

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування інституту, факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавр

(ступінь вищої освіти)

на тему ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ДЛЯ МЕНЕДЖМЕНТУ
КОЛЕКТИВНОЇ ОРЕНДИ ЖИТЛА
SOFTWARE IMPLEMENTATION OF THE MANAGEMENT
SYSTEM FOR COLLECTIVE HOUSING RENTAL

Виконав: студент 4 курсу, групи КНТ-119сп

Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

Перегуда В.Б.

(прізвище та ініціали)

Керівник Олійник А.О.

(прізвище та ініціали)

Рецензент Гофман Є.О.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування закладу вищої освіти)

Інститут, факультет ФКНТ
 Кафедра програмних засобів
 Ступінь вищої освіти бакалавр
 Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
 Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
 “ ” 2022 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА

Перегуди Владислава Борисовича
(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Програмна реалізація системи для менеджменту
колективної оренди житла.
Software Implementation of the Management System for Collective Housing Rental

керівник проєкту (роботи) Олійник Андрій Олександрович, д.т.н., професор,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від “27” квітня 2022 року № 102

2. Строк подання студентом проєкту (роботи) 07 червня 2022 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне
завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблеми та постановка завдань дослідження.
2. Матеріали і методи. 3. Опис програми. 4. Експлуатація, тестування та експериментальне дослідження програми.

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

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	Олійник А.О., д.т.н., професор		
Нормоконтроль	Липовець М.В., асистент		

7. Дата видачі завдання « 04 » квітня 2022 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	2 тиждень	Розділ 2
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент

(підпис)

Перевода В.Б.

(прізвище та ініціали)

Керівник проєкту (роботи)

(підпис)

Олійник А.О.

(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 92 с., 34 рис., 15 табл., 4 дод., 16 джерел.

WEB, JAVASCRIPT, REACT, БД, ОПЕНДА, ЖИТЛО, POSTGRESQL, MVC, NODE, DOCKER, DEPLOYMENT.

Об'єкт дослідження – процес функціонування, підтримки та супроводу електронної системи для менеджменту житла, призначеного для оренди у вигляді вебзастосунку.

Предмет дослідження – програмні засоби для програмної реалізації вебзастосунку з метою підтримки системи для менеджменту колективної оренди житла.

Мета роботи – розробка вебзастосунку для менеджменту колективної оренди житла.

Матеріали, методи та технічні засоби: об'єктно-орієнтоване програмування та функціональне програмування, мова програмування JavaScript, персональний комп'ютер з процесором IntelPentium 4415U під управлінням операційної системи Linuxз дистрибутивом Ubuntu 22.04.

Практична цінність роботи полягає у розробці програмного забезпечення у вигляді кроссплатформного вебзастосунку для менеджменту колективної оренди житла.

Результати. Створено вебзастосунок, що дозволяє ефективно управляти житлом, а також спрощує його пошук і подальшу оренду.

Висновки. Розроблено програмну систему, яка призначена для орендодавців та орендарів, щоб спростити процес оренди.

Галузь використання – користувачі мережі Інтернет, молодь, студентство, біженці.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 92 pages, 34 figures, 15 tables, 4 appendixes, 16 sources.

WEB, JAVASCRIPT, REACT, БД, RENT, HOUSE, POSTGRESQL, MVC, NODE, DOCKER, DEPLOYMENT.

Object of study is a process of functioning, support and maintenance of the housing rental management system in the form of a web-application.

The subject of the research is a way for software implementation of a web-application for servicing a collective rental management system.

The purpose of the work is to create a web-application for collective rental management.

Materials, methods, and tools: object-oriented and functional programming, programming language JavaScript, IBM-compatible personal computer with an Intel Pentium 4415U processor under the control of the Linux operating system with Ubuntu 22.04 distributive.

Results. The web-application has been created that allows effectively manage a house and simplifies the search and further rental.

The practical value of the work lies in the development of cross-platform software for managing collective rent.

Conclusions. A software system has been developed to simplify the rental process for landlords and tenants.

Scope of use – Internet users, youth, students, refugees.

ЗМІСТ

	С.
Перелік скорочень та умовних познач	8
Вступ	9
1 Аналіз предметної області	10
1.1 Предметна область	10
1.2 Дослідження існуючих аналогів	12
1.3 Постановка завдання	15
1.3 Висновки за розділом 1	18
2 Розробка архітектури програми	19
2.1 Вибір інструментарію розробки	19
2.2 Структурна схема розробки	27
2.3 Висновки за розділом 2	29
3 Основні рішення щодо реалізації компонентів системи	30
3.1 Функціонування системи	30
3.2 Інфологічне проєктування	30
3.3 Фізичне проєктування	31
3.4 Проєктування бази даних	36
3.5 Проєктування дизайну	41
3.6 Висновки за розділом 3	41
4 Експлуатація, тестування та експериментальне дослідження програми	42
4.1 Опис застосунку	42
4.2 Умови виконання програми	42
4.3 Інструкція по експлуатації програми	42
4.4 Методика та результати тестування	44
4.5 Висновки за розділом 4	51
Висновки	52
Перелік джерел посилання	53
Додаток А Технічне завдання	55

Додаток Б Опис програми	61
Додаток В Код програми	66
Додаток Д Слайди презентації	84

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API – Application Programming Interface;

AWS – Amazon Web Services;

НОС – Higher-Order Component;

HTML – HyperText Markup Language;

JS – JavaScript;

JSON – JavaScript Object Notation;

БД – база даних;

СКБД – система керування базами даних.

ВСТУП

Може бути важко знайти відповідне місце, щоб зателефонувати в свій наступний будинок, а знайти його до того, як його розкупить інший орендар, може бути ще складніше. На конкурентному ринку вільні квартири з'являються і зникають в одну мить, і день відкритих дверей може більше схожим на протистояння з іншими орендарями, ніж на екскурсію по приміщенню.

Індустрія онлайн-оголошень про продаж або оренду квартир конкурує за надання більш точних даних і кращих варіантів пошуку, що приносить користь орендарям, оскільки більш корисна інформація стає доступною одним натисканням пальця. У міру розвитку застосунків для оренди вони не тільки полегшують пошук вільних квартир, але й надають потенційним орендодавцям фінансову інформацію про орендарів і дозволяють орендарям платити орендну плату після підписання договору оренди.

Метою дипломної кваліфікаційної роботи бакалавра є розробка застосунку для спрощення пошуку і менеджменту житла для оренди.

На сьогоднішній день поняття «колективної оренди» є новим для України. Але у Європі та Великобританії цей спосіб оренди користується високим попитом, особливо серед студентів. Адже це дозволяє студентам орендувати доволі якісне житло при цьому сплачуючи лише частину коштів, бо сума оренди рівномірно ділиться між усіма орендарями. Для формування коректно розподілених комунальних платіжок існують спеціальні сервіси.

Актуальність роботи підтверджується тим, що питання проживання і якості житла є досить розповсюдженою проблемою серед українських студентів, а розробка сервісу для колективної оренди може спростити пошук і менеджмент житла для оренди.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Предметна область

Тематикою дипломної кваліфікаційної роботи бакалавра є вебзастосунок для підтримки оренди житла серед студентів.

Оренда житла – є найгострішою потребою і проблемою серед студентів. Оскільки якість проживання у гуртожитках залишає бажати кращого, то наразі виникає попит оренди квартир або невеликих будинків студентами. Але комунальні платежі становлять більше половини вартості оренди житла, тому студенти намагаються орендувати житло групами.

Основною задачею вебдодатку є комутація комунальних платежів усередині системи, формування «платіжок» та розсилання їх орендарям, зручний і швидкий пошук житла за допомогою фільтрів, перегляд історії показів лічильників та платежів. Додаток дозволяє уникати проблем між орендодавцями та орендарями за допомогою попередження надмірного використання ресурсів.

1.1.1 Поняття оренди житла

Оренда житла – вид договору між орендодавцем та орендарем, при укладанні якого нерухомість передається у тимчасове володіння і використання орендарю за певну плату (зазвичай помісячну). У цьому випадку орендар має право проживати у орендованому житлі на термін, який узгоджений у договорі, але не має прав розпоряджатися майном як власник [1].

Процес оренди, зазвичай, є юридично узгодженим і передбачає укладання нотаріально завіреного договору між орендарем і орендодавцем. При відсутності договору обидві сторони не можуть виносити претензії один одному.

Оренда житлової нерухомості завжди має високий попит серед населення і має такі переваги:

- збереження мобільності, що дозволяє з легкістю переїздити у інші міста;
- не потрібно купувати власне житло, на яке, на жаль, у більшості громадян не вистачає коштів;
- різноманітність нерухомості, що дозволяє обрати квартиру або будинок на власний смак, з будь-яким плануванням та інтер'єром.

1.1.2 Поняття колективної оренди

Колективна оренда – процес оренди житла певним колом людей, наприклад друзів. Цей вид оренди дозволяє розділити вартість оренди та комунальних платежів порівну і знизити загальну вартість проживання[2].

Колективна оренда користується дуже великим попитом у багатьох країнах Європи, але, на жаль, ще не знайшла свого розповсюдження в Україні. Однією з перших країн, яка ввела поняття «колективної оренди» або «flat-sharing» є Великобританія [3]. Там такий вид оренди наразі є дуже популярним і, навіть, будуються окремі райони, будинки в яких розраховані саме на колективне проживання.

Колективна оренда змогла завоювати високу популярність завдяки багатьом перевагам. По-перше, це дозволяє зберегти власні кошти без погіршення якості проживання. По-друге, допомагає студентам соціалізуватись, адже сумісне проживання обов'язково пов'язане з щоденною комунікацією з однолітками. По-третє, сусідами можуть бути люди різних національностей та культур, що дозволяє розвивати свою особистість в різних напрямках.

1.2 Дослідження існуючих аналогів

Нажаль, такий вид сервісу не має широкого поширення в Україні, тому знайти аналоги не є можливим. Основним способом орендування житла українськими студентами є його пошук на дошках оголошень. Але такий спосіб має як недоліки, так і переваги.

Недоліки:

- відсутність історії платежів;
- неможливість перегляду історії показів лічильників для побудування графіків;
- складність управління житлом для орендодавців.

Переваги:

- відсутність додаткових витрат для орендодавців;
- розповсюдженість дошок оголошень;
- широкий вибір житла для орендарів.

Підбиваючи підсумки можна сказати, що пошук житла для подальшої оренди є розповсюдженим способом в Україні, але такий спосіб не є зручним та ефективним. За основу додатка було взято зарубіжні аналоги, які знайшли своє розповсюдження у Великобританії.

1.2.1 Аналог Uhihomes

Uhihomes – сервіс для пошуку житла для студентів. Має зручний та сучасний інтерфейс, широкий набір параметрів для пошуку, а також можливість порівняння якості умов серед декількох варіантів. Сервіс одразу відображає вартість оренди з урахуванням вартості комунальних послуг, які нараховуються по середньому використанню ресурсів. Загальний вигляд вебсайту Uhihomes зображено на рисунку 1.1.

1.2.2 Аналог StuRents

Сервіс StuRents дуже схожий на свій аналог, теж має зручний інтерфейс і можливість «глибокого» пошуку за допомогою різноманітних фільтрів. Але основною відмінністю є наявність 3Dтуру по будинку для перегляду інтер'єру. Загальний вигляд вебсайту StuRents зображено на рисунку 1.2.

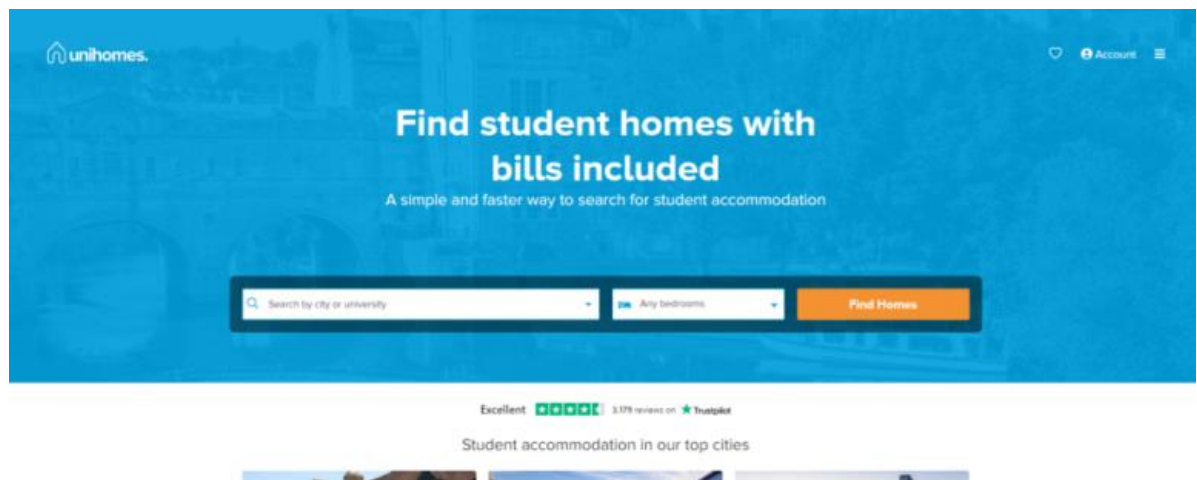


Рисунок 1.1 – Сервіс Unihomes

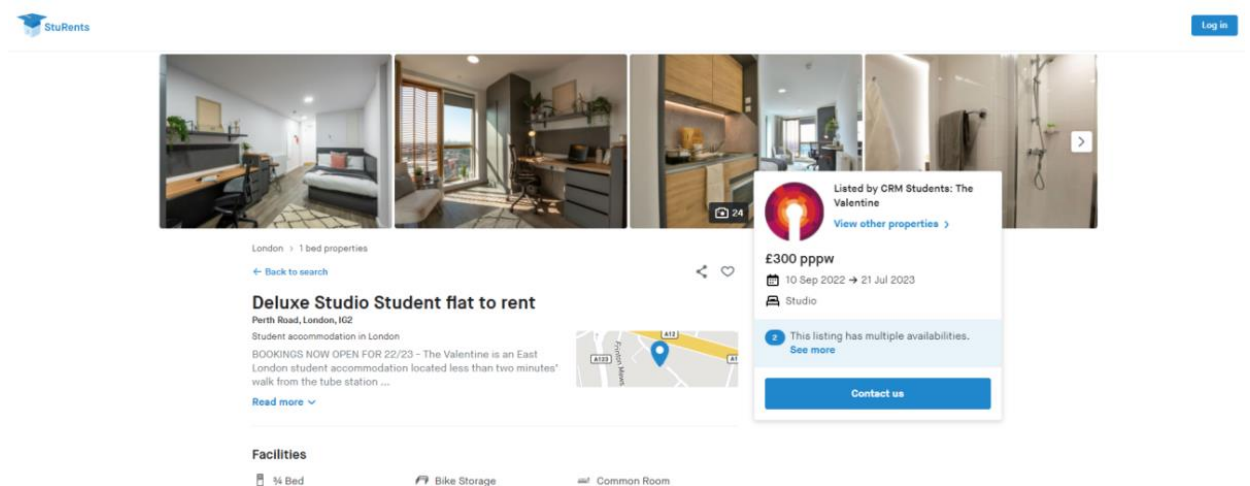


Рисунок 1.2 – Сервіс StuRents

1.2.3 Порівняння існуючих аналогів

Проведемо порівняння за певними критеріями, хід якого представимо у вигляді таблиці (табл. 1.1). У таблиці будуть порівнюватися попередні аналоги із за стосунком, що розробляється – «Сервіс для менеджменту колективної оренди житла». Серед критеріїв виділимо наступні:

- розумний пошук критерій, що характеризує наявність в системі функції виведення спочатку будинків, які знаходяться ближче до поточної локації користувача;
- висока швидкість роботи – критерій, який відображає швидкість опрацювання запитів користувача системою;
- комутація комунальних платежів – критерій, що характеризує наявність механізму для формування та відображення комунальних платежів всередині системи;
- зручність та сучасність дизайну – загальна характеристика системи, яка передбачає, що усі компоненти системи мають візуально гарний вигляд, а також розташовані на місцях, які є інтуїтивно зрозумілими для користувача;
- можливість реєстрації як орендодавця і додавання власних будинків – критерій, що характеризує наявність механізму для додавання власних будинків орендодавцем для розповсюдження їх серед користувачів системи.

