МАГІСТЕРСЬКА РОБОТА

МР. ІПм - 23.00.00.000 ПЗ

Група ІПм-23-2

Іванишин Ростислав

2024

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

Іванишин Ростислав Миколайович

(прізвище, ім'я, по батькові)

УДК 004.4
(індекс)

МАГІСТЕРСЬКА РОБОТА

Моделі, методи та алгоритми створення інноваційного програмного забезпечення в сфері e-commerce

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

Іванишин Р.М.
(підпис, ініціали та прізвище здобувача освітнього ступеня)

Науковий керівник Козак Олексій Федорович, ст. викл.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Допущено до захисту

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

Нормконтроль

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

Освітньо-кваліфікаційний рівень магістр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**
доц. В.В. Бандура
“ 04 ” вересня 2024 р.

ЗАВДАННЯ

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Іванишину Ростиславу Миколайовичу

(прізвище, ім’я, по-батькові)

1. Тема магістерської роботи “Моделі, методи та алгоритми створення інноваційного програмного забезпечення в сфері e-commerce**”**
керівник проєкту (роботи) Козак Олексій Федорович, ст. викл.
затверджені наказом закладу вищої освіти від **“** 22 ” листопада 2024 р. № 781/7

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

3. Вихідні дані до проєкту (роботи) Архітектура, формальний опис та алгоритми функціонування систем електронної комерції

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

1. Процес переходу до електронної комерції (e-commerce)
2. Дослідження використання масштабованих та розподілених рішень
3. Аналіз та обробка великих обсягів даних
4. Розробка програмної моделі та алгоритмів для ефективних рішень в електронній комерції

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

1. Пошукові двигуни та їх популярність за роками (рис. 2.1, ст. 44)
2. Архітектура RESTful Web Service (рис. 3.1, ст. 51)
3. Процесс автентифікації користувача (рис. 3.2, ст. 54)
4. UML Діаграма класів інтерфейсів сервісного рівня (рис. 3.4, ст. 58)
5. UML Діаграма класів реалізації інтерфейсів сервісного рівня (рис. 3.5, ст. 58)
6. UML Діаграма класів реалізації рівня Controller (рис. 3.7, ст. 62)

6. Консультанти розділів проєкту (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

№ п/п	Назви етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Підбір і вивчення літератури	19.09.2024	виконано
2	Аналіз інноваційних технологій в сфері e-commerce	02.10.2024	виконано
3	Дослідження ефективності використання масштабованих рішень	10.10.2024	виконано
4	Платформа Java Spring	25.10.20234	виконано
5	Формулювання вимог та архітектури програмного рішення	04.11.2024	виконано
6	Програмна реалізація рішення	21.11.2024	виконано
7	Затвердження пояснювальної записки роботи завідувачем кафедри	15.12.2024	виконано

Студент – магістр _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

Магістерська робота: 83 с., 22 рис., 45 джерел.

Тема: Моделі, методи та алгоритми створення інноваційного програмного забезпечення в сфері e-commerce

Об'єкт дослідження: моделі, методи та алгоритми створення інноваційного програмного забезпечення в сфері електронної комерції

Мета роботи: розробка ефективних моделей, методів та алгоритмів для створення інноваційного програмного забезпечення, яке відповідає вимогам електронної комерції та здатне забезпечити високу продуктивність, масштабованість і безпеку бізнес-процесів

Предмет дослідження: методи та алгоритми застосовувані в розробці інноваційного програмного забезпечення в сфері електронної комерції. Вони включають в себе вивчення наявних методологій розробки, засобів інтеграції та забезпечення підтримки програмного продукту на протязі всього життєвого циклу.

Результати дослідження:

Проведено аналіз інноваційних технологій створення та підтримки програмного забезпечення, які сприяють ефективному переходу до електронної комерції. На основі отриманих результатів розроблено прототип, в якому продемонстрована практична реалізація досліджених підходів і технологій.

Висновок:

У результаті дослідження було отримано власну систему готельного бізнесу, яка базується на проаналізованих інноваційних технологіях для сфери електронної комерції. Застосовані методології, архітектурні моделі та алгоритми забезпечують необхідну гнучкість, масштабованість і продуктивність програмного забезпечення.

ІННОВАЦІЙНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, АВТОМАТИЗАЦІЯ БІЗНЕС-ПРОЦЕСІВ, ЕЛЕКТРОННА КОМЕРЦІЯ, JAVA SPRING FRAMEWORK, РОЗПОДІЛЕНІ ОБЧИСЛЮВАЛЬНІ СИСТЕМИ, МАСШТАБОВАНІ ПРОГРАМНІ РІШЕННЯ, ОБРОБКА ВЕЛИКИХ ОБСЯГІВ ДАНИХ.

ANNOTATION

Master's work: 83 p., 22 fig., 45 sources.

Topic: Models, methods, and algorithms for developing innovative software in the E-Commerce domain

Object of research: Models, methods, and algorithms for creating innovative software in the field of e-commerce.

Purpose: Develop effective models, methods, and algorithms for creating innovative software that meets the requirements of e-commerce and ensures high performance, scalability, and security of business processes.

