МАГІСТЕРСЬКА РОБОТА

МР. ІПм - 45.00.00.000 ПЗ

Група ІПм-23-2

Королевич Ігор

2024

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

Королевич Ігор Романович

(прізвище, ім'я, по батькові)

УДК 004.942

(індекс)

МАГІСТЕРСЬКА РОБОТА

Моделі, методи та алгоритми управління онлайн-бронюванням житла на основі Spring Boot та Stripe API з інтеграцією Docker та Telegram API

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

Королевич І.Р.

(підпис, ініціали та прізвище здобувача освітнього ступеня)

Науковий керівник Григорчук Любомир Іванович, к.т.н., доцент

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Допущено до захисту
Завідувач кафедри
доц. Бандура В.В
(посада) (підпис) (дата) (ініціали та прізвище)

Нормоконтроль

доц. Вовк Р.Б.
(посада) (підпис) (дата) (ініціали та прізвище)

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

Івано-Франківськ – 2024

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

Ступінь вищої освіти магістр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою ***ІПЗ***

доц. В.В. Бандура

“ 04 ” вересня 2024 р.

ЗАВДАННЯ
НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Королевич Ігор Романович

(прізвище, ім’я, по-батькові)

1. Тема магістерської роботи **“** Моделі, методи та алгоритми управління онлайн-бронюванням житла на основі Spring boot та Stripe API з інтеграцією Docker та Telegram API**”**

керівник проекту (роботи) Григорчук Любомир Іванович, к.т.н., доцент

затверджені наказом закладу вищої освіти від **“ 22 ”** листопада 2024 р. № 781/7

2. **Строк подання студентом проекту (роботи)** 15 грудня 2024 р.

3. ***Вихідні дані до проекту (роботи)*** Архітектура, формальний опис та алгоритми функціонування систем оренди житла користувачами

4. ***Зміст розрахунково - пояснювальної записки (перелік питань, які потрібно розробити)***

1. Аналіз вимог: Аналіз функціональних вимог
2. Аналіз сучасних технологій для автоматизації
3. Аналіз проблем управління орендою
4. Формулювання вимог та алгоритмів функціонуання системи
5. Програмна реалізація рішення

5. ***Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)***

1. Архітектура проекту (рис. 3.1, ст. 47)
2. Процес оплати приміщення(рис. 3.2, ст. 48)
3. Взаємодія з телеграмом(рис. 3.3, ст. 50)
4. Взаємодія з контроллером(рис. 3.4, ст. 52)
5. Логування(рис. 3.5, ст. 59)

6. **Консультанти розділів проекту (роботи)**

Розділ	Консультант	Підпис, дата
Перевірка на плагіат	доц. к.т.н. Вовк Р. Б.	

7. **Дата видачі завдання** 04 вересня 2024 р.

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назви етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Аналіз вимог: Аналіз функціональних вимог	01.10.2024	виконано
2	Аналіз сучасних технологій для автоматизації	12.10.2024	виконано
3	Аналіз проблем управління орендою	25.10.2024	виконано
4	Формулювання вимог та алгоритмів функціонування системи	05.11.2024	виконано
5	Програмна реалізація рішення	22.11.2024	виконано
6	Затвердження пояснювальної записки роботи завідувачем кафедри	15.12.2024	виконано

Студент – магіст _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

Магістерська робота: 89 с., 6 рис., 45 джерел.

Тема: Моделі, методи та алгоритми управління онлайн-бронюванням житла на основі Spring boot та Stripe API з інтеграцією Docker та Telegram API.

Об'єкт дослідження: є відомі програмні рішення управління орендою житла та новітні технології й алгоритми які можуть їх покращити.

Мета роботи: удосконалення платформи на основі spring яка автоматизує процес оренди житла і буде зручною в користуванні іншими користувачами або бізнесам які захочуть використати данне програмне рішення. Дана платформа повинна високий рівень безпеки, високий рівень гнучкості й має достатній рівень можливостей які будуть виділяти платформу на фоні інших програмних рішень.

Предмет дослідження: є моделі, методи та алгоритми для управління орендою житла, реалізація платформи на основі сучасних фреймворків та сервісів, таких як Spring Boot, PostgreSQL, AWS, Telegram API, Stripe API.

Результати дослідження: створення платформи для користувачів та адміністраторів, яка дозволила ефективно додавати, переглядати й змінювати об'єкти житла та удосконаленя новтніх способів у взаємодії з платформою.

Висновок: при виконанні магістерської роботи розроблено платформу для бронювання житла, яка вдосконалює сервіс оренди житла. Данна платформа забеспечує оптимальне ціноутворення для орендодавців та клієнтів, пошук варіантів житла на основі вподобань людей, пошук оптимальних маршрути до місця оренди нерухомості.

ПЛАТФОРМА ДЛЯ БРОНЮВАННЯ ЖИТЛА, STRIPE API, TELEGRAM API, УПРАВЛІННЯ ОРЕНДОЮ, БЕЗГОТІВКОВІ ПЛАТЕЖІ, SPRING BOOT.

ANNOTATION

Master's work: 89 p., 6 fig. 45 sources.

Topic: models, methods and algorithms for managing online accommodation booking based on Spring boot and Stripe API with Docker and Telegram API integration

Object of research: an accommodation booking system that allows users to make reservations, check real-time availability of real estate, and make cashless payments.

Purpose: to improve the spring-based platform that automates the rental process and will be easy to use for other users or businesses that want to use this software solution. This platform should have a high level of security, a high level of flexibility and a sufficient level of features that will distinguish the platform from other software solutions.

Subject of research: models, methods, and algorithms for managing rental housing, implementation of the platform based on modern frameworks and services such as Spring Boot, PostgreSQL, AWS, Telegram API, Stripe API.

Research results: the creation of a platform for users and administrators that will allow them to efficiently add, view and modify housing objects and the addition of the latest ways to interact with the platform that distinguishes it from other off-the-shelf solutions.

Conclusion: during my master's thesis, I developed a booking platform that improves the rental service. This platform provides optimal pricing for landlords and clients, search for housing options based on people's preferences, and search for optimal routes to the rental property.

HOUSING BOOKING PLATFORM, STRIPE API, TELEGRAM API, PROPERTY MANAGEMENT, CASHLESS PAYMENTS, SPRING BOOT.

ЗМІСТ

Стр.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API (Application Programming Interface) – інтерфейс прикладного програмування, що надає повний набір інструментів для створення програмного забезпечення.

Stripe API – платіжний інтерфейс для безпечних онлайн-транзакцій.

HTTPS (HyperText Transfer Protocol Secure**)** – протокол безпечної передачі даних.

TLS (Transport Layer Security) – забезпечує безпечну передачу даних.

AWS (Amazon Web Services) – хмарна платформа, яка надає можливості для зберігання даних, обчислювальних ресурсів та інших послуг.

Telegram API – API бібліотека для надсилання сповіщень в платформі телеграм.

JWT (JSON Web Token) – стандарт передачі інформації, що використовується для автентифікації та обміну даними що забезпечує безпеку при вході на платформи.

Docker – для контейнеризації додатків, що спрощує їх розгортання та управління.

Spring Boot – фреймворк для розробки та запуску Java-додатків із мінімальною конфігурацією.

Spring Data JPA – модуль Spring, який спрощує роботу з базами даних за допомогою об'єктно-реляційного відображення у виді обєктів (ORM).

JUnit – бібліотека для створення повторюваних тестів у Java-програмах.

Liquibase – інструмент для версіонування баз даних та відстеження змін у їх структурі.

Lombok – Java-бібліотека для зменшення рутинного коду за допомогою анотацій.

MapStruct – Java-бібліотека для чіткого та автоматичного мапінгу між об’єктами.

Swagger – інструмент для документування REST API.

ВСТУП

Актуальність роботи

У сучасному світі є досить актуальною потреба в новітніх платформах для оренди житла й потреби людей у короткочасних місцях для проживання через високі ціни на купівлю власного житла. Виникає необхідність у створенні нового рішення яке зробить користування платформою легким та автоматизує процеси оренди житла, так що платформа буде корисною для користувачів після сплати за житло при впровадженні новітніх ідей до даної платформи.

Зокрема, актуальність роботи також визначається використання сучасних технологій й фреймворків, які полегшують процес створення платформи такі як Spring Boot, Stripe API, Docker, Telegram API та AWS, дозволяють створювати високопродуктивні та масштабовані платформи, які зможуть обробляти велику кількість запитів.

Порівняння з відовими рішенням проблеми

Сьогодні є багато платформ де можна взяти в оренду житло. В їх список входять, наприклад, Airbnb та Booking. Ці сервіси підходять для міжнародного ринку але у них є недолік - неможливо інтегрувати з іншими платформами і не можна налаштувати під конкретний бізнес. Ще однією проблемою є низький рівень гнучкості автоматизації та обмежені можливості під час користування ними, які зупиняються лише на тому щоб орендувати житло. Створення рішення що дає людям змогу користуватись платформою й поза платформою завдяки телеграм боту та алгоритму який допоможе знайти найкращий маршрут до житла Також платформа дозволяє інтегрувати її з іншими бізнесами через точки доступу які забезпечують доступ до основних алгоритмів платформи захищеним SSH ключем і відкритий код що був розроблений за стандартами REST і Solid що робить його легким для підтримання та читабельним для інших розробників.

Мета і задачі дослідження

Метою магістерської роботи є удосконалення платформи на основі spring яка автоматизує процес оренди житла і буде зручною в користуванні іншими користувачами або бізнесам які захочуть використати данне програмне рішення. Дана платформа повинна високий рівень безпеки, високий рівень гнучкості й має достатній рівень можливостей які будуть виділяти платформу на фоні інших програмних рішень.

Для досягнення мети поставлено наступні задачі:

- провести аналіз існуючих систем автоматизації управління орендою;
- визначити вимоги до програмного забезпечення з урахуванням потреб користувачів і ринкових тенденцій;
- розробити архітектуру і модель програмного рішення;
- реалізувати ключові функції системи, такі як бронювання, управління об’єктами нерухомості та здійснення безпечних транзакцій;
- провести тестування програмного забезпечення та оцінити його ефективність;

Об’єкт дослідження

Об'єктом дослідження є відомі програмні рішення управління орендою житла та новітні технології й алгоритми які можуть їх покращити.

Предмет дослідження

Моделі, методи та алгоритми для управління орендою житла, реалізація платформи на основі сучасних фреймворків та сервісів, таких як Spring Boot, PostgreSQL, AWS, Telegram API, Stripe API.

Методи дослідження

У роботі використано технології аналізу синтезу і моделювання. Для розробки програмного рішення було використано методи spring з інтеграцією spring boot і

технологіями stripe, telegram й junit для тестів з mokito і реляційна база даних, а саме postgresql.

Наукова новизна отриманих результатів

Використано удосконалені новітні алгоритми, які полегшили користування платформою, розроблено спосіб, що дозволить покращити запити до бази щоб покращити роботу з об'єктами. Застосовано потоки й ланцюги фільтрів(filter chains), що дозволило прискорити роботу платформи і зменшило помилки під час навантаження платформи.

Практичне значення отриманих результатів

Удосконалено платформи для оренди житла, яка рекомендована для впровадженя компаніям, які займаються орендою житла. Дане рішення зменшило час розробки, допомогло в оптимізації процесів: безпеці транзакцій та підвищення задоволення клієнтів компанії через додавання фіч, що допоможе користувачам під час пошуку житла і після знаходження житла.

Структура магістерської роботи.

Магістерська робота викладена на 89 сторінках друкованого тексту, який складається з вступу, трьох розділів, висновків, списку використаних джерел (40 найменувань). Робота містить 6 рисунків.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ОСНОВИ СИСТЕМ УПРАВЛІННЯ ОРЕНДОЮ НЕРУХОМОСТІ ТА АВТОМАТИЗАЦІЯ ПРОЦЕСІВ

1.1 Теоретичні відомості моделей управління орендою

1.1.1 Вступ до систем управління орендою

Система управління орендою - це система яка надає автоматизацію для пошуку й оплати житла та спрощує взаємодію орендодавця з клієнтом.

Системи для управління орендою поділяються на різні типи для різних потреб, давайте розглянемо основні два типи:

- веб-платформа – це онлайн сервіси, які надають користувачам веб сайт через який можна взаємодіяти з системою (Airbnb, Booking.com);
- CRM системи – це комплекс інструментів які надають оптимазію для бізнес процесів компанії що є корисною для фінансового прогнозуваня. Ця система надає корисну інформацію до прикладу середній час за який знаходиться клієнт для оренди житла або середня вартість оренди конкретного житла (AppFolio, Buildium);

Ці системи суттєво відрізняються функціональностю та орієнтуються на різні

аудиторію CRM для бізнесів, а веб платформи для користувачів. В них застосовується різні підходити до автоматизації процесів й використовують різні технології. Давайте розглянемо ці підходити та технології на основі двох популярних систем Booking яка відноситься до веб-платформи та AppFolio яка відноситься до CRM системи.

Booking це одна з найпопулярніших платформ для оренди житла. Він орієнтований на готелі, апартаменти, будинки й інші типи житла. Одними з ключових напрямків цієї платформи є автоматизації обробки та пошуку великого обсягу даних. Система використовує алгоритм Elasticsearch для обробки мільйону запитів. Синхронізація даних між клієнтською стороною і серером відбувається через Restful Api і використовує в собі технології Google BigQuery та Apache Spark що дозволяє обробити велику кількість даних за короткий час.

З плюсів даної платформи можна зазначити такі процеси автоматизації як підтрика клієнтів за допомогою штучного інтелекту, використання алгоритму Elasticsearch що пришвидчує обробку великої кількості даних, високий рівень захисту даних через використання алгоритмів шифрування AES-256 та хмарних сервісів Amazon.

З мінусів платформи є складність у пошуку житла, навіть зазначення потрібних характеристиків не гарантує що платформа надасть те що потрібно користувачу. Високі комісійні збори для орендодавців та готелів також є проблемою даного сервісу вони можуть досягати до 15-20% від суми бронювання що є сутєвою проблемою для малого бізнесу. Також платформа має недієвий захисту від фейкових відгуків, конкуренти можуть скористатися сервісами які залищуть негативні відгуки які негативно повпливають на іншого орендодавця. Скасування оренди не поверне повну суму яку користувач оплатив що створить для нього незручності при зміні планів.

Сама платформа використовує java + spring для бекенду, реляційні бази даних, платіжну систему Stripe та хмарний сервер від AWS.

AppFolio це одна з найпопулярніших систем для бізнесів яка спрямована на автоматизіції процесів управління нерухомості.

З плюсів системи можна зазначити автоматизацію процесів фінансових операцій таких як нарахування плати за оренду та нагадує про платежі, автоматична створення звітів за кожне житло. Ключова особливість AppFolio є оптимізація запитів орендарі, система зберігає історію запитів і дозволяє менеджерам швидко відреагувати на запити. Систему можна налаштувати для відправки нагадувань про продовження договору.

З мінусів системи можна зазначити високу вартість підписки що може бути проблемою для малих бізнесів. AppFolio є складною для інтеграції системою що збільшує час розробки платформ на осноі цієї системи, забираючи час та ресурси компанії.

Система використовує Rubi та Java + spring для бекенду, реляційну базу даних PostgreSQL, Сервер Amazon та використовує шифрування даних за допомогою AES-

256.

Проблеми наявних систем є складність для нових користувачів та складність у знаходженні потрібного житла, яка займає багато часу на пошуки, і часто пропонує варіанти, які не підходять користувачу. Для цього потрібно розробити алгоритм який показує житла, які найбільше підходять його вподобанням подібно до рекомендацій у популярним соцмережах.

Нижче надано опис основних технологій які застосовані у цих системах, а також теоретичні знання для них.

- Spring Boot: популярний фреймворк який оптимізує більшість процесів створення вебдодатків завдяки використанню патернів таких як Spring mvc Spring data web й інших і зберігання об'єктів в бінах та надає доступ до великої кількості бібліотек до прикладу tomcat що спрощує налаштування і запуску сервера [1].
- Spring Data JPA: бібліотека для роботи з базою даних яка аптомізовує роботу з базою даних прибираючи необхідність в обробці логіки jpa, автоматично створюючи імплементацію для нього [3].
- Docker: технологія для контейнеризації програмного рішення яка в рази спрощує розгортання системи на сервері, працює на основі Лінукс ядра [21].
- Stripe API**:** бібліотека яка дозволяє керувати транзакціями через їхню систему що спрощує керування фінансами на сайті [32].
- PostgreSQL: реляційна база даних яка дозволяє швидко отримувати й змінювати дані [26].

1.2 Теоретичні відомості про проблеми при автоматизації процесів та їх вирішення.

1.2.1 Вступ до автоматизації

Автоматизація бронювання нерухомості є ключовим етапом у створенні сучасних платформ для управління орендою житла. Завдяки автоматизації досягається оптимізація часу, зменшення людських помилок та підвищення

ефективності операцій.

1.2.2 Основні етапи

Основними етапами для автоматизації оренди житла є:

Перевірка доступність житла в режимі реального часу.

Необхідно перевіряти в реальному часу доступність житла коли користувач пробує зробити оренду, щоб уникнути моментів коли дві людини одночасно зробили оренду житла або коли користувач спробує зробити оренду житла в дні які вже зарезервовані для іншого користувача.

Щоб вирішити цю проблему системи оренди житла використовують запит до бази даних що в першу чергу допоможе розв'язувати проблему перевірки чи житло вже орендоване застосовуючи принцип черги що гарантує що дві людини не зможуть орендувати житло одночасно. Також необхідно кешувати дані для того, щоб при обємному кількості запиті система могла швидко надати інформацію користувачам без втрати часу на опрацювання всіх запросів користувачів. Коли користувач зробив вибір житла необхідно завершити його бронювання змінюючи статус житла в базі даних.