Таблиця 1.1 – Порівняння існуючих аналогів

Критерій порівняння	Unihomes	StuRents	Мій застосунок
1	2	3	4
Розумний пошук	+	–	–
Висока швидкість роботи	+	–	+

Продовження таблиці 1.1

1	2	3	4
Комутація комунальних платежів	—	—	+
Зручність та сучасність дизайну	+	+	±
Можливість реєстрації як орендодавця і додавання власних будинків	—	—	+

З результатів порівняння можна зробити висновок, що сервіси-аналоги пропонують користувачам зручний пошук житла для оренди, але не надають можливість ефективного управління цим житлом.

Саме тому актуальною залишається задача розробки нового за стосунку, який буде мати високу швидкість роботи, сучасний дизайн, а головне – комутувати комунальні платежі і розділяти їх для реалізації зручної колективної оренди.

1.3 Постановка завдання

За результатами аналізу проблемної області та дослідження існуючих аналогів для вирішення проблемних питань, сформулюємо постановку задачі.

Метою є розробка нового за стосунку для полегшення пошуку житла студентами. Система повинна реалізувати такі функціональні вимоги:

- реєстрація та авторизація в системі;
- пошук будинків та їх фільтрація;
- перегляд інформації користувача;
- додавання нових будинків орендодавцями;

- перегляд історії платежів та показів лічильників;
- здійснення платежів в системі;
- формування комунальних «платіжок» та їх розсилка;
- передавання показів лічильників орендарями через систему;
- перегляд графіків використання ресурсів.

Система також повинна бути стійкою до збоїв, працювати як на десктопах, так і на мобільних пристроях, бути адаптованою під різні розширення екранів і легкою для подальшої підтримки і масштабування.

1.3.1 Функціональність системи

Робота із застосунком–орендар:

- реєстрація та авторизація;
- пошук житла;
- здійснення платежів в системі;
- передавання показів лічильників;
- перегляд історії платежів та показів лічильників;
- перегляд комунальних «платіжок»;
- перегляд та коригування особистої інформації.

Редагування сайту – адміністратор:

- перегляд усіх користувачів в системі;
- перегляд усіх будинків;
- перегляд історії платежів та показів лічильників;
- формування комунальних «платіжок» за певний період та їх

розсилка.

Робота з сайтом – орендодавець:

- реєстрація та авторизація;
- додавання будинків в систему;
- перегляд доданих будинків;

- перегляд комунальних «платіжок» та показів лічильників;
- перегляд графіків використання ресурсів.

1.3.2 Вимоги до інтерфейсу

Для забезпечення ергономічності, зручності та зрозумілості інтерфейс розроблюваного застосунку повинен бути інтуїтивно зрозумілим, простим та сучасним. Також, додаток має працювати однаково гарно на десктопах та мобільних пристроях.

1.3.3 Нефункціональні вимоги до продукту

Мінімальні системні вимоги та програмне забезпечення, що необхідні для функціонування серверної частини мобільного застосунку:

- операційна система: Linux;
- процесор з двома ядрами і з тактовою частотою не менше 2 ГГц;
- оперативна пам'ять: 8 Гбайт;
- встановлене програмне забезпечення Docker;
- дисковий простір: SSD 30Гбайт.

Вимоги для клієнтської сторони: для роботи з сервісом потрібно мати сучасний браузер (Google Chrome, Mozilla Firefox, Safari) та стабільне підключення до мережі Інтернет.

1.3.4 Усвідомлення середовища користувача

Для роботи з додатком потрібно обов'язкова реєстрація в системі. Для зареєстрованих користувачів сервіс надає особисту інформацію користувача, список будинків, доступних до оренди, а також доступ у особистий кабінет, у якому користувач може переглядати інформації про будинок.

1.3.5 Оцінка надійності, продуктивності і підтримки

Для забезпечення достатньої надійності та продуктивності система має відповідати певним вимогам:

- стійкість до збоїв;
- захищене зберігання користувацьких даних;
- обробка помилок та виведення їх у зрозумілому для користувача вигляді;
- додаток повинен видавати повідомлення при некоректному заповненні полів;
- перевірка введених даних як на стороні клієнта, так і на стороні сервера.

1.3 Висновки за розділом 1

В першому розділі було розглянуто основні поняття, що стосуються сервісів для пошуку житла. У тому числі було проведено порівняння вже існуючих аналогів із розроблюваним програмним продуктом.

У ході аналізу було виявлено, що в Україні подібні сервіси не розповсюджені, але попит на них є тож тему можна вважати актуальною.

Було сформовано функціональні та не функціональні вимоги до розроблюваного додатку, переглянуто вимоги до користувацького інтерфейсу, а також проведена оцінка надійності та продуктивності майбутнього програмного продукту.

2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

В попередньому розділі було досліджено предметну область. Було проаналізовані існуючі рішення з виявленням їх основних переваг та недоліків. Основним недоліком аналогічних сервісів є дуже низька їх розповсюдженість в Україні. Результати аналізу та порівняння показують, що обрана тема є дуже актуальною на сьогоднішній день.

Надалі необхідно визначитися із технологією, фреймворками, мовою програмування, а також середовищем розробки і патерном для побудови архітектури застосунку.

2.1 Вибір інструментарію розробки

2.1.1 Вибір мови програмування

JavaScript – це об’єктно-орієнтована мова програмування з динамічної типізацією. Основним середовищем використання є розробка інтерактивних Web-сторінок. Використання JavaScript надає можливість асинхронно взаємодіяти з сервером, змінювати зовнішній вигляд та структуру сторінки. За допомогою технологій мови JavaScript створюються односторінкові сайти (React.js, Angular, Vue.js), серверна частина сайту (Node.js), мобільні додатки (ReactNative) та стаціонарні додатки (Electron) [4]. Стандартом мови є ECMAScript. Станом на 2012 рік, усі сучасні браузері повністю підтримують ECMAScript 5.1 [5]. Важливою перевагою використання JavaScript для розробки WEB-додатків є можливість його використання як для створення клієнтської частини, так і серверної. Це стало можливим за допомогою програмної платформи Node.JS, заснованої на базі V8 від Google [6]. Також, JS має дуже широке коло спільноти, велику кількість різноманітних фреймворків та бібліотек, які значно спрощують рутинну роботу і дозволяють, навіть, створювати мобільні додатки.

Як і у випадку з будь-якою іншою технологією, JavaScript має як переваги, так і недоліки. Але коли справа доходить до цієї конкретної мови програмування, недоліки JavaScript повністю покриваються його перевагами, тому давайте розглянемо їх більш детально [4]-[6].

Вічно зелена і популярна технологія. JavaScript існує тут вже більше 25 років. І це якраз той випадок, коли технологія ще не застаріла. Замість цього на шляху свого розвитку JavaScript придбав велику і підтримуючу спільноту, і число його варіантів використання зростало [4]-[6].

У той час як в світі налічується 1,8 мільярда вебсайтів [4]-[6], JavaScript підтримує 95% з них, і його популярність з плином часу досить стабільна. Це один з найпопулярніших мов програмування за всю історію, і завдяки оновленням, що вносяться популярними браузерами, продуктивність JavaScript рік від року стає ще краще [4]-[6].

Великий кадровий резерв. З вищевказаних причин однією з переваг JavaScript є те, що він володіє досить великим кадровим резервом. У світі налічується 13,8 мільйонів розробників JavaScript, і їх населення продовжує зростати через попит на послуги розробки JavaScript і простоти освоєння цієї технології [4]-[6].

Таким чином, якщо ваш проєкт працює на JavaScript, ви навряд чи зіткнетеся з нестачею талантів. Навпаки, у вас є найбільший кадровий резерв на вибір, і агентство Tіno також готове надати вам підтримку в розробці користувальницьких вебсайтів з використанням JavaScript і пов'язаних з ними технологій [4]-[6].

Майже різнобічний характер. При професійному використанні одного JavaScript буде достатньо для створення як зовнішнього, так і внутрішнього інтерфейсу програми [4]-[6]. Більш того, JavaScript ідеально сумісний з популярними браузерами; він дозволяє створювати адаптивні дизайни і, як правило, не залежить від платформи. Ось чому JavaScript є одним з найпопулярніших варіантів для крос-платформної розробки [4]-[6].

Багаті та креативні інтерфейси для створення. JavaScript досить гнучкий; він має багато можливостей для творчості та реалізації ефективних ідей інтерфейсу користувача. Наприклад, JavaScript ідеально підходить для створення приголомшливих слайдерів, функцій перетягування, анімації та інших компонентів інтерфейсу користувача [4]-[6].

Тонни інструментів і фреймворків. Список переваг JavaScript можна продовжувати і продовжувати, але було б неправильно випускати з уваги той факт, що JavaScript поставляється з безліччю інструментів і фреймворків, що дозволяють краще адаптувати цю мову програмування до конкретного варіанту використання і розкрити його кращі можливості [4]-[6]. Наприклад, в React.JS бібліотека – це набір готових елементів користувацького інтерфейсу, які розробники можуть миттєво використовувати повторно. Angular. JavaScript це ще один вічно зелений фреймворк, який в основному використовується для розробки вебсайтів зі складною архітектурою, але при цьому з високою продуктивністю [4]-[6].

Такий широкий спектр інструментів, розширень, фреймворків і бібліотек значно спрощує і прискорює процес розробки за допомогою JavaScript, що є одним з основних переваг JavaScript.

У таблиці 2.1 проведено порівняння популярних мов програмування для розробки WEB-додатків за наступними критеріями:

- суворі типізація – критерій, що показує який вид типізації має мова програмування – динамічну чи статичну;
- підтримка парадигм ООП – критерій, що відображає підтримку основних принципів ООП (інкапсуляція, наслідування, поліморфізм) мовою програмування;
- можливість розробки клієнтської частини – критерій, що показує чи можливе використання мови програмування для розробки клієнтських інтерфейсів;

- можливість розробки серверної частини – критерій, що показує чи можливе використання мови програмування для розробки серверної логіки;
- легкість вивчення та використання – критерій, що відображає наскільки легко вивчити мову програмування і зручно використовувати її для розробки.

Таблиця 2.1 – Порівняння мов програмування для WEB-розробки

Критерій	JavaScript	PHP	C#	Java
Суворі типізація	—	—	+	+
Підтримка парадигм ООП	+	+	+	+
Можливість розробки клієнтської частини	+	—	+	+
Можливість розробки серверної частини	+	+	+	+
Легкість вивчення та використання	+	+	±	±

Результати порівняння показують, що обрана мова програмування є найбільш доречною для розробки програмного продукту, тому що має невисоку складність, а також дозволяє використовувати її для розробки як серверної, так і клієнтської частини, що дозволяє зекономити час на розробку.

2.1.2 Обґрунтування вибору програмної надбудови

У якості середовища розробки було обрано IntelliJ IDEA. Це дуже потужне середовище програмування від JetBrains, яке має вбудований відладчик та термінал, зручне керування системами контролю версій,

влаштований плагін для роботи з базами даних, авто доповнення коду тощо. Вбудований аналізатор коду може вказати на можливі проблеми з кодом у майбутньому, підкаже як краще написати ту чи іншу ділянку коду, проінформує про дублікати коду. Обране середовище розробки має дуже велику кількість плагінів, які дозволяють розширити вже існуючий функціонал. Наприклад, плагін для роботи з базами даних має підтримку більшості СКБД і дозволяє зручно виконувати «рутинні» операції такі, як вибірка, створення, оновлення та видалення даних, а також він представляє результати вибірки у зручному табличному вигляді [7].

Основними перевагами IntelliJ IDEA є широкий вибір інструментів, які вона надає для допомоги в розробці програмного забезпечення, а також ефективність і прискорення процесу розробки [7].

IntelliJ IDEA має багатий набір інструментів, які дозволяють більш точно, ефективно і продуктивно розробляти програмне забезпечення [7]. IDEA має дуже розумну ергономіку, зручний редактор для написання коду, а також має величезну кількість вбудованих інструментів для підтримки багатьох сучасних мов програмування [7].

Цікавою перевагою обраного середовища є розумне авто доповнення коду, що значно спрощує написання рутинних задач. Авто доповнення дозволяє програмісту отримати список найбільш релевантних символів, функцій, операторів, які можуть бути використанні в даному контексті. Також цей список не є статичним, а залежить від частоти використання того чи іншого оператора. Зазвичай, цей список підібраний дуже точно, що значно прискорює швидкість написання коду [7].

У таблиці 2.2 проведено порівняння обраного середовища з іншими.

Результати порівняння показують, що обране середовище програмування задовольняє усі потреби, які можуть виникнути під час розробки програмного продукту, а також має багато переваг серед аналогів.

Таблиця 2.2 – Порівняння середовищ розробки

Критерій	InteliJIDEA	Vim	WebStorm	SublimeText
Розширюваність	+	+	+	+
Вбудований аналізатор коду	+	±	+	—
Зручна робота з БД	+	—	±	—
Підтримка систем контролю версій	+	±	+	—
Налаштування віддаленого доступу	—	+	—	—

2.1.3 Система керування базами даних

У якості системи керування базами даних була обрана PostgreSQL. PostgreSQL – об’єктно-реляційна система керування базами даних. Основними перевагами PostgreSQL є висока продуктивність та швидкість роботи, надійні механізми транзакцій, вбудована процедурна мова PL/pgsql та підтримка типу JSON. PostgreSQL підтримує вдосконалені типи даних та функції оптимізації запитів. Важливим компонентом PostgreSQL є розширюваність та кастомізація, що дозволяє модифікувати деякі частини СКБД, наприклад оператори. Також ця СКБД має розширений механізм блокування який дозволяє блокувати не тільки таблиці, а і константні вирази [8].

Особливості PostgreSQL. Ось ключові особливості PostgreSQL:

- допомагає розробникам створювати за стосунки [8];
- це дозволяє адміністраторам створювати відмовостійке середовище, захищаючи цілісність даних [8];
- сумісність з різними платформами, що використовують всі основні мови і проміжне програмне забезпечення [8];
- він оснащений найдосконалішим механізмом блокування [8];

- підтримка управління паралелізмом декількох версій [8];
- розвинена функціональність програмування на стороні сервера;
- відповідає стандарту ANSI SQL;
- повна підтримка мережевої архітектури клієнт-сервер;
- SSL реплікації на основі журналів і тригерів;
- резервний сервер і висока доступність;
- об'єктно-орієнтований і сумісний з ANSI-SQL 2008;
- підтримка JSON дозволяє зв'язуватися з іншими сховищами даних, такими як NoSQL, які виступають в якості об'єднаного центру для баз даних polyglot [8].

Переваги PostgreSQL. Нижче наведено основні переваги / переваги PostgreSQL [8]:

- PostgreSQL може запускати динамічні вебсайти та вебзастосунки як опцію стека LAMP [8];
- ведення журналу з випередженням запису в PostgreSQL робить його дуже відмовостійкою базою даних [8];
- вихідний код PostgreSQL знаходиться у вільному доступі під ліцензією з відкритим вихідним кодом. Це дозволяє розробнику вільно використовувати, змінювати та впроваджувати його відповідно до потреб вашого бізнесу [8];
- PostgreSQL підтримує географічні об'єкти, тому Ви можете використовувати його для служб на основі місцезнаходження та географічних інформаційних систем;
- PostgreSQL підтримує географічні об'єкти, тому його можна використовувати в якості сховища геопросторових даних для служб, заснованих на місцезнаходження, і географічних інформаційних систем;
- щоб вивчити Postgres, розробнику не потрібно багато тренуватися, так як він простий у використанні [8];

– низькі витрати на технічне обслуговування та адміністрування як для вбудованого, так і для корпоративного використання PostgreSQL [8].

2.1.4 Розподіл та реалізація бізнес-логіки

Архітектурно за стосунок побудований з використанням схеми Model-View-Controller. Ця схема передбачає використання моделі (Model) у якості централізованого сховища даних, контролера (Controller), що використовується для оновлення даних у моделі, а також представлення (View), призначеного для відображення даних з моделі[9]. Користувач при взаємодії з представленням не змінює дані напряму у моделі, а робить це через контролер. Використання цієї схеми дозволяє працювати окремо з будь-якою з частин застосунку, використовувати різний набір технологій для побудови кожного з компонентів схеми та легко розширювати будь-яку частину [9].

Загалом, архітектурний шаблон Model-View-Controller розділяє застосунок на три основні групи компонентів: моделі, уявлення і контролери. Цей шаблон допомагає домогтися поділу проблем. Використовуючи цей шаблон, запити користувачів направляються на контролер, який відповідає за роботу з моделлю для виконання дій користувача та/або отримання результатів запитів. Контролер вибирає подання для відображення користувачеві і надає йому будь-які необхідні дані моделі [9].