Subject of research: Methods and algorithms applied in the development of innovative software for e-commerce. This includes examining existing development methodologies, integration tools, and ensuring product support throughout its lifecycle.

Research results:

An analysis of innovative technologies for software development and maintenance was conducted, facilitating an efficient transition to e-commerce. Based on the results, a prototype was developed that demonstrates the practical implementation of the researched approaches and technologies.

Conclusion:

As a result of the research, a proprietary hotel business system was created, based on the analyzed innovative technologies for the e-commerce field. The applied methodologies, architectural models, and algorithms provide the necessary flexibility, scalability, and performance of the software.

INNOVATIVE SOFTWARE, BUSINESS PROCESS AUTOMATION, E-COMMERCE, JAVA SPRING FRAMEWORK, DISTRIBUTED COMPUTING SYSTEMS, SCALABLE SOFTWARE SOLUTIONS, BIG DATA PROCESSING.

ЗМІСТ

Стр.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

e-commerce – електронна комерція

API - прикладний програмний інтерфейс

RESTful API - прикладний програмний інтерфейс передачі репрезентативного стану

SDK - набір із засобів розробки

OpenCV - бібліотека комп’ютерного зору з відкритим вихідним кодом

ПЗ - програмне забезпечення

DDOS - Атака на відмову в обслуговуванні

ОС - операційна система

TLS - Transport Layer Security

HTTPS - протокол безпечної передачі даних

RSA - криптографічний алгоритм з відкритим ключем

AES - симетричний алгоритм блочного шифрування

ВСТУП

Актуальність роботи

У зв’язку із зростаючою потребою в розробці інноваційних програмних рішень для електронної комерції актуальність даної роботи залишається високою оскільки цифровий сектор економіки продовжує стрімко розвиватися. У світлі глобалізації та цифровізації бізнес-процесів, багато підприємств знаходяться в пошуку ефективних способів інтеграції сучасних технологій для оптимізації власної діяльності. Рішення на основі інноваційного програмного забезпечення дозволяють компаніям зменшити витрати, підвищити продуктивність та забезпечити персоналізований підхід до обслуговування клієнтів, що є важливим для конкуренції на сучасному ринку.

Перехід до електронної комерції для багатьох підприємств є необхідністю для підтримки власної конкурентоздатності, однак даний процес вимагає висококваліфікованих розробників та технологічних експертів. Правильно обрані та впроваджені моделі з алгоритмами створення інноваційних рішень можуть забезпечити безперебійну роботу програмних сервісів і їх здатності адаптуватися до зміни попиту, нових технологій та інтересів споживачів. З чого випливає, що питання розробки ефективного програмного забезпечення, яке забезпечує гнучкість і масштабованість для розвитку електронної комерції, набуває особливої важливості.

Використання передових технологій у сфері розробки програмного забезпечення дозволяє підприємствам отримати високий рівень автоматизації процесів, це в свою чергу значно покращує взаємодію з клієнтами і підвищує ефективність роботи самого бізнесу. Для прикладу технології на кшталт машинного навчання, аналізу великих даних та штучний інтелект, стають важливими інструментами для забезпечення персоналізованих пропозицій та покращення якості обслуговування. В зв’язку з чим наведені технології дедалі частіше застосовуються в електронній комерції, тому розробка методів для їх успішної інтеграції є одним з ключових аспектів дипломної роботи.

Особливо важливою є роль створення та підтримки програмних сервісів з високими вимогами до масштабованості та доступності, через що e-commerce платформи схильні до значних навантажень під час пікових періодів. Тому наукове дослідження в даній сфері, зосереджене на розробці розподілених систем і технологій хмарних обчислень, сприяє створенню більш продуктивних і стійких рішень для бізнесу. Вміле використання наявних ресурсів, завдяки застосуванню відповідних засобів дозволяє підприємствам зменшити витрати на інфраструктуру, надавати якісні послуги та відповідати на сучасні виклики.

Мета і задачі дослідження

Метою магістерської роботи є розробка ефективних моделей, методів та алгоритмів для створення інноваційного програмного забезпечення, яке відповідає вимогам електронної комерції та здатне забезпечити високу продуктивність, масштабованість і безпеку бізнес-процесів. Дослідження зосереджене на вивченні сучасних підходів до розробки програмних рішень, таких як розподілені програмні системи, використання хмарних технологій, а також інтеграції новітніх технологій, зокрема придатних для обробки великих даних, забезпечення безпеки банківських операцій, для автоматизації та оптимізації операцій в сфері електронної комерції.

До переліку завдань входить визначення найбільш ефективних методів підтримки та розвитку програмних продуктів після їх впровадження, у тому числі інтеграції з існуючими корпоративними системами, масштабування рішень у відповідь на зростання обсягів даних і трафіку, забезпечення гнучкості для постійної адаптації до нових потреб користувачів. Мета роботи також передбачає виявлення основних аспектів забезпечення переходу підприємства до електронної комерції, та розробку практичних рекомендацій щодо вирішення основних проблем з впровадженням інноваційних програмних рішень.

