МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА СЕРВЕРНОЇ ЧАСТИНИ СИСТЕМИ ПОШУКУ ТА КОРОТКОСТРОКОВОЇ ОРЕНДИ ЖИТЛА ЗАСОБАМИ DJANGO»**

Виконав: студент 2 курсу, групи 8.1213-іпз-1
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми інженерія програмного забезпечення
(назва освітньої програми)
А.О. Моторнюк
(ініціали та прізвище)

Керівник доцент кафедри фундаментальної та прикладної математики, доцент, к.ф.-м.н. Панасенко Є.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент доцент кафедри комп'ютерних наук, доцент, к.т.н. Матвіїшина Н.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Моторнюку Артему Олександровичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка серверної частини системи пошуку та короткострокової оренди житла засобами Django

керівник роботи Панасенко Євген Валерійович, к.ф.-м.н., доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 01 » березня 2024 року № 385-с

2. Строк подання студентом роботи 25.11.2024 р.

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Реалізація та тестування серверної частини.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

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

7. Дата видачі завдання 07.03.2024 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	20.03.2024	
2.	Збір вихідних даних.	01.05.2024	
3.	Обробка методичних та теоретичних	18.06.2024	
	джерел.		
4.	Розробка першого та другого розділу.	28.08.2024	
5.	Розробка третього розділу.	25.10.2024	
6.	Оформлення та нормоконтроль	20.11.2024	
	кваліфікаційної роботи магістра.		
7.	Захист кваліфікаційної роботи.	12.12.2024	

Студент _____ А.О. Моторнюк
(підпис) (ініціали та прізвище)

Керівник роботи _____ Є.В. Панасенко
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

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

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка серверної частини системи пошуку та короткострокової оренди житла засобами Django»: 49 с., 13 рис., 8 джерел, 2 додатки.

API, DATABASE, DJANGO, DJANGO-REST-FRAMEWORK, FRAMEWORK, POSTGRESQL, PYTHON.

Об’єкт дослідження – додаток, інструменти взаємодії Django та PostgreSQL.

Мета роботи: розробити серверну частину системи пошуку та короткострокової оренди житла.

Метод дослідження – моделювання, проєктування, програмний, аналітичний.

У роботі розглянуто розробку серверної частини системи пошуку та короткострокової оренди житла засобами Django. Проєктування системи виконано із застосуванням UML-діаграм, зокрема діаграм варіантів використання, діяльності, послідовності та розгортання.

Реалізовано функції реєстрації та аутентифікації користувачів, управління оголошеннями та процес бронювання житла. Проведено юніт- та інтеграційне тестування.

Таким чином, розроблена серверна частина забезпечує гнучкість і продуктивність для інтеграції з будь-яким клієнтським інтерфейсом. Використання Django, PostgreSQL, Redis і Docker дозволило створити надійну та масштабовану інфраструктуру, готову до подальшого розширення та інтеграції.

SUMMARY

Master’s qualifying paper “Development of a Server Part of a Search and Short-term Housing Rental System Using Django”: 49 pages, 13 figures, 8 references, 2 supplements.

API, DATABASE, DJANGO, DJANGO-REST-FRAMEWORK, FRAMEWORK, POSTGRESQL, PYTHON.

The object of the study is an application and tools for interaction between Django and PostgreSQL.

The aim of the study is to develop the backend for a system of search and short-term rental of housing.

The methods of research are modeling, design, programming, and analytical methods.

This work examines the development of the backend for a system of search and short-term rental of housing using Django. The system design was performed using UML diagrams, including use case diagrams, activity diagrams, sequence diagrams, and deployment diagrams.

The implementation includes user registration and authentication features, management of property listings, and the process of booking housing. Unit and integration testing were conducted to ensure the correctness of the implemented functionality.

Thus, the developed backend provides flexibility and performance for integration with any client interface. The use of Django, PostgreSQL, Redis, and Docker enabled the creation of a reliable and scalable infrastructure, ready for further expansion and integration.

ЗМІСТ

Завдання на кваліфікаційну роботу 2
Реферат 4
Summary 5
Вступ 8
1 Огляд систем оренди житла та вимоги до них 10
1.1 Загальна характеристика ринку оренди житла 10
1.2 Популярні платформи для оренди житла та їх особливості 11
1.3 Функціональні вимоги до систем оренди житла 12
1.4 Нефункціональні вимоги до систем оренди житла 12
1.5 Аналіз сильних та слабких сторін існуючих рішень 13
1.6 Узагальнення вимог до розроблюваної системи 13
2 Особливості архітектури та проєктування системи 14
2.1 Архітектура системи 14
2.2 Uml-діаграми 16
2.2.1 Діаграма варіантів використання 16
2.2.2 Опис варіантів використання 18
2.2.3 Діаграма діяльності 20
2.2.4 Діаграма послідовності 22
2.2.5 Діаграма розгортання 25
3 Реалізація та тестування 29
3.1 Опис інструментів розробки 29
3.2 Реалізація основних функціональних модулів 30
3.3 Конфігурація середовища 33
3.3.1 Налаштування docker та docker compose 33
3.4 Процес тестування 34
3.5 Керівництво власника житла 36
3.5.1 Авторизація в системі 36

3.5.2 Створення оголошення..... 37
3.6 Керівництво користувача-гостя..... 38
Висновки ……...…….40
Перелік посилань..... 42
Додаток А Dockerfile..... 43
Додаток Б Docker-compose.local.yml..... 47

ВСТУП

Сучасний світ характеризується активним розвитком цифрових технологій та зростанням попиту на зручні онлайн сервіси, які забезпечують швидкий доступ до необхідних послуг. Однією з важливих сфер є ринок оренди житла, де особливо популярними стали послуги короткострокової оренди. Такі сервіси дозволяють користувачам легко знайти і забронювати житло на короткий період часу, що є зручним як для подорожуючих, так і для власників житла, які прагнуть отримати додатковий дохід. З огляду на високий попит та стрімкий розвиток цього ринку, виникає потреба у створенні високопродуктивних систем, які б забезпечували зручність, швидкість і безпеку користувачів при пошуку та бронюванні житла.

Актуальність теми обумовлена тим, що на сьогоднішній день існує великий попит на системи пошуку та оренди житла, які мають бути надійними, швидкими у роботі та здатними забезпечити захист особистих даних користувачів. Створення серверної частини такого сервісу за допомогою сучасного фреймворку Django дозволяє забезпечити ефективну обробку запитів, високу продуктивність, а також масштабованість системи для подальшого розширення функціональності.

Мета дослідження – розробка серверної частини системи пошуку та короткострокової оренди житла засобами Django, яка забезпечить основні функції платформи, включаючи управління користувачами, пошук та фільтрацію оголошень, а також резервування житла.

Досягнення поставленої мети передбачає реалізацію наступних завдань:

- проаналізувати існуючі системи пошуку та оренди житла, їх функціональні особливості та вимоги до продуктивності;
- дослідити архітектуру Django та її можливості для створення серверної частини подібних систем;
- розробити базову архітектуру серверної частини з урахуванням вимог до масштабованості та безпеки;

– реалізувати API для взаємодії з клієнтською частиною та забезпечення функціональності пошуку та оренди житла;
– протестувати систему на відповідність вимогам продуктивності та надійності.

Об’єктом дослідження є системи для пошуку та оренди житла, що включають серверну та клієнтську частини для забезпечення повного циклу взаємодії з користувачами.

Предмет дослідження – серверна частина системи, яка забезпечує обробку запитів, управління даними, авторизацію та безпеку користувачів у рамках процесу оренди житла.

Методи дослідження включають методи об’єктно-орієнтованого програмування, розробку вебдодатків із використанням фреймворку Django, REST API для інтеграції з клієнтською частиною, а також методи забезпечення безпеки даних користувачів.

Наукова новизна та практичне значення роботи полягають у розробці нової архітектури серверної частини системи для пошуку та оренди житла, що дозволить забезпечити високий рівень продуктивності та масштабованості. Отримані результати можуть бути використані як у подальших дослідженнях, так і в комерційних проєктах, що орієнтовані на ринок оренди житла.

Основні положення і результати кваліфікаційної роботи доповідалися та обговорювалися на науково-практичних конференціях, що підтверджує актуальність і значущість дослідження.

Кваліфікаційна робота складається зі вступу, трьох розділів, висновків, переліку посилань і додатків. У першому розділі наведено огляд систем оренди житла та вимоги до них, у другому розглянуто особливості архітектури серверної частини на Django, а третій розділ присвячено реалізації та тестуванню створеної системи.

1 ОГЛЯД СИСТЕМ ОРЕНДИ ЖИТЛА ТА ВИМОГИ ДО НИХ

Ринок оренди житла є однією з найдинамічніших галузей в економіці, що стрімко розвивається завдяки зростанню попиту на короткострокове житло для різних категорій користувачів. Короткострокова оренда житла користується популярністю серед туристів, бізнесменів, студентів, а також осіб, які працюють віддалено. З розвитком цифрових технологій процес оренди житла значно спростився: з’явилися онлайн-платформи, які надають змогу зручно знаходити та бронювати житло, не виходячи з дому.

Одним із головних аспектів успішної роботи подібних платформ є наявність розробленої серверної частини, яка забезпечує обробку запитів, управління даними, а також гарантує безпеку та захист інформації користувачів. На сьогоднішній день існує декілька популярних платформ, які домінують на ринку оренди житла, кожна з яких має свої унікальні характеристики та функціональні особливості. У цьому розділі буде розглянуто сучасні тенденції ринку оренди житла, проаналізовано функціональні та нефункціональні вимоги до подібних систем, а також виділено сильні та слабкі сторони існуючих рішень.