Такий розподіл обов'язків допомагає розробнику масштабувати застосунок з точки зору складності, оскільки простіше кодувати, налагоджувати і тестувати щось (модель, уявлення або контролер), що має одне завдання. Складніше оновлювати, тестувати і налагоджувати код, який має залежності, розподілені по двох або більше з цих трьох областей [9]. Наприклад, логіка інтерфейсу користувача, як правило, змінюється частіше, ніж бізнес-логіка. Якщо код подання та бізнес-логіка об'єднані в одному об'єкті, об'єкт, що містить бізнес-логіку, повинен змінюватися кожного разу

при зміні інтерфейсу користувача. Це часто призводить до помилок і вимагає повторного тестування бізнес-логіки після кожної мінімальної зміни інтерфейсу користувача [9].

2.2 Структурна схема розробки

Структурно застосунок поділений на серверну частину, клієнтську частину, модуль роботи з AWS, утиліти, а також API для роботи з базою даних.

На рисунку 2.1 наведена структурна схема програмного продукту.

У таблиці 2.3 наведено опис модулів реалізованого додатку з визначенням їх основних функцій.

Таблиця 2.3 – Опис модулів реалізованого додатку з визначенням основних функцій

Модуль	Опис
Авторизація	Використовується для авторизації користувачів у системі, а також для зберігання сесій
Реєстрація	Використовується для реєстрації нових користувачів у системі
AWS	Використовується для роботи з Amazon Web Services, а саме з сервісами S3 (зберігання файлів користувачів) та SES (розсилка електронних листів)
База даних	Використовується для роботи з базою даних – побудування запитів, формування структури за допомогою міграцій
Фронтенд	Використовується для роботи користувача з системою

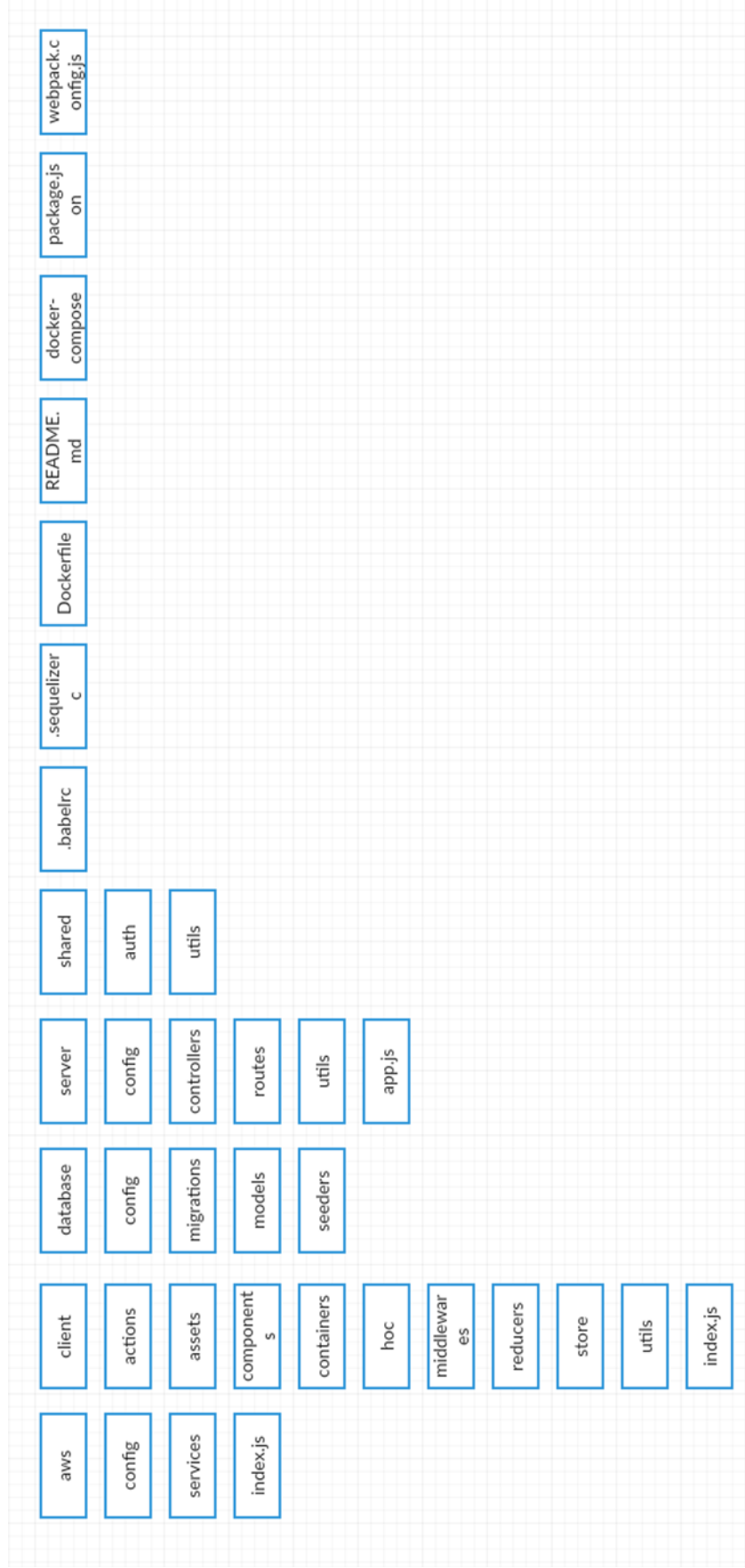


Рисунок 2.1 – Структурна схема программного продукта

2.3 Висновки за розділом 2

У другому розділі було описано особливості обраної мови програмування, середовища розробки і СУБД. Також було наведено структурну схему додатку і розглянуто основні модулі розроблюваної системи.

У якості середовища розробки було обрано IntelliJ IDEA. Також було проаналізовано аналоги і у результаті порівняння доведено доречність використання середовища розробки.

У якості патерну було обрано MVC.

3 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

3.1 Функціонування системи

Застосунок побудований з використанням клієнт-серверної архітектури. На основі цієї архітектури побудовані всі сайти на інтернет-сервіси. Вона передбачає наявність клієнтів, яких може бути безліч, сервера, який оброблює всі запити від клієнтів, а також централізованого сховища даних – бази даних (БД). Клієнтська частина необхідна для взаємодії користувача з системою, а також для відображення даних. Серверна частина реалізує всю бізнес-логіку додатку, оброблює запити за алгоритмами, використовуючи при цьому дані з БД. Зазвичай, клієнти «спілкуються» з сервером за допомогою API – інтерфейсу, який описує набір роутів, за якими може звертатися клієнт, а також дані, які необхідно надати і структуру відповіді. Основними перевагами клієнт-серверної архітектури є:

- уникання дублювання коду – завдяки тому, що вся бізнес-логіка міститься на сервері її не потрібно дублювати для кожного клієнта;
- безпека – усі користувацькі дані зберігаються у БД, деякі навіть в зашифрованому вигляді, що ускладнює їх отримання злодіями [10].

3.2 Інфологічне проєктування

Для опису моделі даних застосунку на етапі інфологічного проєктування було обрано модель «Сутність-Зв'язок» (Entity-Relationship – ER-модель). Ця модель відображає яким чином окремі сутності системи відносяться один до одного. Модель відображає сутності, їх атрибути, а також вид зв'язок між ними. На рисунку 3.1 зображена ER-діаграма застосунку.

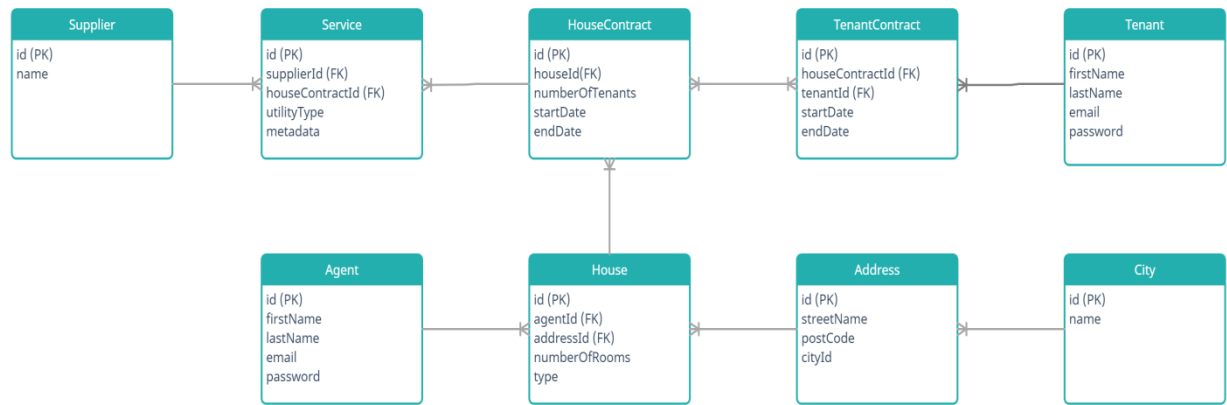


Рисунок 3.1 – ER-діаграма застосунку

3.3 Фізичне проєктування

Розроблюваний додаток поділяється на серверну та клієнтську частину.

Для розробки серверної частини була використана платформа Node.js. Це JavaScript-оточення, яке створена на основі двигуна V8 і призначена для повноцінного використання JavaScript для розробки бекенду. Ця платформа надає можливість взаємодіяти з пристроями введення/виведення, файловою системою, а також з безліччю іншими низькорівневими інтерфейсами. Платформа написана на мові C++, а в її основі лежить подійно-орієнтоване програмування з неблокуючим введенням/виведенням через EventLoop [11].

Для розробки клієнтської частини була використана бібліотека React.JS. Вона призначена для написання клієнтських інтерфейсів і є однією з найпопулярніших бібліотек, яка дозволяє створювати так звані SPA-застосунки (Single Page Application). Бібліотека реалізує набір методів, за допомогою яких можна зручно керувати DOM-ом, виконувати асинхронні операції, виконувати роутинг (навігацію) по сторінках без повного перезавантаження [12].

Для створення вебсерверу було побудовано клас Server. Цей клас призначений для створення вебсерверу, його конфігурування, налаштування

API, який необхідний для роботи з клієнтом, а також запуску. Серед методів класу можна виділити наступні:

- `setupHelmet` – метод, який призначений для конфігурації бібліотеки `helmet`, яка дозволяє зробити API більш захищеною за допомогою встановлення HTTP-заголовків;
- `setupBodyParser` – метод, який призначений для конфігурації бібліотеки `bodyParser`, яка дозволяє «парсити» дані запитів з заголовками `application/json` та `multipart/form-data`;
- `setupRoutes` – метод, який призначений для конфігурації API, який використовується клієнтом для взаємодії з сервером та БД;
- `setupStatic` – метод, який додає логіку завантаження статичних файлів, таких як іконки, статичні малюнки, `html` сторінки, клієнтом;
- `bootstrap` – основний метод, який викликає усі вищеописані методи, а також запускає вебсервер.

Для роботи з базою даних було реалізовано клас `DatabaseAPI`. Клас призначений для створення «пулу» підключень, зберігає SQL запити, а також запускає файли-міграції, які формують структуру БД. Основні методи класу:

- `migrate` – метод, який необхідний для конфігурації та запуску файлів-міграцій;
- `pool` – метод, який створює «пул» підключень;
- `openTransaction` – метод, який починає нову транзакцію у БД та надає зручний інтерфейс для керування нею.

Діаграма класів зображена на рисунку 3.2.

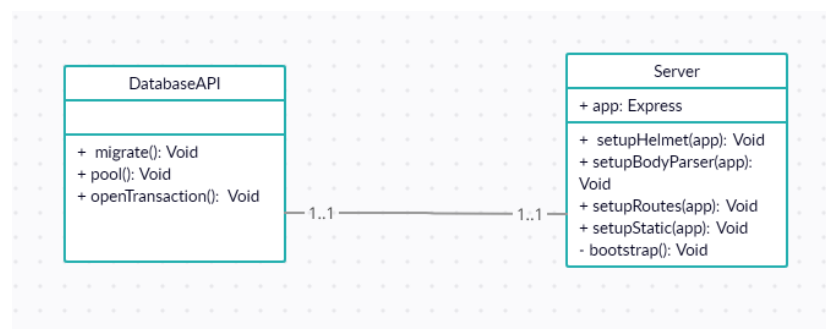


Рисунок 3.2 – Діаграма класів

Для повноцінної роботи застосунку було реалізовано багато алгоритмів, опис основних з них наведено у таблиці 3.1.

Таблиця 3.1 – Опис модулів реалізованого додатку з визначенням основних функцій

Модуль	Опис
Авторизація	Використовується для авторизації користувача в системі, генерації JWT-токена, а також перевірки його на валідність при кожному запиті
Реєстрація	Використовується для створення нових користувачів у системі
AWS	Модуль, який містить допоміжні функції для роботи з Amazon Web Services
База даних	Модуль для роботи з базою даних. Містить усі необхідні запити для повторного використання, формує структуру за допомогою міграцій, створює «пул» підключень
Фронтенд	Використовується для роботи клієнта з системою

3.3.1 Проєктування логіки взаємодії

Авторизація у застосунку відбувається з використанням JWT токенів. При вході в систему користувач має вказати електронну адресу та пароль. Далі відбувається пошук користувача у БД за вказаною поштою і

здійснюється порівняння введеного паролю у вигляді хешу. При успішній авторизації сервер генерує JWT токен і надсилає його клієнту, який зберігає його у браузерному сховищі. При кожному запиті на сервер відправляється токен і сервер перевіряє його валідність. На рисунку 3.3 зображена діаграма станів для алгоритму авторизації.

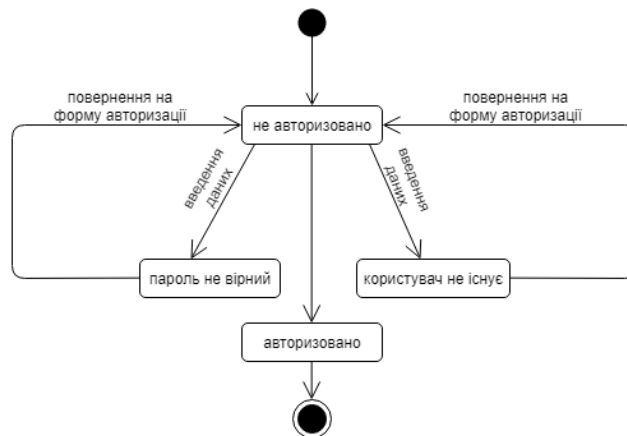


Рисунок 3.3 – Діаграма станів для алгоритму авторизації

Для відновлення забутого паролю користувач повинен перейти за відповідним посиланням на сторінці авторизації і вказати електронну адресу. Далі сервер перевіряє наявність користувача з вказаною адресою в системі і створює запис у таблиці ResetToken з випадковим uuid, термін дії якого складає 15 хвилин, а також відправляє посилання зі згенерованим uuid на електронну пошту. Після переходу за посиланням користувач буде мати змогу відновити пароль. На рисунку 3.4 зображена діаграма станів для алгоритму відновлення паролю.

Для збереження користувацьких зображень був використаний сервіс AWS S3. Цей сервіс зберігає файли у «бакеті», кожен файл повинен мати свій унікальний ключ. При завантаженні зображень відбувається конвертація їх у масив байтів, а також формується ключ зображення. Після цього зображення завантажується у сховище, а його ключ зберігається в таблиці Upload. Отримати зображення можна за ключем. На рисунку 3.5 зображено діаграму станів для алгоритму завантаження зображень.

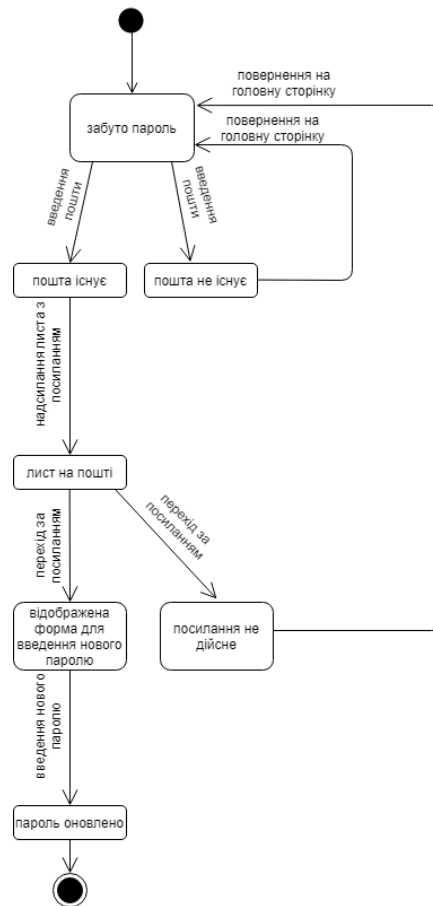


Рисунок 3.4 – Діаграма станів для алгоритму відновлення паролю



Рисунок 3.5 – Діаграма станів для алгоритму завантаження зображень

3.4 Проектування бази даних

Для розробки додатку була використана СКБД PostgreSQL. Основними її перевагами є висока швидкість роботи, розширюваність, підтримка JSON типу, а також декілька вбудованих процедурних мов, що дозволяють створювати процедури не лише з використанням SQL, а й, наприклад, Python. На рисунку 3.6 зображено концептуальну модель бази даних.