Алгоритми обробки запитів повинні містити в собі наступне:

Як вже було зазначено алгоритми повинні обробляти кейси коли кілька користувачів роблять оренду одночасно в таких випадках допомогає технологія реляційної бази даних як ACID.

ACID грає важливу роль в оптимізації паралельних запитів за допомогою нього платформа може обробляти кілька запитів у базу даних без блокування інших запитів і користувачам не доведеться очікувати на обробку минулих запитів.

ACID поділяється на декілька правил:

- Атомарність

Транзакції зазвичай містять багато операцій. Атомарність гарантує, що жодна транзакція не виконується частково. Або всі операції, задіяні в транзакції, будуть виконані, або жодна з них не буде виконана. Якщо під час виконання однієї з операцій сталася помилка і операція була відхилена, усі інші зміни, внесені в

транзакцію, також будуть відхилені. З урахуванням перебоїв, помилок і збоїв система в кожному випадку повинна бути атомарною. Гарантії атомарності запобігають частковим оновленням бази даних, що насправді може спричинити більше проблем, ніж оновлення всієї бази даних за одну транзакцію.

Прикладом атомарної транзакції є переказ коштів з одного рахунку на інший, який проходить дві операції: зняття коштів з першого рахунку та внесення їх на другий рахунок. Виконання цієї операції в атомарній транзакції забезпечує узгодженість бази даних, а це означає, що якщо будь-яка з операцій виявиться невдалою, кошти не будуть вираховані або зараховані.

- Узгодженість

Згідно з вимогою узгодженості система повинна бути в узгодженому стані до початку транзакції та після її завершення. У той же час під час виконання транзакції можна перебувати в неузгодженому стані, але через інші властивості - атомарності та ізольованості, ця невідповідність не видно за межами транзакції.

Узгодженість гарантує незмінність бази даних: кожен елемент, записаний до бази даних, повинен відповідати всім визначеним правилам, включаючи обмеження, каскади, тригери та будь-яку їх комбінацію. Це запобігає пошкодженню бази даних через неправильні транзакції, але не гарантує правильності транзакцій. Посилальна цілісність забезпечується через асоціацію унікальних ключів і зовнішніх ключів.

Наприклад, переказуючи кошти з одного рахунку на інший, ви можете зняти кошти з першого рахунку, а потім внести їх на другий рахунок. Таким чином, після зняття коштів, але до отримання грошей, система перебуває в неузгодженому стані: на жодному рахунку немає коштів. Однак після завершення транзакції повна сума буде перерахована на другий рахунок (або на перший рахунок, якщо транзакцію скасовано).

- Ізоляція

Ізоляція означає, що жодні проміжні зміни не видно за межами транзакції, доки вона не буде завершена. Питання ізоляції стає дуже важливим, коли багато транзакцій з однаковими даними працюють одночасно. Відповідно до цієї вимоги,

якщо дві транзакції намагаються змінити ті самі дані, одна з транзакцій буде відхилена або призупинена, доки інша транзакція не завершиться.

- Довговічність

Надійність гарантує, що, незалежно від інших проблем, результати завершених транзакцій залишаться незмінними після відновлення системи. Іншими словами, якщо користувач отримує повідомлення про успішне завершення транзакції, то він може бути впевнений, що дані будуть збережені та відновлені в разі збою.

Підтвердження або скасування бронювання.

Важливо коректно обробляти процеси підтвердження або скасування бронювання тому платформа повинна підтвердити бронювання в випадках коли все добре і відхилити якщо сталась якась помилка або коли користувач який здає житло в оренду не має можливості через якусь подію здати житло в оренду в задану дату. Для цього користувача необхідно сповістити крім сайту також в популярних системах сповіщень до прикладу SMS або Telegram, щоб не було ніяких непорозумінь в майбутньому.

Ці алгоритми повині будуватись на таких перевірках як:

Перевірка вірності введених даних до прикладу дата(тривалість проживання), кількість людей які будуть там проживати, їхній вік, та спеціальні умови(до прикладу наявність стоянка) .

Можливість оплати: система повинна обробити платіж користувача до того як підтверджувати замовлення і відмінити платіж у випатку якщо хтось оплатив житло першим на ту саму дату.

Оплата може проходити по різному, до прикладу через різні види карт або навіть через криптовалюту головне. Сюди можуть входити такі способи оплати як кредитні карточки, банківські перекази, emoney тощо.

1.2.3 Алгоритми та методи автоматизації

Давайте тепер розглянемо сучасні системи алгоритмів бронювання:

Такі алгоритми використовують автоматичний пошук найкращого житла за заданим алгоритмом пошуку.

Зазвичай їхні алгоритми для пошуку вільних будинків у базі даних має складність N та подальше сортує їх за різними параметрами пошуку, що може бути дуже довгим процесом та нагружати нашу платформу тому наш алгоритм пошуку житла повинен мати алгометричну складність Log(N).

Також застосовують динамічне програмування яке розв'язує декілька проблем до прикладу заповнення параметрів у таблиці таких як ціна, дата, час або ж кількість що проживає там людей.

Деякі рішення також застосовують аі та машинне навчання, щоб спростити роботу підтримки або підвищити точність прогнозу і покращення обробки даних які своєю чергою дають змогу покращувати алгоритм бронювання або спрогнозувати попити на житло або ж різну корисну інформацію для користувача що надає квартиру в оренду.

Також системи часто застосовують алгоритми розподілу ресурсів що дуже корисно готелям які мають обмежені ресурси такі як кількість доступних номерів або кількість людей що можуть орендувати житло.

1.2.4 Інструменти та методи автоматизації

Системи бронювання нерухомості використовують різні технології для автоматизації процесів. Фреймворк як Spring Boot часто використовують в системах так як він спрощує побудову бекенд частини платформи і підтримує багато функцій запезпечення безпеки. Також більшість систем використовують готові рішення для автоматизації оплати по типу Stripe API що надає додаткову безпеку для користувачів при оплаті житла та дозволяє реалізувати автоматичну денну, помісячну оплату [23].

1.3 Аналіз методів автоматизації процесів управління орендою

Важливо для сервісу мати алгоритми автоматизації процесів управління орендою житла, бо цим самим можна дуже ефективно управляти орендою також він є дуже зручним для користувачів платформи. У наступній частині буде представлено

небанальні методи автоматизації, які пов’язані з орендою нерухомості: бронювання, управління договорами, платежі та інформація для користувачів.

1.3.1 Вступ до теми

Автоматизація управління орендою є частиною орендних платформ, які цінують не тільки орендарі, а й орендодавці, адже на процес оренди йде значно менше часу і зусиль [10].

Процес бронювання, обробка та підпис договорів, платежі та різного роду запити які потрібно автоматизувати.

Платформи з управління нерухомістю використовують безліч видів автоматизацій, щоб мінімізувати вплив людського фактору та помилок, таким чином підвищити ефективність процесів. Це дає прозорість операціям та довіру користувачів до сервісу.

1.3.1.1 Автоматизація оренди

Основні процеси автоматизації в управлінні орендою:

- Автоматизація бронювання: автоматична перевірка, що перевіряє вільні місця та запити на бронювання.
- Управління договорами: підписання, оновлення та моніторинг договорів.
- Автоматизація платежів: впровадження та оновлення платіжних систем для платежів.

Інтеграція сервісів: інформування користувачів, зокрема про підтвердження платежу , статус бронювання, тощо.

1.3.2 Методи управління

Ключовим етапом управління орендою є автоматизація бронювання, яка включає перевірку вільних місць та обробку запитів на оренду. Існує кілька методів, які залежать від сучасних технологій:

1.3.2.1 Оптимізація бази данних

Є багато інших способів, щоб перевірити наявність місць за допомогою запитів до бази даних. Процес кешування прискорує перевірки, дозволяє зменшити навантаження на базу даних.

Метод оптимізації до бази даних , використаний мною є також дуже популярним.

1.3.2.2 Механізми резервування

Як вже було зазначено для авторизації бронювання застосовано ACID що дозволяє уникнути моментів коли дві людини одночасно нажали оренду житла і сталось б що дві людини мають одне замовлення на той самий день система ACID використовує найбільший рівень захисту запитів, а саме систему черг тому це прибере можливість орендувати житло двом користувачам одночасно що своєю чергою з заданими нам вимогами до побудови програмного рішення розподілить ресурси створивши пріоритетність та ефективне розподілення ресурсів.

1.3.3 Автоматизація адміністрування оренди

Також згідно з законом України потрібно створити договір оренди житла і при шляху України до диджиталізації її потрібно зробити її створення автоматичним.

1.3.3.1 Інструменти для автоматизації

При створені договорів є системи генерації документів, які допомагають вводити всі дані в процесі створення документа, наприклад: ім’я, дані про оренду, ціна, термін укладання угоди. Така автоматизація дає змогу швидше підписувати договори та фактор людської помилки є значно нижчий.

1.3.3.2 Виконання та обробка

Можливість електронного підпису за допомогою спеціальних сервісів це також автоматизація. Він дозволяє значно спростити та пришвидити підписання договору, адже не потрібно йти в офіс або обмінюватись паперовими документами.

Автоматизація платежів

Автоматизація обробки платежів – також дуже важливий аспект в управлінні орендою, адже він дозволяє швидко здійснювати та підтверджувати її, а також є можливість відстежувати інформацію та статус бронювання.

.3.4.1 Інтеграція з платіжними системами

Щоб оптимізувати процес оплати при використанні кредитних карточок або банківських переказів підключають різні системи з'єднування з платіжними системами (сервісами) потрібно не обмежувати користувача у виборі способу оплати це відбувається за допомогою різних api.

1.3.4.2 Оплата

У цій роботі також важливою частиною є підтвердження оплати платіжні системи дають відповідь у разі успішної або ж на наоборот не успішної оплати. Також важливою частиною є відмова від оплати у разі скасовування оренди житла.

Додаткові сервіси

Розглянемо також додаткові важливі частини сервісу які потрібно реалізувати в цьому програмному рішенні:

Sms-повідомлення та електронна пошта

Для автоматичного інформування користувачів системи часто використовують sms-системи та системи розсилки електронною поштою. Такі системи автоматично надсилають повідомлення про підтвердження або скасування бронювання, а також нагадують про дату початку оренди.

Штучний інтелект для подальшої оптимізації

Також для покращення автоматизації управління орендою можна застосувати штучний інтелект він може зручно вираховувати корисні дані для користувачів або для визначення оптимальних цін та умов для оренди.

.1 Машинне навчання

При навчанні моделі враховується величезна кількість факторів це може бути як цінові фактори за порою року або економічні тенденції це допоможе надати кращі рекомендації користувачам і зменшити ризики при бронюванні.

1.3.7 Можливості автоматизації лізингу

Та як інформаційні технології розвиваються і їхні способи автоматизації також, логічно очікувати що нові технології будуть більш масштабовані й використовувати блокчейни що надасть прозорість та безпеку у транзакціях.

Використання сучасних технологій покращить точність дій користувача такі як перевірка наявності житла або підписати договір або обробити платежі що підвищить точність і швидкість операцій та спростить інтегрування з іншими програмними рішеннями.

1.4. Висновки до розділу

У розділі було проведено аналіз методів та розглянуто види різних систем для оренди житла показано відмінність між веб-платформою та CRM системи і визничили їхні позитивні і негативні сторони. для вдосконалення роботи систем було розглянуто і визначено технології які там використовуються.

Окрім того проведено досліження проблем які можуть виникнути при автоматизації процесів оренди житла та способи їх вирішення. Часто виникає проблема одночасного запиту двома і більше користуачами. Також було розглянуто складність алгоритмів пошуку у сучасних системах.

РОЗДІЛ 2
ДОСЛІДЖЕННЯ СУЧАСНИХ ТЕХНОЛОГІЙ БЕЗПЕКИ ТА ІНТЕГРАЦІЇ ПЛАТІЖНИХ СИСТЕМ

2.1 Алгоритми безпечного обміну даними

Безпека даних є одним із ключових аспектів сучасних інформаційних технологій, особливо коли йдеться про онлайн платформи що обробляють фінансову інформацію та приватні дані користувачів. У системах управління орендою житла це має велике значення адже користувачі дають конфіденційну інформацію такі як платіжні дані або деталі бронювання. Втрата цієї інформації або їхнє потрапляння до сторонніх осіб може завдати великих збитків для користувачів і компанії яка керує платформою.

Одним з нових рішень, які використовуються, це використання протоколу Безпечного Обміну Даних (SDEP), який надає безпечний обмін даними між орендодавцем та орендарем. Цей протокол дозволяє робити разові ключі для кожної угоди, які самознищуються по завершенні. Це сильно зменшує ризик втрати даних навіть якщо основні ключі викрадені.

1.Види шифрування та хешування

- Шифрування AES, DDES, 3DES можна використати для шифрування великих об'ємів даних, а саме головне зробити це швидко.

Формула цього алгоритму шифрування виглядає ось так

$$0 \leq i < Nb * (Nr + 1) \tag{2.1}$$

де Nb – це 32 біта у слові що шифрується;

Nr – це кожний 4 байт у слові.

- Асиметричне шифрування(RSA, ECC) шифрування в вигляді 2 ключів де 1 має ключ шифрування, а другий ключ розшифрування

Алгоритм шифрування RSA виглядає ось так

$$c = m^e \ mod \ n \quad (2.2)$$

де m – початкове повідомлення;

e – публічний показник (експонента);

n – модуль, що є основою шифрування;

c – зашифроване повідомлення.

2. Захист транспортного рівня

- Криптографічний протокол який захищає дані на траспортному рівні(TLS)
- Протокол для безпечної передачі файлів Secure File Transfer Protocol(SFTP)
- Приватна віртуальна мережа дозволить приватній мережі використовувати публічну мережу через захисний тунель (VPN)

3. Однофакторна аутентифікація

- Однофакторна аутентифікація це аутентифікація просто за допомогою логіну і паролю(1FA)

Середній рівень OSI

1.HTTPS

Це суміш http та TLS-шифрування він забезпечує секретність разом із цілісністю даних що відправляються за допомогою шифрування даних.

Таким чином шифрування даних запобігає отримання цих самих даних третіми особами.

Для того, щоб користувач переконався що сайт справжній і він підключився до сервера що він хотів застосовують ssl-сертифікати.

2. TLS (Transport Layer Security)

Це асиметричне шифрування яке складається з :

- Ініціювання сеансу асиметричного шифрування.
- Автоматичне перемикання на симетричне шифрування для випадків швидкого обміну даних після встановлення з'єднання.

3. WebSocket Secure

Постійне двостороннє з'єднання між клієнтом і сервером забезпечує протокол WSS за допомогою TLS. Для систем що працюють в режимі реального часу це є

обовязковим для таких системи як сповіщення, оренда житла. Щоб закрити вразливості, систему потрібно регулярно оновлювати. Саме тому варто звернути увагу на застосування політики оновлення програмного забезпечення.

Підходи до безпеки та захисту

1. Наскрізне шифрування (End-to-End Encryption).

Спосіб шифрування в якому дані шифруються на пристрої відправника й розшифровуються на пристрої користувача що отримує дані.

- Приклад використання: Месенджери типу Telegram.

- Переваги: захист від недоліку людина в середині.

2. Безпека на рівні додатків.

В самому програмному рішенні повинен бути захист розглянемо захист на його рівні:

- Токенізація: це процес заміни конфіденційних даних на токен який не має значення для людини яка його дістане.

- Обфускація коду: це процес ускладнення для сторонніх осіб аналізу коду.

Використання хмари

Також хорошим рішенням є використати хмарні середовища через те, що вони пропонують багато додаткових інструментів для захисту даних.

Можливості хмарного середовища які забезпечуть безпеку:

- Автоматичне шифрування даних що знаходяться у хмарі. (KMaaS): AWS KMS.

- Добавлення ролей та політики користування.

2.2 Stripe Api: особливості інтеграції для забезпечення безпечних транзакцій

Stripe API - Технологія яка дозволяє здійснювати безпечні транзакції через платформу для управління оренди житла.

Зараз один з найпопулярніших технологій для запровадження системи платежу в програмне рішення воно дозволяє здійснювати транзакції без небезпеки для

користувача що ним користується за допомогою цього сервісу компанії можуть інтегрувати оплату різними способами для своїх необхідностей.

Основні можливості API Stripe

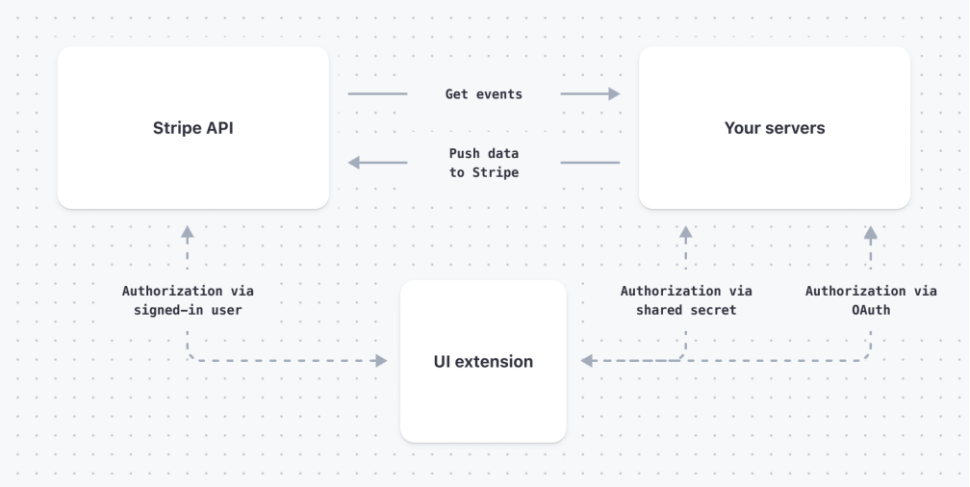


Рис. 2.1. Алгоритм роботи Stripe Api

Для того, щоб додати stripe у платфому можна використати різні способи.