Метою цього розділу є створення комплексного уявлення про сучасні системи оренди житла та визначення вимог до розробки нової платформи, що враховує поточні потреби користувачів і сучасні стандарти безпеки та ефективності. Аналіз існуючих рішень дозволить ідентифікувати основні функції та можливості, які мають бути реалізовані у новій системі, що сприятиме підвищенню її конкурентоспроможності на ринку.

1.1 Загальна характеристика ринку оренди житла

Сучасний ринок оренди житла є динамічним і швидко розвивається, що зумовлено стрімким зростанням попиту на короткострокову оренду. Такі

послуги користуються популярністю серед мандрівників та людей, які прагнуть орендувати житло на певний короткий період. Використання цифрових технологій у цій галузі значно спростило процес пошуку житла, бронювання та оплати, роблячи його більш ефективним та зручним для користувачів. Зростання попиту на оренду житла зумовлене рядом факторів, таких як збільшення числа туристів, зростаюча популярність віддаленої роботи, а також бажання багатьох осіб тимчасово змінити місце проживання для кращого поєднання праці та відпочинку. Крім того, вплив пандемії COVID-19 призвів до зростання попиту на оренду житла у передмістях та на природі, що дає людям змогу працювати віддалено, зберігаючи комфортні умови проживання.

1.2 Популярні платформи для оренди житла та їх особливості

Ринок оренди житла представлений різними онлайн-платформами, серед яких найбільш популярними є Airbnb, Booking.com та Vrbo.

Airbnb – спеціалізується на короткостроковій оренді унікального житла, зокрема будинків, квартир, та навіть нетрадиційних об’єктів, таких як домики на дереві або яхти. Платформа надає можливість спілкування між орендодавцями та орендарями, а також функцію системи відгуків, що підвищує довіру між користувачами.

Booking.com – є універсальною платформою, що пропонує як готельні послуги, так і приватне житло для оренди. Вона орієнтована на широке коло користувачів, включаючи як туристів, так і ділових людей. Сильними сторонами платформи є широкий вибір житла та інтеграція з різноманітними платіжними системами.

Vrbo – фокусується на оренді житла для сімейного відпочинку, пропонуючи великі будинки та вілли для комфортного розміщення груп людей. Платформа користується популярністю серед сімейних туристів, які шукають затишне житло для довготривалого проживання.

1.3 Функціональні вимоги до систем оренди житла

Розробка системи для оренди житла передбачає реалізацію ряду функціональних можливостей, зокрема:

- пошук та фільтрація: система повинна підтримувати пошук житла за різними критеріями (розташування, ціна, тип житла), що дозволяє користувачам швидко знайти відповідні варіанти;
- керування обліковими записами користувачів: користувачам необхідно надати можливість реєстрації, авторизації та управління профілем;
- бронювання житла: система повинна мати функціонал перегляду доступності житла та забезпечувати процес бронювання, включаючи скасування або зміну умов бронювання;
- комунікація між орендодавцем та орендарем: можливість спілкування між сторонами для уточнення деталей оренди, що може бути реалізовано через внутрішній чат;
- система відгуків та оцінок: система повинна дозволяти користувачам залишати відгуки, що підвищує довіру до платформи та покращує загальний досвід користування;
- обробка платежів: інтеграція з платіжними системами для забезпечення безпечної оплати послуг;
- захист даних: система повинна гарантувати безпеку персональних даних відповідно до сучасних вимог захисту конфіденційності.

1.4 Нефункціональні вимоги до систем оренди житла

Окрім функціональних, система повинна також задовольняти наступні нефункціональні вимоги:

- продуктивність – система повинна забезпечувати швидку обробку запитів користувачів, мінімізуючи затримки;

- масштабованість – розробка повинна враховувати можливість розширення системи для обробки високого навантаження;
- надійність та відмовостійкість – система має забезпечувати безперебійну роботу навіть при збоях в окремих компонентах;
- безпека – захист від поширених атак, таких як SQL-ін’єкції, XSS та CSRF-атаки, є критичним для безпеки даних;
- юзабіліті – інтерфейс повинен бути інтуїтивно зрозумілим і зручним для користувачів;
- конфіденційність – система повинна забезпечувати дотримання вимог законодавства щодо захисту персональних даних.

1.5 Аналіз сильних та слабких сторін існуючих рішень

Популярні платформи для оренди житла, як-от Airbnb, Booking.com і Vrbo, мають свої переваги та недоліки. Серед основних сильних сторін є великий вибір житла, зручність користування та функціональність. Однак деякі платформи мають високі комісії або обмеження у гнучкості бронювання. Високий попит на подібні послуги інколи призводить до перевантаження платформ, що може негативно впливати на продуктивність.

1.6 Узагальнення вимог до розроблюваної системи

На основі проведеного аналізу вимог та особливостей популярних платформ було сформовано узагальнені вимоги до системи для оренди житла, яку буде розроблено в рамках цього дослідження. Основними вимогами є підтримка функціональності пошуку, управління обліковими записами, бронювання житла та забезпечення безпеки даних. Нефункціональні вимоги, такі як продуктивність, масштабованість та відмовостійкість, є критичними для забезпечення високого рівня користувацького досвіду.

2 ОСОБЛИВОСТІ АРХІТЕКТУРИ ТА ПРОЄКТУВАННЯ СИСТЕМИ

Проєктування архітектури інформаційної системи є важливим етапом, який визначає, як саме працюватиме система, які компоненти її складатимуть і як вони взаємодіятимуть між собою. Це забезпечує основу для надійності, масштабованості та продуктивності всього програмного забезпечення. У цьому розділі розглянуто архітектуру системи, обґрунтовано вибір інструментів і технологій, а також проілюстровано ключові аспекти роботи за допомогою UML-діаграм.

На початку подано опис загальної архітектури системи: її складових, взаємодії компонентів та функцій, які виконує кожен із них. Далі представлені UML-діаграми, які демонструють, як різні частини системи працюють разом, як користувачі взаємодіють із платформою та як обробляються основні процеси. Діаграми ілюструють сценарії використання, послідовності виконання дій, структуру компонентів та їхнє розташування в середовищі розгортання. Після цього пояснено вибір інструментів, таких як Django, PostgreSQL і Docker, які дозволяють забезпечити надійність і гнучкість системи. Завершує розділ підсумок ключових архітектурних рішень та оцінка їхньої відповідності визначеним вимогам.

Матеріали цього розділу не лише пояснюють, як функціонує система, але й демонструють, що її архітектура є гнучкою, стабільною та здатною задовольнити сучасні вимоги користувачів. Розглянуті підходи та рішення обрано на основі їхньої ефективності та відповідності поставленим завданням.

2.1 Архітектура системи

Архітектура системи пошуку та короткострокової оренди житла – це серце проєкту, яке визначає, як усе працює разом. Вона побудована за клієнт-

серверною моделлю, яка дозволяє чітко розділити завдання між серверною частиною, що обробляє дані, і клієнтською частиною, яка забезпечує зручну взаємодію з користувачем.

Основою серверної частини є фреймворк Django. Це потужний інструмент, який вже давно зарекомендував себе як надійна основа для вебдодатків. Щоб забезпечити ефективну взаємодію клієнта із сервером, використовується Django REST Framework (DRF) [5]. Завдяки цьому фреймворку реалізовано RESTful API, яке дозволяє клієнтській частині надсилати запити до сервера та отримувати потрібні дані.

Для зберігання даних обрана PostgreSQL – реляційна база даних, яка добре справляється з великими обсягами інформації та складними запитами. Вона використовується для організації зберігання даних про користувачів, оголошення, бронювання, відгуки та інші об’єкти системи.

Клієнтська частина, яка може бути вебінтерфейсом або мобільним додатком, працює через API. Вона дозволяє користувачам переглядати оголошення, здійснювати бронювання, керувати своїми профілями та залишати відгуки. Усе це робить взаємодію з платформою максимально зручною та зрозумілою.

Окремої уваги заслуговує процес розгортання системи. Тут використовується Docker та Docker Compose – сучасні інструменти контейнеризації, які роблять процес автоматизованим і простим. Кожен компонент системи (сервер, база даних тощо) працює у своєму контейнері, що забезпечує ізоляцію та стабільність. До того ж, це рішення дозволяє легко масштабувати систему у разі зростання навантаження.

Архітектура побудована з урахуванням принципів модульності, безпеки та масштабованості. Модульна структура забезпечує гнучкість, дозволяючи легко додавати нові функції. Масштабованість підтримується можливістю горизонтального розширення серверної частини. А вбудовані механізми безпеки Django захищають систему від таких загроз, як XSS, CSRF та SQL-ін’єкції [1].

Ця архітектура розроблена з акцентом на сучасні вимоги до продуктивності та надійності, що робить її ідеальним рішенням для розробки платформи оренди житла.

2.2 UML-діаграми

2.2.1 Діаграма варіантів використання

Діаграма варіантів використання демонструє, як різні користувачі взаємодіють із системою. У нашій системі визначено три основні ролі: Гість, Власник та Адміністратор. Кожна з цих ролей має свої функціональні можливості.

Гість може реєструватися та авторизовуватися в системі, переглядати оголошення, здійснювати бронювання житла, залишати відгуки на оголошення та переглядати свої попередні бронювання. Власник, у свою чергу, має змогу створювати оголошення про житло, редагувати та видаляти їх, а також переглядати відгуки, залишені гостями. Адміністратор здійснює контроль якості системи, схвалюючи або відхиляючи оголошення, а також модеруючи відгуки користувачів.

Діаграма варіантів використання відображає ці сценарії взаємодії та допомагає візуалізувати, які функції системи доступні для кожної ролі. Це також дозволяє побачити основні функціональні залежності між акторів та системою.

На діаграмі (див. рис. 2.1) варіантів використання відображено ключові сценарії взаємодії основних користувачів із системою пошуку та оренди житла. Вона ілюструє три категорії користувачів: Гість, Власник і Адміністратор, кожен з яких взаємодіє із системою у свій унікальний спосіб.