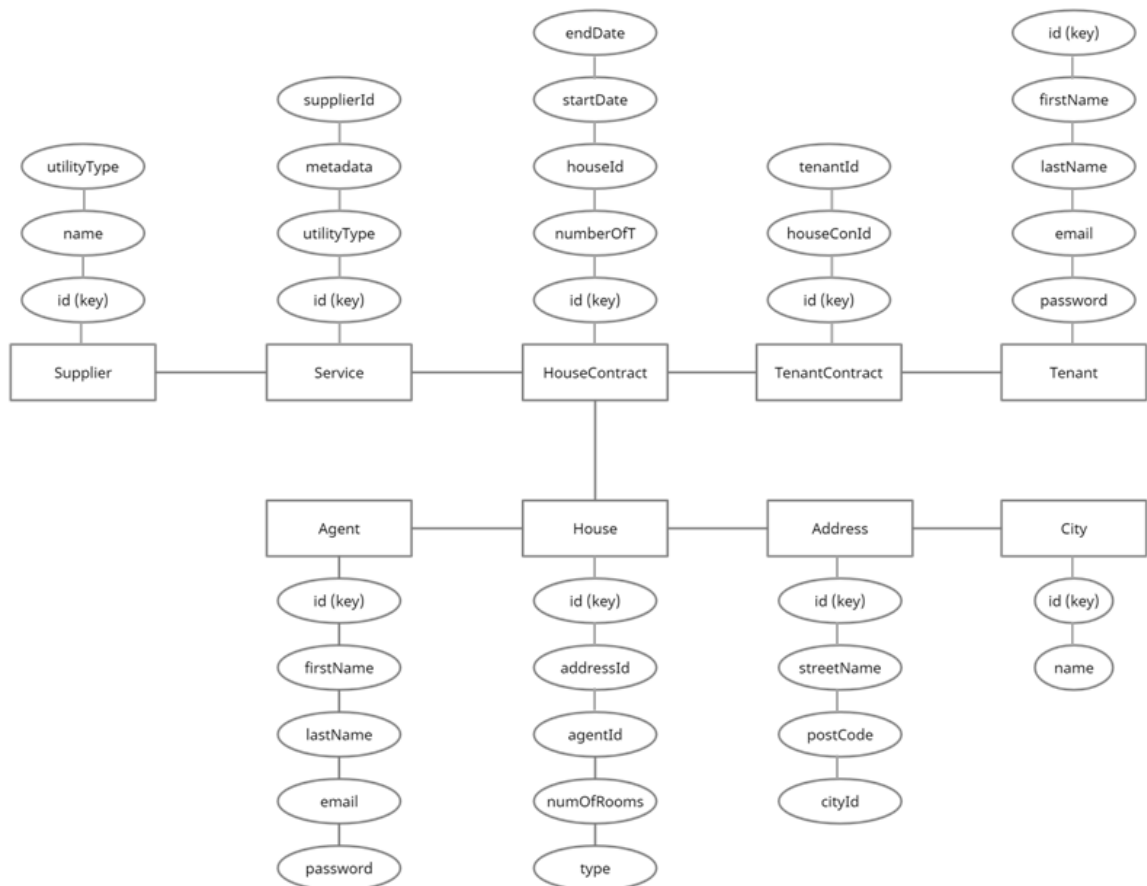


Рисунок 3.6 – Концептуальна модель бази даних

Сутність House (*id*, *addressId*, *agentId*, *numberOfRooms*, *type*) – зберігає дані про будинки. Опис полів сутності та їх типу наведено у таблиці 3.2.

Сутність Agent (*id*, *firstName*, *lastName*, *password*, *email*) – зберігає дані про орендодавців. Опис полів сутності та їх типу наведено у таблиці 3.3.

Таблиця 3.2 – Структура таблиці «House»

Назва поля	Тип поля	Розмір поля	Опис
id	Лічильник	Довге ціле	Призначене для унікальності записів у таблиці
addresId	Числовий	Довге ціле	Зберігає ідентифікатор адреси
agentId	Числовий	Довге ціле	Зберігає ідентифікатор орендодавця
numberOfRooms	Числовий	Довге ціле	Зберігає кількість кімнат
type	Текстовий	10	Зберігає тип будинку – будинок, квартира

Таблиця 3.3 – Структура таблиці «Agent»

Назва поля	Тип поля	Розмір поля	Опис
id	Лічильник	Довге ціле	Призначене для унікальності записів у таблиці
firstName	Текстовий	20	Зберігає ім'я
lastName	Текстовий	20	Зберігає прізвище
password	Текстовий	50	Зберігає хеш паролю
email	Текстовий	50	Зберігає електронну адресу

Сутність Address (id, streetName, postCode, cityId) – зберігає дані про усі адреси в системі. Опис полів сутності та їх типу наведено у таблиці 3.4.

Сутність City (id,name) – зберігає дані про усі міста в системі. Опис полів сутності та їх типу наведено у таблиці 3.5.

Сутність Tenant (id,firstName, lastName,password, email) – зберігає дані про орендарів. Опис полів сутності та їх типу наведено у таблиці 3.6.

Таблиця 3.4 – Структура таблиці «Address»

Назва поля	Тип поля	Розмір поля	Опис
id	Лічильник	Довге ціле	Призначене для унікальності записів у таблиці
streetName	Текстовий	20	Зберігає назву вулиці та будинок
postCode	Текстовий	10	Зберігає поштовий індекс
cityId	Числовий	Довге ціле	Зберігає ідентифікатор міста

Таблиця 3.5 – Структура таблиці «City»

Назва поля	Тип поля	Розмір поля	Опис
id	Лічильник	Довге ціле	Призначене для унікальності записів у таблиці
name	Текстовий	20	Зберігає назву міста

Таблиця 3.6 – Структура таблиці «Tenant»

Назва поля	Тип поля	Розмір поля	Опис
id	Лічильник	Довге ціле	Призначене для унікальності записів у таблиці
firstName	Текстовий	20	Зберігає ім'я
lastName	Текстовий	20	Зберігає прізвище
password	Текстовий	50	Зберігає хеш паролю
email	Текстовий	50	Зберігає електронну адресу

Сутність TenantContract (id,houseContractId, tenantId,startDate, endDate)

– зберігає дані про контракти орендарів. Реалізує зв'язок «багато до

багатьох» між будинками та орендарями. Опис полів сутності та їх типу наведено у таблиці 3.7.

Таблиця 3.7 – Структура таблиці «TenantContract»

Назва поля	Тип поля	Розмір поля	Опис
id	Лічильник	Довге ціле	Призначене для унікальності записів у таблиці
houseContractId	Числовий	Довге ціле	Зберігає ідентифікатор контракту на будинок
tenantId	Числовий	Довге ціле	Зберігає ідентифікатор орендаря
startDate	Дата		Зберігає дату початку контракту
endDate	Дата		Зберігає дату кінця контракту

Сутність HouseContract (id,numberOfTenants, houseId,startDate, endDate) – зберігає дані про контракти будинків. Опис полів сутності та їх типу наведено у таблиці 3.8.

Сутність Service (id,utilityType, metadata,supplierId) – зберігає дані про сервіси у будинку. Опис полів сутності та їх типу наведено у таблиці 3.9.

Таблиця 3.8 – Структура таблиці «HouseContract»

Назва поля	Тип поля	Розмір поля	Опис
1	2	3	4
id	Лічильник	Довге ціле	Призначене для унікальності записів у таблиці

Продовження таблиці 3.8

1	2	3	4
numberOfTenants	Числовий	Довге ціле	Зберігає максимальну кількість орендарів для цього контракту
houseId	Числовий	Довге ціле	Зберігає ідентифікатор будинку
startDate	Дата		Зберігає дату початку контракту
endDate	Дата		Зберігає дату кінця контракту

Таблиця 3.9 – Структура таблиці «Service»

Назва поля	Тип поля	Розмір поля	Опис
id	Лічильник	Довге ціле	Призначене для унікальності записів у таблиці
utilityType	Текстовий	15	Зберігає тип сервісу
metadata	JSON		Зберігає додаткові дані про сервіс
supplierId	Числовий	Довге ціле	Зберігає ідентифікатор постачальника

Сутність Supplier (id,utilityType, metadata,supplierId) – зберігає дані про постачальників. Опис полів сутності та їх типу наведено у таблиці 3.10.

Таблиця 3.10 – Структура таблиці «Supplier»

Назва поля	Тип поля	Розмір поля	Опис
id	Лічильник	Довге ціле	Призначене для унікальності записів у таблиці
name	Текстовий	40	Зберігає назву постачальника
utilityType	Текстовий	15	Зберігає тип сервісу

3.5 Проєктування дизайну

Інтерфейсу додатка була приділена велика увага, адже користувачу повинно бути зручно і приємно працювати з сайтом. Для побудови інтерфейсу були використані стандартні HTML– компоненти, такі як кнопки, списки, блоки, таблиці і т.д. Також, були використані вже готові рішення для бібліотеки React.JS, що дозволяють ефективно використовувати елементи, що повторюються безліч разів.

3.6 Висновки за розділом 3

У третьому розділі було описано архітектуру застосунку, наведено основні діаграми, що стосуються проєктування, зображено концептуальну схему бази даних та описано її сутності.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Опис застосунку

Функціональне призначення застосунку полягає у спрощенні пошуку житла для оренди, а також ефективного управління ним.

4.2 Умови виконання програми

Для нормального функціонування застосунку, повинні виконуватися наступні умови до апаратної частини сервера:

- операційна система: Linux;
- процесор з двома ядрами і з тактовою частотою не менше 2 ГГц;
- оперативна пам'ять: 8 Гбайт;
- встановлене програмне забезпечення Docker;
- дисковий простір: SSD 30Гбайт.

Для нормального функціонування застосунку, повинні виконуватися наступні умови до апаратної частини клієнта:

- браузер Google Chrome, Mozilla Firefox. Opera, Safari або Edge;
- стабільне підключення до мережі Інтернет.

4.3 Інструкція по експлуатації програми

4.3.1 Характеристики програми

Застосунок виконує наступні функції:

- реєстрація та авторизація користувачів в системи;
- додавання житла та перегляд інформації про нього орендодавцем;
- пошук житла орендарем;
- додавання показів лічильників;

— генерація комунальних платіжок.

4.3.2 Використані технічні засоби

При розробці була використана бібліотека Webpack. Це пакувальник модулів, який призначений для застосунків, розроблених з використанням мови JavaScript. Основна задача пакувальника полягає у тому, щоб побудувати коректний граф залежностей усіх модулів системи і згенерувати єдиний файл, який може бути завантажений у HTML файл. Також Webpack може завантажувати й інші файли, такі як зображення, CSS, шрифти, файли і т.д. за допомогою спеціальних завантажувачів. Для роботи з бібліотекою необхідно створити файл конфігурації, в якому вказати точку входу (основний JavaScript файл), точку виходу та додати необхідні плагіни і завантажувачі. Webpack дозволяє розробникам створювати власні плагіни та завантажувачі, що робить його дуже розширюваним [13].

4.3.3 Звернення до програми

Для роботи з клієнтської частини застосунку необхідно перейти за адресою, яку було налаштовано при розгортанні.

Для розгортання серверної частини краще за все підходить AWS. Потрібно створити серверну машину AWS EC2, яка буде підпадати під вказані апаратні вимоги. Далі потрібно налаштувати БД, з використанням наприклад AWS RDS. Так як додаток зберігає дані на AWS S3, то потрібно створити «бакет» і отримати персональні ключі, які потрібно вказати в конфігураційному файлі додатку. Після цього потрібно перейти на серверну машину, завантажити додаток з репозиторію, встановити Docker [14] та створити контейнер. Після створення контейнеру потрібно увійти в нього і запустити додаток [15].

4.3.4 Вхідні й вихідні дані

Вхідними даними до програми є база даних, що повинна знаходитись на сервісі, який підтримує роботу з PostgreSQL, наприклад AWS RDS.

Вихідними даними програми є HTML – сторінка.

4.4 Методика та результати тестування

Тестування буде проведено за допомогою «димового тестування». Димове тестування – це мінімальний набір тестів на явні помилки програмного забезпечення. Димове тестування дозволяє перевірити базовий функціонал системи і його готовність до релізу [16]. Тестування буде проведено за таким сценарієм:

- перевірка авторизації;
- перевірка реєстрації;
- перевірка завантаження зображень;;
- перевірка додавання житла для оренди;
- перевірка відображення базової інформації про житло.

Тестування буде проводитись на ПК з такими характеристиками:

- операційна система Linux, дистрибутив Ubuntu 20.04;
- процесор Intel Pentium 4415U;
- відеокарта Nvidia GeForce MX110;
- оперативна пам'ять 12 Гб;
- програмне забезпечення Docker 18.09.5;
- браузер Google Chrome 101.

4.4.1 Тестування функціоналу

Для роботи з за стосунком необхідно створити нового користувача, для прикладу створимо акаунт орендодавця. Для цього натиснемо кнопку

Signup, що розташована у навігаційній панелі та оберемо варіант реєстрації, у нашому випадку As agent. Після чого відкриється форма реєстрація, на якій потрібно заповнити усі обов’язкові поля на всіх кроках. Для перевірки завантаження зображень спробуємо додати зображення у форму першого кроку реєстрації. При успішному завантаженні зображення воно з’явиться в зменшеному розмірі у формі, що відображено на рисунку 4.1. У разі успішного створення користувача на останньому кроці реєстрації буде відображено повідомлення, зображене на рисунку 4.2.

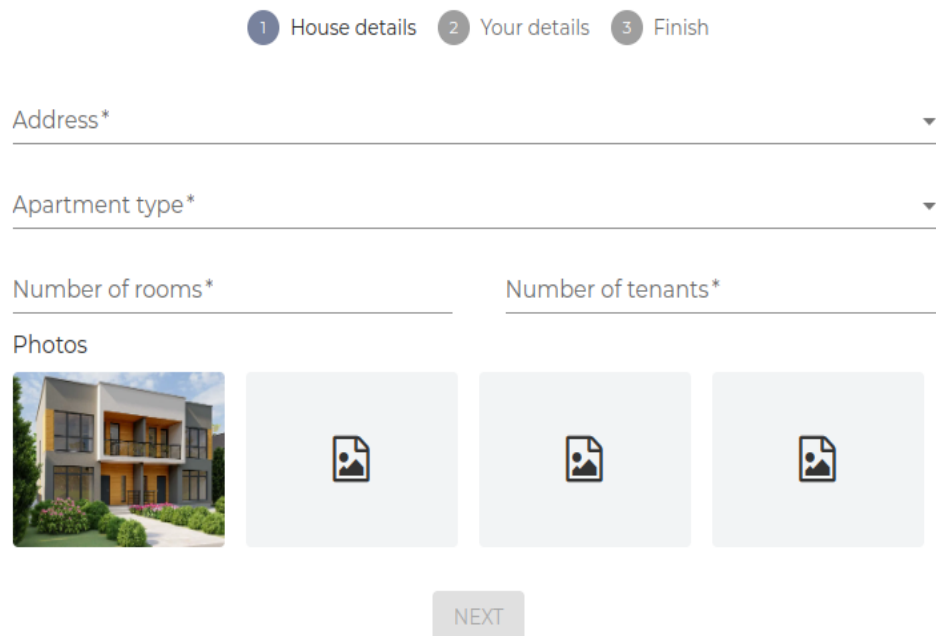


Рисунок 4.1 – Вигляд форми при успішному завантаженні зображення

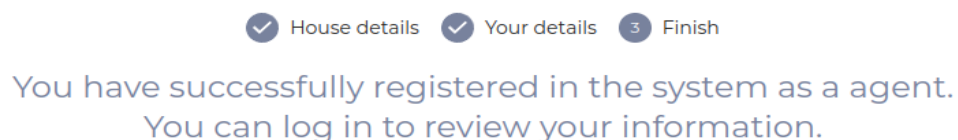


Рисунок 4.2 – Повідомлення про створення нового користувача

Далі виконаємо вхід в систему. Для цього натиснемо кнопку Login у навігаційній панелі та оберемо варіант порталу, в нашому випадку Agent portal. Далі потрібно ввести дані користувача, які було вказано при реєстрації. Порядок роботи системи авторизації зображено на рисунку 4.3. При успішному вході в систему користувача буде автоматично перенаправлено на домашню сторінку.