Об’єктом дослідження є моделі, методи та алгоритми створення інноваційного програмного забезпечення в сфері електронної комерції

Предметом дослідження є методи та алгоритми застосовувані в розробці інноваційного програмного забезпечення в сфері електронної комерції. Вони включають в себе вивчення наявних архітектурних моделей, методологій розробки, засобів інтеграції та забезпечення підтримки програмного продукту на протязі всього життєвого циклу.

Методи дослідження включають комплексний підхід до аналізу, проєктування та впровадження програмних рішень для електронної комерції. Вони містять теоретичне дослідження наявних рішень та процесу переходу бізнесу до електронної комерції, розглядають засоби досягнення високої продуктивності, надійності та безпеки, включають в себе експерементальне моделювання, разом з розробкою та тестуванням прототипу цільового програмного продукту.

Наукова новизна одержаних результатів

Здійснено аналіз інноваційних технологій створення та підтримки програмного забезпечення, що дозволяють бізнесу ефективно переходити до електронної комерції. У роботі запропоновано нові підходи до використання сучасних методів проєктування адаптованих під потреби динамічного бізнес-середовища, також досліджено алгоритми оптимізації процесів інтеграції програмних продуктів із існуючими системами.

Практичне значення одержаних результатів.

На основі проведеного дослідження було розроблено функціонуючий прототип програмного забезпечення для готельного бізнесу, який дозволяє автоматизувати обробку пошуку, рекомендацій та бронювання житла і є готовим до інтеграції з платіжними, бухгалтерськими або аналітичними системами для подальшого розширення наявних послуг.

Особистий внесок

1. Описано ключові етапи та особливості переходу бізнесу до електронної комерції
2. Досліджено ефективність та доцільність використання масштабованих програмних рішень, наведені недоліки та переваги основних технологій
3. Розглянуто методи оптимізації алгоритмів, які використовуються в електронній комерції, зокрема для обробки великих обсягів даних, покращення продуктивності, забезпечення безпеки та створення ефективних механізмів управління й впровадження програмного продукту.
4. Запропоновано архітектуру, алгоритм функціонування та реалізацію програмної системи для готельного бізнесу з високим рівнем оптимізації, масштабованості та надійності.

Публікації. За результатами наукових досліджень, проведених у магістерській роботі, підготовлено доповідь для участі на науково-технічній конференції студентів інституту інформаційних технологій Івано-Франківського національного технічного університету нафти і газу, а також представлені тези доповіді та презентація на всеукраїнській науково-практичній конференції молодих учених і студентів “Інформаційні технології в освіті, техніці та промисловості” (ІТОТП-2024), яка відбулась 10 жовтня 2024 року в м. Івано-Франківськ.

Структура магістерської роботи.

Магістерська робота викладена на 83 сторінках друкованого тексту, який складається з вступу, трьох розділів, висновків, списку використаних джерел (45 найменувань). Робота містить 22 рисунки.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ВІДОМОСТІ ПРО ІННОВАЦІЙНІ ТЕХНОЛОГІЇ В СФЕРІ E-COMMERCE

1.1 Теоретичні відомості про сферу цифрової економіки

1.1.1 Поняття електронної комерції (e-commerce)

Електронна комерція - це електронна торгівля, яка передбачає проведення бізнес-транзакцій які здійснюються в мережі Інтернет, через мобільні пристрої та інші цифрові засоби. Електронна комерція охоплює різні види діяльності, включаючи купівлю та продаж товарів і послуг, переказ коштів і даних, а також обмін діловою документацією з іншими компаніями.

Електронна комерція спирається на різноманітні технології та інструменти, які забезпечують взаємодію між споживачами, компаніями та постачальниками за допомогою онлайн-платформ, включаючи електронну пошту, онлайн-каталоги, пошукову оптимізацію, протокол передачі файлів і веб-сервіси. До переваг електронної комерції можна віднести підвищену доступність товарів і послуг, цілодобову доступність, швидкість доступу та глобальне охоплення [1].

Сьогодні транзакції електронної комерції можуть відбуватися як між бізнесом і споживачами, так і між самими споживачами, розвиток електронної комерції суттєво змінив традиційну бізнес-модель завдяки створенню нових каналів продажів та можливостей для зростання малого та середнього бізнесу. Термін «електронна комерція» часто використовується як взаємозамінний з терміном «бізнес», який позначає ширший спектр діяльності, пов'язаної з веденням бізнесу в Інтернеті, включаючи маркетинг, рекламу та обслуговування клієнтів [2].

1.1.2 Категорії інноваційного програмного забезпечення

Окрім e-commerce, також існують інші типи програмних продуктів, котрі охоплюють різноманітні сфери життя та бізнесу. Нижче наведений список основних категорій:

- E-Learning (електронне навчання) – це програми та платформи для онлайн-освіти, курсів, тренінгів, платформи для дистанційного навчання. Приклади: Moodle, Coursera, Duolingo.
- Fintech (фінансові технології) – це програмне забезпечення для управління фінансами, цифрові гаманці, онлайн-банкінг, кредитні та інвестиційні платформи. Приклади: PayPal, Revolut, Robinhood.
- EdTech (освітні технології) – це інструменти для поліпшення процесу навчання, включаючи онлайн-платформи для взаємодії викладачів і студентів. Приклади: Kahoot!, Google Classroom.
- Social Media (соціальні мережі) – це платформи для взаємодії людей в Інтернеті, обміну інформацією, створення та поширення контенту. Приклади: Facebook, Instagram, Twitter.
- HealthTech (технології в охороні здоров'я) – це програмне забезпечення для медичних установ, моніторингу здоров'я пацієнтів, дистанційного консультування. Приклади: MyChart, Teladoc.
- GovTech (урядові технології) - це платформи для автоматизації урядових процесів, надання громадських послуг через Інтернет. Приклади: Gov.uk, e-Government Services.
- Smart City (розумне місто) – програмне забезпечення для управління міськими ресурсами, інфраструктурою, транспортом і енергоспоживанням. Приклади: Citymapper, UrbanFootprint.
- PropTech (технології в нерухомості) – це програмні рішення для управління нерухомістю, покупок, продажів і оренди. Приклади: Zillow, Redfin.
- Cybersecurity (кібербезпека) – програмне забезпечення для захисту інформаційних систем від атак, захист персональних даних. Приклади: Norton, McAfee.
- HRTech (технології для управління людськими ресурсами) – це інструменти для автоматизації процесів підбору, управління персоналом, оцінки ефективності. Приклади: BambooHR, Workday, LinkedIn.

- IoT (Internet of Things - Інтернет Речі) – програмне забезпечення для управління пристроями та системами, підключеними до Інтернету (розумні будинки, транспортні системи тощо). Приклади: SmartThings, Nest.
- LegalTech (правові технології) – програмні інструменти для автоматизації правових процесів, надання юридичних послуг онлайн. Приклади: Clio, Rocket Lawyer.
- AgriTech (сільськогосподарські технології) – це програмне забезпечення для аграрної індустрії, що допомагає в управлінні фермами, оптимізації виробництва. Приклади: John Deere Operations Center, CropX.
- AI & Machine Learning Applications (Штучний інтелект та машинне навчання) – це алгоритми та програмні продукти, що використовують штучний інтелект для автоматизації, аналітики даних, обробки мовних запитів тощо. Приклади: Google AI, OpenAI, IBM Watson.

Поданий список був наведений у зв’язку з тим, що дані категорії можуть бути комбіновані між собою або охоплювати інші специфічні напрямки, в залежності від потреб і інновацій в конкретній галузі. Електронна комерція часто тісно зв’язана з декількома із поданих категорій. Для прикладу технології Fintech можуть бути застосовані для автоматизації онлайн оплати послуг та сервісів, а LegalTech сформує та оформить необхідну фінансову звітність, тим самим зменшуючи навантаження на бухгалтерію та інший персонал бізнесу.

1.1.3 Види та переваги електронної комерції

Електронна комерція поділяється на кілька видів залежно від типу учасників, які беруть участь у здійсненні транзакцій. Основними типами електронної комерції є:

– B2B (business-to-business),
– B2C (business-to-consumer),
– C2C (consumer-to-consumer),
– C2B (consumer-to-business).

Тип B2B передбачає транзакції між підприємствами, що здійснюються через інтернет-платформи або інші електронні канали. Така форма комерції охоплює

оптову торгівлю, закупівлю матеріалів, постачання товарів для подальшої реалізації та інші операції між організаціями. B2B-системи дозволяють значно знизити витрати, покращити ефективність постачання та збільшити швидкість бізнес-процесів, оскільки вони автоматизують взаємодію між компаніями [2].

B2C є найбільш поширеним і включає онлайн-торгівлю, де підприємства безпосередньо продають свої товари та послуги кінцевим споживачам. Цей тип комерції дозволяє компаніям досягати великої аудиторії через інтернет-магазини, платформу електронних платежів, маркетингові кампанії та інші онлайн-ресурси.

C2C передбачає торгівлю між споживачами, зазвичай через онлайн-аукціони або платформи для обміну товарами й послугами, що створюють можливість для індивідуальних осіб здійснювати покупку та продаж без посередників. В свою чергу, C2B включає моделі, де споживачі пропонують продукти або послуги бізнесам. Це може бути, наприклад, модель фріланс-платформ, де користувачі надають свої професійні послуги компаніям, або ж модель, при якій споживачі надають бізнесу свої дані або ресурси для подальшого використання. Кожен з розглянутих вище типів електронної комерції відрізняється за механізмами взаємодії між учасниками та специфікою використання технологій для організації бізнес-процесів [2].

Електронна комерція надає численні переваги, які роблять її привабливою як для підприємців, так і для кінцевих споживачів, що в свою чергу значно трансформує традиційні бізнес-моделі.

Однією з основних переваг є зручність, оскільки покупці мають можливість здійснювати покупки в будь-який час і з будь-якої точки світу, за умови наявності доступу до Інтернету. Це є особливо актуальним в умовах сучасного ритму життя, коли люди прагнуть максимально ефективно використовувати свій час, а також шукають способи оптимізації процесу покупки. На додачу, зручність використання електронних платформ дозволяє значно знизити час, витрачений на пошук і придбання товарів, що сприяє підвищенню рівня задоволеності споживачів.