Технологія Stripe дозволяє підтримувати різні способи оплати такі як: кредитні картки, банківські перекази, цифрові гаманці, або ж google pay та apple pay.

- Stripe працює у більш ніж 100 країнах тому не буде проблемою для користувачів з усього світу оплатити бронювання різними валютами.

- Stripe також дозволяє налаштовувати автоматичне знання коштів протягом якогось періоду.

- Адміністрація Stripe допоможе у випадку коли потрібне швидке розв'язання питань з поверненням коштів.

- Автоматичні звіти: stripe дозволяє отримати доступ до платіжної аналітики у випадку коли вам необхідна статистика або її потрібно комусь надати з користувачів.

Як вже не раз було сказало однією з найбільшою перевагою Stripe є її безпека Всі дані користувачів перевіряються і передаються в зашифрованому виді що унеможливлює перехоплення даних сторонніми користувачами. Stripe суворо відповідає всім стандартам PCI DSS, що робить всі взаємодії даних платіжних карт дуже суворо регламентованими.

Також Stripe використовує технологію токенізації тому під час всіх транзакцій їхня інформація буде замінюватись унікальним токеном що своєю чергою знижу ризики для користувачів навіть при витоку даних.

Сам Stripe розроблений для полегшення інтегрування з різними мовами програмування включаючи java джаву цим самим спрощуючи роботу для розробників. Також спрощує роботу з ним сам факт наявності документації що є частою проблемою.

Ключові етапи:

1. Реєстрація і отриманя ключів: щоб почати роботу з Stripe потрібно отримати унікальні ключі також він надає тестові ключі для тестування роботи Stripe api.
2. Імплементація платіжної форми: платформа Stripe надає доступ до їхніх інструментів створення зручних і безпечних інтерфейсів при оплаті користувачами.
3. Обробка відповідей API: сама платформа надає детальну інформацію в якості відповіді на всі транзакції як успішні, так і не успішні.
4. Середовище для тестування: також Stripe надає доступ до всіх потрібних функцій тестування що дозволить мінімалізувати шанс на отримання помилки перед запуском платформи.

Особливості підготовки

Stripe дозволяє налаштувати систему під себе для бізнесу в випадку оренди житла він може автоматизувати процес транзакції та створення віртуальних чеків для користувачів також в ньому можна налаштувати систему нарахування комісій під себе.

Системи захисту від шахрайства

Платформа також надає можливості виявлення шахрайських операцій. Ця система була розроблена самою компанією з використанням Машиного навчання цим самим дозволить виявити аномалії зменшуючи шахрайства і підвищиючи довіру клієнтів до нашої платформи.

Гнучкість.

Stripe допомагає розробникам побудувати платіжний процес під себе це охоплювати таке як вигляд платіжної форми, способи оплати, повідомлення про стан транзакцій це своєю чергою робить його більш гнучкішим для автоматизації бізнес-процесів оплати.

Тепер давайте трішки зачіпимо недоліки використання Stripe api одна з них це комісія за використання цієї платформи що в перспективу зменшать дохід також важливо додати що залежно від регіону можуть виникнути юридичні складнощі з використанням цієї платформи й також платформа буде повністю залежними від неї через те, що всі операції з грошима будуються на цій платформі.

Проте завдяки своїй масштабованості Stripe є однією з найкращих платформ для обробки платежі що сильно виділяє її серед інших платформ не важливо яких розмірів і типів. Масштабованість - це здатність системи справлятись з будь-який обсягами транзакцій без втрати продуктивності системи встановлюючи високі стандарти в цьому.

Всі ці ключові елементи які були обговорені зверху дозволяють працювати з максимальних ефективностей як для малих, так і для великих бізнесів. Він буде однаково ефективно працювати незалежно чи це стартап з малими кількостями транзакцій так і для бізнесів з великими кількостями транзакцій на день чи навіть хвилину через те, що ця система гарантує стабільне та безперебійну обробку даних.

Сама архітектура Stripe працює на хмарній технології що своєю чергою дозволяє зручно і динамічно обробляти й розприділяти ресурси залежно від вхідного навантаження що своєю чергою означає що платформа дозволяє забезпечити швидку обробку платежів. Ця технологія забезпечує високе масштабування додатків.

Ще одна важлива особливість платформи Stripe підтримка мультивалют при транзакціях це є дуже важливою для тих бізнесів які хочуть працювати в декілька країн через те, що Stripe підтримує більше ніж 100 різних валют дозволяючи працювати на різних ринках що собою дуже спрощує обробку міжнародних платежів в різних країнах [44].

Також Stripe підтримує масштабну автоматизацію в процесі масштабування що у майбутньому дозволить не перейматися зросту навантаження на платформу й

уникнути перевантаження через те, що їхній сервіс балансує запити користувачів.

Це все зверху що обговорювали надає велику перевагу для міжнародних великих компаній оскільки це дозволить інтегрування з корпоративними системами що своєю чергою дозволяє гнучко контролювати фінансові операції глобального рівня з можливостями зростання у майбутньому.

Що ще з корисного є для компаній так це можливість моніторингу й прогнозування майбутніх навантажень на систему що своєю чергою дозволить компаніям передбачувати пікові періоди й готуватися до них що є дуже важливим для компаній що працюють у сферах де потрібно комерції або ж використовувати їх, щоб визначити ціни які потрібно виставити на свята й інших святкових транзакцій.

Також з масштабування є важливим момент можливості інтеграції з іншими платформами й сервісами. Доприкладу можна використати Stripe api разом з telegram.

Stripe можна масштабувати до конкретних потреб бізнесу таких як підтримки розробки індивідуальних рішень для оплати або ж до конкретних вимог клієнта. для початку малі компанії можуть застосовувати цей сервіс для обробки малої кількості транзакцій та коли компанії розвивається і збільшується кількість транзакцій тоді Stripe може легко збільшити масштабованість системи що збільшить гнучкість.

Stripe може спокійно змінити способи ведення бізнесу через те, що архітектура сфокусована на розширенні й забезпеченні можливостей для підприємств. Вона дозволить бути конкурентно спроможним для всіх типів бізнесу і дозволяє налаштовувати її під себе у будь-який час [4].

2.3 Протокол передачі даних HTTPS у платіжних системах

Протокол передачі даних HTTPS: є одним з найважливіших елементів в забезпеченні систем платежів, так як обробка конфіденційних даних в ньому є основною вимогою. HTTPS додає ще один важливий рівень безпеки до протоколу HTTP.

Основи HTTPS

SSL/TLS використовується в HTTPS щоб створити зашифрований канал між

клієнтом і сервером, технологія робить неможливим підміну даних чи перехоплення їх між клієнтами. Звісно ж цей протокол вважається найважливішим при передачі фінансових даних, платежів, номерів карток чи реквізитів, особистих даних та паролів.

HTTPS являє собою основний механізм безпеки, завдяки чому передається інформація при здійсненні безпечних транзакцій. Звісно ж всі найсучасніші платіжні системи такі як Stripe, PayPal та інші платіжні шлюзи - потребують використання HTTPS щоб зашифрувати дані. Що виключає певні загрози, наприклад атаки під час яких зловмисники перехоплюють дані (MITM), що б дозволили злочинцю бачити да змінювати дані клієнта.

Як працює HTTPS.

HTTPS використовує криптографічний протокол SSL/TLS, вони ж в свою чергу шифрують дані між сторонами клієнта та сервера. Дане ж з'єднання може забезпечувати:

1. Повне шифрування особистих даних: таких як номер вашої картки в банку чи інші реквізити клієнта які можуть бути потрібними в процессі, повнестю зашифровані дані не можуть бути переглянутими чи зміненими сторонніми особами.

2. Аутентифікація сервера: платіжна система SSL/TLS аутентифікує систему платежів за допомогою своїх сертифікатів, цим же вона гарантує клієнтам підключення до правильного сервера.

3. Забезпечення цілісності даних: гарантується що дані клієнта ніяким чином не були змінені при передачі, цей фактор є надважливим для безпечних транзакцій.

HTTPS у платіжних системах.

Для платіжних систем HTTPS являється тандартною технологією для забезпечення конфіденційності клієнта. Щоразу як клієнт здійснює транзакцію, браузер користувача використовує HTTPS для безпечної передачі його даних.

Як HTTPS захищає дані під час передачі

1. Збереження інформації платежу: всі дані платежа та реквізити такі як термін дії картки CVV та номер картки, перед тим як бути надісланими на сервер ці дані будуть зашифровані. Тобто якщо злочинець навіть зможе перехопити певні дані

транзакції він не зможе їх використати оскільки вони будуть зашифровані.

2. Гарнатія автентичності сервера: у всіх платіжних системах повинні застосовуватись сертифікати SSL/TLS, Дані сертифікати забезпечують підключення до існуючого сервера, а не до можливих шахрайських копій.

3. Захищення даних: HTTPS захищає дані які передаються за допомогою криптографічних ключів, що передаються. В разі якщо дані будуть перехоплені, зловмисник не зможе ніяк ними скористатись оскільки вони повністю зашифровані.

4. Забезпечення захисту від перехоплення даних: будь які дані передаються в зашифрованому вигляді, що унеможливлює будь яке втручання зловмисника таке як спроба зміни чи модифікації даних під час транзакції. Зокрема під час використання відкритих Wi-Fi мереж.

Значення HTTPS для конфіденційності користувачів

HTTPS забезпечує надійний захист даних користувача під час транзакцій**.** Саме тому всі сучасні платіжні системи переходять на використання протоколу HTTPS , для гарантіїї безепеки даних своїх клієнтів. Використання даного протоколу також сприяє довірі клієнтів до платіжної системи оскільки так вони будуть впевнені в захищеності своїх даних

Використання HTTPS для мобільних платежів.

В мобільних технологіях використання HTTPS є надважливим для забезпечення платежів. Тому такі великі платформи як Apple Pay та Google Pay, звісно ж вимагають використання протоколу HTTPS для проведення транзакцій. Плюси HTTPS в таких системах

1. Захищеність даних: HTTPS захищає персональні дані користувачів, і цей фактор є надважливим при роботі з персональними даними.

2. Забезпечення довіри: застосування даного протоколу показує високу надійність системи транзакцій, це в свою чергу сприяє підвищенню довіри користувачів до системи.

3. Підтримка сучасних стандартів безпеки: протокол HTTPS відповідає вимогам таких стандартів як PCI та DSS, цей фактор є обов’язковим для платіжних систем.

Тому звісно ж протокол HTTPS є наважливішим, що забезпечує надійність платежів. Цей протокол забезпечує повний захист даних користувачів таких як номер картки та інших особистих даних які потрібно захистити від сторонніх. Користуючись протоколом HTTPS платіжна система забезпечує цілісність та безпеку даних що сприяє безмежній довірі клієнта до сервісу.

Протокол HTTPS шифрує дані криптографічними протоколами SSL/TLS, що забезпечують канали звязку для клієнта та сервера. Особливістю його є те що дані аутентифікуються. Користувач знає що працює саме з тим сервером до якого звертався , а не з якоюсь шахрайською копією що спробує викрасти дані користувача.

З цього зрозуміло що в сфері фінансів дані повинні бути надійно захищеними усіма способами. Оскільки в разі порушення протоколу дані клієнта чи його фінансові активи можуть бути викрадені. А при задіянні механізмів протоколу HTTPS клієнту гарантується що його дані будуть на сто відсотків захищені, а будь які транзакції пройдуть успішно.

Автентифікація: на відміну від HTTP, HTTPS включає надійну автентифікацію через протокол SSL/TLS. Сертифікат SSL/TLS веб-сайту містить відкритий ключ, який веб-браузер може використовувати для підтвердження того, що документ, надісланий сервером (наприклад, сторінка HTML), має цифровий підпис особи, яка є власником документа. закритий ключ. Якщо сертифікат сервера було підписано загальнодовіреним центром сертифікації (CA) (наприклад, SSL.com), браузер погодиться з тим, що будь-яка ідентифікаційна інформація, що міститься в сертифікаті, була перевірена надійною третьою стороною.

Сайти HTTPS також можна налаштувати для взаємної автентифікації, коли веб-браузер надає облікові дані клієнта, які ідентифікують користувача. Взаємна автентифікація корисна для таких ситуацій, як віддалена робота, коли потрібно включити багатофакторну автентифікацію, щоб зменшити ризик фішингу або інших атак, пов’язаних із крадіжкою облікових даних. Щоб отримати додаткові відомості про налаштування клієнтських сертифікатів у веб-переглядачі, прочитайте цю інструкцію.

Цілісність: кожен файл (наприклад, веб-сторінка, зображення або файл

JavaScript), надісланий у браузер веб-сервером HTTPS, містить цифровий підпис, за допомогою якого веб-браузер може визначити, що файл не змінено чи не змінено третьою особою. Сервер обчислює криптографічний хеш вмісту файлу, включаючи його цифровий сертифікат, а браузер може самостійно обчислювати хеш, щоб довести, що цілісність файлу не порушена.

Https є не тільки просто технічним рішенням, але і є важливим елементом для довіри користувачів до нашого програмного рішення через те, що у сьогодні в нинішньому світі де є загрози витоку інформації або кібератаки стали звичним явищем HTTPS відіграє ключову роль у процесі забезпечення безпеки й стала основним критерієм для довіри користувачам.

Одна з ключових переваг HTTPS є шифрування даних що передаються між двома точками користувач - сервер кожний запит проходить через шифрування що гарантує що Мен ін дзе мідл не буде здатним прочитати дані що він піймав.

Якщо брати безпеку платіжну систему, то HTTPS включає механізми виявлення та запобіганню доступу сторонніх людей до даних. Сертифікатибезпеки сайту також гарантує доступ що користувач перейшов на потрібний сайт, а не на копію. HTTPS надає безпеку передачі даних, а сертифікат його легітимність.

Для нашої платформи використання https це також хороший знак для клієнтів що будуть використовувати нашу платформу через те, що клієнти будуть відчувати себе більше захищеним.

HTTPS можливо і також важливо поєднувати з різними іншими механізмами безпеки доприкладу міжмережеві екрани, або ж багатофакторною автентифікацією чи з системою виявлення аномалій. В платформі важливо об'єднати це в інфраструктуру через, яку можна легко реагувати на будь-яку загрозу або спробу злому. наш HTTP є першим бар'єром для запобігання доступу до наших даних.

Сам по собі HTTPS є вимогою безпеки яка введена у міжнародний стандарт як PCI DSS, дотримуватись цієї вимоги є обов'язковим для будь-якої платформи яка має обробляти транзакції. використання HTTP забезпечує захист для даних і є необхідним щоб не отримати штраф.

Https є надзвичайно ефективним якщо бізнес платформа хоче зберегти свою

платформу від кібератак, а саме фішинг, коли створюється підроблені вебсайти схожі на наші, щоб викрасти дані клієнтів. Використання сертифікації надзвичайно дієво ускладнює такі атаки через те що сучасні веббраузери попереджують користувачів про такі загрози.

Але потрібно не забувати що впровадження https не є фінальним для запровадження безпеки це тільки 1 з багатьох кроків. Як мінімум потрібно використовувати методи шифрування даних на стороні клієнта ще важливим моментом є моніторинг мережевої активності. Проте не зважаючи на це HTTPS є обов'язковою технологією для усіх платформ що використовують платіжну систему.

З сучасним розвитком технологій HTTPS буде відігравати все більш важливу роль через те, що своєю чергою це важливий протокол для захисту даних від кібератак і допомагає системам збільшити рівень безпеки у майбутньому.

2.4 Системи виявлення та запобінання загрозам у сучасних платформах

Безпека даних це дуже важлива частина сьогодення через те, що застосовано великі обсяги фінансових транзакцій і важливою частиною є захист від систем вторгнень (IDS) і систем запобігання вторгнень(IPS), що дозволяє моніторити, аналізувати протидію загрозам зі сторони правопорушників.

Базовий принцип роботи IDS

Таке як IDS служить для виявлення загроз аналізуючи трафік з подіями що відбуваються в мережі та також своєю чергою на кінцевих точках. Він дізнається про підозрілу активність яку можна рахувати як спробу атаки та включає сканування портів скануючи чи є шкідливі програмні забезпечення або вразливості.

Також є СПП який швидко блокує підозрілу дію в реальному часу і зупиняє атаку ще до того як вони почались що само собою забезпечує активний захист. IDS можна розглядати як пасивного спостерігача, а IPS за активного який перериває мережеві процеси і може блокувати небажані дії [10].

- Методи виявлення загроз.

Сам по собі IDS/IPS має в собі багато способів які допоможуть виявити загрози

на стороні.

1. Аналіз сигнатури - він порівнює трафік з базою даних відомих сигнатурних атак і цим самим цей підхід ефективний проти більшості відомим нам загроз, але він може бути мало ефективним для нових типів атак.

2. Аналіз аномалій - він використовує модель нормальною поведінкою в системі тому активності які не відповідають цим моделям буде вважатись підозрілим. Це дозволяє виявити невідомі, а також нові загрози, але і також може мати багато хибних спрацювань.

3. Підхід гібридний - він поєднує обидва методи які були описані зверху щоб отримати баланс між ними.

Використання IDS/IPS у платіжних системах

Наше програмне рішення може бути одним з головних цілий для кіберзлочинців оскільки вони будуть містити в собі цінні фінансові й персональні інформації. IDS/IPS має критичну роль у безпеці забезпечуючи такі види захисту:

- Захист від дедос атаків блокування спроб дедосу.