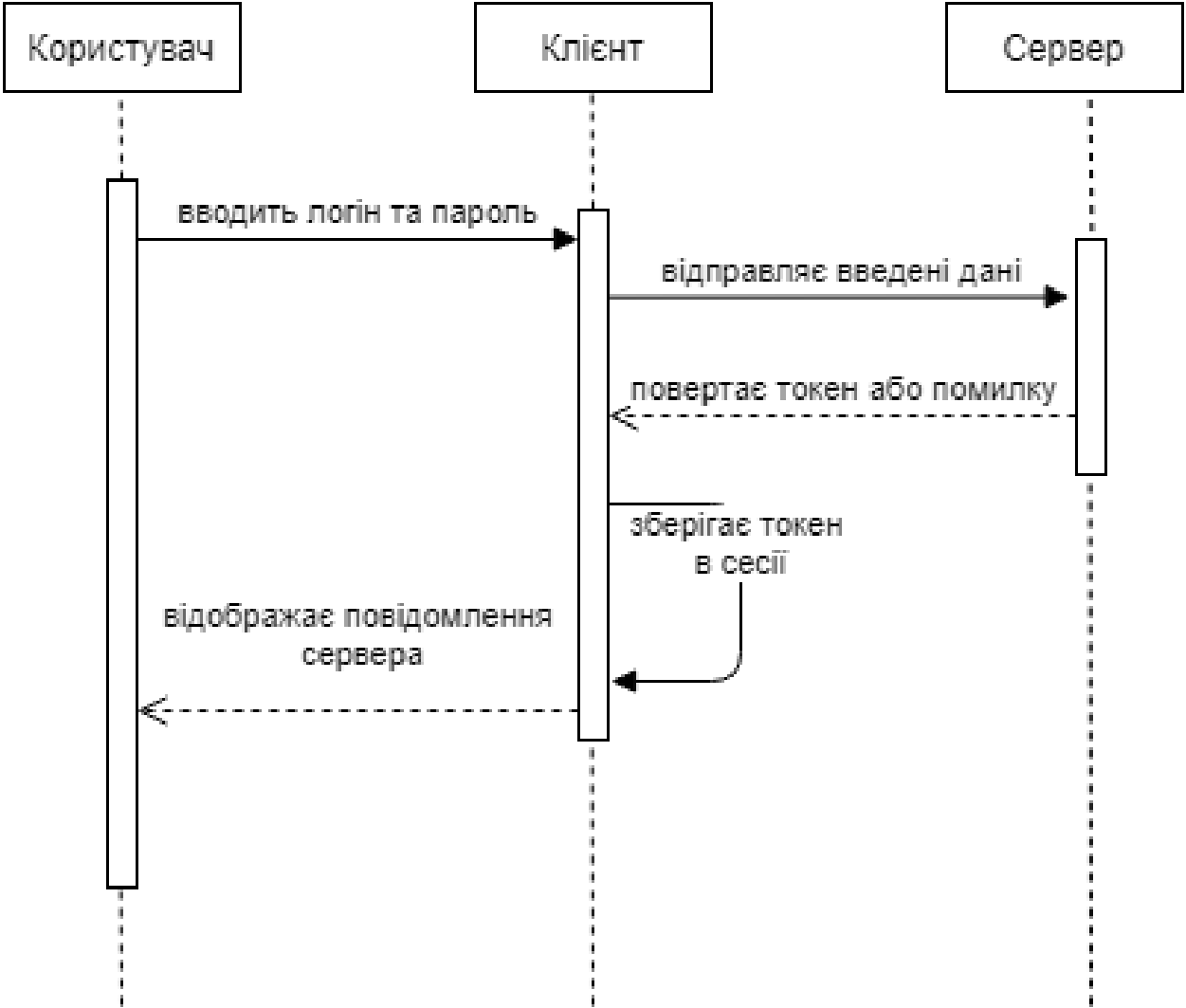


Рисунок 4.3 – Діаграма станів для алгоритму авторизації

Загальний вигляд домашньої сторінки порталу орендодавця зображено на рисунку 4.4.

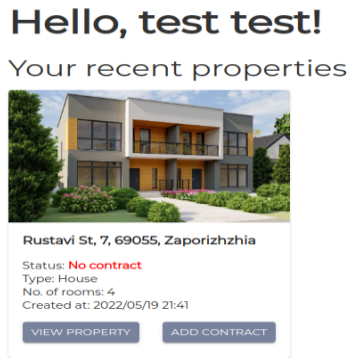


Рисунок 4.4 – Домашня сторінка порталу орендодавця

Далі спробуємо додати новий контракт для створеного будинку. Для цього натиснемо кнопку Add contract. При натисканні кнопки буде відображено модальне вікно, загальний вигляд якого зображено на рисунку 4.5. Для створення контракту необхідно заповнити всі обов'язкові поля форми.

Для перегляду інформації про будинок необхідно натиснути кнопку View property, при цьому буде відкрито нове вікно, яке зображено на рисунку 4.6.

Рисунок 4.5 – Форма додавання нового контракту

Rustavi St, 7, 69055, Zaporizhzhia



Status: **No contract**
 Type: House
 No. of rooms: 4
 Created at: 2022/05/19 21:41

Property information

CONTRACTS TENANTS SERVICES

ID	Type	Number of Tenants	Start Date	End Date	Monthly cost
Empty data					

Rows per page: 5 ▾ 0-0 of 0 < >

Рисунок 4.6 – Вікно інформації про будинок

4.4.2 Тестування верстки

Для тестування верстки був використаний Chrome DevTools, який дозволяє переглядати сайт при різній роздільній здатності.

Загальний вигляд форми реєстрації на iPhone XR зображено на рисунку 4.7.

Домашня сторінка на iPad Air зображена на рисунку 4.8.

RentIt! Login Sign Up

1 House details 2 Your details 3 Finish

Address*

Apartment type*

Number of rooms* Number of tenants*

Photos

Main

NEXT

RentIt! Created by Vladyslav Perehuda © 2022

Рисунок 4.7 – Вигляд форми реєстрації на iPhone XR

RentIt! My account Logout Sign Up

Hello, test test!

Your recent properties

Zvenigorodska St, 2, 69093, Zaporizhzhia

Status: Empty
Type: House
No. of rooms: 3
Created at: 2022/05/09 23:23

VIEW PROPERTY ADD CONTRACT

Rustavi St, 7A, 69000, Zaporizhzhia

Status: Tenanted
Type: House
No. of rooms: 3

Рисунок 4.8 – Вигляд домашньої сторінки на iPad Air

Таблиці інформації про будинок на Google Pixel зображено на рисунку 4.9.

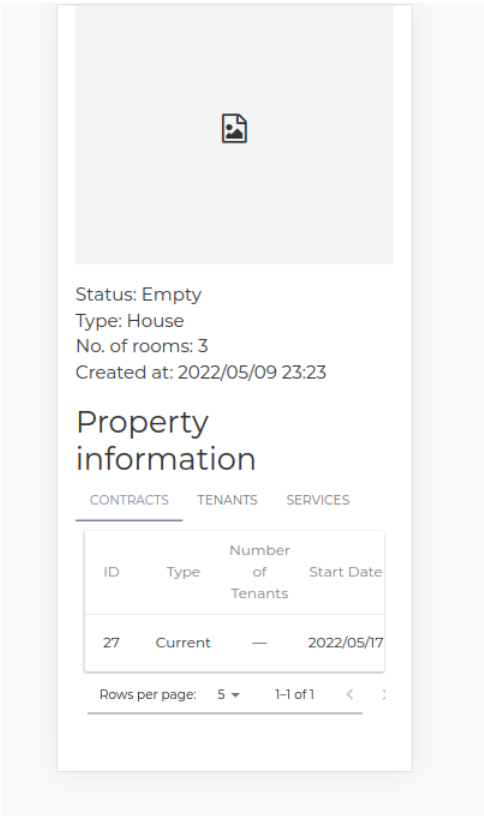


Рисунок 4.9 – Вигляд таблиць інформації про будинок на Google Pixel

4.4.3 Чек-листи тестування

Після проведення тестування було сформовано чек-лист з результатами, який зображено у таблиці 4.1.

Таблиця 4.1 – Чек-лист для тестування функціоналу додатку

Функціонал	Linux		Windows
	Google Chrome	Mozilla Firefox	Google Chrome
1	2	3	4
Авторизація	Пройдено	Пройдено	Пройдено
Реєстрація	Пройдено	Пройдено	Пройдено
Завантаження зображень	Пройдено	Пройдено	Пройдено

Продовження таблиці 4.1

1	2	3	4
Додавання житла	Пройдено	Пройдено	Пройдено
Відображення базової інформації про житло	Пройдено	Пройдено	Пройдено
Верстка при різних дозволах екрану	Пройдено	Не пройдено	Пройдено
Сумісність	Пройдено	Пройдено	Пройдено

4.5 Висновки за розділом 4

У 4 розділі було описані основні вимоги для нормального функціонування застосунку, описано вхідні та вихідні дані, надано інструкцію для розгортання додатку, а також проведено тестування базового функціоналу і верстки.

ВИСНОВКИ

В результаті виконання дипломної кваліфікаційної роботи бакалавра було розроблено застосунок, який призначений для спрощення оренди житла.

Під час роботи з першим розділом було детально проаналізовано предметну область, доведено актуальність роботи, а також було розглянуто сайти-аналоги з порівнянням їх з розроблюваним за стосунком для з'ясування доцільності розробки.

У другому розділі було розглянуто мову програмування, описано її особливості і проведено її порівняння з іншими мовами програмування. Також було розглянуто використані технічні засоби і наведено структурну схему застосунку.

У третьому розділі було зображено основні діаграми для розроблюваного застосунку, описано модулі, наведено концептуальну схему бази даних, а також описано її сутності.

При тестуванні додатку не виникло помилок які б призвели до нестабільної або некоректної роботи.

В подальшому можливо модернізувати за стосунок, додавши механізм виплат орендодавцям, розробити алгоритм для підрахунку середнього споживання ресурсів і формувати звіти на основні цих даних.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Оренда житлових приміщень / [Електрон. ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/%D0%9E%D1%80%D0%B5%D0%BD%D0%B4%D0%B0_%D0%B6%D0%B8%D1%82%D0%BB%D0%BE%D0%B2%D0%B8%D1%85_%D0%BF%D1%80%D0%B8%D0%BC%D1%96%D1%89%D0%B5%D0%BD%D1%8C/.
2. Houseshare vs halls / [Electronic resource]. – Access mode: <https://www.matthewsofchester.com/house-share-vs-halls/>.
3. Flat-sharing for students CMS / [Electronic resource]. – Access mode: <https://www.essentialliving.co.uk/blog/flat-sharing-etiquette-101/>.
4. Flanagan D. JavaScript: The Definitive Guide / David Flanagan. – California: O'Reilly Media, 2020. – 706p.
5. JavaScript | MDN / [Электрон. ресурс] – Режим доступа: <https://developer.mozilla.org/ru/docs/Web/JavaScript/>.
6. Пауэрс Ш. Изучаем Node. Переходим на сторону сервера / Санкт-Петербург: Питер Пресс, 2017. – 304 с.
7. How to Master IntelliJ to Boost Your Productivity / [Electronic resource] – Access mode: <https://medium.com/free-code-camp/how-to-master-intellij-to-boost-your-productivity-44b9da20c556/>.
8. Obe R. PostgreSQL. Up & Running / Regina Obe. California: O'Reilly Media, 2017. – 314 p.
9. Мартін Р. Чистий код / Роберт Мартін. Харків: Фабула, 2019. – 448 с.
10. Клиент-серверная архитектура в картинках / [Электрон. ресурс] – Режим доступа: <https://habr.com/ru/post/495698/>.
11. Многопоточность на Node.js. Eventloop / [Электрон. ресурс] – Режим доступа: <https://habr.com/ru/post/479062/>.

12. Subramarian V. Pro PERN Stack / Vasan Subramarian. – London: Apress, 2017. – 32 с.
13. Основы Webpack: Часть 1 / [Электрон. ресурс] – Режим доступа: <https://codeguida.com/post/454>.
14. Docker / [Electronic resource] – Access mode: <https://www.docker.com/>.
15. Deploy your app / [Electronic resource] – Access mode: <https://docs.docker.com/language/java/deploy/>.
16. Culberston R., Brown C., Cobb G. Rapid Testing / Rober Culberston, Chris Brown, Gary Cobb. – Hoboken: Prentice Hall PTR, 2002. – 394 p.

17.

ДОДАТОК А
Технічне завдання

Вступ

Назва програми – «Програмна реалізація системи для менеджменту колективної оренди житла». Система призначена для спрощення пошуку і подальшої оренди житла серед студентів.

A.1 Підстава для розробки

Підставою для розробки слугує завдання для дипломної кваліфікаційної роботи на тему «Програмна реалізація системи для менеджменту колективної оренди житла», затверджене наказом Національного університету «Запорізька політехніка» № 102 від 27 квітня 2022 р.

A.2 Призначення розробки

Даний проєкт у першу чергу призначений для студентів і дозволяє спростити пошук та оренду житла. Створений ресурс дозволить комутувати комунальні платежі в одному місці, а також спрощує взаємозв'язок між орендодавцем та орендарем. За допомогою цього проєкту студенти зможуть колективно винаймати житло, не думаючи про рівний поділ платежів – система буде робити це автоматично.

A.3 Основні вимоги до програми, що розробляється

A.3.1 Вимоги до функціональних характеристик

Робота із застосунком–орендар:

- реєстрація та авторизація;
- пошук житла;

- здійснення платежів в системі;
- передавання показів лічильників;
- перегляд історії платежів та показів лічильників;
- перегляд комунальних «платіжок»;
- перегляд та коригування особистої інформації.

Редагування сайту – адміністратор:

- перегляд усіх користувачів в системі;
- перегляд усіх будинків;
- перегляд історії платежів та показів лічильників;
- формування комунальних «платіжок» за певний період та їх

розсилка.

Робота з сайтом – орендодавець:

- реєстрація та авторизація;
- додавання будинків в систему;
- перегляд доданих будинків;
- перегляд комунальних «платіжок» та показів лічильників;
- перегляд графіків використання ресурсів.

A.3.2 Вимоги до надійності

Вся інформація про користувачів системи буде зберігатись у базі даних AWSRDS, файли будуть розміщуватись на хмарному сховищі AWS S3, додаток буде розгорнутий у контейнері Docker з використанням сервісу AWS EC2.

Захист БД забезпечується за допомогою приватної мережі, тобто підключення до бази даних можливе лише користувачами та сервісами, які мають доступ до цієї мережі.

A.3.3 Умови експлуатації

Код програми буде розміщений у приватному репозиторії GitLab і може бути наданий певному колу людей.

Експлуатація програми здійснюється згідно експлуатаційної документації на розроблену програму, яка відповідає стандартам і містить інформацію, необхідну для її освоєння та експлуатації.

A.3.4 Вимоги до складу та параметрів технічний засобів

Мінімальні системні вимоги та програмне забезпечення, що необхідні для функціонування серверної частини мобільного застосунку:

- операційна система: Linux;
- процесор з двома ядрами і з тактовою частотою не менше 2 ГГц;
- оперативна пам'ять: 8 Гбайт;
- встановлене програмне забезпечення Docker;
- дисковий простір: SSD 30Гбайт.

Вимоги для клієнтської сторони: для роботи з сервісом потрібно мати сучасний браузер (Google Chrome, Mozilla Firefox, Safari) та стабільне підключення до мережі Інтернет.

A.3.5 Вимоги до маркування і пакування

Програмний код буде розміщений у репозиторії GitLab і може бути розповсюджений через цей сервіс.

A.3.6 Вимоги до транспортування й зберігання

Вимоги до транспортування й зберігання аналогічні вимогам, пропонованим до носія, на якому записана програма.

A.3.7 Вимоги до програмної документації

Програмний продукт повинен поставлятися у наступному складі:

- технічне завдання;
- опис програми;
- текст програми.