Широкий вибір товарів і послуг, котрі доступні через інтернет-магазини, є ще однією ключовою перевагою електронної комерції. Завдяки великому асортименту покупці можуть швидко знайти необхідний товар, а можливість порівняння цін та

характеристик з різних джерел дає змогу здійснити обґрунтований вибір на основі детальної інформації. Для підприємств електронна комерція означає суттєве зменшення витрат, пов'язаних з орендою комерційних приміщень, утриманням персоналу та іншими операційними витратами, оскільки значна частина процесів автоматизується.

Можливість збору та аналізу даних про поведінку користувачів на онлайн-платформах є також важливим аспектом інтеграції електронної комерці, що дозволяє компаніям персоналізувати свої пропозиції та маркетингові стратегії, орієнтуючись на індивідуальні потреби споживачів. Вдосконалена аналітика дозволяє компаніям отримувати глибше розуміння потреб своєї цільової аудиторії, а також створювати більш ефективні стратегії взаємодії. Також електронна комерція відкриває можливості для виходу бізнесів на глобальні ринки, що є важливим фактором для малих і середніх підприємств, які раніше не мали доступу до міжнародної конкуренції. Глобалізація через Інтернет дозволяє таким компаніям значно розширювати свою аудиторію та залучати клієнтів з різних країн, що стимулює розвиток підприємництва в умовах жорсткої конкурентної боротьби.

1.1.4 Етапи створення та функціонування сегменту електронної комерції

Створення та функціонування успішного сегменту електронної комерції включає в себе кілька основних етапів, кожен з яких вимагає ретельної підготовки та огляду [2].

Першим етапом є концептуалізація та розгляд бізнес-моделі. На цьому етапі підприємство повинно визначити основні цілі, які воно хоче досягти через електронну комерцію, а також обрати тип електронної комерції (B2B, B2C, C2C тощо), що буде відповідати специфіці його діяльності. Також необхідно провести дослідження цільового ринку, вивчити потреби аудиторії і конкурентне середовище, для формування унікальних провозиці, котрі дозволяють бізнесу відрізнятися від інших компаній.

Другим етапом є розробка та створення онлайн-платформи. Цей етап включає розробку веб-сайту або мобільного додатку, котрий має бути зручним, інтуїтивно

зрозумілим і технологічно стабільним. Вибір платформи для електронної комерції повинен враховувати можливості інтеграції з іншими системами, на зразок платіжних шлюзів, систем управління запасами та аналітичними інструментами. Важливо забезпечити надійність безпеки даних, захист від кібератак і реалізацію методів захисту конфіденційної інформації клієнтів, оскільки вона є критично важливою для підтримання довіри користувачів.

Третім етапом є налаштування та інтеграція платіжних систем. Він включає в себе вибір та налаштування методів оплати, наприклад кредитних карток, електронних гаманців, банківських переказів і систем оплати через мобільні пристрої. На даному етапі є важливим забезпечення зручності та безпеки платіжних операцій, оскільки від цього залежить рівень задоволення користувачів і довіра до компанії. Платіжні системи повинні бути інтегровані з платформою таким чином, щоб процес оплати був швидким, зручним і безперешкодним для користувачів, при цьому всі операції мають відповідати вимогам безпеки, зокрема стандарту PCI DSS (Payment Card Industry Data Security Standard).

Четвертий етап полягає в розробці та реалізації маркетингових стратегій, спрямованих на залучення та утримання клієнтів. Це включає пошукову оптимізацію (SEO), контекстну рекламу, використання соціальних медіа, email-маркетинг, а також акції та знижки для стимулювання попиту. Важливо враховувати специфіку поведінки користувачів в інтернеті, їх переваги та уподобання, що дозволяє створювати персоналізовані рекламні пропозиції. Крім того, на цьому етапі слід реалізувати стратегії взаємодії з клієнтами через різноманітні канали комунікації, зокрема через чат-боти, служби підтримки та зворотній зв'язок для підтримки високого рівня клієнтського сервісу.

П'ятим та останнім етапом є моніторинг та оптимізація процесів, що дозволяє забезпечити безперервне покращення функціонування сегменту електронної комерції. Після запуску платформи необхідно постійно відслідковувати її ефективність, наприклад застосовуючи аналітичні засоби - Google Analytics або спеціалізовані платформи для аналізу поведінки користувачів. Даний етап включає аналіз продажів, ефективності маркетингових кампаній, а також рівня задоволеності

клієнтів. На основі зібраних даних можна коригувати стратегії, покращувати функціональність сайту, пропонувати нові продукти або послуги, та впроваджувати інновації. Все перелічене дозволяє підтримувати конкурентоспроможність і своєчасно відповідати на динамічні вимоги ринку.

1.1.5 Перспективи розвитку

З кожним роком все більше споживачів віддають перевагу онлайн-покупкам, що стимулює розвиток нових технологій та бізнес-моделей. Електронна комерція відкриває широкі можливості для бізнесу та споживачів. Вона забезпечує зручність, доступність, великий вибір товарів і послуг, а також дозволяє бізнесам швидко адаптуватися до змін на ринку, використовувати нові можливості і залучати більше клієнтів [1].