Гість є користувачем, який здійснює пошук житла для оренди. Його можливості включають реєстрацію або авторизацію в системі, що дозволяє створити обліковий запис або увійти до вже існуючого. Далі гість може

переглядати доступні оголошення, фільтруючи їх за різними критеріями. Коли підходяще житло знайдено, система дозволяє здійснити бронювання. Після завершення проживання гість має змогу залишити відгук про житло та сервіс.

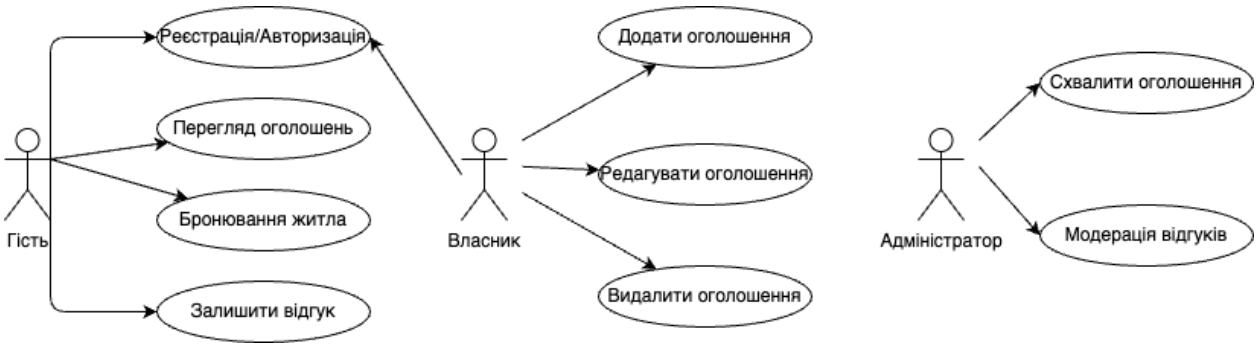


Рисунок 2.1 – Діаграма варіантів використання

Власник, у свою чергу, взаємодіє із системою з іншою метою. Його основна роль полягає у додаванні нових оголошень про житло, редагуванні існуючих, якщо є потреба оновити інформацію, та видаленні тих, які більше не актуальні. Це забезпечує динамічність та актуальність платформи для інших користувачів.

Адміністратор виконує функції модерації та управління системою. Він перевіряє нові оголошення перед їхньою публікацією, щоб забезпечити відповідність правилам платформи. Також адміністратор здійснює модерацію відгуків, контролюючи, щоб вони відповідали етичним та професійним стандартам.

Діаграма варіантів використання є надзвичайно важливим інструментом при розробці програмних систем, оскільки вона допомагає краще зрозуміти, як саме система взаємодіє з користувачами. Вона створює зрозуміле уявлення про те, які функції є критично важливими для різних типів користувачів, та як ці функції можуть бути реалізовані. Це не просто спосіб візуалізації – це ключ до ефективної комунікації між розробниками, дизайнерами, замовниками і кінцевими користувачами.

Ця діаграма дозволяє чітко побачити, хто саме використовуватиме систему, які завдання вони виконуватимуть і яким чином. Завдяки їй можна

структурувати вимоги до функціоналу, уникнути непорозумінь у команді та зрозуміти, де необхідно зосередити увагу під час розробки. Наприклад, для системи оренди житла важливо окреслити, що гість може здійснювати пошук і бронювання житла, тоді як власник має додавати і редагувати оголошення. Без цього розуміння легко упустити критичний функціонал або витратити ресурси на другорядні завдання.

Діаграма варіантів використання також виступає як основа для проєктування деталей системи. Вона не лише описує сценарії використання, але й допомагає уявити, як вони будуть реалізовані на технічному рівні. Завдяки цій діаграмі можна ефективно планувати розвиток системи, а також перевіряти, чи відповідає реалізація початковим вимогам. Це спрощує як процес розробки, так і подальшу підтримку програмного забезпечення.

2.2.2 Опис варіантів використання

Прецедент «Реєстрація користувача».

Призначення: цей варіант використання забезпечує можливість створення нового користувача в системі.

Основний потік подій: сценарій починається, коли клієнт надсилає запит на API-ендпойнт реєстрації. Запит містить необхідні дані, такі як ім’я, електронна пошта та пароль. Сервер перевіряє валідність отриманих даних. У разі успіху система створює новий обліковий запис у базі даних та повертає підтвердження у форматі JSON.

Виняткова ситуація 1: введені невалідні дані – сервер повертає код помилки 400 (Bad Request) із детальним описом помилки.

Виняткова ситуація 2: користувач із вказаною електронною поштою вже існує – сервер повертає код помилки 409 (Conflict).

Виняткова ситуація 3: серверна помилка – повертається код помилки 500 (Internal Server Error).

Прецедент «Авторизація користувача».

Призначення: цей варіант використання дозволяє існуючому користувачу увійти до системи.

Основний потік подій: клієнт надсилає POST-запит на ендпойнт авторизації із вказанням електронної пошти та паролю. Сервер перевіряє відповідність введених даних обліковому запису. У разі успішної перевірки система генерує токен доступу (JWT), який повертається клієнту для подальшого використання.

Передумова: обліковий запис користувача має існувати в базі даних.

Виняткова ситуація 1: введені дані не відповідають запису – сервер повертає код помилки 401 (Unauthorized).

Виняткова ситуація 2: серверна помилка – повертається код 500 (Internal Server Error).

Прецедент «Додавання оголошення».

Призначення: цей варіант використання дозволяє власнику створити нове оголошення про здачу житла.

Основний потік подій: клієнт надсилає запит на відповідний API-ендпойнт із вказанням деталей житла, таких як назва, опис, ціна та адреса. Сервер обробляє запит, перевіряє дані на валідність, зберігає нове оголошення у базі даних та повертає підтвердження.

Передумова: користувач має бути авторизованим для виконання цієї дії.

Виняткова ситуація 1: невалідні дані – сервер повертає код помилки 400 (Bad Request) із деталізацією проблеми.

Виняткова ситуація 2: серверна помилка – повертається код 500 (Internal Server Error).

Прецедент «Бронювання житла».

Призначення: цей варіант використання дозволяє гостю створити бронювання для обраного житла.

Основний потік подій: клієнт надсилає запит до API-ендпойнта бронювання із зазначенням ID оголошення, дат початку та закінчення оренди.

Сервер перевіряє доступність житла у вказаний період, створює запис у базі даних про бронювання та повертає підтвердження.

Передумова: користувач має бути авторизованим для виконання цієї дії.

Виняткова ситуація 1: житло недоступне на вибрані дати – сервер повертає код помилки 409 (Conflict).

Виняткова ситуація 2: серверна помилка – повертається код 500 (Internal Server Error).

Прецедент «Схвалення оголошення».

Призначення: цей варіант використання забезпечує можливість адміністратору перевірити та схвалити оголошення перед його публікацією.

Основний потік подій: адміністратор надсилає запит до API-ендпойнта модерації оголошень, вказуючи ID оголошення та його статус. Сервер оновлює запис у базі даних, змінюючи статус оголошення, і повертає підтвердження успішної операції.

Передумова: оголошення має бути додано до бази даних і знаходитись у статусі «Очікує схвалення».

Виняткова ситуація 1: оголошення не знайдено – сервер повертає код помилки 404 (Not Found).

Виняткова ситуація 2: серверна помилка – повертається код 500 (Internal Server Error).

2.2.3 Діаграма діяльності

Діаграма діяльності є важливим інструментом для моделювання бізнес-процесів, алгоритмів і робочих потоків. У нашій системі вона дозволяє описати послідовність дій та взаємодій між компонентами серверної частини під час виконання ключових сценаріїв. Ця діаграма сприяє кращому розумінню логіки процесів, полегшує їхній аналіз і дозволяє виявити можливі точки оптимізації.

Діаграма діяльності побудована на основі UML і відображає послідовність дій у вигляді потоків управління. Для кожного процесу моделюються активності, переходи, точки прийняття рішень та завершення. Наприклад, процес бронювання житла включає перевірку доступності житла, створення бронювання в базі даних, а також відправку повідомлення з підтвердженням на електронну пошту користувача.

У нашому випадку, діаграма діяльності також ілюструє, як сервер взаємодіє з базою даних і зовнішніми сервісами, такими як система оплати або відправка повідомлень. Це дозволяє нам оцінити залежності між компонентами, їхню взаємодію та потенційні ризики.

На рисунку 2.2 зображено процес додавання нового оголошення. Він починається із запиту користувача до API-ендпойнта, після чого сервер виконує перевірку даних, зберігає інформацію в базі даних і завершує процес поверненням підтвердження. Такий підхід забезпечує прозорість роботи системи та дозволяє швидко впроваджувати необхідні зміни в процесах.