- Захист від крадіжки даних по типу мен ін дзе мідл

- Виявлення та блокування фішингових атак, а саме блокування спроб таких атак.

Виклики в реінжинірингу

Але як і ефективність вони мають і деякі виклики давайте їх розглянемо:

2. IDS/IPS можуть мати аномалії під час аналізую і будуть блокувати легітимні операції.

3. Існують проблеми при швидких атаках вони можуть поширюватись швидше ніж системи IDS/IPS встигають оновлюватись з потрібними сигнатурами.

Вплив на репутацію.

Він також важливий для довіри користувачів через те, що системи працють з персональними даними людей витік такої інформації може призвести до негативного ставлення користувачів до нашої платформи й фінансовим втратам.

Що далі для IDS/IPS.

Сучасні тенденції в розвитку IDS/IPS мають собі інтеграцію такого як хмарного середовища і їхніх технологій й може містити в собі Машине навчання що дозволить системі швидше адаптуватись до нових викликів зменшуючи негативний вплив на користувачів платформи й забезпечуючи їм безпеку.

Такі системи які виявляють і запобігають загрозам є невіддільна частина інформаційних технологій і також їхнє значення росте з ростом складності архітектурам і обсягом даних якими оперують ці платформи. Якби великі платформи які обслуговують тисячі, а то і мільйони користувачів щосекунди подавалась потенційним загрозам ними б було не реально користуватись.

Ці системи IDS/IPS стали охоронцями цифрового простору вони не тільки спостерігають, але й також аналізують мільйони сигналів в секунду відокремлюючи простий графік від небезпечного й за допомогою високого рівня автоматизації та інтеграції зі штучним інтелектом вони адаптуються до нових загроз.

Через те, що загрози зростають вимагаються і нові способи захисту від них. На сьогоднішні дні вони крім виконання своєї роботи на мережевому рівні, але і також інтегруються з іншими рівнями захисту таких як кінцеві точки або ж захистом кінцевих точок контролем доступу, наприклад сучасні платформи як Stripe.

Ще одним перспективним аспектом IDS/IPS є інтеграція в хмарні обчислення. З переходом багатьох платформ у хмару з'являються нові виклики у сфері безпеки. Виклики пов'язані з розподіленим даних. Хмарні IDS/IPS були б ідеальним рішенням не тільки для забезпечення безпеки, але й для масштабування зі збільшенням трафіку.

2.5 Рішення для автентифікації та забезпечення конфіденційності даних

Так як автентифікація й конфіденційність є невідємною частиною сучасних систем безпеки, зараз коли зростає обсяг цифрових інформацій підвищуються і ризики, потрібно мати ефективні способи аутентифікації яка буде гарантувати що тільки справжні користувачі отримають доступ до наших ресурсів, а засоби конфіденційності збереже інформацію користувачів від витоку під час зберігання або передачі даних.

Інформація якою володіє тільки користувач таким як паролі або пін-коди можуть бути вкрадені через найпоширеніші атаки фішінгу або з використанням атаки грубої сили.

У користувача може бути, наприклад фізичний пристрій або токени чи смарткарта, чи дані з одноразовими кодами. Це більше безпечне рішення оскільки зловмисники не зможуть отримати доступ до пристрою.

Якщо додати біометричні дані такі як відбитки пальців, сканування обличчя або оболонку ока система буде безпечнішою через те що їх важко підробити [45].

Мультифакторна автентифікація (MFA)

Теперішні системи безпеки зазвичай маю в собі багатофакторну автентифікацію до прикладу для доступу до банківського рахунку можна оскористатись паролей і підтвердити за допомогою мобільного телефону. Коли використовується ця система шанси що користувача зламають стають значно низькими.

Прикладом популярних протоколів автентифікації є:

- OAuth 2.0: За допомогою нього можна отримати доступ до ресурсів без необхідності в прямої передачі даних користувача з іншого сайту.
- SAML: Security Assertion Markup Language - За допомогою нього можна обмінюватись автентифікаційною інформацією яка використовується в корпоративних середовищах.
- OpenID Connect: розширення OAuth 2.0 з додатковими функціями для автентифікації, такими як перевірка користувача.

Для того, щоб запобігти витоку інформацій які перебувають у стані спокою й у стані руху використовуються алгоритми які роблять дані не читабельними будь-кому хто не володіє правильним ключем. Основні способи шифрування є:

Такі способи шифрування як алгоритм Advanced Encryption Standard(AES), використовує тільки один ключ для шифрування та розшифрування даних.

RSA це асиметричне шифрування яке використовує пару ключів відкритий і закритий що є досить корисним при надсиланні даних через незахищену мережу.

Гібридні системи інтегрують обидві системи, щоб знайти правильний баланс

між безпекою та швидкістю обробки даних.

SSl та TLS - це протокол безпеки який використовується у вебкомунікаціях. Вони разом підтримують асиметричне шифрування для встановлення з'єднання з сервером після чого переходять до іншого типу шифрування симетричного для передачі даних. цим воно захищає від підробки повідомлень що надходять і додавати конфіденційність.

Захист даних.

Сучасні атаки зазвичай націлені на кінцеві точки такі як пристрої користувачів тому необхідно додати додаткові заходи безпеки:

- Антивірусні програмні забезпечення захищає від шкідливих програмних забезпечень.
- Шифрування дисків захищає інформацію на випадки втрати або крадіжок пристрою. Механізми автентифікації або захисту інформації пропонують інструменти сучасних способів захисту.Хмарні сервіси пропонують інструменти які допоможуть інструменти які полегшують виконання функцій що описані зверху. Для прикладу можна сказати про Amazon Cognito[31]: адміністрування ідентифікації та автентифікації користувачів або ж можна надати приклад Google Identity Platform надає технологію для автентифікації. Технології завжди розвиваються як і способи обману, а при розвитку квантових комп'ютерів можуть і в рази ускладнити способи захисту тому необхідно новітні способи розвитку їхнього захисту. Саме через такі виклики й було розроблені новітні способи аутентифікація як по біометрії та алгоритми шифрування які є стійкими до квантових комп'ютерів.

Зараз найбільшою проблемою є і буде в інформаційній безпеці це автентифікація та забезпечення конфіденційності даних через що забезпечення ефективного захисту є особливою потребою коли інформація передається з неймовірною швидкістю як до прикладу диджиталізація програмного рішення.

Ідентифікація: Основні стратегії та виклики.

Головним тут є дотримання балансу між зручності для роботи користувачів та високою рівнем безпеки й таке як традиційні паролі вже не є достатнім і легко піддаються загрозам атак грубої сили, проте використання біометрії може бути

складним для користувачів [9].

Такі популярні біометричні системи які засновані на скануванні обличчя або відбитків пальців появляються мало не щодня саме через їхню зручність. Однак існує багато проблем які зв'язані зі сферою конфіденційності біометричних даних, бо їхня компрометація може мати довгострокові наслідки які унеможливлюють зміни у майбутньому.

2.6 Висновки до розділу

У розділі розглянуто сучасні алгоритми шифрування та способи підтвердження особи і захисту даних їх негативні і позитивні сторони. Визначено основні проблеми при збереженні паролів та кодів доступу, при застосуванні фізичних гаджетів, біометричних відомостей та їх значення у підвищенні безпеки системи.

Розглянуто багатофакторну автентифікації (MFA) як популярний засіб захисту доступу до ресурсів, а також відомі протоколи автентифікації, як OAuth 2.0, SAML і OpenID Connect що дають змогу безпечно обмінюватись інформацією про автентифікацію.

Увагу звернули на методи шифрування даних, такі як AES, RSA та популярні системами, які забезпечують захист даних як під час зберігання так і передачі. Знайшли баланс між безпекою й швидкістю.

РОЗДІЛ 3
РОЗРОБКА АЛГОРИТМУ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ОРЕНДИ

3.1 Опис проблем управління орендою та мотивація створення сучасних систем автоматизції

Управління орендою є невіддільною частиною сучасного ринку нерухомості, однак наявні платформи часто стикаються з численними проблемами. Основні виклики включають: складність у масштабуванні систем, ручне управління цінами та умовами оренди, затримки у реагуванні на запити користувачів, а також недостатній рівень безпеки й прозорості процесів.

На базі зробленого аналізу була покращенна платформа для керування орендою, що вирішує ці проблеми через використання сучасних технологій і підходів. Головні акценти ставляться на:

- Масштабованість – використання мікросервісної архітектури на базі Spring Boot, що дає можливість змінювати систему в разі збільшення кількості користувачів платформи і збільшенні обсягів даних.
- Гнучкість – автоматизоване управління динамічними цінами на основі аналізу попиту, доступності житла та сезонності.
- Легкість у використанні – зробити зрозумілим інтерфейс для спілкування як з боку тих хто орендує, так і тим хто дає в оренду.

Порівняння з існуючими платформами

Існуючі сайти, як Airbnb або Booking.com, дають прості функції для бронювання але мають ряд проблем:

- Недостатня інтеграція автоматизованих рішень – відсутність ефективних механізмів динамічного ціноутворення, яке враховує реальний попит, доступність і конкурентні пропозиції.
- Низький рівень безпеки даних - часто використовують старі способи шифрування або не до кінця перевірені системи обробки фінансових угод.

Шляхи виходу з проблем

Розроблене рішення вносить нові ідеї, що сильно покращує роботу платформи.

Автоматизація процесів бронювання – використання алгоритмів машинного навчання для аналізу історії попиту й автоматичної оптимізації умов оренди.

Реалізація технологій реального часу – інтеграція з Telegram API для миттєвих повідомлень про підтвердження бронювань, зміни цін або наявності об'єктів.

Підвищення безпеки – впровадження Stripe API для фінансових транзакцій із сучасними протоколами шифрування, а також механізмів багаторівневої автентифікації [44].

Інтерактивність – можливість орендодавців додавати прайс-листи, умови оренди та іншу інформацію, яка автоматично синхронізується з усіма користувачами в реальному часі.

Давайте зараз розглянемо приклад коду який може виконуватись у фоновому режимі в системі автоматизації під час процесу бронювання житла тут надаються інформація про компанії які надають послуги оренди житла.

Лістинг 3.1.

```
Map<String, String> workPages = new HashMap<>();
workPages.put("company1", "https://example.com/offer1");
робочі сторінки.put("company2", "https://example.com/offer2");
workPages.put("company3", "https://example.com/offer3");
for (String company : workPages.keySet()) {
Рядок offerPage = workPages.get(компанія
// Логіка, щоб визначити, чи відповідні пропозиції для сторінки
System.out.println("Перевірити наявність пропозиції для компанії:
" + компанія);

}
```

Частина коду яка представлена вище слугує для зберігання даних різних фірм які здають житла в оренду і можна перевірити актуальність пропозицій. Використання такої структури дає можливість автоматизувати оновлення та перевірку наявності житла коли нам буде зручно що становить значну частину автоматизації систем.

Також є важливість у забезпечення безпеки та конфіденційності даних через те,

що традиційні методи управління орендою як паперові записки й таблиці excel не можуть надати високий рівень захисту даних. Тому відсутність методів шифрування та захисту призведе до витоку інформації.

Це робиться за допомогою автоматизації системи за допомогою методів та алгоритмів у системі які дозволять нам працювати з платіжними шлюзами й функціями безпеки для них потрібно застосувати шифрування даних і автентифікація користувачів розгляньмо один з прикладів як можна застосувати платіжну систему.

Лістинг 3.2.

```
public void processPayment(Order order) {
спробувати {
stripeService
// Платіжні картки
System.out.println("Оплата замовлення успішно оброблена для ідентифікатора замовлення:

} catch (StripeException e) {
System.out.println("Помилка під час обробки платежу: " + e.getMessage());

}

}
```

Тут зазначено як можна здійснити обробку платежу, щоб фінансові операції були точними, а головне безпечними що своєю чергою спростить процес оренди для користувача та орендодавця.

Потреби в автоматизації.

Потреба в автоматизації допоможе усвідомити що автоматизація є головною потребою у розробці програмного рішення і зменшить кількість людських помилок та прискорить обробку даних з підвищенням ефективності роботи. Користувачі завжди зможуть скористатися актуальними інформаціями про доступність житла для оренди та швидко і зручного оплатити житло. наведений вище код демонструє як можна автоматизувати процеси моніторингу та оновлення даних доступних пропозицій.

Приклад автоматизації процесу оплати за допомогою Stripe

Давайте тепер розглянемо код який застосований в проєкт для оплати, він оброблює результат платежу та статус бронювання. Обробка платежу здійснюється своєю чергою за допомогою Stripe про яку ми вже не раз говорили вище.

Лістинг 3.3.

```
    @Override
    public                                          PaymentCreateResponseDto
createPaymentSession(PaymentRequestDto requestDto) {
        SessionCreateParams params = SessionCreateParams.builder()
                .setSuccessUrl(successUrl)
                .setCancelUrl(cancelUrl)
                .addLineItem(SessionCreateParams.LineItem.builder()
                        .setPrice(getPrice(requestDto.getTotal()))
                        .setQuantity(DEFAULT_PRODUCTS_QUANTITY)
                        .build()
                )
                .setMode(SessionCreateParams.Mode.PAYMENT)
                .build();
        Session session;
        try {
            session = Session.create(params);
        } catch (StripeException e) {
            throw new PaymentFailedException(CHECKOUT_FAILURE_MESSAGE
+ e);
        }
        paymentService.save(requestDto, session);
        return new PaymentCreateResponseDto(session.getUrl());
    }
```

Код створює автоматичну платіжну сесію через Stripe де клієнт оплачує оренду житла й у разі успішної оплати система автоматично змінює статус бронювання на "підтверджено" у разі відмови на "відхилено". Це автоматизує процес і мінімалізує шанс людської помилки.

Цей клас автоматизує процес оплати що робить його зрозумілішим і прискорює обробку в пару разів що своєю чергою робить його більш гнучким і масштабованим.

Автоматизоване розподіл для підвищення продуктивності

Через те, що ми використовуємо Stripe він автоматизує процеси оплати й підтвердження бронювання що своєю чергою надає нам швидку, а головне безпечну обробку цих транзакцій знижуючи ризики на помилки при обробці цих платежів що своєю чергою покращує взаємодію з клієнтами які будуть користуватись нашим

програмним рішенням.

Лістинг 3.4.

```
    if (customer.getChatId() !=
    notificationService.sendToUserPayment(customer.getChatId(),
оплата);
    }
```

notification_service

В коді що показано вище описаний процес сповіщення користувачів про успішну оплату в випадку коли користувач підключився до телеграм бота.

Використання нових автоматизованих методів та алгоритмів для управління орендою пов'язано з потребою покращення ефективності, зменшенню помилок людей і забезпечення швидкого реагування на зміни в навколишньому світі. Створення таких систем – лише зробити процеси оренди кращими для клієнтів. Автоматизація платіжної системи надає можливість власникам квартир підтримувати свіжу інформацію, одночасно оновлюючи статуси бронювань і платежів.

Сучасні технології, як Stripe, вже можуть обробляти платежі і ще й робити статуса бронювання в реальному часі. Розглянем автоматизацію платежів на основі коду платіжного сервісу який використовує Stripe для створення платіжних сеансів:

Лістинг 3.5.

```
    @Override
    public                                    PaymentCreateResponseDto
createPaymentSession(PaymentRequestDto requestDto) {
        SessionCreateParams params = SessionCreateParams.builder()
                .setSuccessUrl(successUrl)
                .setCancelUrl(cancelUrl)
                .addLineItem(SessionCreateParams.LineItem.builder()
                        .setPrice(getPrice(requestDto.getTotal()))
                        .setQuantity(DEFAULT_PRODUCTS_QUANTITY)
                        .build()
                )
                .setMode(SessionCreateParams.Mode.PAYMENT)
                .build();
        Session session;
        try {
```

```
            session = Session.create(params);
        } catch (StripeException e) {
            throw new PaymentFailedException(CHECKOUT_FAILURE_MESSAGE
+ e);
        }
        paymentService.save(requestDto, session);
        return new PaymentCreateResponseDto(session.getUrl());
    }
```

У цьому коді робиться сесія для оплати за допомогою Stripe, де використовується сума платіжу, валюта а також створюється посилання для клієнтів щоб зробити платіж. Якщо сесія була вдало створена інформація про платіж зберігається в бази даних. Такий підхід дозволяє швидко генерувати платіжні посилання для кожного бронювання.

Після успішної оплати, система сама оновлює статус бронювання

Лістинг 3.6.

```
    @Override
    @Transactional
    public void succeed(String sessionId) {
        Payment                          payment                          =
paymentRepository.findBySessionId(sessionId).orElseThrow(
                ()                              ->                          new
EntityNotFoundException(PAYMENT_NOT_FOUND_MESSAGE + sessionId)
        );
        payment.setStatus(Status.SUCCEED);
        paymentRepository.save(payment);

        Booking booking = payment.getBooking();
        booking.setStatus(Booking.Status.CONFIRMED);
        bookingRepository.save(booking);
        Customer customer = booking.getCustomer();
        if (customer.getChatId() != null) {

notificationService.sendToUserPayment(customer.getChatId(), payment);
        }
        notificationService.paymentMessage(payment);
    }
```

Цей метод змінює статус бронювання після підтвердження успішного платежу, підтверджуючи бронювання й посилаючи клієнту повідомлення про завершення операції. Це дає можливість заощадити час, зменшити число помилок і підвищити

задоволеність клієнтів.