У найближчому майбутньому мобільна комерція і соціальна комерція продовжуватимуть активно розвиватися, і це стане основною рушійною силою для багатьох бізнесів. Як правило нові користувачі використовують мобільні пристрої для здійснення покупок, що зумовлює потребу в оптимізації онлайн-магазинів під мобільні платформи. Соціальні мережі, виступаючи в ролі інструменту для просування товарів та послуг набувають все більшої популярності, у зв’язку з тим, що платформи на кшталт Instagram та Facebook стають каналами для прямого продажу та взаємодії з кінцевими споживачами. У цьому контексті, важливим аспектом буде інтеграція новітніх технологій, для прикладу штучного інтелекту та машинного навчання, для персоналізації користувацького досвіду та покращення ефективності маркетингових стратегій.

Не менш важливим напрямом є вдосконалення логістичних і доставочних процесів, котрі суттєво покращать ефективність електронної комерції. Рішення, засновані на автоматизації, роботизації та використанні дронів для доставки, можуть значно знизити витрати на транспортування та скоротити час доставки товарів. Інноваційні логістичні технології ідентично допоможуть покращити управління запасами та передбачити попит, що сприятиме оптимізації ресурсів і зменшенню витрат для компаній. Новітні підходи до управління ланцюгами постачання

сприятимуть зменшенню логістичних затримок і забезпечать більшу прозорість на всіх етапах процесу доставки [3].

Розвиток електронної комерції відкриває нові горизонти для бізнесу, забезпечуючи створення робочих місць і розвитку нових індустрій. Іноваційні технології дають підприємствам можливість вийти на нові ринки, знижувати витрати та покращувати якість обслуговування. Перспективи даного сектору є великими, оскільки цифрові платформи постійно еволюціонують, і компанії, які готові адаптуватися до нових умов, можуть значно підвищити свою конкурентоспроможність.

1.2 Процес розробки та підтримки порталів електронної комерції

1.2.1 Ключові етапи переходу бізнесу

Для успішного здійснення переходу до електронної комерції, бізнесу необхідно пройти декілька ключових етапів, кожен з яких вимагає детального планування та виконання для забезпечення успішної трансформації [3].

На першому етапі відбувається оцінка готовності бізнесу до електронної комерції, під час якої компанія проводить аналіз доступних ресурсів, потреби цільової аудиторії, визначає цілі які повинні бути досягнуті під час інтеграції. Також відбувається дослідження конкурентного середовища та потенційних ризиків

На другому етапі відбувається безпосередня розробка та впровадження онлайн-платформи. Тобто створення веб-сайту або мобільного додатку, інтеграцію з платіжними системами та організацію процесів управління товарами, обробки замовлень і логістики. Важливо забезпечити зручну навігацію та привабливий дизайн, що сприятиме позитивному зворотному відгуку у відвідувачів платформи. Дизайн сайту має відповідати бренду компанії і враховувати потреби цільової аудиторії, також він повинен бути адаптивним під різні типи пристроїв. Від вибору цільової платформи залежить складність та вартість створення інтерфейсу, забезпечення безпеки та захисту конфіденційних даних.

Третім етапом є реалізація маркетингової стратегії для залучення клієнтів. Вона включає в себе оптимізацію сайту для пошукових систем (SEO), контекстну рекламу, просування через соціальні мережі, email-маркетинг та інші канали комунікації з клієнтами. SEO передбачає добавлення якісного контенту для залучення трафіку, використання ключових слів, створення метатегів, оптимізацію швидкості завантаження сайту та інші технічні аспекти.

На четвертому етапі відбувається налаштування і оптимізація процесів обслуговування клієнтів. Вона включає інтеграцію платіжних систем і логістики, системи підтримки клієнтів, рекомендаційних систем. Надання висококласного обслуговування в електронній комерції є важливим аспектом для збереження клієнтської лояльності та підвищення репутації бізнесу.

Останній етап це моніторинг, аналіз та адаптація стратегії розвитку. Після запуску платформи необхідно активно збирати та аналізувати великі обсяги інформації, чого можна досягнути через застосування аналітичних засобів, наприклад Google Analytics, для оцінки поведінки користувачів, кількості замовлень, показників конверсії і задоволеності клієнтів. Аналітика може допомогти виявити можливі проблеми в роботі сайту чи платформи, а також знайти слабкі місця у маркетингових кампаніях. Опираючись на отримані дані бізнес може корегувати стратегію розвитку, вдосконалювати інтерфейс, змінювати ціноутворення і загалом покращувати досвід користувачів сервісів.

1.2.2 Вибір та інтеграція e-commerce рішення для конкретного бізнесу

Опираючись на етапи переходу бізнесу та отримавши список вимог до функціональності платформи, можна приступити до ретельного розгляду підходящого рішення. Для малого чи середнього бізнесу може бути достатньо готових рішень. На ринку таку роль виконують CMS – системи уплавління вмістом веб-сайтів, ключовими технологіями виступають Shopify або WooCommerce, які забезпечують основні функції для онлайн-продажів але не підходять для великих підприємств або компаній з унікальними потребами. Для них може знадобитися

розробка індивідуальної платформи, що дозволить максимально адаптувати систему до специфічних бізнес-задач [4].