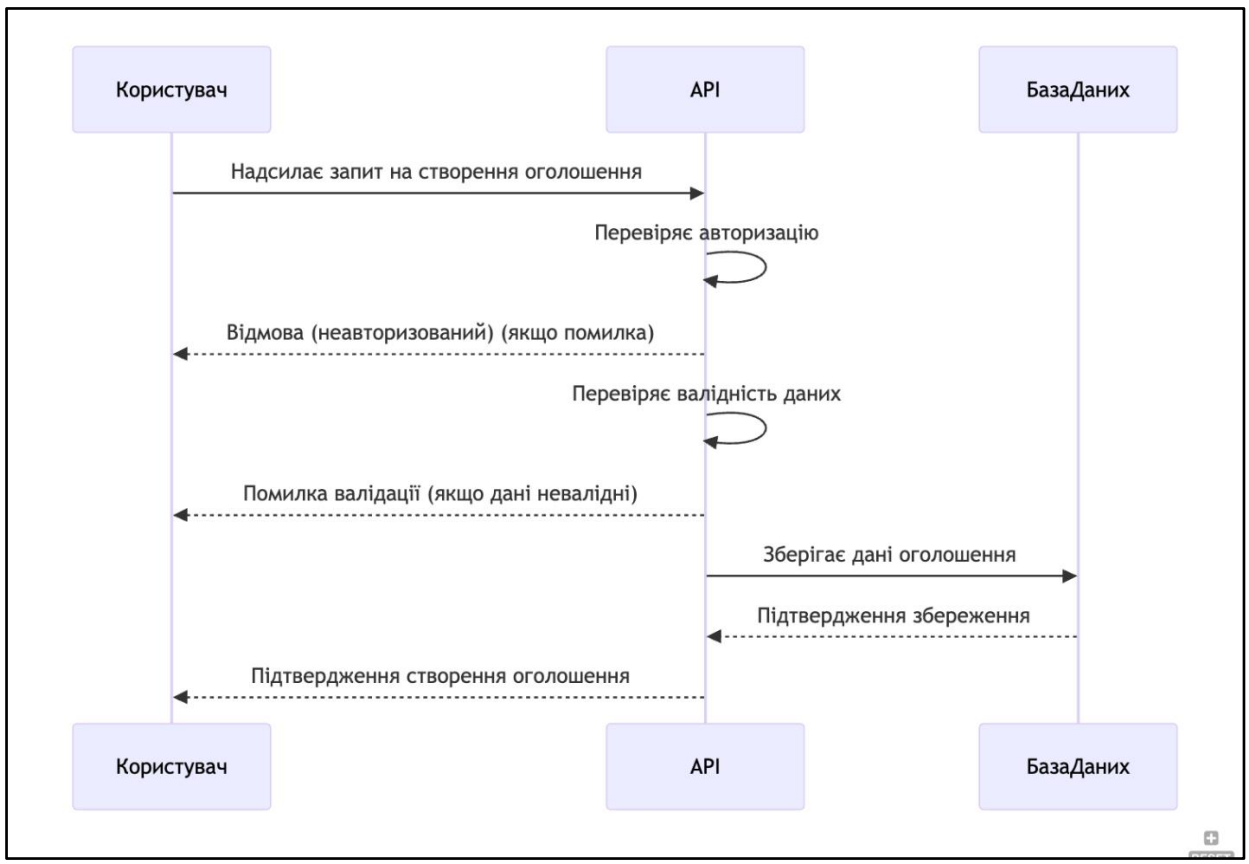


Рисунок 2.2 – Діаграма діяльності

Діаграми діяльності використовуються на різних етапах розробки. На початковому етапі вони допомагають аналізувати вимоги та візуалізувати основні процеси. Під час реалізації вони виступають основою для написання коду та перевірки його логіки. Також вони корисні на етапі тестування, оскільки дозволяють перевірити всі можливі сценарії, включно з винятковими ситуаціями, такими як помилки в обробці даних або недоступність сервісів.

Таким чином, діаграма діяльності є незамінним інструментом для моделювання функціональної поведінки системи, її аналізу та вдосконалення. Вона дозволяє виявити слабкі місця у процесах і покращити їх, що є важливим аспектом для створення якісного програмного забезпечення.

2.2.4 Діаграма послідовності

Діаграма послідовності, яка є частиною UML, відіграє ключову роль у візуалізації порядку взаємодій між об’єктами або компонентами системи у часі. Цей тип діаграми дозволяє детально розглянути, як саме серверна частина системи виконує певний функціонал через обмін повідомленнями між її компонентами. Вона не лише представляє технічну складову, але й сприяє більш глибокому розумінню логіки роботи системи.

На такій діаграмі відображаються учасники процесу, які взаємодіють один з одним у вигляді прямокутників, що розташовані горизонтально зверху. Під кожним учасником знаходиться вертикальна лінія життя, яка вказує на існування цього об’єкта протягом сценарію. Повідомлення між учасниками представлені у вигляді стрілок, що вказують напрямок комунікації. Важливим аспектом є те, що діаграма послідовності читається зверху вниз, що дозволяє легко зрозуміти хронологію дій.

Для нашої системи короткострокової оренди житла діаграма послідовності зосереджується на ключових сценаріях. Наприклад, у сценарії додавання нового оголошення спочатку користувач надсилає запит до API. Серверна частина перевіряє, чи авторизований користувач. Якщо авторизація

успішна, система перевіряє валідність даних оголошення. Після цього сервер зберігає інформацію у базі даних та надсилає відповідь користувачу із підтвердженням.

Ця діаграма корисна не тільки для розробників, але і для інших учасників проєкту, таких як аналітики чи тестувальники. Вона допомагає зрозуміти, які саме етапи виконує система, в яких місцях можуть виникати помилки, і як ці помилки можна вирішити. Зокрема, у разі виникнення виключних ситуацій, таких як помилки у валідації даних або збої у базі даних, діаграма показує, як система реагує на такі події.

Діаграма послідовності також є чудовим інструментом для планування тестування. Вона дозволяє створювати тест-кейси, які покривають як основні, так і альтернативні сценарії. Наприклад, для функції бронювання житла важливо протестувати як успішне створення бронювання, так і ситуації, коли житло недоступне або виникають проблеми з базою даних.

Таким чином, діаграма послідовності є важливим елементом моделювання системи, який забезпечує прозорість і зрозумілість процесів, що відбуваються у серверній частині. Це не лише полегшує розробку, але й підвищує загальну якість програмного забезпечення, що відповідає сучасним стандартам у галузі ІТ.

На рисунку 2.3 описана діаграма послідовності прецеденту «Додавання оголошення». На цій діаграмі послідовності відображено порядок взаємодії між різними компонентами серверної частини системи короткострокової оренди житла під час виконання процесу додавання нового оголошення. Вона демонструє, як користувач взаємодіє із сервером через API, а сервер виконує внутрішні операції, взаємодіючи з іншими компонентами.

Перший крок на діаграмі – це надсилання користувачем запиту до API для створення оголошення. Серверна частина спочатку виконує перевірку авторизації через компонент IsAuthenticated. Якщо користувач не авторизований, сервер завершує процес, повертаючи помилку авторизації. У разі успішної авторизації запит передається до компонента CreatePropertyApi, який обробляє створення оголошення.

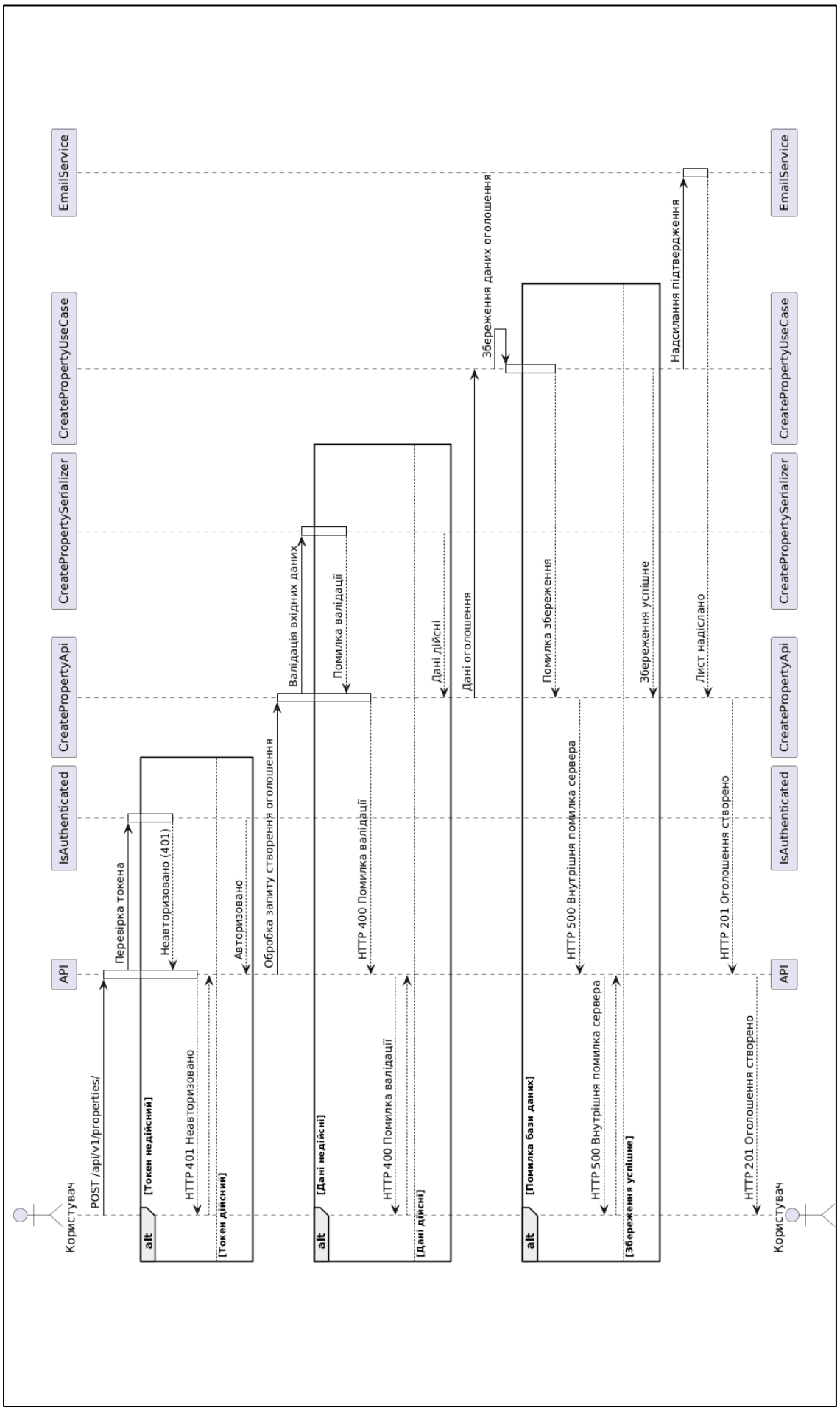


Рисунок 2.3 – Діаграма послідовності

Далі дані проходять перевірку валідності за допомогою компонента CreatePropertySerializer. У разі виявлення помилок у введених даних сервер повертає повідомлення про помилку валідації. Якщо дані є коректними, запит передається до компонента CreatePropertyUseCase, який відповідає за збереження даних оголошення в базі даних. У разі виникнення проблем із базою даних сервер генерує повідомлення про внутрішню помилку, але якщо збереження успішне, процес переходить до фінальної стадії.