A.4 Стадії та етапи розробки

Виділяють такі етапи в розробці програмного забезпечення:

- формування вимог;
- проектування;
- реалізація;
- тестування;
- впровадження;
- експлуатація та підтримка.

A.5 Порядок контролю та приймання системи

Приймання системи здійснюється відповідно до процедури проходження дипломування.

Під час тестування розглядають дві основні стратегії: тестування функціоналу та тестування верстки. Звітність з результатів тестування оформлюється у вигляді скріншотів та заповнених чек-листів ходу тестування.

Для тестування функціональності за стосунку обов'язково наводиться діаграм моделювання логіки взаємодії користувача із системою – діаграма послідовностей з набору діаграм UML. Також окремо наводиться опис

апаратного (та за використання особливого програмного) забезпечення, яке використовувалося під час тестування.

Тестування верстки відбувається шляхом розгортання та перегляду дизайну за стосунку на різних апаратних платформах. Тому у якості звіту можуть використовуватися скріншоти з реальних пристроїв або емуляторів, що засвідчать роботу за стосунку.

Звіт про проведену роботу направляється керівникові дипломної кваліфікаційної роботи бакалавра д.т.н., професору Олійнику А.О. Нормоконтроль за оформленням пояснювальної записки виконує асистент Липовець М.В. Представлення та захист результатів роботи виконується в присутності екзаменаційної комісії.

Вимоги до оформлення проміжних звітів і підсумкового звіту узгоджуються із науковим керівником дипломної кваліфікаційної роботи бакалавра, нормо контролером та завідуючим кафедрою програмних засобів.

ДОДАТОК Б
Опис програми

Б.1 Загальні відомості

Тема дипломної кваліфікаційної роботи бакалавра -«Програмна реалізація системи для менеджменту колективної оренди житла». Метою роботи є розробка застосунку для спрощення пошуку та менеджменту житла для оренди. Він забезпечує пошук житла для подальшої колективної оренди, а також надає можливість зручно і ефективно управляти житлом орендодавцями.

Б.2 Функціональне призначення

Система повинна реалізувати такі функціональні вимоги:

- реєстрація та авторизація в системі;
- пошук будинків та їх фільтрація;
- перегляд інформації користувача;
- додавання нових будинків орендодавцями;
- перегляд історії платежів та показів лічильників;
- формування комунальних «платіжок»;
- передавання показів лічильників орендарями через систему.

Б.3.1 Алгоритм авторизації

Авторизація у за стосунку відбувається з використанням JWT токенів. При вході в систему користувач має вказати електронну адресу та пароль. Далі відбувається пошук користувача у БД за вказаною поштою і здійснюється порівняння введеного паролю у вигляді хешу. При успішній авторизації сервер генерує JWT токен і надсилає його клієнту, який зберігає його у браузерному сховищі. При кожному запиті на сервер відправляється токен і сервер перевіряє його валідність.

Б.3 Опис логічної структури програми

Структурна схема застосунку зображена на рисунку Б.1.

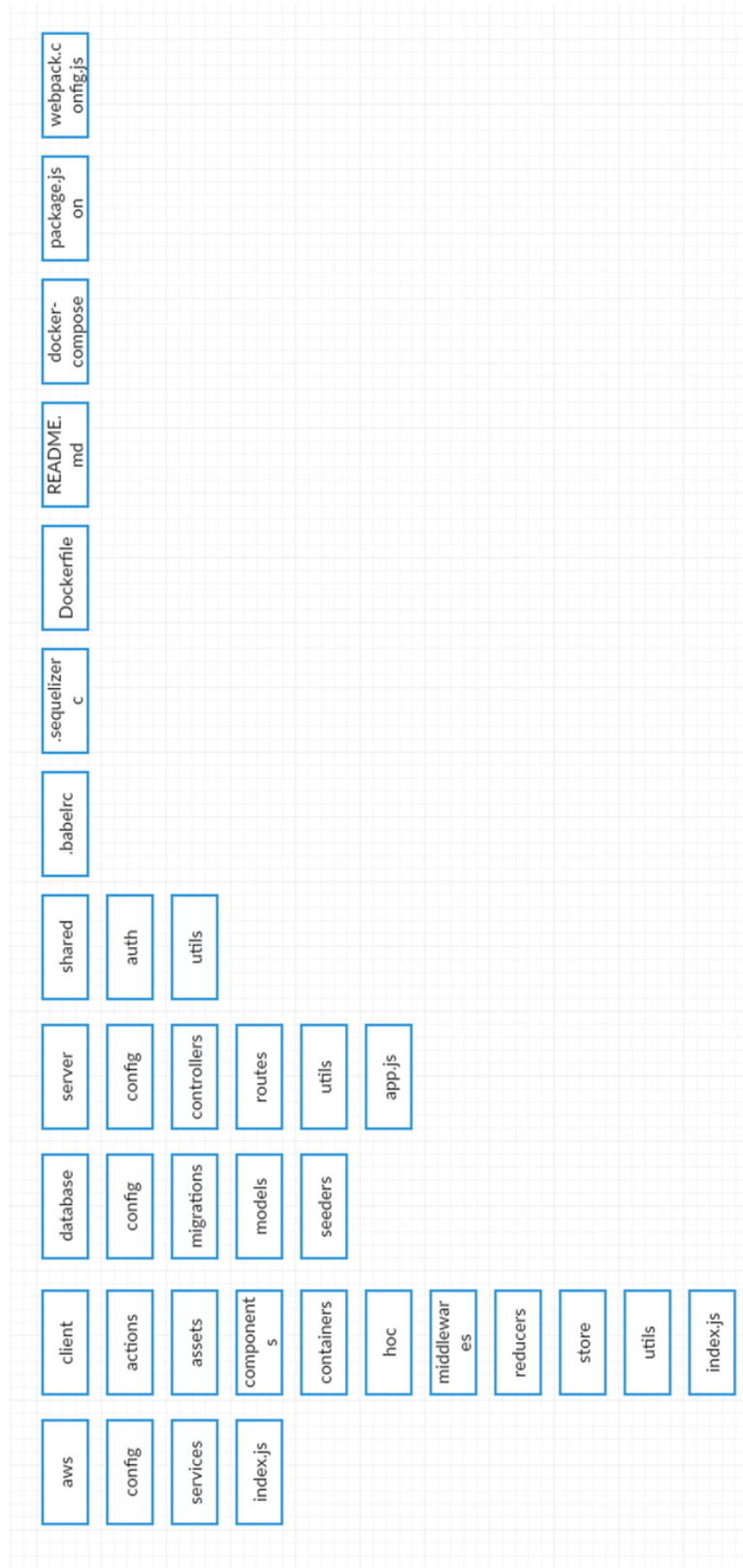


Рисунок Б.1 – Структурна схема програмного продукту

Б.3.2 Алгоритм реєстрації

Алгоритм авторизації використовується для створення нових користувачів у системі. Реєстрація відбувається за електронною поштою і паролем, який повинен бути довжиною не менше 8 символів і латинських літер, містити у собі хоча б один символ і латинську літеру у верхньому регістрі. Після успішної реєстрації користувач буде мати змогу увійти в систему за вказаними даними.

Б.3.3 Алгоритм завантаження зображень

Для збереження користувацьких зображень був використаний сервіс AWS S3. Цей сервіс зберігає файли у «бакеті», кожен файл повинен мати свій унікальний ключ. При завантаженні зображень відбувається конвертація їх у масив байтів, а також формується ключ зображення. Після цього зображення завантажуються у сховище, а його ключ зберігається таблиці Upload. Отримати зображення можна за ключем.

Б.4 Використані технічні засоби

Застосунок було розроблено з використанням мови JavaScript. Цю мову було обрано за простоту використання та легкість вивчення, також сучасні технології дозволяють використовувати цю мову як для написання серверної частини, так і клієнтської. Також, для розробки була використана бібліотека React.JS, платформа Node.JS та СКБД PostgreSQL. Для побудови готово «бандлу» була використана бібліотека Webpack.

Б.5 Виклик і завантаження

Для роботи з додатком необхідно перейти за адресою, яку було налаштовано при розгортанні застосунку.

Для розгортання серверної частини краще за все підходить AWS. Потрібно створити серверну машину AWS EC2, яка буде підпадати під вказані апаратні вимоги. Далі потрібно налаштувати БД, з використанням наприклад AWS RDS. Так як додаток зберігає дані на AWS S3, то потрібно створити «бакет» і отримати персональні ключі, які потрібно вказати в конфігураційному файлі додатку. Після цього потрібно перейти на серверну машину, завантажити додаток з репозиторію, встановити Docker та створити контейнер. Після створення контейнеру потрібно увійти в нього і запустити додаток.

Б.6 Початкові дані

При запуску додатку структура БД автоматично формується і оновлюється за допомогою файлів-міграцій.

Вихідними даними є HTML-сторінки з даними.

ДОДАТОК В**Код програми**

```

src/server/index.js
import express from 'express';
import bodyParser from 'body-parser';
import helmet from 'helmet';
import { authMiddleware } from 'lib/auth';
import db from 'server/db';
import cors from 'cors';

import apiV1 from './api/v1';

class Server {
  app = express();

  setupHelmet(app) {
    app.use(helmet.frameguard());
    app.use(helmet.hidePoweredBy());
    app.use(helmet.xssFilter());
    app.use(helmet.hsts());
    app.use(helmet.ieNoOpen());
    app.use(helmet.noSniff());

    app.use(cors({ origin: '*' }));
  }

  setupBodyParser(app) {
    app.use(bodyParser.urlencoded({ limit: '50mb', extended: true }));
    app.use(bodyParser.json({ limit: '50mb' }));
  }

  setupRoutes(app) {
    const router = express.Router();

    router.use(authMiddleware());
    router.use('/api/v1', apiV1);

    router.get('*', (req, res, next) => {
      if (!req.rewritten) {
        req.rewritten = true;
        req.url = req.url.startsWith('/admin') ? '/admin/index.html' : '/index.html';
        app.handle(req, res, next);
      } else {
        next();
      }
    });

    app.use(router);
  }

  setupStatic(app) {
    app.use(express.static('dist/client'));
  }

  async dbMigrate() {
    await db.migrator.migrate();

    console.log('migration completed');
  }

  async bootstrap() {
    await this.dbMigrate();
    this.setupHelmet(this.app);
    this.setupBodyParser(this.app);
    this.setupStatic(this.app);
  }
}

```

```

this.setupRoutes(this.app);

this.app.listen(3001,()=>console.log('Server started'));
}
}

newServer().bootstrap();

src/server/db.js
import{ Pool}from'pg';
importPostgratorfrom'postgrator'
importconfigfrom'shared/config/server';
import _from'lodash';

classDatabaseAPI{
static_instance;

staticgetInstance(){
returnDatabaseAPI._instance=DatabaseAPI._instance||newDatabaseAPI();
}

getpool(){
if(!this._pool){
this._pool=newPool(config.db);
}

returnthis._pool;
}

getmigrator(){
if(!this._migrator){
this._migrator=newPostgrator(config.db);
this._migrator.on('validation-started',(m)=>console.info(` validating ${m.filename}`));
this._migrator.on('migration-started',(m)=>console.info(` running ${m.filename}`));
}

returnthis._migrator;
}

asyncopenTransaction(){
constclient=awaitthis.pool.connect();

awaitclient.query('BEGIN');

client.commit=()=>client.query('COMMIT');
client.rollback=()=>client.query('ROLLBACK');

returnclient;
}

asyncauth(role,email,pass){
constdata=awaitthis.pool.query('SELECT auth($1, $2, $3)',[role,email,pass]);

return_.get(data,'rows.0.auth');
}

asyncprocessSignUp(data,transaction=this.pool){
constqueryResult=awaittransaction.query('SELECT process_sign_up($1)',[data]);

return_.get(queryResult,'rows.0.process_sign_up');
}

asyncisPossibleAddress(data){

```

```

return await this.pool.query('SELECT * FROM is_possible_address($1)', [data]);
}

async getCurrentUser({userRole, userId}){
const data = await this.pool.query('SELECT get_current_user($1, $2)', [userRole, userId]);

return _.get(data, 'rows.0.get_current_user');
}

async getAgentProperties(filters = {}){
const data = await this.pool.query('SELECT * FROM filter_agent_properties($1)', [filters]);

return _.get(data, 'rows');
}

async createUpload(data, transaction = this.pool){
return await transaction.query(`
    INSERT INTO "Upload" (bucket, url, type, "houseId", metadata, "mimeType")
    VALUES ($1, $2, $3, $4, $5, $6)
`, [data.bucket, data.url, data.type, data.houseId, data.metadata, data.mimeType]);
}

async getUploadByKey(key){
const data = await this.pool.query(`
    SELECT *
    FROM "Upload"
    WHERE url = $1
`, [key]);

return _.get(data, 'rows.0');
}

async createHouseContract(data, transaction = this.pool){
const queryResult = await transaction.query('SELECT create_house_contract($1)', [data]);

return _.get(queryResult, 'rows.0.create_house_contract');
}

async getPropertyById(id){
const queryResult = await this.pool.query(`
    SELECT *
    FROM "AgentPropertiesView"
    WHERE id = $1
`, [id]);

return _.get(queryResult, 'rows.0');
}

async filterHouseContracts(filters = {}){
const queryResult = await this.pool.query('SELECT * FROM filter_house_contracts($1)', [filters]);

return _.get(queryResult, 'rows');
}

async filterTenantContracts(filters = {}){
const queryResult = await this.pool.query('SELECT * FROM filter_tenant_contracts($1)', [filters]);

return _.get(queryResult, 'rows');
}

async filterServices(filters = {}){
const queryResult = await this.pool.query('SELECT * FROM filter_services($1)', [filters]);

```

```

return_.get(queryResult,'rows');
}

asyncfilterSuppliers(filters={}){
constqueryResult=awaitthis.pool.query('SELECT * FROM filter_suppliers($1)',[filters]);

return_.get(queryResult,'rows');
}
}

exportdefaultDatabaseAPI.getInstance();

src/server/api/v1/auth.js
importexpressfrom'express';
importjwtfrom'jsonwebtoken';
importdbfrom'server/db';
importconfigfrom'shared/config/server';
import{jwtVerify}from'lib/auth';
importmulterfrom'multer';
importawsfrom'lib/aws';
importpathfrom'path';
importdayjsfrom'dayjs';

constupload=multer({
storage:multer.memoryStorage()
});

constrouter=express.Router();
exportdefaultrouter;

router.post('/login',async(req,res)=>{
try{
constjwtData=awaitdb.auth(req.body.role,req.body.email,req.body.password);
constjwtToken=jwt.sign(jwtData,config.jwtSecret,{expiresIn:'1 day'});

res.json({
status:'ok',
token:jwtToken,
});
}catch(err){
if(err.message.indexOf('errors.login.incorrectData')>=0){
returnres.status(401).json({
status:'error',
message:'Incorrect data',
})
}
}

console.error('Auth error: ',err);
res.status(500).json({
status:'error',
message:err.message,
});
}
});

router.get('/check-auth',jwtVerify(),async(req,res)=>{
try{
returnres.json({
status:'ok',
userId:req.auth.userId,
userRole:req.auth.userRole,
});
}catch(err){

```

```

res.status(500).json({
  status:'error',
  message:err.message,
});
}
});

router.post('/signup/tenant',async(req,res)=>{
  try{
    awaitdb.processSignUp({...req.body,signupType:'tenant'});

    res.json({
      status:'ok'
    });
  }catch(err){
    console.error('Signup error: ',err);

    leterrMessage='Internal server error';

    if(err.message.indexOf('Tenant_unique_email')>=0){
      errMessage='Tenant with the specified email already exists!';
    }

    res.status(500).json({
      status:'error',
      message:errMessage,
    });
  }
});

router.post('/signup/agent',upload.fields([ {name:'images'} ]),async(req,res)=>{
  consttransaction=awaitdb.openTransaction();

  try{
    consthouseData=JSON.parse(req.body.houseData);
    constagentData=JSON.parse(req.body.agentData);
    constimages=req.files.images;

    const{houseId}=awaitdb.processSignUp({houseData,agentData,signupType:'agent',transaction})||{};

    for(const[idx,{buffer,mimetype,originalname}]ofimages.entries()){
      constext=path.extname(originalname);
      consttimestamp=dayjs().format('YYYYMMDD_HH:mm');

      const{Key:url,Bucket:bucket}=awaitaws.s3.upload({
        Key:`houses/${houseId}/image_${idx}_${timestamp}${ext}`,
        Body:buffer
      }).promise();

      awaitdb.createUpload({
        url,bucket,type:'HOUSE',houseId,mimeType:mimetype,metadata:{main:idx===0},
      },transaction);
    }

    awaittransaction.commit();

    res.json({
      status:'ok'
    });
  }catch(err){
    awaittransaction.rollback();

    console.error('Signup error: ',err);
  }
});