На сучасному ринку для високотехнологічних та продуктивних систем розглядають в основному два напрямки - C# ASP.NET та Java Spring. Менш ефективними, але швидкими і дешевими у розробці вважаються напрямки Python Django і JavaScript.

Окремо розглядають технології для розробки користувацького інтерфейсу, найбільш популярними є Vue.js, Angular і React.

Для мобільних пристоїв кожна платформа вимагає власного засобу, наприклад Andriod Studio і Swift для Apple iPhone iOS. Однак деякі сучасні програмні засоби розробки інтерфейсу починають пропонувати універсальність і мультиплатформенність, наприклад React Native підтримує як Android так і iOS системи.

1.3 Дослідження ефективності використання масштабованих програмних рішень для виробничих систем

1.3.1 Актуальність використання масштабованих програмних рішень

Актуальність використання масштабованих програмних рішень зростає в умовах швидкого розвитку технологій і постійного збільшення обсягів даних, котрі в свою чергу потрібно обробляти. Сучасні веб-додатки та платформи потребують здатності обробляти великий потік користувацьких запитів і значні обсяги інформації в реальному часі. Інтеграція масштабованих програмних рішень дозволить динамічно налаштовувати кількість обчислювальних ресурсів, своєчасно реагуючи на зміни і забезпечуючи стабільне функціонування системи при зростанні кількості користувачів або даних. Розподілені системи, в свою чергу, забезпечують високу доступність та надійність для критично важливих сервісів, а також підвищують стійкість систем до відмов окремих вузлів та компонентів, тобто підвищуючи загальну стійкість до звоїв. Представлені засоби дозволяють вирішити перелічені проблеми, або зменшити їхній вплив на бізнес, що відповідно спонукає компанії

розглядати перспективу переходу в сторону масштабованості та розподіленості власних сервісів [6].

1.3.2 Типи архітектури програмного забезпечення

Монолітна архітектура - це класичний підхід, який використовується при створенні програмних додатків. Він був домінуючим підходом до розробки програмного забезпечення протягом багатьох років і все ще широко використовується на даний момент [7].

Монолітна архітектура об'єднує і запускає всі компоненти програми як єдину, уніфіковану систему. Тому будь-які зміни, внесені в одну частину програми, можуть потенційно вплинути на всю систему.

Також варіантом архітектури для веб-сервісу є СОА (сервіс-орієнтована архітектура) - багатошарова (три або більше рівнів) архітектура, кожен шар якої відповідає за певний обов'язок:

- 1 шар - користувальницький інтерфейс, найчастіше веб-інтерфейс
- 2 шар - бізнес-логіка: розрахунки, агрегація даних тощо.
- 3 шар - база даних (сховище)

У сервіс-орієнтованій архітектурі (SOA) сервіси функціонують незалежно і надають функціональність або обмін даними своїм споживачам. Споживач запитує інформацію і надсилає вхідні дані в службу. Сервіс обробляє дані, виконує завдання і надсилає відповідь. Наприклад, якщо додаток використовує сервіс авторизації, він надає йому ім'я користувача та пароль. Сервіс перевіряє їх і повертає відповідну відповідь [7]. Основні принципи даної архітектури є наступними:

Забезпечення сумісності. Кожен сервіс в SOA включає документи опису, які визначають функціональність сервісу та пов'язані з ним умови. Будь-яка клієнтська система може запустити сервіс, незалежно від базової платформи або мови програмування. Наприклад, бізнес-процеси можуть використовувати сервіси, написані як на C#, так і на Python. Оскільки немає прямих взаємодій, зміни в одному сервісі не впливають на інші компоненти, що використовують цей сервіс.

Слабка взаємозв’язність. Сервіси в SOA мають бути слабко зав’язаними, мати якомога менше залежностей від зовнішніх ресурсів, таких як моделі даних або інформаційні системи. Вони також мають бути статичними, не зберігаючи жодної інформації з минулих сесій або транзакцій. Таким чином, якщо змінити сервіс, це не матиме істотного впливу на клієнтські додатки та інші сервіси, що використовують цей сервіс.

Абстрагування. Клієнти або користувачі сервісів у SOA не зобов'язані знати логіку коду сервісу або деталі його реалізації. Для них сервіси мають виглядати як чорний ящик. Клієнти отримують необхідну інформацію про те, що робить сервіс і як його використовувати, через контракти на обслуговування та інші документи з описом сервісу.

Степінь деталізації. Сервіси в SOA повинні мати відповідний розмір і обсяг, в ідеалі упаковуючи одну дискретну бізнес-функцію в один сервіс. Розробники можуть використовувати кілька сервісів, щоб створити комбінований сервіс для виконання складних операцій.

Сервісно-орієнтована архітектура (SOA) може підійти для створення великих корпоративних застосунків, однак для неї необхідна більша гнучкість в процесі масштабування невеликих бізнес-додатків. SOA містить наступні обмеження:

- Корпоративна сервісна шина (ESB) з'єднує кілька сервісів разом, що робить її критичною точкою при нагрузках.
- Усі сервіси використовують спільний репозиторій даних. Це ускладнює індивідуальне управління сервісами.
- Кожен сервіс має широку сферу застосування. Таким чином, відмова одного із сервісів вплине на весь бізнес-процес.