Опис структури

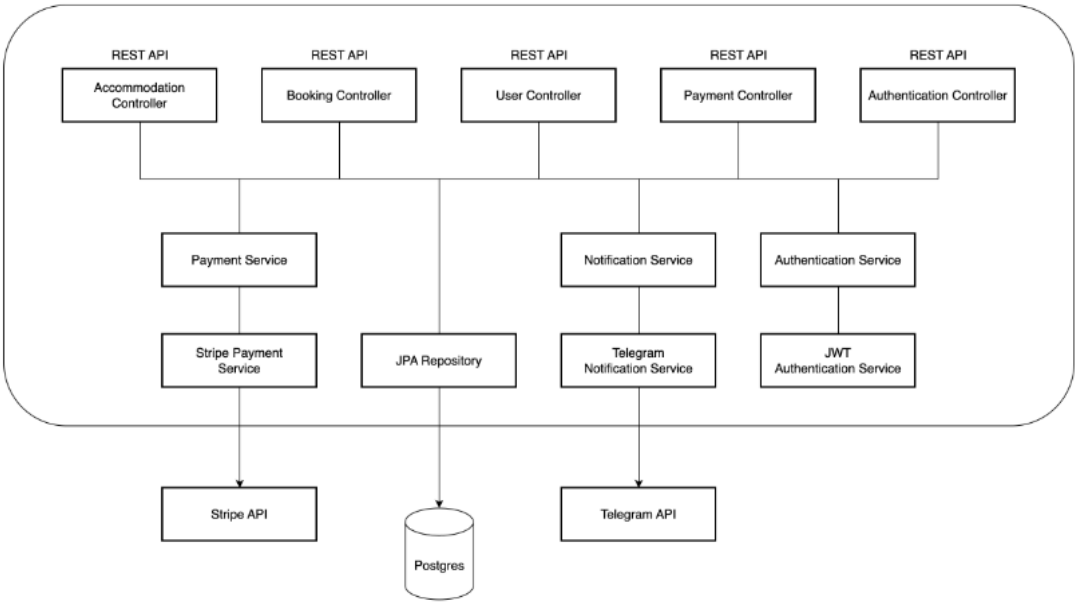


Рис. 3.1. Архітектура проекту

На малюнку згори видно будову системи для онлайн-бронювання житла з використанням технології Spring Boot. Тут показано, як різні частини розподілені зі зазначенням мети кожного окремого завдання у загальному функціонуванні. Головні частини, які видно на схемі, ми розглянемо далі.

Загальний опис

Архітектура базується на RESTful API, за яким клієнтська частина працює з сервером. Кожен контролер REST API дає функцію відповідних модулів, що робить всю систему розширювальною, модульною та масштабованою. Всі методи контролерів і сервісів взаємодіють з реляційною базою даних Postgres - для зберігання всіх потрібних даних [14].

1. API REST:

Accommodation contoller: цей контроллер контролює дані про житло (як-от, його опис, ціна, зручності і так далі). Він може обробляти запити для отримання списку житла або оновлювати інформацію про певне житло.

- Контролер оренди: має справу з орендою. Він отримує запити на створення,

перегляд або скасування бронювань працює з даними користувачів та даними житла;

- Контролер юзера: керує даними юзера, такими як запис, профілі юзерів та їх дії в системі.

- Контролер грошей: керує грошовими угодами та з'єднується з зовнішніми службами для безпечної оплати за бронювання;

- Контролер перевірки: управляє всіма справами, пов'язаними з перевіркою та дозволом користувачів, як вхід, реєстрація або оновлення токенів доступу.

2. Оплата

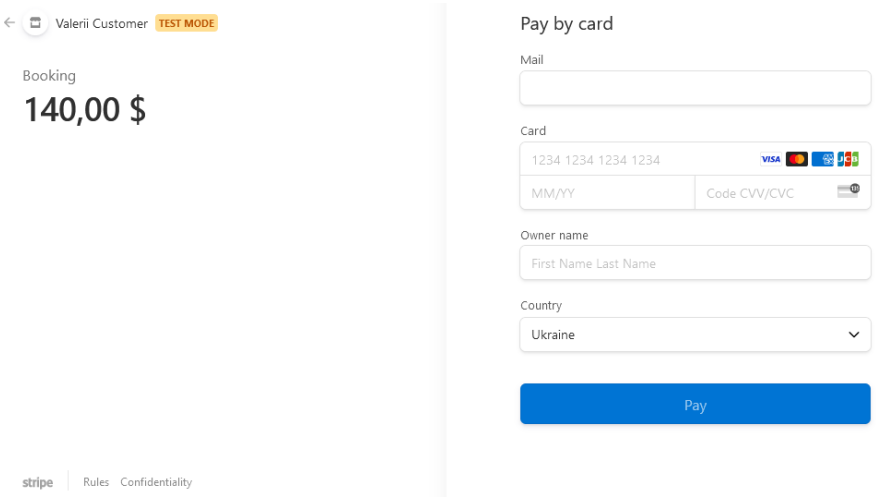


Рис. 3.2. Процес оплати приміщення

Payment Service: відповідає за підготовку моделів даних для платіжних операцій;

Сервіс сповіщення використовується для відправлення повідомлень до користувачів з допомогою сервісу сповіщення Telegram [31].

Конторолер підтвердження: підтверджує юзерів і працює з службою підтвердження JWT для створення та перевірки токенів доступу;

3. Інтеграція із зовнішніми API:

Система має інтеграцію із зовнішнім API:

Stripe API надає структуру що дозволяє людям робити онлайн-оплати без потреби у самостійному виконанні важких платіжних процесів.

Телеграм API надає функції, яка дозволяє відправляти людям повідомлення

через телеграм месенджер наприклад, посилати поідомлення про підтвердження бронювання зміну статусу або інші важливі новини.

4. Репозиторій JPA

Всі сервіси звертаються до JPA Сховища, працюють з базою даних постгрес і дають змогу зберігати всі дані про користувачів - умови бронювання, оплати тощо. Власне JPA спростить роботу з базою даних за рахунок того що запити можуть бути створені автоматично на основі методів JPA.

Процес бронювання:

- Користувач посилає запит до Контролера бронювання. Контролер відправляє запит до сервісу, що бронює, яка через репозиторій дивиться наявність місць.
- Якщо житло є, система робить бронювання та зберігає запис в базі даних за допомогою репозиторія JPA.
- Після створення замовлення контролер грошей починає платіж через сервіс.
- Платіжна сервіс говорить з платіжною службою Stripe, що потім відправляє запит до Stripe API для зроблення транзакції [32].

До контролера для перевірки особи звертаються з запитом на вхід. Алгоритм, що перевіряє особу, підтверджує дані користувача у базі даних. Алгоритм перевірки особи бере токен, перевіряє і повертає токен назад користувачу.

Інтеграція сторонніх API буде допомагати реагувати на дії користувача у реальному часі.

Безпека: використання JWT для аутентифікації та Stripe API для платежів;

Зручність: API Telegram дає змогу людям повідомляти про стан їхнього бронювання легким способом.

3.2 Вимоги до програмного рішення та архітектура проектованої системи

Зважаючи на головні потреби ринку наша платформа повинна в собі мати наступне:

1. Модульність

Модульна структура має бути такою, де кожен елемент відповідає за окрему та певну роль. При цьому система мусить бути простою в підтримці та розширенні .

2. Інтеграція із зовнішніми сервісами

Декілька потрібних з'єднань стосуються роботи з грошима через API платіжної системи, Stripe API, в той же час як інші створені для обміну швидкими повідомленнями як-от Telegram API.

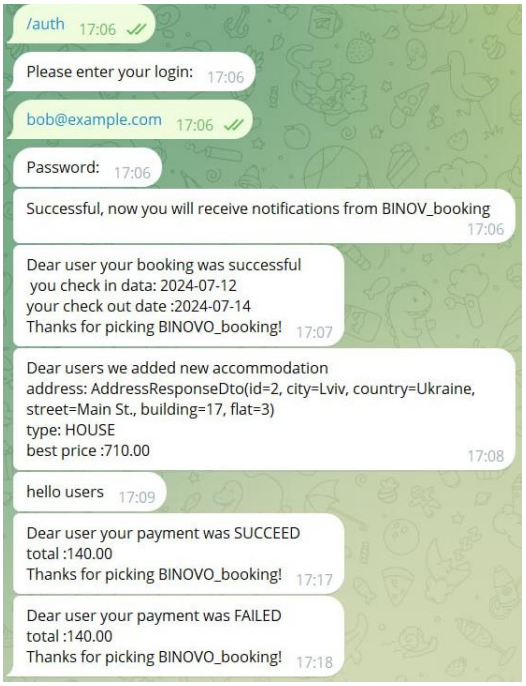


Рис. 3.3. Взаємодія з телеграмом

3. JWT

Процеси підтвердження особи мають проходити через JWT (JSON Веб Токени), що надає захист даних користувачів.

4. Реляційні дані зберігання

Усі ці дані потрібно буде тримати в зв'язкових базах даних PostgreSQL, що дасть можливість мати до них шивдкий доступ за допомогою JPA [23].

5. REST API

Доступ до функцій системи має бути через REST API, що забезпечує сумісність з клієнтськими програмами.

6. Telegram

Статус резервування та сплат має автоматично обновлятись в Телеграмм.

7. Витривалість

Система має бути в стані підтримувати багато запитів разом [4].

Будова системи багатошарова, що дає змогу чітко розділити обов'язки між частинами системи. Його головні рівні та їх функції показані нижче:

1. Ступінь кінцевих точок

На цьому рівні стоят контролери, що обробляють запити користувачів и дають доступ до можливостей системи.

- Контролер житла: слідкує за даними про житло;

- Контролер бронювання: обробка оренди житла;

- Контролер користувача: керування даних користувачів;

- Payment Controller: контроль платежів;

- Контролер автентифікації: авториз;

2. Ступінь обробки бізнес логіки(Сервісно-алгоритмічний блок коду)

- Платіжна служба : управління платежами;

- Платіжний сервіс Stripe, з'єднання із API Stripe для створення платіжних меж.

- Сервіс сповіщення: відправлення;

- Сервіс повідомлень використовує АПІ Телеграм для надсилання повідомлень.

- Сервіс автентифікації: обробка авторизації;

3. Ступінь бази данних

Усі дані будуть зберігатися в базі даних PostgreSQL, а доступ до них буде через JPA Repository який забезпечує легке спілкування програми з базою даних і підтримує транзакції [23].

4. Ступінь інтегрування

Ця архітектура має з'єднання з іншими сервісами.

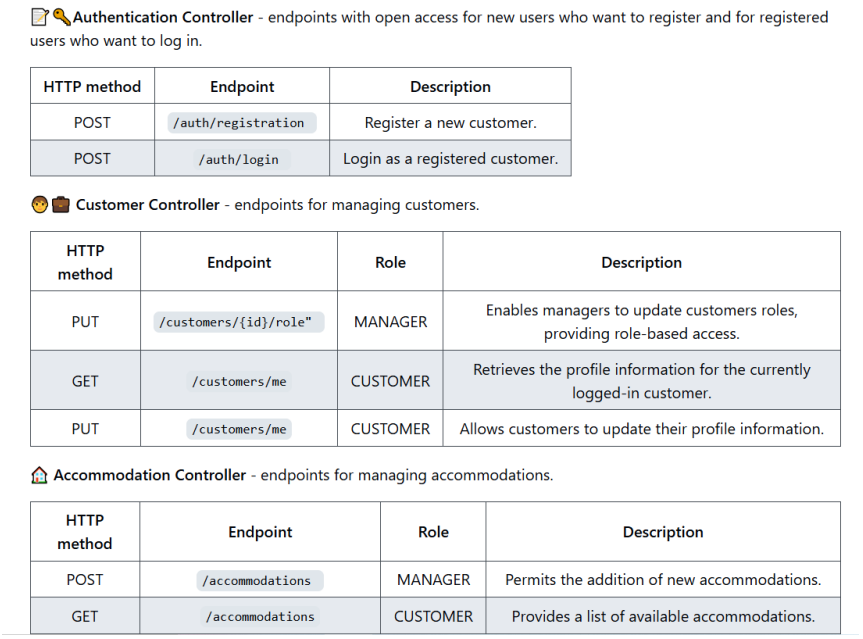
- Stripe API для обробки грошей.

– API для Телеграм, щоб сповістити користувачів про статус запиту та оплату.

RESTAPI використовують моделі, методи та алгоритми, які потім працюють із базою даних або сторонніми API. Він має в собі такі риси системи, як модульність, можливість росту і легкість у додаванні нових функцій [26].

- Ясне ділення завдань: кожен модуль відповідає тільки за свою частину бізнес-логіки або з'єднання що зменшує ризик помилки;

- Гнучка: архітектура є модульною та підтримує добавку та зміни для нових чи старих служб;



Authentication Controller - endpoints with open access for new users who want to register and for registered users who want to log in.

HTTP method	Endpoint	Description
POST	`/auth/registration`	Register a new customer.
POST	`/auth/login`	Login as a registered customer.

Customer Controller - endpoints for managing customers.

HTTP method	Endpoint	Role	Description
PUT	`/customers/{id}/role"`	MANAGER	Enables managers to update customers roles, providing role-based access.
GET	`/customers/me`	CUSTOMER	Retrieves the profile information for the currently logged-in customer.
PUT	`/customers/me`	CUSTOMER	Allows customers to update their profile information.

Accommodation Controller - endpoints for managing accommodations.

HTTP method	Endpoint	Role	Description
POST	`/accommodations`	MANAGER	Permits the addition of new accommodations.
GET	`/accommodations`	CUSTOMER	Provides a list of available accommodations.

Рис. 3.4 Взаємодія з контроллером

У розробленій програмній системі були впроваджені декілька важливих REST-контролерів, кожен з них відповідає за окрему функцію. Такий підхід дає можливість мати модульність та зручність в використанні, спрощуючи підтримку та розвиток система. Контролери виконують різні аспекти роботи системи, починаючи від входу користувачів і закінчуючи управлінням житлом. У цьому розділі буде докладно описано кожен із цих контролерів, їх головне призначення, функціональність та внесок в загальну архітектуру системи.

Контролери застосовують сервіси де є бізнес логіка.

Контролер автентифікації

Контролер для автентифікації є важливим елементом впровадження безпекової політики та управлінні доступу до системи. Він керує, та впроваджує два основні кінцеві точки:

- POST /auth/registration

Щоб зареєструвати нового користувача у системі. Ця точка мусить бути відкритою для всіх адже вона дає змогу клієнтам створювати акаунти використовуючи тільки свої імена електронні адреси та паролі.

- POST /auth/login

Призначений для входу в систему користувачів. Ця точка повертає токени доступу після перевірки даних. Для цього застосуватно JWT (JSON Web Token) щоб використовувати спосіб створення безпечної та великої системи управління сеансами

Ролі та токени є важливим чином для кожної веб-системи. Це дає безпечний вхід до інших частин з контролерами, бо лише дозволені особи можуть користуватися основними функціями.

Контролер клієнта

Контролер клієнта створений для керування даними про замовників та надання індивідуального сервісу для кожного клієнта. Він має такі кінцеві точки:

- PUT /customers/{id}/role

Дає можливість змінювати ролі користувачів для менеджерів. Це дуже важливо для системи доступу, де клієнти та менеджери або адміністратори мають різні права. Зміна роль може вказувати на будь-яку дію, наприклад, підвищення користувача до менеджера або зменшення прав користувача.

- GET /customers/me

Надає можливість користувачам дивитися їх профілі. Ця річ допомагає мати ясність та легкість в даних для людей, що допомагає хорошому досвіду користувача.

- PUT /customers/me

Дозволяє користувачам оновлювати свої особисті дані, такі як ім'я адреса електронна пошта.

Даний контролер відіграє важливу роль у створенні гнучкої системи, що дозволяє користувачам самостійно керувати своїми даними та ролями.

Контролер помешкань

Контролер помешкань був створен для управління даними про квартири, які можна орендувати. Він має багато різних можливостей:

- POST /accommodations

Дає можливість менеджеру ввести нову модель в систему. Ця кінцева точка дозволяє вводити опис, місце розтування, ціну та умови оренди що стосуються житла.

- GET /accommodations

Дозволяє клієнтам бачити всі оренди з яких вони можуть вибрати;

- GET /accommodations/{id}

Дає точну інформацію про житло. Це може бути зображення, опис, присутність вільних місць та відгуки інших користувачів

- PUT /accommodations/{id}

Дає змогу змінити дані про об'єкт. Наприклад якщо ціна, або умови оренди були змінились, це надає оновлення інформації.

- PUT /accommodations/{id}/address

Кінцева точка для оновлення адреси житла. Це зручно, коли житло може переміститися або є потреба в отриманні більш точних даних про адресу.

- DELETE /accommodations/{id}

Цей метод дозволяє менеджерам видаляти з системи житлові об'єкти.

Оренда житла

Контролер бронювання - це головна частина методів та алгоритмів, що буде використовуватися для управління процесами бронювання. Його кінцеві точки дозволять клієнтам створити, подивитися та оновити бронювання, а менеджерам - слідкувати за статусом цих бронювань. Функція цього контролера має в собі такі кінцеві точки:

- PATCH /bookings

Ця кінцева точка надає менеджерам повноваження змінювати статус бронювання. Це може бути статус підтвердження, статус скасування або, можливо, перерозподіл в інший статус на основі бізнес-логіки.

- POST / bookings

Ця кінцева точка надає зможу бронювання нове житло клієнтами. Клієнт своєю чергою робить запит для отримання оренди житла на вибрану дату.

- GET / bookings

Це кінцева точка яка надає змогу клієнтам отримати інформацію про всі бронювання що вони зробили. Він повертає список бронювань з інформацією про дату, статус бронювання і ціну.

Менеджери можуть використовувати цей ендпоід, щоб отримати список бронювань користувачів за заданими умовами як наприклад фільтрація бронювання за статусом або унікальним номером нашого клієнта.

Ця кінцева точно дуже корисна для того, щоб дозволити клієнтам зробити потрібні їм зміни у боронуванні до прикладу як зміна дати або деталі бронювання.