На останньому етапі сервер надсилає підтвердження через EmailService, використовуючи SMTP для передачі листа користувачу. Після успішного завершення всіх дій сервер повертає користувачу повідомлення про успішне створення оголошення.

Ця діаграма наочно ілюструє всі ключові етапи виконання запиту, враховуючи як основні сценарії, так і виняткові ситуації. Вона дозволяє детально побачити послідовність взаємодій між компонентами системи, їхні ролі та типи обробки даних, що є важливим для аналізу та оптимізації процесів.

2.2.5 Діаграма розгортання

Діаграма розгортання є одним із найважливіших елементів моделювання фізичної архітектури програмного забезпечення. Вона дозволяє наочно представити розташування компонентів системи на фізичних або віртуальних вузлах, а також продемонструвати, як ці вузли взаємодіють між собою. У контексті розробки сучасних вебдодатків, таких як система короткострокової оренди житла, діаграма розгортання дозволяє зрозуміти, як саме програмна система інтегрується в інфраструктуру.

У діаграмі розгортання використовуються кілька ключових елементів: вузли (Node), компоненти (Component) та зв’язки (Connector). Вузли представляють фізичні або віртуальні пристрої, на яких працюють різні частини системи. Це можуть бути сервери, клієнтські пристрої, бази даних або

зовнішні сервіси. Наприклад, у нашій системі вузли включають вебсервер, сервер бази даних та зовнішній SMTP-сервер для відправлення електронної пошти. Кожен вузол може містити один або більше компонентів, які виконують певні функції, наприклад, обробку запитів користувача або управління даними.

Компоненти, своєю чергою, відображають програмні модулі або функціональні частини системи, які виконують конкретні завдання. У нашій системі це, зокрема, серверна частина, реалізована за допомогою Django, яка виконує роль основного компонента для обробки HTTP-запитів. Також серед компонентів виділяється база даних PostgreSQL, яка зберігає всю інформацію про оголошення, користувачів та бронювання. Ще одним важливим компонентом є система Celery Worker, що обробляє фонові завдання, наприклад, надсилання повідомлень або виконання асинхронних операцій.

Зв’язки між вузлами або компонентами, які часто називають конекторами, ілюструють типи взаємодій між елементами системи. У нашій системі вони відображають передачу HTTP-запитів між браузером користувача та сервером, запити SQL до бази даних або комунікацію через протокол SMTP для відправлення електронних листів. Такі зв’язки можуть бути реалізовані через мережеві протоколи, що забезпечують надійну взаємодію між різними частинами системи. Наприклад, HTTP/HTTPS гарантує безпечний обмін даними між клієнтом і сервером, а SQL-запити дозволяють серверу виконувати складні операції з базою даних.

Розгортання сучасних систем часто базується на хмарній інфраструктурі, що забезпечує гнучкість і масштабованість. У нашій діаграмі розгортання вузли вебсервера та сервера бази даних розташовані в хмарному середовищі, наприклад, на платформах AWS або Azure. Це дозволяє легко масштабувати систему, додаючи нові екземпляри серверів або оптимізуючи використання ресурсів відповідно до потреб користувачів.

Окрім цього, діаграма розгортання є важливим інструментом для забезпечення безпеки та стійкості системи. Завдяки чіткому уявленню про

взаємодії між компонентами можна визначити можливі точки відмови або ризики. Наприклад, якщо вузол бази даних перевантажений, це може призвести до затримок у роботі всієї системи. В таких випадках важливо передбачити механізми резервування або реплікації даних, які дозволять зменшити навантаження та забезпечити безперебійну роботу.

У підсумку, діаграма розгортання є незамінним інструментом для візуалізації фізичної архітектури програмної системи (рис. 2.4). Вона забезпечує розробникам, адміністраторам та іншим зацікавленим сторонам можливість отримати чітке уявлення про те, як саме компоненти взаємодіють між собою та з користувачами. Завдяки цьому можна краще зрозуміти, як оптимізувати роботу системи, зробити її більш ефективною та надійною.

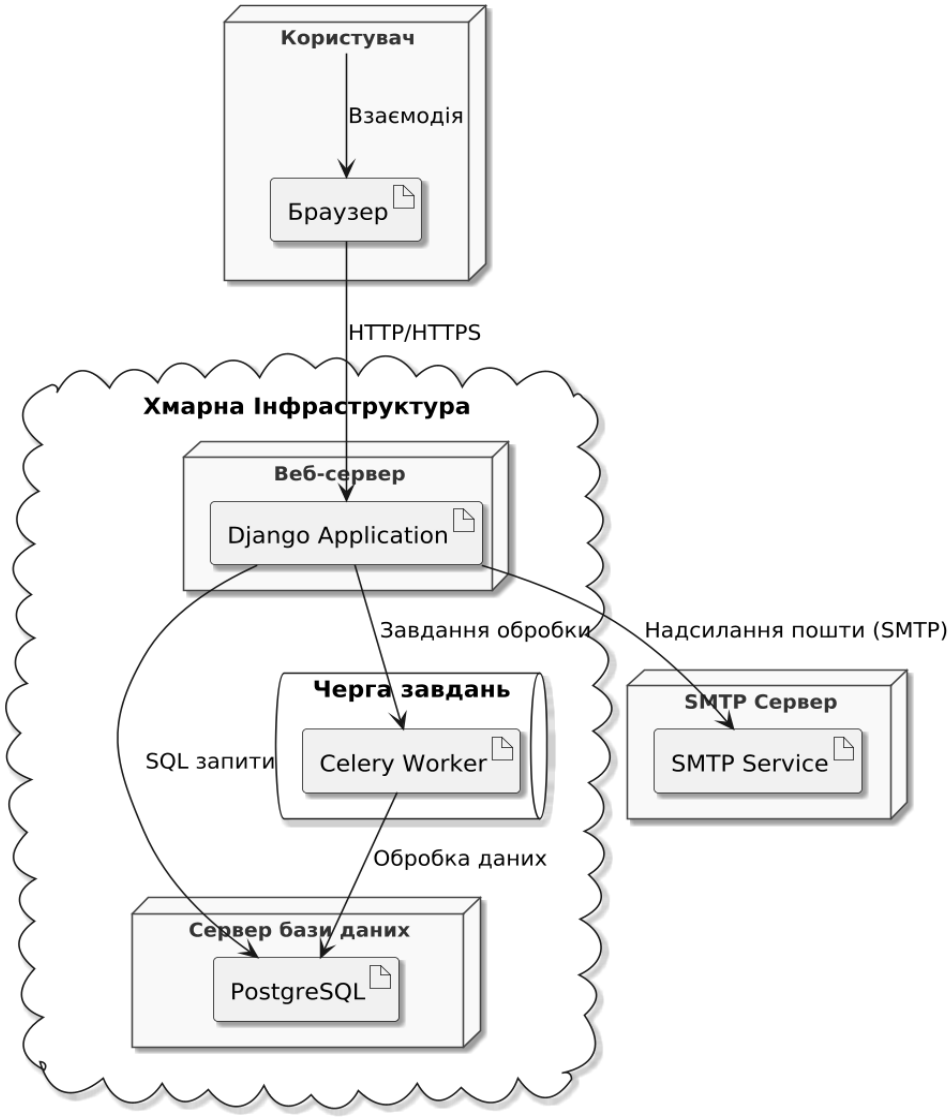


Рисунок 2.4 – Діаграма розгортання

На цій діаграмі розгортання (див. рис. 2.4) відображено фізичну архітектуру системи короткострокової оренди житла. Вона демонструє, як різні компоненти програмного забезпечення взаємодіють між собою в реальному середовищі. Центральним елементом є вузол вебсервера, який працює на хмарній інфраструктурі та обслуговує всі запити користувачів. На ньому розташовано основний компонент – серверна частина, реалізована за допомогою Django.

Також на діаграмі показано сервер бази даних, де розміщено PostgreSQL. Цей компонент відповідає за зберігання всіх даних системи, таких як інформація про користувачів, оголошення та бронювання. Взаємодія між вебсервером і базою даних здійснюється через SQL-запити, що забезпечує швидкий і надійний доступ до даних.

Додатково, на діаграмі представлено компонент Celery Worker, який відповідає за обробку асинхронних завдань. Це включає виконання фонового надсилання електронних листів, підготовку повідомлень або інші операції, які не потребують миттєвої відповіді від системи. Його взаємодія з іншими компонентами, зокрема базою даних, забезпечує ефективність і паралельність виконання задач.

Окремо зображено зовнішній вузол – SMTP-сервер, який використовується для надсилання електронних листів, таких як підтвердження реєстрації або сповіщення про бронювання. Вебсервер передає запити до SMTP-сервера через стандартний протокол, забезпечуючи комунікацію з користувачами поза межами основної системи.

Загалом, діаграма ілюструє взаємодії між вузлами та компонентами, підкреслюючи роль кожного з них у забезпеченні роботи системи. Вона також демонструє використання сучасних технологій для створення масштабованого та надійного середовища, яке може адаптуватися до зростаючих потреб користувачів.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1 Опис інструментів розробки

Реалізація серверної частини системи короткострокової оренди житла потребувала використання низки сучасних інструментів та технологій, які забезпечують ефективну розробку, налаштування середовища та тестування. У цьому підрозділі буде детально описано ключові інструменти, які використовувались під час створення системи, а також обґрунтовано їх вибір.