```

```

let errMessage = 'Internal server error';

if (err.message.indexOf('Agent_unique_email') >= 0) {
  errMessage = 'Agent with the specified email already exists!';
}

res.status(500).json({
  status: 'error',
  message: errMessage,
});
} finally {
  transaction.release();
}
});

router.get('/current-user', jwtVerify(), async (req, res) => {
  try {
    const { userRole, userId } = req.auth;
    const { firstName, lastName, email } = await db.getCurrentUser({ userRole, userId });
  } catch (err) {
    res.status(500).json({
      status: 'error',
      message: err.message,
    });
  }
});

src/client/commin/router.js
import React from 'react';
import { applyMiddleware, combineReducers, createStore } from 'redux';
import { composeWithDevTools } from 'redux-devtools-extension';
import thunk from 'redux-thunk';
import { Provider as ReduxProvider } from 'react-redux';
import { checkLogin } from 'client/reducers/user';
import { ThemeProvider } from '@mui/material/styles';
import { AdapterDayjs } from '@mui/x-date-pickers/AdapterDayjs';
import { LocalizationProvider } from '@mui/x-date-pickers';

import muiTheme from '../muiTheme';
import reducers from '../reducers';

export default class Router extends React.Component {
  state = {
    store: null
  };

  constructor(props, context) {
    super(props, context);

    const store = createStore(
      combineReducers(reducers),
      {},
      composeWithDevTools(applyMiddleware(thunk))
    );

    let tokenCheck = Promise.resolve();

```

```

if(localStorage.getItem('token')){
  tokenCheck=store.dispatch(checkLogin());
}

this.state={store,tokenCheck};
}

getRouter(){
returnnull;
}

render(){
return(
<ReduxProviderstore={this.state.store}>
<ThemeProvidertheme={muiTheme}>
<LocalizationProviderdateAdapter={AdapterDayjs}>
{this.getRouter()}
</LocalizationProvider>
</ThemeProvider>
</ReduxProvider>
)
}
}

src/client/common/ImagePreloader/index.js
importReactfrom'react';
importCircularProgressfrom'@mui/material/CircularProgress';

exportdefaultclassImagePreloaderextendsReact.Component{
state={
loading:false,
error:false,
};

componentDidMount(){
constimageSrc=this.props.src;

if(imageSrc){
this.setState({loading:true});

constimg=newImage();
img.src=imageSrc;
img.onload=()=>this.setState({loading:false,loadedImage:imageSrc});
img.onerror=(err)=>{
console.error('Image loading error: ',err);
this.setState({loading:false,error:true});
};
}
}

renderImage(){
const{loadedImage,loading,error}=this.state;

if(loading){
return(
<CircularProgresssize={40}/>
);
}

if(error){
return(
<className="fa fa-2x fa-chain-broken"aria-hidden="true"/>
)
}
}

```

```

    }

    return(
    <img {...this.props} alt="Image" src={loadedImage}/>
    )
  }

  render(){
    return this.renderImage();
  }
}

src/lib/dec.js
import React from 'react';
import { connect } from 'react-redux';
import _ from 'lodash';

export function withConnect(...args){
  return function(descriptor){
    if(descriptor.kind === 'class'){
      descriptor.finisher = (...fargs) => {
        const Connect = connect(...args)(...fargs);

        return (props) => React.createElement(Connect, props);
      };
    } else if(typeof descriptor === 'function' || typeof descriptor === 'object'){
      return connect(...args)(descriptor);
    } else {
      throw new Error('Unsupported element for decoration!');
    }
  };
}

export function withQuery(query, options){
  return function(descriptor){
    if(descriptor.kind === 'class'){
      descriptor.finisher = (Component) => (
        class extends React.Component {
          state = {
            data: null,
            loading: false,
            error: null,
          };

          isMounted = false;

          updateQueryState(state){
            if(this.isMounted){
              this.setState(state);
            }
          }

          async runQuery(props = this.props){
            try{
              if(this.isSkipped()){
                return;
              }

              const params = (_isFunction(options.params) ? options.params(props) : options.params) || {};

              this.updateQueryState({loading: true});

              const data = await query(params);

```

```

this.isMounted&&this.updateQueryState({data});
} catch(err){
  console.error('Query error: ',err);
  this.updateQueryState({error:err.message});
} finally{
  this.updateQueryState({loading:false});
}
}

componentWillReceiveProps(nextProps){
  if(!_isFunction(options.params)){
    if(!_isEqual(options.params(this.props),options.params(nextProps))){
      this.runQuery(nextProps);
    }
  }
}

asyncComponentDidMount(){
  this.isMounted=true;
  awaitthis.runQuery()
}

componentWillUnmount(){
  this.isMounted=false;
}

getQueryProps(){
  constqueryName=options.name||'data';

  if(this.isSkipped()){
    returnnull;
  }

  return{
    [queryName]:{
      ...this.state,
      refetch:this.runQuery.bind(this)
    }
  };
}

isSkipped(){
  return_isFunction(options.skip)?options.skip(this.props):options.skip;
}

render(){
  return(
    <Component {...this.props} {...this.getQueryProps()} />
  );
}
}
)
} else{
  thrownewError('Unsupported element for decoration!');
}
}
}

src/lib/forms/index.js
import {withFormik,FieldArray} from'formik';
importdayjsfrom'dayjs';
import _from'lodash';

```

```

importFieldfrom'./Field';
importAddressFieldfrom'./AddressField';
importDateFieldfrom'./DateField';
importImageLoaderfrom'./ImageLoader';
importCheckboxFieldfrom'./CheckboxField';

functionwithForm(options){
returnfunction(descriptor){
if(descriptor.kind==='class'){
constdefaultOptions={
validateOnMount:true,
mapPropsToValues:(props)=>({props}),
};

descriptor.finisher=(Component)=>withFormik(_merge(defaultOptions,options))(Component);
}else{
thrownewError('Unsupported element for decoration!');
}
}
}

functionmapDeep(collection,iteratee=_identity,path=[]){
functionpredicate(value,key){
constsubpath=path.concat([key]);
constval=_find([iteratee(value,key,subpath),value],_negate(_isUndefined));

returnval===value?mapDeep(value,iteratee,subpath):val;
}

switch(true){
case_isArray(collection):
return_.map(collection,predicate);
case_isPlainObject(collection):
return_.mapValues(collection,predicate);
default:
returncollection;
}
}

functionval(error,cb,strict=false){
return(v)=>(strict?cb(v):v===null||v===''||cb(v))?null:error;
}

constvalidators={
required:val('Required',(v)=>v!=null&&v!=='',true),
requiredFile:val('Required',(v)=>v&&v.length>0,true),
email:val('Invalid email',(v)=>(
/^[^<>()[\]\\\.,;:\s@"']+(\.[^<>()[\]\\\.,;:\s@""]+)*)(\.[^<>()[\]\\\.,;:\s@""]+)*@((\[[0-9]{1,3}\. [0-9]{1,3}\. [0-9]{1,3}\. [0-9]{1,3}\)|([a-zA-Z\0-9]+\.)+[a-zA-Z]{2,}))$/).test(v)
)),
name:val('Invalid name',(v)=>/^[a-zA-ÿ\s,.'-]+$/i.test(v)),
noPast:val('Date must not be in the past',(v)=>days(v).startOf('day').add(1,'second').isAfter(daysjs().startOf('day'))),
};

functionvalidate(values,schema={}){
return_.omitBy(mapDeep(schema,(validator,key,path)=>{
constvalue=_get(values,path);
leterrors;

switch(true){
case_isFunction(validator):
returnvalidator(value,path);

```

```

case_.isArray(validator):
errors=_(validator)
.map((key)=>_.isFunction(key)?key(value,path):(validators[key])(value,path))
.flatten().filter(_.identity).value();

returnerrors[0]||null;
case_.isObject(validator):
return;
default:
returnnull;
}
}),_.isNil);
}

export{
withForm,
Field,
FieldArray,

validate,
validators,
val,

AddressField,
ImageLoader,
DateField,
CheckboxField,
};

src/shared/conf/server.js
importconfigsfrom'../../conf/server.json';
importcsfrom'pg-connection-string';

constconfig={...configs[process.env.NODE_ENV]};

if(config.db.connectionString){
config.db={
...cs.parse(config.db.connectionString),
...config.db,
};

deleteconfig.db.connectionString;
}

exportdefaultconfig;

src/client/queries/index.js
importaxiosfrom'axios';
import{apiUrl}from'shared/config/client';

exportfunctionunifiedQueryApi(method,url){
if(!(methodinaxios)){
thrownewError('Wrong axios request method');
}

returnasync(options={})=>{
try{
const{data}=awaitaxios({
method,
url:apiUrl(url),
...options
});

```

```

return data;
} catch (err) {
  throw err;
}
}
}
export function secureQueryApi(method, url) {
  return async (options) => unifiedQueryApi(method, url)({
    ...options,
    headers: { Authorization: `Bearer ${localStorage.getItem('token')}` }
  });
}

// Public
export const isPossibleAddress = (params) => unifiedQueryApi('get', '/api/v1/address')({ params });
export const login = (data) => unifiedQueryApi('post', '/api/v1/auth/login')({ data });
export const processSignupTenant = (data) => unifiedQueryApi('post', '/api/v1/auth/signup/tenant')({ data });
export const processSignupAgent = (data) => unifiedQueryApi('post', '/api/v1/auth/signup/agent')({ data });

// Private
export const getAgentProperties = ({ filters }) => secureQueryApi('get', '/api/v1/agent/properties')({ params: filters });
export const createHouseContract = (data) => secureQueryApi('post', '/api/v1/contract')({ data });
export const getPropertyById = ({ id }) => secureQueryApi('get', `/api/v1/property/${id}`)();
export const filterPropertyContracts = ({ propertyId, filters }) => secureQueryApi('get', `/api/v1/property/${propertyId}/contracts`)({ params: filters });
export const filterPropertyTenants = ({ propertyId, filters }) => secureQueryApi('get', `/api/v1/property/${propertyId}/tenants`)({ params: filters });
export const filterPropertyServices = ({ propertyId, filters }) => secureQueryApi('get', `/api/v1/property/${propertyId}/services`)({ params: filters });
export const checkAuth = () => secureQueryApi('get', '/api/v1/auth/check-auth')();
export const getCurrentUser = () => secureQueryApi('get', '/api/v1/auth/current-user')();
export const filterSuppliers = ({ filters }) => secureQueryApi('get', '/api/v1/supplier')({ params: filters });

src/client/reducers/user.js
import { createReducer, createAction } from 'redux-act';
import { createActionAsync } from 'redux-act-async';
import { checkAuth } from 'client/queries';

import { parseJwtToken } from '../utils';

export const checkLogin = createActionAsync('CHECK_AUTH', checkAuth, { noRethrow: true });
export const loginOk = createAction('LOGIN_OK');
export const logout = createAction('LOGOUT_OK');

const defaultState = {
  loggedIn: false,
  userRole: null,
  userId: null,
  token: localStorage.getItem('token'),
};

export default createReducer({
  [checkLogin.ok]: (state, { response }) => {
    const { userRole, userId } = response;

    return {
      ...state,
      loggedIn: true,
      userRole,
      userId,
    };
  },
  [checkLogin.error]: (state) => {
    localStorage.removeItem('token');
  }
});

```

```

return{
...state,
...defaultState
},
[loginOk]:(state,{token})=>{
const{userRole,userId}=parseJwtToken(token);

localStorage.setItem('token',token);

return{
...state,
loggedIn:true,
userRole,
userId,
token,
},
[logout]:(state)=>{
localStorage.removeItem('token');

return{
...state,
...defaultState
},defaultState)

src/client/commin/TableBase/index.js
import React from 'react';
import Table from '@mui/material/Table';
import TableHead from '@mui/material/TableHead';
import TableBody from '@mui/material/TableBody';
import TableRow from '@mui/material/TableRow';
import TableCell from '@mui/material/TableCell';
import TableContainer from '@mui/material/TableContainer';
import TablePagination from '@mui/material/TablePagination';
import Paper from '@mui/material/Paper';
import Typography from '@mui/material/Typography';
import CircularProgress from '@mui/material/CircularProgress';
import Grid from '@mui/material/Grid';
import _ from 'lodash';

import styles from './index.scss';

export default class TableBase extends React.Component {
  getColumns() {
    return [];
  }

  getRows() {
    return [];
  }

  getRowId(row) {
    return row.id;
  }

  isLoading() {
    return false;
  }

  onPerPageChange = (ev) => {
    this.props.updateUI({offset: 0, limit: ev.target.value});
  }

```

```

onPageChange=(ev,page)=>{
  if(ev!=null||!this.isLoading()){
    this.props.updateUI({offset:page});
  }
}

renderCell=(row,name,value)=>{
  return{value};
};

cellRenderer=(row,columnName)=>{
  constvalue=row[columnName];
  constcell=this.renderCell(row,columnName,value);

  if(React.isValidElement(cell)){
    returncell;

  }else{
    const{value:rowValue,className}={value,...cell};
    constkey=`${this.getRowId(row)}-${columnName}`;

    return(
      <TableCellkey={key}align="center"data-multilineclassName={className}>
        {_.isNil(rowValue)?'':rowValue}
      </TableCell>
    );
  }
};

renderColumns(){
  return(
    <TableRow>
      {
        _.map(this.columns,(column)=>(
          <TableCellkey={column}align="center">{column}</TableCell>
        ))
      }
    </TableRow>
  )
}

renderCells(){
  if(this.isLoading()){
    returnnull;
  }

  return_.map(this.rows,(row)=>(
    <TableRowkey={this.getRowId(row)}>
      {_.map(this.columns,(title,key)=>this.cellRenderer(row,key))}
    </TableRow>
  ));
}

renderLoadingOrEmpty(){
  constrows=this.rows;

  if(this.isLoading()){
    return(
      <GriditemcontaineralignItems="center"justifyContent="center"padding="20px 0">
        <CircularProgresssize={20}/>
      </Grid>
    )
  }
}

```

```

    }

    if(!_isEmpty(rows)){
    return(
    <GriditemcontaineralignItems="center"justifyContent="center"padding="20px 0">
    <Typographyvariant="body1">
    Emptydata
    </Typography>
    </Grid>
    );
    }

    returnnull;
    }

    renderPagination(){
    constpage=_.get(this.props,'ui.offset')||0;
    constrowsPerPage=_.get(this.props,'ui.limit')||5;
    consttotalCount=_.chain(this.rows).first().get('totalCount').value()||0;

    return(
    <TablePagination
    rowsPerPageOptions={[5,10,25]}
    component="div"
    count={+totalCount}
    rowsPerPage={rowsPerPage}
    page={page}
    onPageChange={this.onPageChange}
    onRowsPerPageChange={this.onPerPageChange}
    />
    )
    }

    render(){
    return(
    <divclassName={styles.table}>
    <TableContainercomponent={Paper}>
    <Table>
    <TableHeadclassName="tableHeader">
    {this.renderColumns()}
    </TableHead>
    <TableBodyclassName="tableBody">
    {this.renderCells()}
    </TableBody>
    </Table>
    {this.renderLoadingOrEmpty()}
    </TableContainer>

    {this.renderPagination()}
    </div>
    )
    }
    }

    src/client/common/EditModalBase/index.js
    import _from'lodash';
    importReactfrom'react';
    importButtonfrom'@mui/material/Button';
    importDialogfrom'@mui/material/Dialog';
    importDialogTitlefrom'@mui/material/DialogTitle';
    importDialogActionsfrom'@mui/material/DialogActions';
    importDialogContentfrom'@mui/material/DialogContent';
    importCircularProgressfrom'@mui/material/CircularProgress';