- DELETE / bookings

Надає можливість клієнтам скасувати бронювання у випадках коли у користувача змінились плани що зробило систему більш гнучкішою і збільшує довіру клієнтів.

Цей контролер є одним з найважливіших, бо він відповідає за оренду житла оплату і сповіщення.

Розрахунок плати з персоналом

Контролер оплати керує фінансовими операціями про оренду житла та взаємодіє з Stripe арі що надає надійність і безпеку транзакцій користувача.

нижче описані його кінцеві точки.

- GET /payments

Ця кінцева точка доступна для всіх авторизованих користувачів через те, що він дозволяє отримати історію оплат для конкретних клієнтів і відображає інформацію про всі завершені транзакції, їхні статуси та суми оплат.

- POST / payments

Ця кінцева точка розпочинає платіж. Користувач вибирає бронювання, за яку має заплатити а система веде на сторінку оплати.

- GET / payments

Це остання зупинка на яку йде платіж, вона записує дані, підтверджує резервування і змінює статус бронювання у системі для правильного бронювання. Він також використовується для обробки випадків, коли платіж не пройшов(наприклад, у разі невдалої угоди користувач може ще раз подати заявку або змінити реквізити при новій спробі).

Роль вбудованих контролерів

Це важливі частини загальної забудови системи які повинні мати наступні умови:

Зручність для юзера Юзерам легко бронювати та платити завдяки простоті інтерфейсу.

Ясність процесів, відкритість і зрозумілість на кожному кроці від бронювання до оплати.

Найвищий ступінь захисту угод, - з'єднання з API Страйп.

Масштабування означає, що нові можливості можуть бути просто додані або функція може бути розроблена на меншому рівні без шкоди для стабільності системи. Ці контролери забезпечують постійну роботу системи і допомагає їй працювати більш ефективно задовольняючи потреби користувачів.

Необхідні частини системи:

1. Ясно вказати обов'язок - кожен контролер несе відповідальність за певний комплект завдань що робить систему більш зрозумілішою для інших розробників [33].

2. Розширюваність - додавання нових функцій або зміна старих не вплине на інші частини системи [44].

3. Відкритість для юзерів, - юзери можуть зручно працювати з системою [14].

4. Захист – розподіл ролел для входу на кінцевих точках підвищує захист [17].

Усі зазначені контролери працюють разом з іншими частинами системи, серед яких є сервіси, сховища даних й зовнішні API що створено для будування цілої, ефективної й надійної архітектури. Така структура забезпечить успішність дій користувача і він буде впевнений що все пройшло без проблем і його дані будуть в безпеці.

3.3 Розробка моделей, методів та алгоритмів для управління бронюванням та оплати

Моделі, методи та алгоритми є основною частиною для управління бронюванням і поєднані з платіжною системою Stripe для обробки платежів.

У коді виконується наступне:

- створення бронювання;
- обробка запита на створення (модель DTO з даними про бронювання);
- адреса житла зберігається у базі через AddressRepository;
- модель для бронювання створюється за допомогою мапера (AccommodationMapper) і зберігає в базі даних;
- користувач отримує письмо про те що зробив бронювання через NotificationService.

Лістинг 3.7.

```
    public AccommodationResponseDto save(AccommodationRequestDto
accommodationRequestDto) {
        Address savedAddress =
addressRepository.save(accommodationRequestDto.getAddress());
        Accommodation accommodation =
accommodationMapper.toEntity(accommodationRequestDto);
        accommodation.setAddress(savedAddress);
        return
accommodationMapper.toDto(accommodationRepository.save(accommodation))
;
    }
```

Також у коді що виконується наступне:

- оновлення броні;
- виконується пошук бронювання за ідентифікатором;
- частини що можна змінити, це: вид, величина, вартість тощо;
- зміни зберігають в базу, користувач отримує повідомлення про оновлення.

Лістинг 3.8.

```
public AccommodationResponseDto updateDetailsById(Long id,
```

```
AccommodationUpdateRequestDto requestDto) {
        Accommodation accommodation =
accommodationRepository.findById(id).orElseThrow(() ->
                new EntityNotFoundException(EXCEPTION_MSG_CANNOT_FIND
+ id))
                .setType(requestDto.getType())
                .setPrice(requestDto.getPrice());
        return
accommodationMapper.toDto(accommodationRepository.save(accommodation))
;
    }
```

Алгоритми управління оплатою

Оплата - це один з найголовніших кроків в процесі бронювання. Система робить правила, які створюють сесії обробки оплати, обробляють успішні і неуспішні платежі. Головна логіка для цього сервісу зроблена в класах PaymentServiceImpl та StripePaymentServiceImpl.

Обробка платежів

Сесія робить умови, які містять посилання на вдалу і скасовану оплату.

Лістинг 3.9.

```
    public                                    PaymentCreateResponseDto
createPaymentSession(PaymentRequestDto requestDto) {
        SessionCreateParams params = SessionCreateParams.builder()
                .setSuccessUrl(successUrl)
                .setCancelUrl(cancelUrl)
                .addLineItem(SessionCreateParams.LineItem.builder()
                        .setPrice(getPrice(requestDto.getTotal()))
                        .setQuantity(DEFAULT_PRODUCTS_QUANTITY)
                        .build())
                .setMode(SessionCreateParams.Mode.PAYMENT)
                .build();
        Session session = Session.create(params);
        paymentService.save(requestDto, session);
        return new PaymentCreateResponseDto(session.getUrl());
    }
```

Обробка успіху

- Коректно оплачений переводиться в статус ПІДТВЕРДЖЕНО.
- Користувач отримує письмо про зроблений платіж.
- В випадку помилки прокидуємо кастомну помилку.

Лістинг 3.10.

```
    public void succeed(String sessionId) {
        Payment payment = paymentRepository.findBySessionId(sessionId).orElseThrow(
                () -> new EntityNotFoundException(PAYMENT_NOT_FOUND_MESSAGE + sessionId));
        payment.setStatus(Status.SUCCEED);
        Booking booking = payment.getBooking();
        booking.setStatus(Booking.Status.CONFIRMED);
        bookingRepository.save(booking);
        notificationService.paymentMessage(payment);
    }
```

Обробка не відбулася

Оновлюється статус платежу на FAILED.

Для скасованих бронювань є статуси REJECTED. Інтеграція з NotificationService важлива для комунікації з користувачами. Використовують сервіс NotificationService для відправлення повідомлень про: створення нового бронювання, позитивну або негативну зміну статусу оплати. Ці алгоритми дають можливість легко керувати не тільки бронюванням й оплатою, але і прозорістю процесами які логуються і мають дуже високу надійність і безпечну інтеграцією Stripe API.

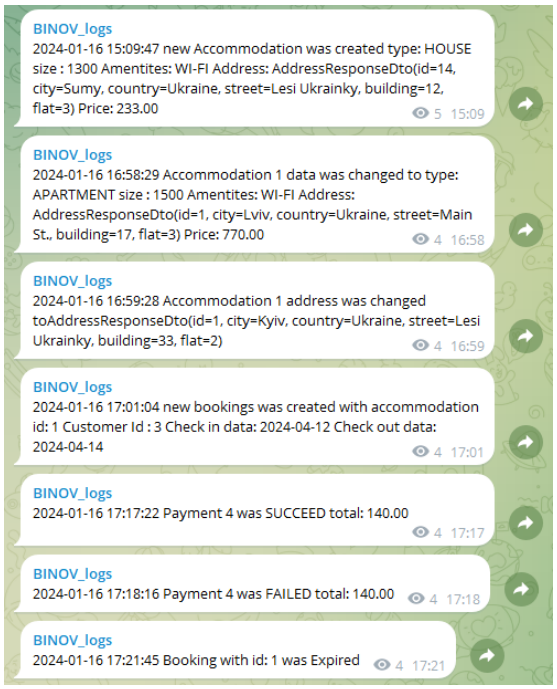


Рис. 3.5. Логування

Алгоритм пошуку найкращого шляху до житла.

За основу було взято алгоритми Дейкстри та A*. Я зробив гібрид цих двох алгоритмій використовуючи карту опен стріт, яку я розбив на граф вузлів(на основі координати перехресть і точків цікавості).

Для нього використовую наступну формулу

$$H(n) = \text{Відстань}(n,\ end) \tag{3.1}$$

де H- це перспектива вузла;

n-це поточний вузол;

end – кінцевий вузол.

Цим самим прискорю швидкість обробки мого алгоритму. Давайте розглянемо ключові моменти.

Лістинг 3.11.

```
        public List<String> findShortestPath(Graph graph, String
startId, String endId) {
            Node startNode = graph.getNode(startId);
            Node endNode = graph.getNode(endId);
            if (startNode == null || endNode == null) {
                throw new NoPathFoundException("Start or end node not
found in the graph.");
            }
            PriorityQueue<PathNode> openSet = new
PriorityQueue<>(Comparator.comparingDouble(PathNode::getEstimatedCost)
);
            Map<Node, Double> gScores = new HashMap<>();
            Map<Node, Node> cameFrom = new HashMap<>();
            Set<Node> visited = new HashSet<>();
            gScores.put(startNode, 0.0);
            openSet.add(new PathNode(startNode, 0.0,
heuristic(startNode, endNode)));
            while (!openSet.isEmpty()) {
                PathNode current = openSet.poll();
                Node currentNode = current.getNode();
                if (currentNode.equals(endNode)) {
                    return reconstructPath(cameFrom, currentNode);
                }
                if (visited.contains(currentNode)) continue;
                visited.add(currentNode);
```

```
                for (Map.Entry<Node, Double> neighborEntry :
currentNode.getNeighbors().entrySet()) {
                    Node neighbor = neighborEntry.getKey();
                    double tentativeGScore =
gScores.getOrDefault(currentNode, Double.MAX_VALUE) +
neighborEntry.getValue();
                    if (tentativeGScore <
gScores.getOrDefault(neighbor, Double.MAX_VALUE)) {
                        gScores.put(neighbor, tentativeGScore);
                        double estimatedCost = tentativeGScore +
heuristic(neighbor, endNode);
                        openSet.add(new PathNode(neighbor,
tentativeGScore, estimatedCost));
                        cameFrom.put(neighbor, currentNode);
                    }
                }
            }
            throw new NoPathFoundException ("No path found from " +
startId + " to " + endId);
        }
```

В цьому коді використовую Graph graph як граф з ребрами між заданими точками та stratId й endId як стартовий вузл і кінцевий. Перевіряється чи існують задані вузли якщо таких нема прокидується кастомна помилка для того щоб можна було швидше розпізнати проблему яку прокидує код. Після чого оцінюємо вартість шляху(швидкість за яким користувач зможе добратись до точки) і вираховуємо шляхи використовуючи таблицю оцінки шляхів gScores.

Система для управління бронюванням та платежами зроблена з акцентом на співпраці методів. Зв’язок між різними частинами, як-от сервіси бронювання, оплати і сповіщення, дають змогу детально подивитись на процеси управління. Нижче розглянемо ближче декілька моментів.

Зв'язок бронювання та оплати

Синхронізація станів:

Логіка оплати тісно поєднана з бронюванням. Наприклад, після вдалої плати статус бронювання автоматично стає ПІДТВЕРДЖЕНИЙ що дозволяє підтримувати актуальний стан бронювання в системі без зайвих запитів. Такий процес узгодження статусів значно спрощує управління процесом уникаючи неузгодженості даних. Та підвищує рівень автоматизації процесів бронювання для користувача платформи.

Лістинг 3.12.

```
    public void succeed(String sessionId) {
        Payment payment = paymentRepository.findBySessionId(sessionId).orElseThrow(
                () -> new EntityNotFoundException(PAYMENT_NOT_FOUND_MESSAGE + sessionId));
        payment.setStatus(Status.SUCCEED);
        Booking booking = payment.getBooking();
        booking.setStatus(Booking.Status.CONFIRMED);
        bookingRepository.save(booking);
    }
```

Невдалі переходи

Якщо якісь платіж не вийшов, чи то користувач скасував його, чи сталась якась помилка - стан бронювання тепер буде FAILED! Це дозволяє звільнити заброньоване житло для інших людей, розповісти користувачеві, що він зробив не так та дати йому ще 1 спробу.

Лістинг 3.13.

```
    public void cancel(String sessionId) {
        Payment payment = paymentRepository.findBySessionId(sessionId).orElseThrow(
                () -> new EntityNotFoundException(PAYMENT_NOT_FOUND_MESSAGE + sessionId));
        payment.setStatus(Status.FAILED);
        Booking booking = payment.getBooking();
        booking.setStatus(Booking.Status.REJECTED);
        bookingRepository.save(booking);
    }
```

Stripe API.

Зв'язок з Stripe API дає швидке та легке проходження платежів.

Саме з допомогою Stripe API можна автоматично рахувати ціни в вказаній валюті на послуги що надаються. Так сервери Stripe отримую створену модель цін, PriceCreateParams.

Лістинг 3.14.

```
    private String getPrice(BigDecimal total) {
    PriceCreateParams params = PriceCreateParams
    .set
    .setUnitAmount(total.longValue() * PRICE_VALUE_IN_CENTS)
```

```
.title(DEFAULT
.compile()
.
return price.create(params)
}
```

- URL-адреси успіху та скасування допомагають поставити статус кожної справи через яку проходять ініційовані платіжні сесії,
- Ключ Stripe`secretKey дає змогу кодувати дані так що гарантує безпеку від викрадення грошей від сторонніх осіб.

Обробка виключень:

Stripe API може дати помилки у вападку з лімітами або не вистачає грошей, тощо. Тому потрібно їх обробляли.

Лістинг 3.15.

```
try {
session = Session.create(params);
} catch (StripeException e) {
throw new PaymentFailedException(C
}
```

Сповіщення - одна з ключових функцій UX.

Сповіщення про майбутні події:

Всі важливі події які трапляються в системі приходять з повідомленнями що надсилаються до NotificationService. Посеред таких подій:

- повідомлення для юзера про вдале створення броні;
- сповіщення про стан платежу.

notificationService.sendToUserPayment(customer.getChatId(), payment);

Система швидко перевірить, чи є контакти користувача, наприклад, чи є chatId, а потім відправить повідомлення в Telegram збагачуючи досвід людей і зменшуючи потребу в ручному управлінні сповіщеннями

Можливість масштабування

Розширення функцій: сервісно-орієнтована будова дозволяє додати нові способи бронирования або управління платою без великих змін у вже розробленій логіці; Наприклад, Додати здатність часткової оплати. Введення відсутності

людського фактора. Додавання з інші платіжні системи, такі як PayPal та Google Pay. Надійність роботи система підтримки під час великих нагрузок забезпечується тим, що керування транзакціями та обмін повідомленнями проходить асинхронно що дозволяє впоратись з високим навантаженням одночасних запитів [10].

3.4 Програмна реалізація основних методів системи

Подивимося на деякі важливі частини виконання на прикладі сервісу BookingServiceImpl, що відповідає за керування бронюванням в системі. Основні функції сервісу бронювання:

1. Створення нового бронювання

Метод save(Long customerId, BookingRequestDto requestBookingDto) виконує наступні дії:

- перевіряє наявність вільних місць в будинку на цей період, щоб уникнути конфлікту бронювань;
- змінює запит DTO в Booking за мапером;
- зберігає бронювання зі статусом PENDING;
- надсилає повідомлення користувачеві через Telegram, враховуючи його chatId.

Лістинг 3.16.

```
    public BookingResponseDto save(Long customerId, BookingRequestDto
requestBookingDto) {
        // Перевірка доступності житла
        List<Booking>               byAccommodationId                =
bookingRepository.findByAccommodationId(requestBookingDto.getAccommoda
tionId());
        // Логіка перевірки перекриттів бронювань...
        // Збереження бронювання

bookingToSave.setCustomer(customerRepository.findById(customerId).orEl
seThrow(
            ()                              ->                              new
EntityNotFoundException(String.format(CUSTOMER_NOT_FOUND_MESSAGE,
customerId))
```

```
        ));
        // Надсилання повідомлення
        if (customer.getChatId() != null) {

notificationService.sendToUserBookingSuccessful(customer.getChatId(),
savedBookingDto);
        }
        return savedBookingDto;
    }
```

2. Отримання списку бронювань користувача

Метод getAll(Long customerId, Pageable pageable) дає список бронювань які належать певному користувачу. Для оптимізації запитів є пагінація.

Лістинг 3.17.

```
    public List<BookingResponseDto> getAll(Long customerId, Pageable
pageable) {
    return bookingMapper.toDtos(
    bookingRepository.findAllByCustomerId(customerId, pageable)
    }
```

3. Пошук бронювання за його ідентифікатором

Метод findById(Long customerId, Long id) дає перевірка доступу до їх бронювання. Це значить, що якщо юзер не є власником цього бронювання то буде створено виключення UnauthorizedActionException.

Лістинг 3.18.

```
    public BookingResponseDto findById(Long customerId, Long id) {
        Booking booking = bookingRepository.findById(id)
            .orElseThrow(()                    ->                    new
EntityNotFoundException(String.format(BOOKING_NOT_FOUND_MESSAGE,
id)));
        if (!booking.getCustomer().getId().equals(customerId)) {
            throw                                                    new
UnauthorizedActionException(String.format(UNAUTHORIZED_FIND_MESSAGE,
customerId, booking.getCustomer().getId()));
        }
        return bookingMapper.toDto(booking);
    }
```

4. Оновлення статусу або налаштувань бронювання

- Метод updateById з параметрами дозволяє редагувати дані заїзду та виїзду, але тільки для власного бронювання сам метод приймає Long customerId Long id.

- Метод updateStatus оновлює статуси бронювання та відправляє повідомлення за оновлення користувачу.

