	
Повідомлення після відправки форми	Passed	Passed	Passed	Passed	

Рисунок 3.26 – Чек-лист верстки

План тестування	Критерії тестування	Gchrome	FF	Opera	Safari	Notes
1.1 Оформлення елементів сторінки						
1.1.1 Header	Позиціювання елементів header на всіх сторінках;	Passed	Passed	Passed	Passed	
	Корректний розмір і позиціювання графічних елементів;	Passed	Passed	Passed	Passed	
	Орфографія тексту;	Passed	Passed	Passed	Passed	
	Наявність іконки входу у систему	Passed	Passed	Passed	Passed	
1.1.2. Menu	Наявність посилання на всі головні сторінки з контентом сайту	Passed	Passed	Passed	Passed	
	Постійні ширина/висота	Passed	Passed	Passed	Passed	
	Зміна виду кнопок при наведенні курсору	Passed	Passed	Passed	Passed	
	Відображення на всіх сторінках елементів меню	Passed	Passed	Passed	Passed	
1.1.3 Footer	Наявність посилань на додаткову інформацію	Passed	Passed	Passed	Passed	
	Відображення на всіх сторінках елементів підвалу	Passed	Passed	Passed	Passed	
1.2. Додавання оголошення до сайту	Корректне відображення інпутів форми додавання оголошення	Passed	Passed	Passed	Passed	
	Автоматична підстанова міста для оголошення, згідно до вказаного міста рієлтора	Passed	Passed	Passed	Passed	
	Підпис всіх полів форми	Passed	Passed	Passed	Passed	
	Зручний вибір головної фотографії оголошення	Passed	Passed	Passed	Passed	
	Валідація полів форми	Passed	Passed	Passed	Passed	
	Відображення повідомлення в разі помилки, або успішного додавання оголошення	Passed	Passed	Passed	Passed	

Рисунок 3.27 – Чек-лист зручності використання сайту

ВИСНОВКИ

Ознайомившись з теоретичними відомостями пов’язаними з тематикою кваліфікаційної роботи, а також беручи до уваги гіркі події цього року по всій території України, створення онлайн платформ для продажу нерухомості буде дуже затребуване в найближчому майбутньому.

Для створення платформи з продажу нерухомості згідно до завдання кваліфікаційної роботи було обрано наступні технології:

– Node.js та фреймворк, на його основі, Express, для реалізації серверної частини інформаційної системи;
– бібліотека React для створення клієнтської частини інформаційної системи;
– MySQL як базу даних інформаційної системи;
– UML (Unified Modeling Language) для проєктування та моделювання інформаційної системи.

За час виконання завдання кваліфікаційної роботи було розглянуто можливості Node.js, Express та React для розробки веб додатків. Було розглянуто мову проєктування UML (Unified Modeling Language) та ER проєктування систем баз даних. А також отримані навички проєктування архітектури інформаційних систем та реалізація за допомогою всіх вище перерахованих технологій.

Результатом кваліфікаційної роботи є:

– технічне завдання для розробки платформи з продажу нерухомості;
– спроектовані основні бізнес-процеси та база даних платформи з продажу нерухомості;
– реалізована платформа з продажу нерухомості;
– результат автоматичного тестування контролерів та чек листи тестування створеної платформи з продажу нерухомості.

ПЕРЕЛІК ПОСИЛАНЬ

1. Плюси та мінуси React: віртуальна DOM, синтаксис JSX та інші аргументи для суперечки. URL: https://nuancesprog.ru/p/14500/ (дата звернення: 28.09.2022).
2. Node.js. URL: https://uk.wikipedia.org/wiki/Node.js (дата звернення: 28.09.2022).
3. Express.js. URL: https://uk.wikipedia.org/wiki/Express.js (дата звернення: 28.09.2022).
4. Діаграми UML для моделювання процесів і архітектури проекту. URL: https://evergreens.com.ua/ua/articles/uml-diagrams.html (дата звернення: 21.10.2022).
5. Діаграма прецедентів. URL: https://uk.wikipedia.org/wiki/Діаграма_прецедентів (дата звернення: 05.11.2022).
6. Що таке діаграма діяльності? – визначення з техопедії. URL: https://uk.theastrologypage.com/activity-diagram (дата звернення: 06.11.2022).
7. Що таке ER-діаграма та як її створити? URL: https://www.lucidchart.com/pages/ru/erd-диаграмма/#section_0 (дата звернення: 07.11.2022).
8. Тестування програмного забезпечення. URL: https://uk.wikipedia.org/wiki/Тестування_програмного_забезпечення (дата звернення 25.11.2022).
9. Jest. URL: https://jestjs.io/uk/ (дата звернення 25.11.2022).