Основою для розробки серверної частини став фреймворк Django, який є одним із найбільш популярних інструментів для створення вебдодатків [3]. Django пропонує зручну архітектуру MVT, що спрощує розділення логіки додатка, управління базами даних та обробку запитів користувача [7]. Крім того, Django має вбудовані механізми аутентифікації, які полегшують реалізацію функцій реєстрації та авторизації користувачів, що є критично важливими для системи.

Для управління базами даних використовувалася система PostgreSQL, яка відома своєю стабільністю, масштабованістю та підтримкою складних запитів. PostgreSQL забезпечує високу продуктивність під час обробки великих обсягів даних і підтримує розширені функції, такі як JSONB для роботи з напівструктурованими даними, що може бути корисним для динамічних налаштувань системи.

Налаштування та ізоляція середовища виконувались із використанням Docker та Docker Compose. Docker дозволяє створювати контейнери, що включають всі необхідні залежності, забезпечуючи їхню взаємодію в однорідному середовищі. Це спрощує процес розгортання системи на різних етапах розробки, тестування та продакшну. Docker Compose, своєю чергою, дозволяє автоматизувати запуск декількох контейнерів, таких як вебсервер Django, база даних PostgreSQL та Celery Worker.

Редактор PyCharm використовувався як основне середовище розробки. PyCharm забезпечує інтегроване середовище для написання, тестування та відлагодження коду. Його переваги включають інтеграцію з інструментами для контролю версій, такими як Git, а також підтримку багатьох плагінів, які підвищують продуктивність розробників.

Для фонових задач, таких як відправлення електронних листів і виконання асинхронних операцій, використовувався Celery. Цей інструмент інтегрується з Redis як брокером завдань, що дозволяє легко масштабувати виконання задач. Завдяки цьому, навіть за високого навантаження на систему, асинхронна обробка запитів виконується швидко і стабільно.

Загалом, використання зазначених інструментів дозволило створити надійну, масштабовану та ефективну серверну частину системи. Кожен з цих інструментів був обраний з урахуванням специфічних вимог проєкту, а їхнє інтегроване використання забезпечило досягнення поставлених цілей.

3.2 Реалізація основних функціональних модулів

Реалізація основних функціональних модулів системи короткострокової оренди житла виконувалась із дотриманням принципів модульності, масштабованості та зрозумілості коду. Основою для розробки стала архітектура, запропонована фреймворком Django, яка дозволяє розподілити логіку додатка між різними рівнями: моделі, представлення (views) та контролери. У цьому підрозділі буде розглянуто ключові модулі, що забезпечують основну функціональність системи.

Одним із центральних модулів системи є модуль аутентифікації та управління користувачами. Реалізація цієї частини системи базувалася на вбудованих можливостях Django, зокрема модулі Django Allauth, який забезпечує просту інтеграцію механізмів реєстрації, авторизації та відновлення паролів [2, 6]. Для підтримки різних типів користувачів, таких як

гості та власники, були створені розширені моделі профілів, які дозволяють зберігати специфічну інформацію для кожного з них.

Ще одним важливим компонентом є модуль управління оголошеннями. Цей модуль відповідає за створення, редагування та видалення оголошень, а також за їхню публікацію. Усі операції з оголошеннями реалізовані за допомогою REST API, створеного на основі Django Rest Framework. Особливістю реалізації є використання серіалізаторів, які дозволяють ефективно перетворювати дані між форматами JSON і моделями бази даних [4]. Крім того, для забезпечення гнучкої фільтрації та пошуку оголошень реалізовано спеціалізовані фільтри, що дозволяють користувачам швидко знаходити необхідну інформацію. Нижче наведено приклад серіалізатору для отримання списку оголошень (рис. 3.1).

```
class GetAllPropertiesSerializer(serializers.ModelSerializer):
    class Meta:
        model = Property
        fields = ("id", "name", "price_per_night", "max_guests", "rating")
```

Рисунок 3.1 – Приклад серіалізатора

Модуль бронювання житла також займає ключове місце в системі. Він забезпечує можливість бронювання обраного житла на конкретні дати. Реалізація цього модуля передбачала перевірку доступності житла на вибрані дати, використовуючи механізми транзакцій PostgreSQL для уникнення конфліктів. Додатково був реалізований механізм оповіщення користувачів про успішне бронювання за допомогою електронної пошти.

Особливу увагу було приділено модулю асинхронних задач, який реалізовано на основі Celery. Цей модуль забезпечує виконання фонових операцій, таких як надсилання електронних листів або оновлення статусів оголошень. Завдяки використанню Redis як брокера завдань, система здатна ефективно обробляти великий обсяг запитів у режимі реального часу [8]. Нижче наведено приклад асинхронної здачі для відправки електронного листа (див. рис. 3.2).

```
@shared_task()
def send_email(subject: str, message: str, to: str):
    send_mail(subject, message, settings.DEFAULT_FROM_EMAIL, [to])
```

Рисунок 3.2 – Приклад асинхронної задачі

Для кожного модуля були розроблені окремі тести, які перевіряють його функціональність та відповідність вимогам. Усі тести виконувались у рамках інтегрованого середовища тестування Django, що дозволяє забезпечити стабільність і надійність системи. Нижче наведено приклад тесту для API-ендпоінта для отримання всіх оголошень (рис. 3.3).

```python
class TestGetAllPropertiesView(APITestCase):
    def setUp(self):
        self.url = "/properties/"
        self.user = User.objects.create_user(username="test", password="test", email="test@example.com")
        self.client.force_authenticate(user=self.user)
        self.property = Property.objects.create(
            name="Test Property",
            description="Test Description",
            address="Test Address",
            price_per_night=100.0,
            max_guests=5,
            host=self.user,
            rating=4.5,
        )

    def test_get_all_properties(self):
        response = self.client.get(self.url)
        self.assertEqual(response.status_code, second: 200)
        self.assertEqual(
            response.data,
            second: [
                {
                    "id": self.property.id,
                    "name": "Test Property",
                    "price_per_night": 100.0,
                    "max_guests": 5,
                    "rating": 4.5,
                }
            ],
        )

    def test_get_all_properties_unauthenticated(self):
        self.client.force_authenticate(user=None)
        response = self.client.get(self.url)
        self.assertEqual(response.status_code, second: 401)
        self.assertEqual(response.data, second: {"detail": "Authentication credentials were not provided."})

    def test_get_all_properties_invalid_method(self):
        response = self.client.post(self.url)
        self.assertEqual(response.status_code, second: 405)
        self.assertEqual(response.data, second: {"detail": 'Method "POST" not allowed.'})

    def test_get_all_properties_invalid_url(self):
        response = self.client.get("/properties/invalid/")
        self.assertEqual(response.status_code, second: 404)
        self.assertEqual(response.data, second: {"detail": "Not found."})
```

Рисунок 3.3 – Приклад тестів

Загалом, реалізація основних модулів системи була проведена з урахуванням вимог до високої продуктивності, стабільності та масштабованості. У результаті вдалося створити інтегровану інфраструктуру, яка забезпечує надійний та зручний доступ до функціональності системи для всіх категорій користувачів.

3.3 Конфігурація середовища

Процес налаштування середовища є одним із найважливіших етапів у розробці програмних систем, оскільки саме він визначає стабільність, масштабованість та зручність у підтримці проєкту. У нашій системі короткострокової оренди житла було застосовано комплексний підхід до конфігурації середовища, який базується на використанні сучасних інструментів контейнеризації, баз даних і фонових обчислень. Такий підхід дозволяє спростити управління системою та забезпечити її високу продуктивність навіть за значного навантаження.

3.3.1 Налаштування Docker та Docker Compose

Docker є одним із найбільш популярних інструментів для створення ізольованих середовищ розробки, тестування та продакшну. Завдяки Docker ми можемо упакувати всі необхідні залежності, бібліотеки та конфігурації в один контейнер, який можна легко розгортати на будь-якому сервері. Це забезпечує стабільність роботи незалежно від операційної системи або налаштувань хосту. Використання Docker дозволяє уникнути проблем із сумісністю програмного забезпечення, що часто виникають під час розробки.

У нашому проєкті був створений Dockerfile, який чітко визначає, які залежності потрібні для роботи серверної частини. Описана конфігурація

забезпечує автоматичну установку всіх необхідних пакетів, таких як Python і бібліотеки для роботи з Django, а також підготовку середовища для запуску вебсервера. Крім того, робоча директорія налаштовується таким чином, щоб розробники могли легко тестувати локальні зміни коду без необхідності переписувати конфігурації.

Детально ознайомитися з Dockerfile можна в додатку А.

Однак для координації роботи кількох сервісів, таких як вебсервер, база даних і брокер задач, одного Docker-контейнера недостатньо. Для цього використовується Docker Compose, який дозволяє одночасно запускати кілька контейнерів і автоматизувати їх взаємодію. У нашому docker-compose.local.yml описано всі основні сервіси, включаючи вебсервер на основі Django, базу даних PostgreSQL, Redis для асинхронної обробки задач і Celery Worker. Це забезпечує легкість у керуванні всіма компонентами системи.

Детально ознайомитися з файлом docker-compose.local.yml можна в додатку Б.

3.4 Процес тестування

Тестування є важливим етапом розробки програмного забезпечення, який дозволяє виявити помилки, забезпечити стабільність системи та підтвердити відповідність функціональних і нефункціональних вимог. У процесі тестування нашої системи короткострокової оренди житла було застосовано комплексний підхід, який включав як автоматизоване, так і ручне тестування. Такий підхід дозволив забезпечити високу якість програмного продукту та мінімізувати ризики появи помилок у реальному середовищі.