Лістинг 3.19.

```
    public BookingResponseDto updateStatus(Long id,
BookingUpdateStatusRequestDto requestDto) {
        Booking booking = bookingRepository.findById(id).orElseThrow(
            () -> new
EntityNotFoundException(String.format(BOOKING_NOT_FOUND_MESSAGE, id))
        );
        booking.setStatus(requestDto.getStatus());

notificationService.bookingStatusChangedMessage(bookingMapper.toDto(bo
oking));
        return bookingMapper.toDto(bookingRepository.save(booking));
    }
```

5. Скасувати бронювання

Метод deleteById перевірить чи користувач робив бронювання якщо роби, то видаляє бронювання в іншому випадку прокине UnauthorizedActionException.

Лістинг 3.20.

```
    public void deleteById(Long customerId, Long id) {
        Long bookingCustomerId =
bookingRepository.findById(id).orElseThrow(
            () -> new
EntityNotFoundException(String.format(BOOKING_NOT_FOUND_MESSAGE, id))
        ).getCustomer().getId();
        if (customerId.equals(bookingCustomerId)) {
            bookingRepository.deleteById(id);
            notificationService.bookingCanceledMessage(id);
        } else {
            throw new
UnauthorizedActionException(String.format(UNAUTHORIZED_DELETE_MESSAGE,
customerId, bookingCustomerId));
        }
    }
```

6. Пошук за параметрами

Метод search формує запит для пошуку за заданими фільтрами що забезпечує отримування релевантних результатів. Цей метод фільтрує данні що будуть приходити у вигляді списку моделей DTO.

Лістинг 3.21.

```
    public List<BookingResponseDto> search(BookingSearchParametersDto
searchParameters) {
        Specification<Booking>            specification            =
specificationBuilder.build(searchParameters);
        return
bookingMapper.toDtos(bookingRepository.findAll(specification));
    }
```

7. Авто перевірка

Метод checkBookingDate() працює по графіку. Він змінює стан усіх бронювань, чия дата виїзду пройшла, на EXPIRED Усі зміни надсилаються разом з повідомленнями

Лістинг 3.22.

```
    @Scheduled(cron = "@daily")
    private void checkBookingDate() {
        LocalDate nowDate = LocalDate.now();
        List<Booking> bookings = bookingRepository.findAll();
        for (Booking booking : bookings) {
            if      (booking.getCheckOut().isBefore(nowDate)      &&
booking.getStatus() != EXPIRED) {
                booking.setStatus(EXPIRED);
                bookingRepository.save(booking);

notificationService.bookingExpiredMessage(String.format(EXPIRED_MESSAG
E, booking.getId()));
            }
        }
    }
```

3.5 Тестування розробленого програмного продукту на основі тестових даних

Тест є важливою частиною розробки програм, бо воно допомагає знайти помилки на якомога ранішній стадії і показує, що функція відповідає запитам. У цьому розділі буде описано тестування яке робилося на рівні сервісу та доступу до бази даних для модулів які займаються управлінням даних у системі.

Для цих тестів використовується JUnit 5, Mockito та особливі анотації для роботи з базою даних у тестовому середовищі. Ось приклади тестування сервісів `AccommodationServiceImpl` і `AccommodationRepository`.

3.5.1 Тестування

Сервісу AccommodationService є класом, який треба перевірити на правильність відповіді на запит що приходить від користувача, а також на зв'язок з репозиторієм. Перевірка такого класу може бути зроблена за допомогою техніки імітації залежностей оскільки вона допомагає протестувати повадки сервісу без об'єднання справжніх частин коду.

1. Тестування методів

Метод `getAll` повертає списокк всіх доступних будинків з підтримкою пагінації. У тесті перевіряється, чи правильно сервіс обробляє запит і формує список DTO до показу даних.

- Приклад `AccommodationRepository`, що віддає збережені дані.

- Пустий клас `AccommodationMapper` показу екземпляри моделі на рівні DTO.

Лістинг 3.23.

```
    @Test
    void getAll_ValidPageable_ReturnAllAccommodations() {
        Pageable pageable = PageRequest.of(0,10);
        List<Accommodation> accommodations =
List.of(accommodation);
        Page<Accommodation> accommodationPage =
                new PageImpl<>(accommodations, pageable,
accommodations.size());
        List<AccommodationResponseDto> expected =
List.of(accommodationResponseDto);

Mockito.when(accommodationRepository.getAll(pageable)).thenReturn(a
ccommodationPage);
Mockito.when(accommodationMapper.toDto(accommodation)).thenReturn(a
ccommodationResponseDto);
        List<AccommodationResponseDto> actual =
accommodationService.getAll(pageable);
        assertEquals(expected, actual, "Expected accommodations
should be: " + expected
                + " but was: " + actual);
    }
```

Тестовий метод містить ситуацію, де дані успішно повертаються з використанням сервісу. Результат перевіряється на співпадіння з очікуваним результатом за допомогою методу `assertEquals` .

2. Модульне тестування методу findById

Метод `findById` викликається на даному об'єкті житла і передається його ID. Тест перевіряє, чи вірно сервіс створює модель DTO з отриманої моделі сховища.

1. Метод `findById` викликає репозиторій з емуляцією.

2. Маппінг виклик: мапера для DTO.

3. Порівняння фактичного з прогнозом.

Лістинг 3.24.

```
        @Test
        void
  findById_ValidAccommodation_ReturnAccommodationResponseDto() {
            Long id = 1L;
          Mockito.when(accommodationRepository.findById(id))
                  .thenReturn(Optional.ofNullable(accommodation));
          Mockito.when(accommodationMapper.toDto(accommodation))
                  .thenReturn(accommodationResponseDto);
          AccommodationResponseDto expected = accommodationResponseDto;
          AccommodationResponseDto              actual              =
accommodationService.findById(id);
          assertEquals(expected, actual, "Expected accommodation should
be: " + expected
                  + " but was: " + actual);
      }
```

Цей тест успішний лише для випадку, коли дані отримані правильно і виклик залежностей є правильним.

3.5.2 Розміщення Репозиторію

Ми можемо перевірити рівень доступу до бази даних окремо за допомогою анотації `@DataJpaTest`. Це дає можливість робити тести для сховищ, створюючи пробну базу даних з наперед заданими тестовими даними для перевірки методів сховища в ізоляції [40].

1. Тестування метода `findById`.

Тест має перевірити, чи правильно методи застосовуються при пошуку та поверненні об'єкта [40].

Тестові дані можуть загрузитись перед тестом в базу даних за допомогою анотації `@Sql`

Створюються об'єкти `Accommodation` і `Address`, відповідно.

- Виконується перевірка відповідності збережених у базі даних даних з даними очікуваних.

Лістинг 3.25.

```
      @Test
      @Sql(scripts = "classpath:db/"
              +    "accommodations/add-accommodations-and-addresses-to-
tables.sql",
              executionPhase = Sql.ExecutionPhase.BEFORE_TEST_METHOD)
      @Sql(scripts = "classpath:db/"
                + "accommodations/delete-accommodations-and-addresses-
  from-tables.sql",
                executionPhase = Sql.ExecutionPhase.AFTER_TEST_METHOD)
        void findById_AccommodationWithAddress_ReturnAccommodation() {
            Address address = new Address()
                    .setId(1L)
                    .setCountry("Ukraine")
                    .setCity("Lviv")
                    .setStreet("Bohdan Khmelnytsky")
                    .setBuilding(100);
            Accommodation expected = new Accommodation()
                    .setId(1L)
                    .setType(Accommodation.Type.HOUSE)
                    .setAddress(address)
                    .setSize("100 sq.m.")
                    .setAmenities("All")
                    .setPrice(new BigDecimal("100.00"))
                    .setAvailableUnits(1);
            Accommodation                    actual                    =
  accommodationRepository.findById(1L).get();
            assertEquals(expected,  actual,  "Expected  accommodation
  should be: " + expected
                    + " but was: " + actual);
        }
        setAddress(   setSize(«100   sq   .set   .set   .setPrice(new
  BigDecimal(»100.00"))   openAvailable   Accommodation   actual   =
  accommodationRepository.findById(   assertEquals(expected,   actual,
  “Очікуване житло повинно бути: ” + expected) + » але було: « + actual);
  }
```

Це для того, щоб встановити, чи працює коректно пошук в БД, і чи є всі необхідні дані.

Перевірка з допомогою підготовлених SQL-скриптів з первинними даними служать для заповнення бази даних потрібними значеннями дозволяє впевнитися що всі тести йдуть в умовах контролю коли результати можна передбачити.

3.6 Висновки до розділу

У цьому розділі вдосконалено головні аспекти платфори та розроблено моделі, методи й алгорими для керування орендою, котра має на меті покращити чинні рішення за допомогою новітніх технологій. Платформа вирішує труднощі з якими стикаються існуючі сервіси: складнощі масштабування, ручне управління цінами і брак ефективних систем безпеки. Завдяки мікросервісній архітектурі надає високий рівень масштабованості та гнучкості що дозволяє підлаштувати систему до росту кількості користувачів і обсягу великої кількості даних.

Крім того платформа має модульну структуру, що дає можливість легко підтримати, розширити та перевіряти систему. Взаємодія з такими зовнішніми сервісами як Stripe для обробка платежів і Telegram для спілкування з користувачами забезпечує зручний доступ до всіх функцій платформи. Використання JWT для аутентифікації і авторизація підвищує рівень безпеки а також зберігання даних у PostgreSQL дозволяє ефективно працювати з великим обсягом інформації.

Визначено ефективну взаємодію алгоритмів між користувачами й орендодавцями. Розроблені алгоритми дозволяють постійно перевіряти наявність вільних місць під замовлення, мають можливість зберігаюти дані з належними статусами, та автоматично повідомляють користувачів про їх зміни через Telegram та інші меседжери. Таким чином платформа не тільки автоматизує процеси управління орендою а й підвищує рівень зручності і безпеки для користувачів.

ВИСНОВКИ

У магістерській рооботі було покращено платформу для управління орендою житла, що має на меті вирішення проблем сучасних систем. Також увага була приділена на впровадженні новітніх моделей, методів і алгоритмів які забезпечують швидкість роботи платформи, автоматизацію процесів бронювання, плати. Було збільщено зручність для користувачів шляхом підвищенню рівня безпеки даних, а також адаптивністю до зростаючих обсягів даних і кількості юзерів.

Платформа побудована на мікросервісній структурі, що дає високу гнучкість, масштабованість і стабільність роботи. Модульна структура дозволяє легко додати нові функції, адаптувати систему до змінюючих умов роботи платформи, а також спростити підтримку та розширення платформи. Цей спосіб дозволяє зберігати ефективність платформи навіть при значному росту навантаження.

Для покращення роботи платформи були додані зовнішні сервіси, як Stripe API для безготівкових платежів і Telegram API для сповіщень користувачів. Це рішення дозволяє автоматизувати обробку оплат, надсилати повідомлення про статус бронювання та забезпечити зв'язок з юзерами через бота.

Розроблені алгоритми, які перевіряють наявність вільних місць для бронювання, оновлюють статуси в реальному часі та забезпечують правильність даних. Розроблені алгоритми дають змогу працювати з великим обсягом інформації, зменшуючи ризики помилок.

Удосконалено й використано новітні алгоритми, які полегшили користування платформою, розроблено спосіб, що дозволило оптимізувати запити до бази даних покращуючи роботу з об'єктами. Застосовано потоки й ланцюги фільтрів(filter chains), що прискорило роботу платформи і зменшило помилки під час перенавантаження платформи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. "Spring in Action". Craig Walls. Manning Publications. 2022. Archived from the original on 20 March 2023. Retrieved 2023-05-15.
2. "Java: The Complete Reference". Herbert Schildt. McGraw-Hill. 2022. Archived from the original on 15 April 2023. Retrieved 2023-07-01.
3. "Spring Boot in Practice". Somnath Musib. Manning Publications. 2021. Archived from the original on 1 March 2023. Retrieved 2023-04-20.
4. "Clean Code: A Handbook of Agile Software Craftsmanship". Robert C. Martin. Prentice Hall. 2008. Archived from the original on 12 May 2022. Retrieved 2022-09-10.
5. "Effective Java". Joshua Bloch. Addison-Wesley. 2018. Archived from the original on 8 February 2023. Retrieved 2023-03-05.
6. "Head First Java". Kathy Sierra, Bert Bates. O'Reilly Media. 2022. Archived from the original on 4 January 2023. Retrieved 2023-02-15.
7. "Microservices Patterns". Chris Richardson. Manning Publications. 2018. Archived from the original on 10 June 2022. Retrieved 2022-12-01.
8. "Java Performance: The Definitive Guide". Scott Oaks. O'Reilly Media. 2014. Archived from the original on 25 September 2022. Retrieved 2023-01-10.
9. "Building Microservices". Sam Newman. O'Reilly Media. 2021. Archived from the original on 14 October 2022. Retrieved 2023-02-20.
10. "Designing Data-Intensive Applications". Martin Kleppmann. O'Reilly Media. 2017. Archived from the original on 30 March 2022. Retrieved 2022-11-15.
11. "Spring Microservices in Action". John Carnell. Manning Publications. 2017. Archived from the original on 10 August 2022. Retrieved 2022-12-20.
12. "Java Concurrency in Practice". Brian Goetz. Addison-Wesley. 2006. Archived from the original on 3 July 2022. Retrieved 2023-01-05.
13. "Refactoring: Improving the Design of Existing Code". Martin Fowler. Addison-Wesley. 2018. Archived from the original on 9 March 2023. Retrieved 2023-06-01.
14. "RESTful Web Services". Leonard Richardson, Sam Ruby. O'Reilly Media. 2007. Archived from the original on 2 May 2022. Retrieved 2022-08-15.

15. "Learning Spring Boot 3.0". Greg L. Turnquist. Packt Publishing. 2022. Archived from the original on 18 September 2022. Retrieved 2023-02-25.

16. "Test-Driven Development: By Example". Kent Beck. Addison-Wesley. 2002. Archived from the original on 11 January 2023. Retrieved 2023-04-01.

17. "JUnit in Action". Vincent Massol, Ted Husted. Manning Publications. 2020. Archived from the original on 24 May 2022. Retrieved 2023-03-15.

18. "Practical Unit Testing with JUnit and Mockito". Tomek Kaczanowski. Lulu.com. 2019. Archived from the original on 22 November 2022. Retrieved 2022-12-30.

19. "Spring Security in Action". Laurentiu Spilca. Manning Publications. 2020. Archived from the original on 8 February 2023. Retrieved 2023-03-25.

20. "Pro Spring Security". Carlo Scarioni. Apress. 2013. Archived from the original on 1 June 2022. Retrieved 2023-01-15.

21. "Docker: Up & Running". Karl Matthias, Sean Kane. O'Reilly Media. 2020. Archived from the original on 19 March 2022. Retrieved 2022-10-25.

22. "Kubernetes: Up and Running". Brendan Burns, Joe Beda, Kelsey Hightower. O'Reilly Media. 2022. Archived from the original on 16 May 2023. Retrieved 2023-06-20.

23. "PostgreSQL: Up and Running". Regina Obe, Leo Hsu. O'Reilly Media. 2021. Archived from the original on 21 October 2022. Retrieved 2023-01-20.

24. "High Performance PostgreSQL". Gregory Smith, Korry Douglas. O'Reilly Media. 2015. Archived from the original on 2 August 2022. Retrieved 2022-12-05.

25. "Liquibase in Action". Robert Reeves, Nathan Voxland. Manning Publications. 2014. Archived from the original on 15 September 2022. Retrieved 2022-11-01.

26. "Mastering MapStruct". Gunnar Morling. Packt Publishing. 2020. Archived from the original on 28 February 2023. Retrieved 2023-04-10.

27. "Mockito Cookbook". Marcin Grzejszczak. Packt Publishing. 2014. Archived from the original on 20 July 2022. Retrieved 2023-02-05.

28. "Mastering Mockito". Sujoy Acharya. Packt Publishing. 2017. Archived from the original on 4 June 2023. Retrieved 2023-07-10.

29. "Swagger & OpenAPI: Building Robust APIs". Josh Ponelat, Lukas Rosenstock. Manning Publications. 2022. Archived from the original on 13 April 2023. Retrieved 2023-05-25.

30. "Learning AWS". Aurobindo Sarkar, Amit Shah. Packt Publishing. 2018. Archived from the original on 30 March 2022. Retrieved 2022-09-20.

31. "Programming Amazon Web Services". James Murty. O'Reilly Media. 2008. Archived from the original on 5 May 2023. Retrieved 2023-06-15.

32. "Stripe API Integration Guide". Mitesh Soni. Independently Published. 2021. Archived from the original on 9 July 2022. Retrieved 2023-03-10.

33. "Java for Beginners". Sharanam Shah, Vaishali Shah. SPD Publications. 2018. Archived from the original on 22 August 2023. Retrieved 2023-09-05.

34. "Software Architecture in Practice". Len Bass, Paul Clements, Rick Kazman. Addison-Wesley. 2012. Archived from the original on 6 October 2022. Retrieved 2022-11-10.

35. "Domain-Driven Design". Eric Evans. Addison-Wesley. 2003. Archived from the original on 17 April 2023. Retrieved 2023-05-05.

36. "Modern Software Development with Java". Kevin Wittkopf. Springer. 2021. Archived from the original on 25 December 2022. Retrieved 2023-01-15.

37. "Head First Software Development". Dan Pilone, Russ Miles. O'Reilly Media. 2008. Archived from the original on 30 November 2022. Retrieved 2023-01-05.

38. "Programming Telegram Bots". Mark Zamoysky. Independently Published. 2019. Archived from the original on 9 June 2023. Retrieved 2023-07-01.

39. "Building RESTful Web Services with Spring 5". Raja CSP Raman. Packt Publishing. 2018. Archived from the original on 20 October 2022. Retrieved 2023-02-15.