ДОДАТОК А

Реалізація кінцевих точок для роботи з записами оголошень

A.1 ShowOfferList

```
async showOfferList (req, res) {
    try {
      let { city, minPrice = 1, maxPrice, minFloor = 1, maxFloor = 200, minSuperficiality = 1,
        maxSuperficiality = 200, ...params } = req.query

      const page = req.params.page
      const limit = 9
      const offset = limit * page - limit
      const attributes = [
        "id",
        "price",
        "cityId",
        "area",
        "street",
        "photo",
        "rooms",
        "building",
        "square"
      ]
      const priceFilter = minPrice && maxPrice ?
        { [Op.between] : [ minPrice, maxPrice] } :
        { [Op.or] : { [Op.gte] : minPrice, [Op.lte] : maxPrice } }

      const query = {
          cityId : city,
          price: priceFilter,
          floor: {
            [Op.between] : [ minFloor, maxFloor]
          },
```

```
            superficiality: {
                [Op.between] : [ minSuperficiality, maxSuperficiality]
            },
            ...params
        }
        const offers = await Immovables.findAll({
            order: [
                ['price', 'ASC']
            ],
            where: query,
            attributes,
            limit,
            offset
        })
        if(offers.length >= 1) {
            const count = await Immovables.findAndCountAll( { where: query } )
            const pages = Math.ceil(count.count/limit)
            return res.json({result:{offers, pages}})
        }
        return res.status(404).json({error:"Жодної пропозиції за обраними крітеріями"})
    } catch (e) {
        res.status(500).json({error:e.message})
    }
}
```

A.2 ShowOffer

```
async showOffer (req, res) {
    try {
        const id = req.params.id
        const offer = await Immovables.findOne({where:{id}})
        if(offer){
            const city = await offer.getCity()
            const wall = await offer.getWall()
            const offerPhotos = await offer.getPhotos()
            const realtor = await offer.getRealtor({
```

```
                attributes: [
                    'id',
                    'fullName',
                    'rating',
                    'number',
                    'photo'
                ]
            })
            offer.setDataValue('city', city)
            offer.setDataValue('wall', wall)
            offer.setDataValue('realtor', realtor)
            offer.setDataValue('offerPhotos', offerPhotos)
            return res.json({result : offer})
        }
        res.status(404).json({error:"Такої пропозиції не існує"})
    } catch (e) {
        res.status(500).json({error:e.message})
    }
}
```

A.3 CreateOffer

```
async createOffer(req, res) {
    const photo = req.file.filename
    const unlincPath = path.resolve(__dirname, '..', '..', 'public', 'OfferPhotos', photo)
    try {
        const immovable = await Immovables.create({...req.body, photo: photo})
        res.send(JSON.stringify({result:true, id:immovable.id}))
    } catch (e) {
        unlinkSync(unlincPath)
        res.status(500).send({error:"При створенні сталася помилка"})
    }
}
```

A.4 UploadOfferPhotos

```
async uploadOfferPhotos (req, res) {
    const id = req.params.id
    const photos = req.files
    try {
      for(let photo of photos) {
        await Photos.create({photo:photo.filename, immovableId: id})
      }
      res.send(JSON.stringify({result:true}))
    } catch (e) {
      for(let photo of photos){
        const unlincPath = path.resolve(__dirname, '..', '..', 'public', 'OfferPhotos', 'OfferDetailPhotos',
photo.filename)
        unlinkSync(unlincPath)
      }
      return res.status(500).send(JSON.stringify({error:"При завантаженні сталася помилка"}))
    }
```

A.5 DeleteOffer

```
async deleteOffer(req, res) {
    try {
      const id = req.params.id
      const offer = await Immovables.findOne({where:{id}})
      const photos = await offer.getPhotos()
      const oldPhoto = offer.photo
      await Immovables.destroy({where:{id}})
      for(let item of photos){
        const unlincPath = path.resolve(__dirname, '..', '..', 'public', 'OfferPhotos', 'OfferDetailPhotos',
item.photo)
        unlinkSync(unlincPath)
      }
      unlinkSync(path.resolve(__dirname, '..', '..', 'public', 'OfferPhotos', oldPhoto))
      return res.json({result:true})
```

```
    } catch (e) {
      res.status(500).json({error:'При видаленні сталася помилка'})
    }
```

A.6 UpdateOffer

```
async updateOffer(req, res) {
    try {
      const id = req.params.id
      let {oldPhotos} = req.body
      if(req.file){
        const immovable = await Immovables.findOne({where:{id}})
        req.body.photo = req.file.filename
        unlinkSync(path.resolve(__dirname, '..', '..', 'public', 'OfferPhotos', immovable.photo))
      }
      await Immovables.update({...req.body}, {where:{id}})
      if(oldPhotos){
        oldPhotos = JSON.parse(oldPhotos)
        for(let photo of oldPhotos) {
          await Photos.destroy({where:{photo}})
          unlinkSync(path.resolve(__dirname, '..', '..', 'public', 'OfferPhotos', 'OfferDetailPhotos', photo))
        }
      }
      return res.send(JSON.stringify({result:true}))
    } catch (e) {
      if(req.file){
        unlinkSync(path.resolve(__dirname, '..', '..', 'public', 'OfferPhotos', req.file.filename))
      }
      res.status(500).send(JSON.stringify({erorr:e.message}))
    }
  }
```