Одним із основних напрямків тестування стало використання юніт-тестів, які дозволяють ізолювати окремі компоненти системи та перевірити їхню коректність. Юніт-тести були розроблені для основних модулів системи, включаючи обробку запитів API, функції створення оголошень і механізми бронювання. У процесі тестування використовувався вбудований у Django

фреймворк для тестування, який забезпечує просту інтеграцію тестів у середовище розробки. Це дозволило перевірити функціональність кожного модуля в ізольованих умовах і виявити можливі проблеми на ранніх етапах розробки.

Іншим важливим аспектом стало тестування інтеграції між компонентами системи. Оскільки система включає кілька взаємозалежних сервісів, таких як Django, PostgreSQL, Redis і Celery, було необхідно переконатися, що їх взаємодія є коректною. Для цього були створені інтеграційні тести, які перевіряли роботу всієї системи в комплексі, включаючи обробку запитів, збереження даних у базі даних і виконання асинхронних задач. Таке тестування дозволило виявити потенційні проблеми у взаємодії компонентів та забезпечити їхнє узгоджене функціонування.

Окрему увагу було приділено тестуванню API. Для цього використовувався інструмент Postman (рис. 3.4), який дозволяє надсилати HTTP-запити до API та перевіряти їхні відповіді. Тестування API включало перевірку коректності авторизації, створення та отримання оголошень, а також функцій фільтрації та пошуку. Усі ці операції були перевірені як на правильність виконання, так і на відповідність очікуваним результатам.

Для забезпечення автоматизації тестування використовувався фреймворк Pytest, який є потужним інструментом для виконання тестів. Pytest дозволяє створювати як прості, так і складні тестові сценарії, а також надає можливість легко запускати всі тести одразу. Крім того, автоматизація дозволила значно скоротити час на перевірку функціональності системи, особливо на етапі внесення змін у код.

У процесі тестування також були перевірені нефункціональні вимоги системи, такі як продуктивність і безпека. Наприклад, для перевірки продуктивності було виконано тестування навантаження, яке включало створення тисяч одночасних запитів до сервера. Це дозволило переконатися, що система може витримувати високі навантаження без зниження швидкості роботи. Щодо безпеки, були проведені перевірки захисту даних, зокрема правильність обробки токенів авторизації та запобігання SQL-ін’єкціям.

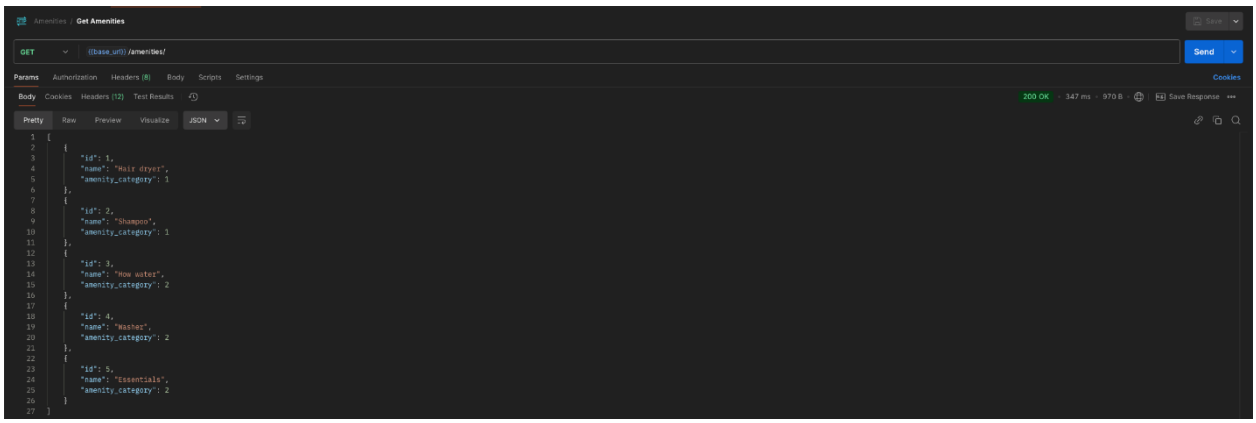


Рисунок 3.4 – Приклад використання Postman

Загалом процес тестування забезпечив виявлення та усунення більшості потенційних проблем на етапі розробки, що значно підвищило якість системи. Завдяки систематичному підходу до тестування вдалося створити стабільну та надійну серверну частину, яка відповідає всім заявленим вимогам і готова до використання в реальному середовищі.

3.5 Керівництво власника житла

Процес взаємодії користувача із системою короткострокової оренди житла базується на інтуїтивно зрозумілих процедурах створення оголошень та їх бронювання. Це керівництво надає детальний опис кроків, які користувач виконує для реалізації основних функціональних можливостей системи. Такий підхід забезпечує не лише ефективну роботу, а й комфортне використання сервісу.

3.5.1 Авторизація в системі

Для роботи з системою користувач повинен бути авторизованим у ній. Це передбачає, що перед виконанням будь-якого запиту користувач отримує токен авторизації через API-ендпойнт /api/auth-token/. Користувач має

відправити POST-запит на цей ендпойнт з полями username та password у тілі запиту. Нижче приведено приклад цього запиту (рис. 3.5).

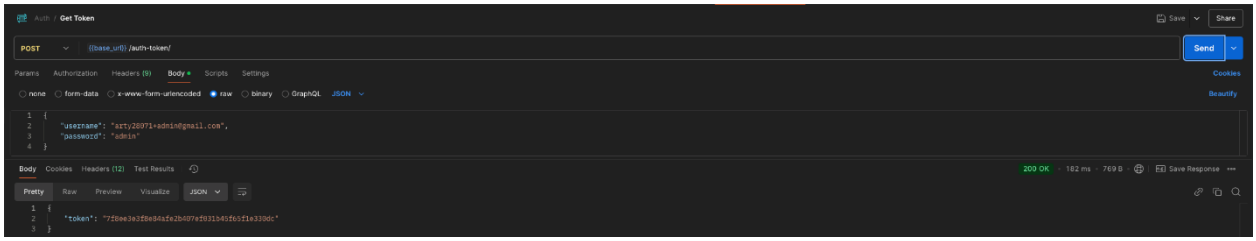


Рисунок 3.5 – Приклад запиту на отримання токену авторизації

Вразі відправки неправильних даних, у відповіді повернеться повідомлення з деталями (рис. 3.6).

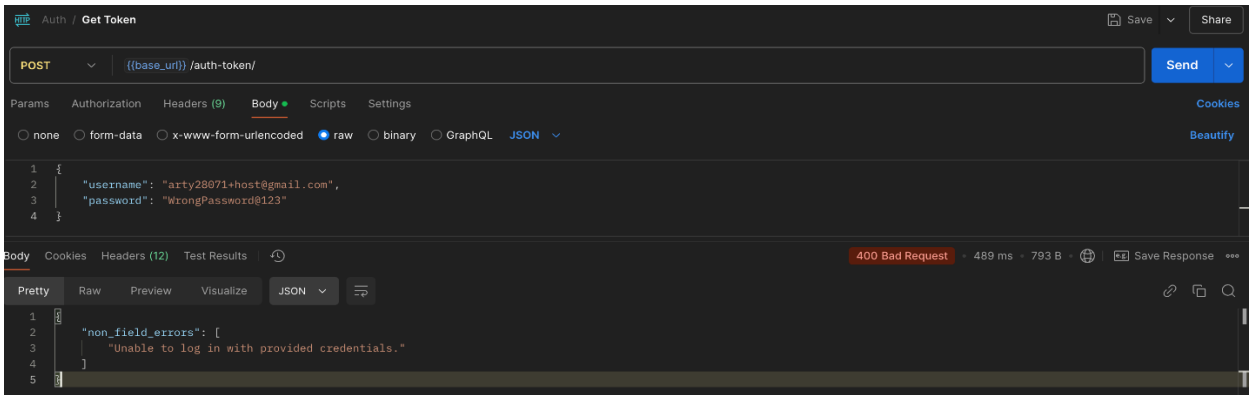


Рисунок 3.6 – Приклад запиту на отримання токену авторизації з неправильними даними

Після цього токен додається до заголовків запиту як Authorization: Token<token> для підтвердження прав доступу.

3.5.2 Створення оголошення

Процес створення оголошення виконується через POST-запит до ендпойнта /api/properties/. У тілі запиту передаються дані у форматі JSON, які містять назву оголошення, його опис, вартість за ніч, місто та країну. Нижче наведено приклад тіла запиту (див. рис. 3.7).

{"title": "Квартира в центрі Києва", "description": "Затишна квартира з усіма зручностями", "price_per_night": 800, "city": "Київ", "country": "Україна"}

Рисунок 3.7 – Приклад тіла запиту на створення оголошення

Система виконує перевірку коректності даних, зокрема наявності обов’язкових полів та допустимих значень, таких як числове значення для поля price_per_night. У разі успішної обробки запиту сервер повертає статус 201 Created та дані створеного оголошення у відповіді. Якщо виникає помилка валідації, сервер повертає статус 400 Bad Request із відповідними повідомленнями про помилки. Нижче наведено приклад відповіді з повідомленнями про помилки (див. рис. 3.8).

{"price_per_night": ["Ensure this value is greater than or equal to 1.0."]}

Рисунок 3.8 – Приклад відповіді з повідомленнями про помилки

3.6 Керівництво користувача-гостя

Для бронювання житла користувач також надсилає авторизований запит, але цього разу до ендпойнта /api/bookings/. Процес бронювання передбачає обрання користувачем дат початку та завершення бронювання, а також ідентифікатора оголошення. Нижче наведено приклад тіла запиту (див. рис. 3.9).