40. "Effective Software Testing". Maurício Aniche. Manning Publications. 2022. Archived from the original on 12 May 2023. Retrieved 2023-06-25.

41. "Spring Boot 2.0 Cookbook". Alex Soto, Jason Porter, and Andy Gumbrecht. Packt Publishing. 2018. Archived from the original on 15 June 2022. Retrieved 2023-05-10.

42. "Clean Architecture: A Craftsman's Guide to Software Structure and Design". Robert C. Martin. Prentice Hall. 2017. Archived from the original on 3 February 2023. Retrieved 2023-04-18.

43. "The Pragmatic Programmer: Your Journey to Mastery". Andrew Hunt, David Thomas. Addison-Wesley. 2020. Archived from the original on 22 September 2023. Retrieved 2023-05-22.

44. "Design Patterns: Elements of Reusable Object-Oriented Software". Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Addison-Wesley. 1994. Archived from the original on 25 July 2022. Retrieved 2023-03-05.

45. "Spring 5 Recipes: A Problem-Solution Approach". Marten Deinum, Daniel Rubio, and Josh Long. Apress. 2018. Archived from the original on 8 June 2023. Retrieved 2023-06-12.

ДОДАТКИ

Додаток А

Фрагменти програмниих репозиторіїв

```
        @Repository
        public interface AccommodationRepository extends
    JpaRepository<Accommodation, Long> {
            //дістаємо житло за конкретною адресою
            @Query("select a from Accommodation a join fetch
    a.address")
            Page<Accommodation> getAll(Pageable pageable);
            //дістаємо житло за конкретним id
          @Query("select a from Accommodation a join fetch a.address
where a.id = :id")
            Optional<Accommodation> findById(Long id);
        }
        @Component
        @RequiredArgsConstructor
        public class BookingSpecificationBuilder implements
    SpecificationBuilder<Booking> {
            private final SpecificationProviderManager<Booking>
    specificationProviderManager;
            @Override
            public Specification<Booking>
    build(BookingSearchParametersDto searchParameters) {
                Specification<Booking> spec =
    Specification.where(null);

                if (searchParameters.customerIds() != null &&
    searchParameters.customerIds().length > 0) {
                    spec =
    spec.and(specificationProviderManager.getSpecificationProvider("cus
    tomer_id")

    .getSpecification(searchParameters.customerIds()));
                }

                if (searchParameters.statuses() != null &&
    searchParameters.statuses().length > 0) {
                    spec =
    spec.and(specificationProviderManager.getSpecificationProvider("sta
    tus")

    .getSpecification(searchParameters.statuses()));
                }

                return spec;
            }
        }
        @Component
        @RequiredArgsConstructor
        public class BookingSpecificationProviderManager implements
```

```
SpecificationProviderManager<Booking> {
        private static final String
CANNOT_FIND_SPECIFICATION_BY_KEY
                = "Can't find specification by key: ";
        private final List<SpecificationProvider<Booking>>
bookingSpecificationProviders;

        @Override
        public SpecificationProvider<Booking>
getSpecificationProvider(String key) {
            return bookingSpecificationProviders.stream()
                    .filter(spec -> spec.getKey().equals(key))
                    .findFirst()
                    .orElseThrow(() -> new
RuntimeException(CANNOT_FIND_SPECIFICATION_BY_KEY + key));
        }
    }
    @Component
    public class CustomerIdSpecificationProvider implements
SpecificationProvider<Booking> {
        private static final String CUSTOMER_ID = "customer_id";

        @Override
        public String getKey() {
            return CUSTOMER_ID;
        }
         //задаємо специфікації для фільтрації
        public Specification<Booking> getSpecification(String[]
params) {
            return (root, query, criteriaBuilder)
                    -> root.join("customer").get("id").in(params);
        }
    }

    @Component
    public class StatusSpecificationProvider implements
SpecificationProvider<Booking> {
        private static final String STATUS = "status";

        @Override
        public String getKey() {
            return STATUS;
        }
    //задаємо специфікації для фільтрації
        public Specification<Booking> getSpecification(String[]
params) {
            return (root, query, criteriaBuilder)
                    -> root.get(STATUS).in((Object[]) params);
        }
    }

  @Repository
```

```
    public interface BookingRepository extends JpaRepository<Booking,
Long> {
        List<Booking> findAll(Specification<Booking> specification);
    //дістаємо тільки список замовлень за id користувачів
         @Query(value = "SELECT b FROM Booking b JOIN FETCH
  b.accommodation"
                + " JOIN FETCH b.customer WHERE b.customer.id =
  :customerID")
        List<Booking> findAllByCustomerId(Long customerID, Pageable
  pageable);
    //дістаємо тільки список замовлень за id житла
        @Query(value = "SELECT b FROM Booking b JOIN FETCH
b.accommodation"
                 + " JOIN FETCH b.customer WHERE b.accommodation.id
  = :accommodationId")
        List<Booking> findByAccommodationId(Long accommodationId);
    }
```

Додаток Б

Фрагменти програмниих контроллерів

```
    @RestController
    @RequiredArgsConstructor
    @RequestMapping("/accommodations")
    @Tag(name = "Accommodation management.",
            description = "Endpoints for managing accommodations.")
    public class AccommodationController {
        private final AccommodationService accommodationService;

        //задаємо дозволені ролі для кінцевої точки
        @PreAuthorize("hasRole('ROLE_MANAGER')")
        @PostMapping
        @ResponseStatus(value = HttpStatus.CREATED)
      // задаємо опис для openId
        @Operation(summary = "Save new accommodation.",
                description = "Permits the addition of new
accommodations.",
                security = @SecurityRequirement(name = "bearerAuth"))
        public AccommodationResponseDto save(
                @RequestBody @Valid AccommodationRequestDto
accommodationRequestDto) {
            return
accommodationService.save(accommodationRequestDto);
        }

        //задаємо дозволені ролі для кінцевої точки
        @PreAuthorize("hasRole('ROLE_CUSTOMER')")
        @GetMapping
        @Operation(summary = "Get all accommodations.",
                description = "Provides a list of available
accommodations.",
                security = @SecurityRequirement(name = "bearerAuth"))
        public List<AccommodationResponseDto> getAll(Pageable
pageable) {
            return accommodationService.getAll(pageable);
        }
        //задаємо дозволені ролі для кінцевої точки
        @PreAuthorize("hasRole('ROLE_CUSTOMER')")
        @GetMapping("/{id}")
        @Operation(summary = "Get accommodation by id.",
                description = "Retrieves detailed information about a
specific accommodation.",
                security = @SecurityRequirement(name = "bearerAuth"))
        public AccommodationResponseDto findById(@PathVariable Long
id) {
            return accommodationService.findById(id);
        }
```

```
        //задаємо дозволені ролі для кінцевої точки
        @PreAuthorize("hasRole('ROLE_MANAGER')")
        @PutMapping("/{id}")
      // задаємо опис для openId
        @Operation(summary = "Update accommodation.",
                description = "Allows updates to accommodation
details.",
                security = @SecurityRequirement(name = "bearerAuth"))
        public AccommodationResponseDto
updateDetailsById(@PathVariable Long id,

@RequestBody @Valid

AccommodationUpdateRequestDto requestDto) {
            return accommodationService.updateDetailsById(id,
requestDto);
        }

        //задаємо дозволені ролі для кінцевої точки
        @PreAuthorize("hasRole('ROLE_MANAGER')")
        @PutMapping("/{id}/address")
      // задаємо опис для openId
        @Operation(summary = "Update accommodation address.",
                description = "Allows updates to accommodation
address.",
                security = @SecurityRequirement(name = "bearerAuth"))
        public AccommodationResponseDto
updateAddressById(@PathVariable Long id,

@RequestBody @Valid
                                                    AddressRequestDto
requestDto) {
            return accommodationService.updateAddressById(id,
requestDto);
        }

        //задаємо дозволені ролі для кінцевої точки
        @PreAuthorize("hasRole('ROLE_MANAGER')")
        @DeleteMapping("/{id}")
        @Operation(summary = "Delete accommodation.",
                description = "Enables the removal of
accommodations.",
                security = @SecurityRequirement(name = "bearerAuth"))
        @ResponseStatus(HttpStatus.NO_CONTENT)
        public void deleteById(@PathVariable Long id) {
            accommodationService.deleteById(id);
        }
    }
    @RestController
    @RequestMapping(value = "/auth")
    @RequiredArgsConstructor
      // задаємо опис для openId
    @Tag(name = "Authentication management.",
```

```
            description = "Endpoints for managing authentication.")
    public class AuthenticationController {
        private final AuthenticationService authenticationService;
        private final CustomerService customerService;

        @PostMapping("/registration")
        public CustomerRegistrationResponseDto register(
                @RequestBody @Valid CustomerRegistrationRequestDto
requestDto) {
            return customerService.register(requestDto);
        }

        @PostMapping("/login")
        public CustomerLoginResponseDto login(@RequestBody @Valid
CustomerLoginRequestDto requestDto) {
            return authenticationService.authenticate(requestDto);
        }
    }
    @RestController
    @RequiredArgsConstructor
    @RequestMapping("/bookings")
    @Tag(name = "Booking management.",
            description = "Endpoints for managing bookings.")
    public class BookingController {
        private final BookingService bookingService;
        //задаємо дозволені ролі для кінцевої точки
        @PreAuthorize("hasRole('ROLE_MANAGER')")
        @PatchMapping("/{id}/status")
        @Operation(summary = "Update booking status.",
                description = "Allows manager to change booking
status.",
                security = @SecurityRequirement(name = "bearerAuth"))
        public BookingResponseDto updateStatus(
                @PathVariable Long id,
                @RequestBody @Valid BookingUpdateStatusRequestDto
requestDto
        ) {
            return bookingService.updateStatus(id, requestDto);
        }

        @PreAuthorize("hasRole('ROLE_CUSTOMER')")
        @PostMapping
        @ResponseStatus(HttpStatus.CREATED)
        @Operation(summary = "Create new booking.",
                description = "Permits the creation of new
accommodation bookings.",
                security = @SecurityRequirement(name = "bearerAuth"))
        public BookingResponseDto create(@RequestBody @Valid
BookingRequestDto requestBookingDto,
                                         Authentication
authentication) {
            Customer customer = (Customer)
authentication.getPrincipal();
```

```
            return bookingService.save(customer.getId(),
requestBookingDto);
        }
        //задаємо дозволені ролі для кінцевої точки
        @PreAuthorize("hasRole('ROLE_CUSTOMER')")
        @GetMapping("/my")
        @Operation(summary = "Get all customer bookings",
                description = "Retrieves customer bookings.",
                security = @SecurityRequirement(name = "bearerAuth"))
        public List<BookingResponseDto> getAll(Pageable pageable,
Authentication authentication) {
            Customer customer = (Customer)
authentication.getPrincipal();
            return bookingService.getAll(customer.getId(), pageable);
        }

        @PreAuthorize("hasRole('ROLE_CUSTOMER')")
        @GetMapping("/{id}")
        @Operation(summary = "Get booking by id.",
                description = "Provides information about a specific
booking.",
                security = @SecurityRequirement(name = "bearerAuth"))
        public BookingResponseDto findById(Authentication
authentication,
                                           @PathVariable Long id) {
            Customer customer = (Customer)
authentication.getPrincipal();
            return bookingService.findById(customer.getId(), id);
        }
        //задаємо дозволені ролі для кінцевої точки
        @PreAuthorize("hasRole('ROLE_CUSTOMER')")
        @PutMapping("{id}")
      // задаємо опис для openId
        @Operation(summary = "Update booking by id.",
                description = "Allows customers to update their
booking details.",
                security = @SecurityRequirement(name = "bearerAuth"))
        public BookingResponseDto update(Authentication
authentication,
                                         @PathVariable Long id,
                                         @RequestBody @Valid
BookingUpdateRequestDto requestDto
        ) {
            Customer customer = (Customer)
authentication.getPrincipal();
            return bookingService.updateById(customer.getId(), id,
requestDto);
        }
        //задаємо дозволені ролі для кінцевої точки
        @PreAuthorize("hasRole('ROLE_CUSTOMER')")
        @DeleteMapping("/{id}")
      // задаємо опис для openId
        @Operation(summary = "Delete booking by id.",
```

```
                description = "Enables the cancellation of
bookings.",
                security = @SecurityRequirement(name = "bearerAuth"))
        @ResponseStatus(HttpStatus.NO_CONTENT)
        public void deleteById(Authentication authentication,
@PathVariable Long id) {
            Customer customer = (Customer)
authentication.getPrincipal();
            bookingService.deleteById(customer.getId(), id);
        }

        @PreAuthorize("hasRole('ROLE_MANAGER')")
        @GetMapping("/search")
    // задаємо опис для openId
        @Operation(summary = "Get bookings by customer and status.",
                description = "Retrieves bookings based on customer
ID and their status.",
                security = @SecurityRequirement(name = "bearerAuth"))
        public List<BookingResponseDto> search(
                BookingSearchParametersDto searchParameters
        ) {
            return bookingService.search(searchParameters);
        }
     }
```

Додаток В

Фрагменти програмниих моделей

```
    @Entity
    @Data
    @Table(name = "accommodations")
    @SQLDelete(sql = "UPDATE accommodations SET is_deleted = true
WHERE id=?")
    @Where(clause = "is_deleted=false")
    @Accessors(chain = true)
    public class Accommodation {
        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        private Long id;
        @JdbcTypeCode(SqlTypes.VARCHAR)
        @Column(nullable = false)
        private Type type;
        @OneToOne(fetch = FetchType.LAZY)
        @JoinColumn(name = "address_id", nullable = false)
        @ToString.Exclude
        @EqualsAndHashCode.Exclude
        private Address address;
        @Column(nullable = false)
        private String size;
        @Column(nullable = false)
        private String amenities;
        @Column(nullable = false)
        private BigDecimal price;
        @Column(nullable = false)
        private Integer availableUnits;
        @Column(nullable = false)
        private boolean isDeleted = false;

        public enum Type {
            HOUSE,
            APARTMENT
        }
    }

    @Entity
    @Data
    @NoArgsConstructor
    @Accessors(chain = true)
    @Table(name = "addresses")
    @SQLDelete(sql = "UPDATE addresses SET is_deleted = true WHERE
id=?")
    @Where(clause = "is_deleted=false")
    public class Address {
        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        private Long id;
```

```
        @Column(nullable = false)
        private String city;
        @Column(nullable = false)
        private String country;
        @Column(nullable = false)
        private String street;
        @Column(nullable = false)
        private Integer building;
        private Integer flat;
        @Column(nullable = false)
        private boolean isDeleted = false;
    }

    @Entity
    @Data
    @Table(name = "bookings")
    @SQLDelete(sql = "UPDATE bookings SET is_deleted = true WHERE
id=?")
    @Where(clause = "is_deleted=false")
    public class Booking {
        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        private Long id;
        private LocalDate checkIn;
        private LocalDate checkOut;
        @ToString.Exclude
        @EqualsAndHashCode.Exclude
        @ManyToOne(fetch = FetchType.LAZY)
        @JoinColumn(name = "accommodation_id", nullable = false)
        private Accommodation accommodation;
        @ToString.Exclude
        @EqualsAndHashCode.Exclude
        @ManyToOne(fetch = FetchType.LAZY)
        @JoinColumn(name = "customer_id", nullable = false)
        private Customer customer;
        @JdbcTypeCode(SqlTypes.VARCHAR)
        @Column(nullable = false)
        private Status status;
        @Column(name = "is_deleted", nullable = false)
        private boolean isDeleted = false;

        public enum Status {
            PENDING,
            CONFIRMED,
            REJECTED,
            EXPIRED
        }
    }

    @Entity
    @Table(name = "customers")
    @Data
```

```
    @SQLDelete(sql = "UPDATE customers SET is_deleted = true WHERE id
= ?")
    @Where(clause = "is_deleted=false")
    public class Customer implements UserDetails {
        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        private Long id;
        @Column(nullable = false, unique = true)
        private String email;
        @Column(nullable = false)
        private String firstName;
        @Column(nullable = false)
        private String lastName;
        @Column(nullable = false)
        private String password;
        @ManyToMany(fetch = FetchType.LAZY)
        @ToString.Exclude
        @EqualsAndHashCode.Exclude
        @JoinTable(
                name = "customers_roles",
                joinColumns = @JoinColumn(name = "customer_id"),
                inverseJoinColumns = @JoinColumn(name = "role_id")
        )
        private Set<Role> roles = new HashSet<>();
        @Column(name = "chat_id")
        private Long chatId;
        @Column(name = "is_deleted", nullable = false)
        private Boolean isDeleted = false;

        @Override
        public Collection<? extends GrantedAuthority>
getAuthorities() {
            return roles;
        }

        @Override
        public String getUsername() {
            return email;
        }

        @Override
        public String getPassword() {
            return password;
        }

        @Override
        public boolean isAccountNonExpired() {
            return !isDeleted;
        }

        @Override
        public boolean isAccountNonLocked() {
            return !isDeleted;
```

```
    }
    @Override
    public boolean isCredentialsNonExpired() {
        return !isDeleted;
    }
    @Override
    public boolean isEnabled() {
        return !isDeleted;
    }
}

@Entity
@Data
@Table(name = "payments")
public class Payment {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @JdbcTypeCode(SqlTypes.VARCHAR)
    @Column(nullable = false)
    private Status status;
    @ToString.Exclude
    @EqualsAndHashCode.Exclude
    @OneToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "booking_id", nullable = false)
    private Booking booking;
    @Column(name = "session_url")
    private String sessionUrl;
    @Column(name = "session_id")
    private String sessionId;
    @Column(name = "total")
    private BigDecimal total;
    public enum Status {
        PROCESSING,
        SUCCEED,
        FAILED
    }
}
```