```

```

export default class EditModalBase extends React.Component {
  static defaultProps = {
    open: true,
    scrollBody: false,
  };

  okLabel = 'Save';
  cancelLabel = 'Cancel';

  close = (...args) => {
    const { onClose, reset } = this.props;
    if (typeof reset === 'function') reset();
    if (typeof onClose === 'function') onClose(...args);
  };

  submit(...args) {
    this.props.handleSubmit(...args);
  }

  renderTitle() {
    return this.props.title;
  }

  renderActions() {
    const { isSubmitting, isValid } = this.props;

    return [
      <Button key={this.cancelLabel} variant="contained" color="secondary" onClick={this.close}>
        {this.cancelLabel}
      </Button>,
      <Button key={this.okLabel} variant="contained" onClick={(...args) => this.submit(...args)}
        disabled={!isValid || isSubmitting}>
        {this.okLabel}
      </Button>
    ];
  }

  renderBody() {
    return null;
  }

  isLoading() {
    return false;
  }

  renderLoading() {
    return <CircularProgress size={40}/>;
  }

  render() {
    const { open, className, classes } = this.props;
    const title = this.renderTitle();

    return open ? (
      <Dialog open
        title={title}
        className={className}
        fullWidth maxWidth="md">
        {title ? <DialogTitle className={_.get(classes, 'title', '')}>{title}</DialogTitle> : null}
        <DialogContent className={_.get(classes, 'content', '')}>{this.isLoading() ? this.renderLoading() : this.renderBody()}</DialogContent>
        <DialogActions className={_.get(classes, 'actions', '')}>{this.renderActions()}</DialogActions>
      </Dialog>
    ) : null;
  }
}

```

```

</Dialog>
):null;
}
}

src/client/common/privateRouter.js
import React from 'react';
import { Route } from 'react-router-dom';
import _ from 'lodash';
import browserHistory from 'lib/history';
import { withConnect } from 'lib/dec';

@withConnect((store)=>({
  user: store.user
}))
export default class PrivateRoute extends React.Component {
  checkAuth() {
    const { redirectPath, checkLogin, roles=[], user: { userRole, token }={} } = this.props;
    const redirectToLogin = () => {
      browserHistory.replace({
        pathname: redirectPath,
        state: { from: this.props.location }
      });
    };

    if (!localStorage.getItem('token')) {
      return redirectToLogin();
    }

    if (!token || !_.includes(roles, userRole)) {
      checkLogin
        .then(({ response=null }={}) => {
          if (!response || (!_.isEmpty(roles) && !_.includes(roles, response.userRole))) {
            return redirectToLogin();
          }
        })
        .then(() => {
          // ...
        });
    }

    render() {
      const { component: Component, ...rest } = this.props;

      return (
        <Route {...rest} render={ (props) => {
          this.checkAuth();

          return <Component {...props} />
        } } />
      );
    }
  }
}

```

ДОДАТОКД
Слайди презентації

Тема роботи

Програмна реалізація системи для менеджменту колективної оренди житла

Виконав:
ст. групи КНТ-119сп

В.Б. Перегуда

Керівник:
д.т.н., професор

А.О. Олійник

Рисунок Д.1 – Слайд №1

Об'єкт, предмет, мета

Об'єктом роботи є процес функціонування, підтримки та супроводу електронної системи для менеджменту житла, призначеного для оренди у вигляді веб-застосунку.

Предметом дослідження є програмні засоби для програмної реалізації веб-застосунку з метою підтримки системи для менеджменту колективної оренди житла.

Метою роботи є розробка веб-застосунку для менеджменту колективної оренди житла.

2

Рисунок Д.2 – Слайд №2

Постановка завдання

За результатами дослідження предметної області та порівняння існуючих аналогів для вирішення проблеми було сформовано постановку завдання.

Метою є розробка нового веб-застосунку для полегшення пошуку житла для оренди і подальшого його зручного менеджменту.

Система повинна бути стійкою до збоїв, працювати на десктопах та мобільних пристроях, бути адаптованою під різні розширення екранів, а також легкою для подальшої модернізації та підтримки.

3

Рисунок Д.3 – Слайд №3

Функціональні вимоги

Система повинна реалізовувати такі функціональні вимоги

- реєстрація та авторизація в системі;
- пошук будинків та їх фільтрація;
- перегляд інформації користувача;
- додавання нових будинків орендодавцями;
- перегляд показів лічильників;
- формування комунальних «платіжок»;
- передавання показів лічильників орендарями через систему.

4

Рисунок Д.4 – Слайд №4

Порівняння існуючих аналогів

Критерій	Власний сервіс	Unihomes	StuRents
Розумний пошук	-	+	-
Висока швидкість роботи	+	+	-
Комутація комунальних платежів	+	-	-
Зручність та сучасність дизайну	±	+	+
Можливість додавання власних будинків	+	-	-

5

Рисунок Д.5 – Слайд №5

Порівняння мов програмування

	JavaScript	PHP	C#	Java
Суворі типізація	-	-	+	+
Підтримка парадигм ООП	+	+	+	+
Можливість розробки клієнтської частини	+	-	+	+
Можливість розробки серверної частини	+	+	+	+
Легкість вивчення та використання	+	+	±	±

6

Рисунок Д.6 – Слайд №6

Порівняння середовищ розробки

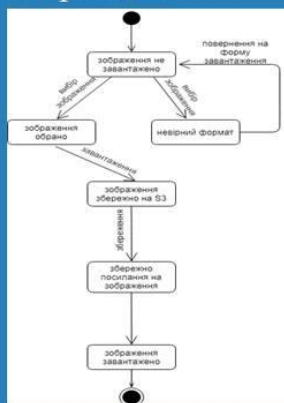
	Intelij IDEA	Vim	WebStorm	Sublime Text
Розширюваність	+	+	+	+
Вбудований аналізатор коду	+	±	+	-
Зручна робота з БД	+	-	±	-
Підтримка систем контролю версій	+	±	+	-
Налаштування віддаленого доступу	-	+	-	-

7

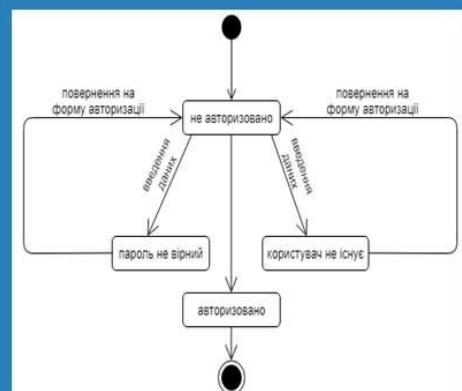
Рисунок Д.7 – Слайд №7

Проектування системи

UML діаграма завантаження зображень



UML діаграма авторизації

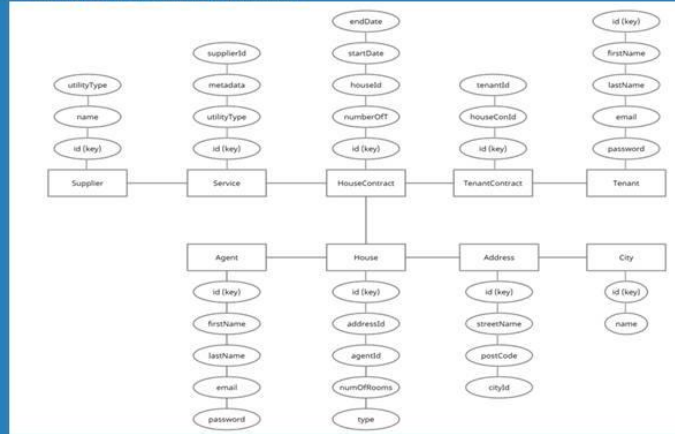


8

Рисунок Д.8 – Слайд №8

Проектування бази даних

Концептуальна модель бази даних

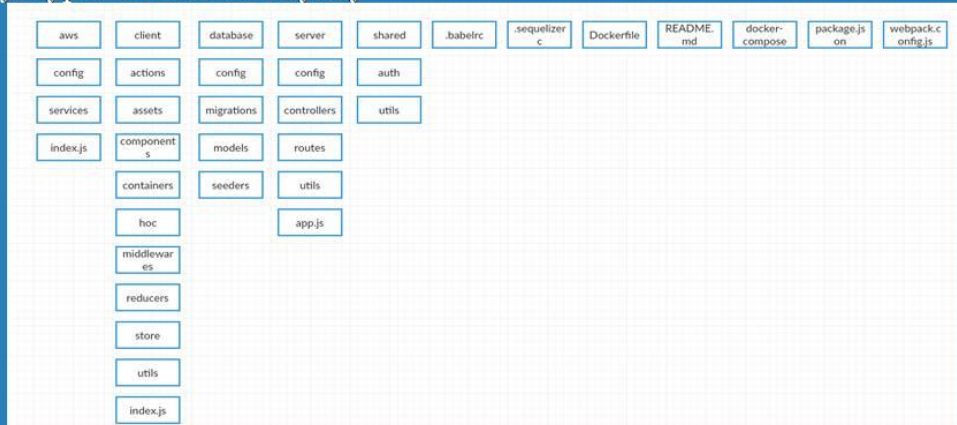


9

Рисунок Д.9 – Слайд №9

Фізичне проектування

Структурна схема застосунку



10

Рисунок Д.10 – Слайд №10

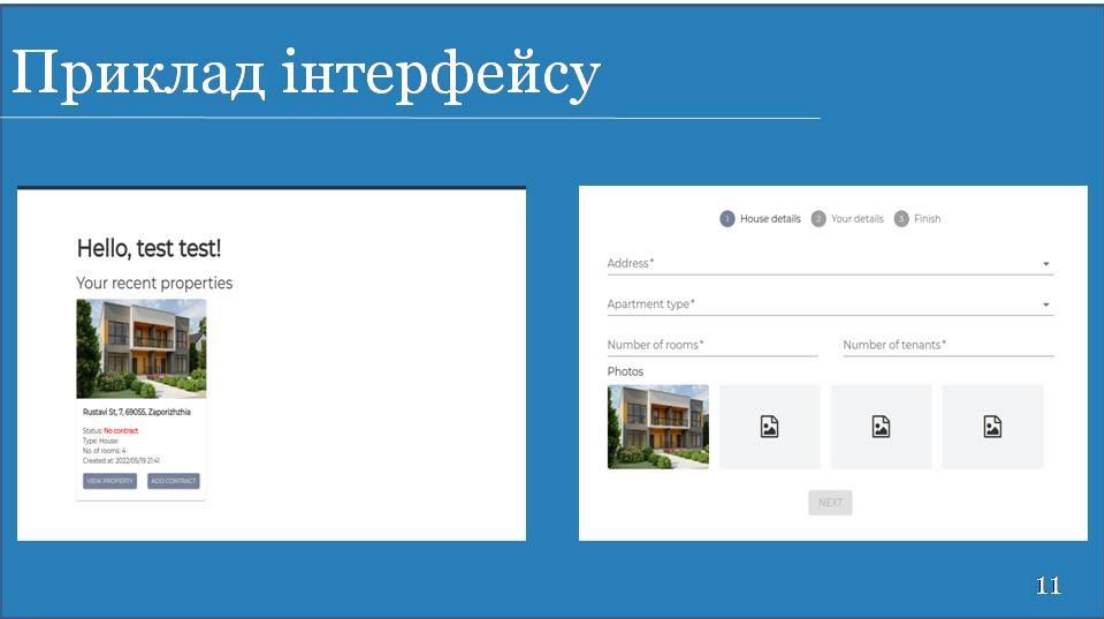


Рисунок Д.11 – Слайд №11

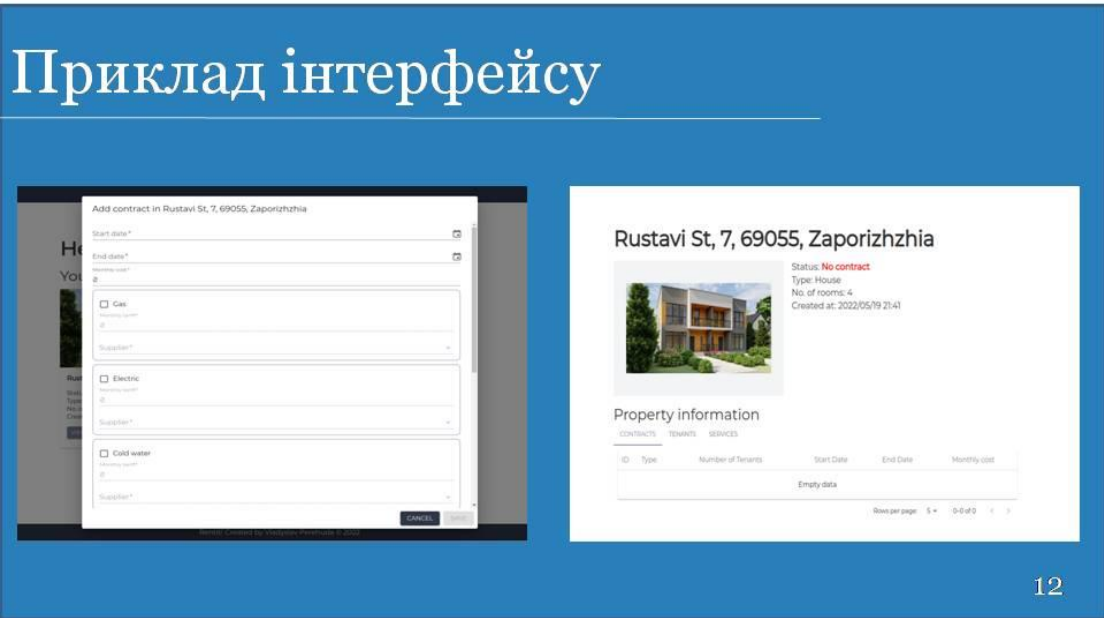


Рисунок Д.12 – Слайд №12

Результати тестування

Функціонал	Linux		Windows
	Google Chrome	Mozilla Firefox	Google Chrome
Авторизація	Пройдено	Пройдено	Пройдено
Реєстрація	Пройдено	Пройдено	Пройдено
Завантаження зображень	Пройдено	Пройдено	Пройдено
Додавання житла	Пройдено	Пройдено	Пройдено
Відображення базової інформації про житло	Пройдено	Пройдено	Пройдено
Верстка при різних дозволах екрану	Пройдено	Не пройдено	Пройдено
Сумісність	Пройдено	Пройдено	Пройдено

13

Рисунок Д.13 – Слайд №13

Висновки

Під час роботи було виконано наступні завдання:

- було проаналізовано предметну область колективної оренди житла;
- досліджено існуючі аналоги та проведено їх порівняння;
- спроектовано застосунок і базу даних;
- обрано технічні засоби розробки: мову програмування, фреймворки, систему керування базами даних;
- описано інструкцію по експлуатації програми.

В результаті виконання роботи було створено веб-застосунок, що спрощує пошук житла, а також надає можливість ефективного і зручного менеджменту ним.

14

Рисунок Д.14 – Слайд №14

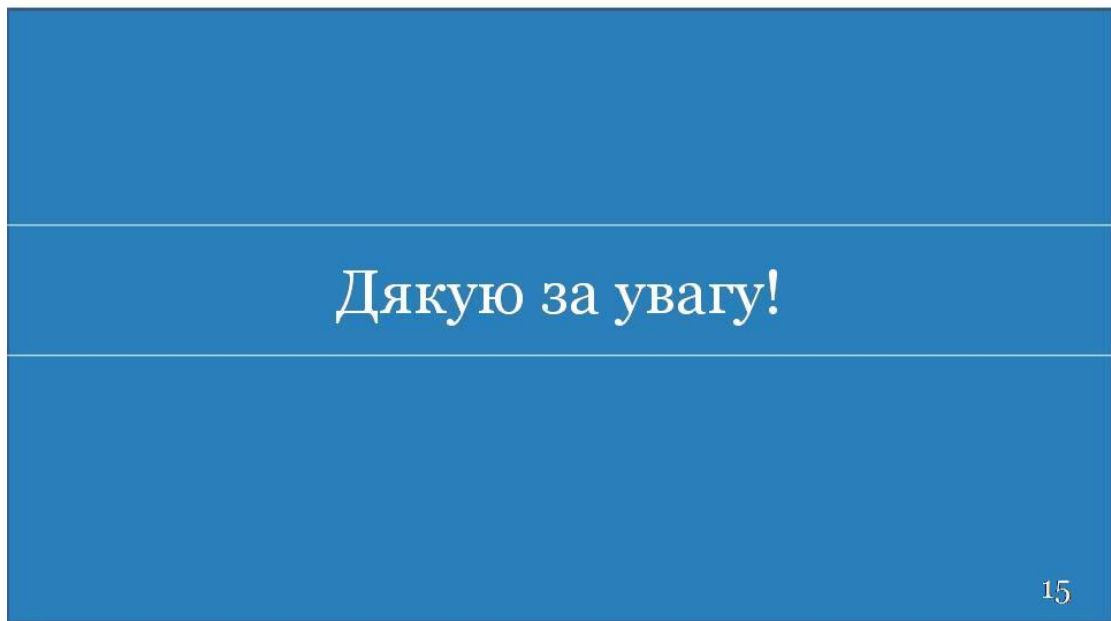


Рисунок Д.15 – Слайд №15