{
"property_id": 12,
"start_date": "2024-12-01",
"end_date": "2024-12-10"
}

Рисунок 3.9 – Приклад запиту для бронювання житла

Після отримання запиту сервер перевіряє доступність житла на зазначені дати. Якщо житло вже заброньоване на ці дати, сервер повертає відповідь зі

статусом 400 Bad Request та повідомленням, що об’єкт недоступний. У разі успішного бронювання сервер відповідає статусом 201 Created та надає інформацію про створене бронювання, включаючи ідентифікатор бронювання, обрані дати та дані про об’єкт.

Важливо зазначити, що всі запити до API обробляються асинхронно. Наприклад, після успішного створення оголошення або бронювання сервер ініціює фонову задачу через Celery для надсилання підтвердження користувачеві електронною поштою. Це гарантує, що користувач отримує актуальну інформацію про виконані дії без затримок у роботі API.

Таким чином, користувач може взаємодіяти з сервером через стандартизовані HTTP-запити до API. Такий підхід забезпечує високу гнучкість та зручність у використанні серверної частини системи, дозволяючи легко інтегрувати її з будь-яким інтерфейсом або зовнішньою системою.

ВИСНОВКИ

У ході виконання даної роботи було розроблено серверну частину системи короткострокової оренди житла, яка реалізована за допомогою фреймворку Django. Основні функціональні можливості серверної частини включають створення та редагування оголошень, бронювання житла, аутентифікацію користувачів, а також обробку асинхронних задач. Система відповідає сучасним вимогам до вебдодатків, забезпечуючи високу продуктивність, надійність та безпеку даних користувачів.

У відповідності з метою кваліфікаційної роботи було реалізовано серверну частину системи оренди житла з використанням наступних технологій:

- Django для створення API та управління бізнес-логікою;
- PostgreSQL для зберігання даних про оголошення, користувачів та бронювання;
- Celery та Redis для обробки асинхронних задач, таких як надсилання повідомлень користувачам;
- Docker для ізоляції середовища та забезпечення простоти розгортання системи.

У відповідності з поставленими задачами були виконані наступні етапи створення серверної частини:

- сформовані вимоги до системи, включаючи функціональні (створення оголошень, бронювання, аутентифікація) та нефункціональні (масштабованість, надійність, безпека), а також проведено аналіз аналогічних систем;
- спроєктована та побудована архітектура серверної частини (створені діаграми варіантів використання, діяльності, послідовності та розгортання, які детально описують логіку та інфраструктуру системи);

- реалізовано ключові модулі серверної частини з використанням REST API, налаштована інтеграція з базою даних, а також забезпечено обробку асинхронних запитів;
- протестована робота системи, включаючи юніт-тести для перевірки коректності основних функцій, інтеграційне тестування для перевірки взаємодії між компонентами та тестування навантаження для оцінки продуктивності системи.

Результатом виконаної роботи стала серверна частина системи, яка відповідає всім заявленим вимогам. Вона забезпечує гнучкість, масштабованість та безпеку, створюючи міцну основу для подальшого розвитку й інтеграції з клієнтською частиною або сторонніми сервісами.

ПЕРЕЛІК ПОСИЛАНЬ

1. Django documentation | Django documentation. Django Project. URL: https://docs.djangoproject.com/en/5.1/ (дата звернення: 10.07.2024).

2. Gagliardi V. Authentication and authorization in the django REST framework. Decoupled django. Berkeley, CA, 2021. P. 153–171. URL: https://doi.org/10.1007/978-1-4842-7144-5_10 (дата звернення: 20.07.2024).

3. GitHub – cookiecutter/cookiecutter-django: cookiecutter django is a framework for jumpstarting production-ready django projects quickly. GitHub. URL: https://github.com/cookiecutter/cookiecutter-django (дата звернення: 10.07.2024).

4. Home – Django REST framework. Home – Django REST framework. URL: https://www.django-rest-framework.org/ (дата звернення: 01.08.2024).

5. Lightweight django: using REST, websockets, and backbone. O'Reilly Media, 2014. 246 p. (дата звернення: 28.08.2024).

6. Quickstart – django-allauth. URL: https://docs.allauth.org/en/latest/installation/quickstart.html (дата звернення: 05.10.2024).

7. Rubio D. REST services with django. Beginning django. Berkeley, CA, 2017. P. 549–566. URL: https://doi.org/10.1007/978-1-4842-2787-9_12 (дата звернення: 12.09.2024).

8. Walter T. Das Python-Framework django. Kompendium der Web-Programmierung. Berlin, Heidelberg, 2008. P. 447–461. URL: https://doi.org/10.1007/978-3-540-33135-3_22 (дата звернення: 05.10.2024).

ДОДАТОК А

Dockerfile

```
# define an alias for the specific python version used in this file.
FROM docker.io/python:3.12.7-slim-bookworm AS python

# Python build stage
FROM python AS python-build-stage

ARG BUILD_ENVIRONMENT=local

# Install apt packages
RUN apt-get update && apt-get install --no-install-recommends -y \
  # dependencies for building Python packages
  build-essential \
  # psycopg dependencies
  libpq-dev

# Requirements are installed here to ensure they will be cached.
COPY ./requirements .

# Create Python Dependency and Sub-Dependency Wheels.
RUN pip wheel --wheel-dir /usr/src/app/wheels  \
  -r ${BUILD_ENVIRONMENT}.txt


# Python 'run' stage
FROM python AS python-run-stage
```

```
ARG BUILD_ENVIRONMENT=local
ARG APP_HOME=/app

ENV PYTHONUNBUFFERED=1
ENV PYTHONDONTWRITEBYTECODE=1
ENV BUILD_ENV=${BUILD_ENVIRONMENT}

WORKDIR ${APP_HOME}



# devcontainer dependencies and utils
RUN apt-get update && apt-get install --no-install-recommends -y \
  sudo git bash-completion nano ssh

# Create devcontainer user and add it to sudoers
RUN groupadd --gid 1000 dev-user \
  && useradd --uid 1000 --gid dev-user --shell /bin/bash --create-home dev-user \
  && echo dev-user ALL=\(root\) NOPASSWD:ALL > /etc/sudoers.d/dev-user \
  && chmod 0440 /etc/sudoers.d/dev-user



# Install required system dependencies
RUN apt-get update && apt-get install --no-install-recommends -y \
  # psycopg dependencies
  libpq-dev  \
  wait-for-it \
  # Translations dependencies
  gettext \
  # cleaning up unused files
  && apt-get purge -y --auto-remove -o
```

```
APT::AutoRemove::RecommendsImportant=false \
  && rm -rf /var/lib/apt/lists/*

# All absolute dir copies ignore workdir instruction. All relative dir copies are wrt to the workdir instruction
# copy python dependency wheels from python-build-stage
COPY --from=python-build-stage /usr/src/app/wheels  /wheels/

# use wheels to install python dependencies
RUN pip install --no-cache-dir --no-index --find-links=/wheels/ /wheels/* \
  && rm -rf /wheels/

COPY ./compose/production/django/entrypoint /entrypoint
RUN sed -i 's/\r$//g' /entrypoint
RUN chmod +x /entrypoint

COPY ./compose/local/django/start /start
RUN sed -i 's/\r$//g' /start
RUN chmod +x /start


COPY ./compose/local/django/celery/worker/start /start-celeryworker
RUN sed -i 's/\r$//g' /start-celeryworker
RUN chmod +x /start-celeryworker

COPY ./compose/local/django/celery/beat/start /start-celerybeat
RUN sed -i 's/\r$//g' /start-celerybeat
RUN chmod +x /start-celerybeat

COPY ./compose/local/django/celery/flower/start /start-flower
```

```
RUN sed -i 's/\r$//g' /start-flower
RUN chmod +x /start-flower


# copy application code to WORKDIR
COPY . ${APP_HOME}

ENTRYPOINT ["/entrypoint"]
```

ДОДАТОК Б

Docker-compose.local.yml

```
volumes:
  airbnb_clone_backend_local_postgres_data: {}
  airbnb_clone_backend_local_postgres_data_backups: {}
  airbnb_clone_backend_local_redis_data: {}

services:
  django: &django
    build:
      context: .
      dockerfile: ./compose/local/django/Dockerfile
    image: airbnb_clone_backend_local_django
    container_name: airbnb_clone_backend_local_django
    depends_on:
      - postgres
      - redis
      - mailpit
    volumes:
      - .:/app:z
    env_file:
      - ./.envs/.local/.django
      - ./.envs/.local/.postgres
    ports:
      - '8000:8000'
    command: /start

  postgres:
```

```
  build:
    context: .
    dockerfile: ./compose/production/postgres/Dockerfile
  image: airbnb_clone_backend_production_postgres
  container_name: airbnb_clone_backend_local_postgres
  volumes:
    - airbnb_clone_backend_local_postgres_data:/var/lib/postgresql/data
    - airbnb_clone_backend_local_postgres_data_backups:/backups
  env_file:
    - ./.envs/.local/.postgres

mailpit:
  image: docker.io/axllent/mailpit:latest
  container_name: airbnb_clone_backend_local_mailpit
  ports:
    - "8025:8025"

redis:
  image: docker.io/redis:6
  container_name: airbnb_clone_backend_local_redis

  volumes:
    - airbnb_clone_backend_local_redis_data:/data



celeryworker:
  <<: *django
  image: airbnb_clone_backend_local_celeryworker
  container_name: airbnb_clone_backend_local_celeryworker
  depends_on:
```

```
    - redis
    - postgres
    - mailpit
  ports: []
  command: /start-celeryworker

celerybeat:
  <<: *django
  image: airbnb_clone_backend_local_celerybeat
  container_name: airbnb_clone_backend_local_celerybeat
  depends_on:
    - redis
    - postgres
    - mailpit
  ports: []
  command: /start-celerybeat

flower:
  <<: *django
  image: airbnb_clone_backend_local_flower
  container_name: airbnb_clone_backend_local_flower
  ports:
    - '5555:5555'
  command: /start-flower
```