Тому розробники звертаються до архітектури мікросервісів для більш детального підходу до створення додатків.

Мікросервісна модель розділяє сервіс SOA на більш дрібні сервіси. Кожен мікросервіс працює в обмеженому контексті та незалежно від інших сервісів. Архітектура мікросервісів має обмежені або відсутні взаємозалежності між окремими

сервісами та знижує ризик відмови всієї системи [7]. Пошукові API є прикладом винесененя логіки пошуку в окремий мікросервіс.

Сервісно-орієнтована архітектура охоплює ширшу сферу діяльності підприємства. Різні бізнес-підрозділи ефективно взаємодіють на єдиній платформі обміну даними. Навпаки, мікросервіси належать до вужчої сфери.

Наприклад, управління складськими запасами буде SOA-сервісом системи електронної комерції. Але мікросервісний підхід дасть змогу розділити управління складськими запасами на більш дрібні сервіси, як-от перевірка доступності, виконання замовлень і бухгалтерський облік.

Розгортання сервісів SOA може виявитися складним завданням, оскільки вони певною мірою пов'язані один з одним. Наприклад, розробники повинні перебудувати весь додаток, якщо вони змінюють або додають новий сервіс. Крім того, додатки SOA не можуть повною мірою скористатися перевагами контейнеризації, яка абстрагує додаток від операційних систем та обладнання. Водночас мікросервіси простіше розгортати, оскільки вони призначені для масштабування в хмарному середовищі. Кожен мікросервіс - це незалежний застосунок, який розробники можуть розміщувати в контейнерах і розгортати в хмарі.

Коли застосовувати Моноліт, SOA або мікросервіси

Монолітна архітектура підходить для малих бізнесів, вона забезпечує достатній рівень продуктивності та значною мірою зменшує складність процесу розробки, тестування, розгортання і підтримки програмного рішення, оскільки весь необхідний функціонал розміщується в одному програмному пакеті. Однак вона стикається із проблемами при значному розширенні функціоналу програми. Для управління десятками розрізнених сервісів компанії вибирають одну із двох наступних архітектур:

SOA – організації із застарілими або автономними корпоративними додатками виграють від архітектури SOA. SOA спрощує звичайну систему програмного забезпечення, розбиваючи її на більш дрібні модульні частини. Це рішення також об'єднує загальні ресурси для оптимізації бізнес-функцій. Замість створення

дублюючих і резервних сервісів розробники можуть повторно використовувати наявні сервіси SOA для впровадження більшої кількості бізнес-рішень.

Мікросервіси - архітектура мікросервісів - варіант для підтримки гнучких команд розробників. Дозволяє швидко і поступово вносити зміни до коду, не впливаючи на стабільність застосунку, використовуючи інструменти безперервної інтеграції та безперервної доставки (CI/CD). Краще використовувати мікросервіси, коли розробники ставлять перед собою такі цілі:

- Використання різних мов програмування, бібліотек або фреймворків для створення одного додатку
- Об'єднання окремих сервісів, побудованих з використанням різних програмних фреймворків
- Надання обчислювальних ресурсів і масштабування окремих сервісів у режимі реального часу

Завдяки мікросервісам компанії можуть скористатися сучасними хмарними можливостями та з легкістю розгортати сотні мікросервісів [7].

1.3.3 Переваги розподілених систем

В контексті масштабованих програмних рішень в сфері електронної комерції, найбільш поширеними реалізаціями розподілених систем на даний момент є наступні:

Клієнт-серверна система – як базова форма розподіленої архітектури, в якій клієнти (браузери, мобільні додатки, інші сервери) взаємодіють із сервером для обробки даних. В поданій системі масштабованість досягається шляхом розподілення на окремі віртуальні та фізичні машини і збереженні даних в цетралізованих або розподілених базах даних [8].

Мікросервісна архітектура – як тип розподіленої системи, де програмна реалізація серверної частини розбивається на мінімізовані сервіси, кожен з яких відповідає за вирішення чіткої задачі і містить конкретний, характерний тільки для неї функціонал. Масштабованість в мікросервісах досягається шляхом роботи на

різних серверах або контейнерах, які застосовують спеціалізоване API для комунікації між собою [9].

Системи черг повідомлень – при застосуванні мікросервісної архітектури постає проблема вибору спеціалізованого API обміну повідомленнями між компонентами і збереженні асинхронності у виконанні процесів. Прикладом реалізації черги надсилання та отримання повідомлень є RabbitMQ, Apache Kafka та AWS SQS

Розподілені бази даних – застосовуються для масштабування даних стрімко зростаючих веб-систем. Тобто дані перестають зберігатись в одній базі даних, а знаходяться в кількох розподілених між фізичними машинами, серверами або дата-цетнрами. Рішення на прикладі Apache Cassandra або Google Bigtable забезпечують високу доступність, масштабованість та стійкість до збоїв, дозволяючи обробляти великі обсяги даних [8].

Системи кешування - Redis або Memcached є прикладом відомих розподілених кеш-систем, які інтегрують у випадку необхідності зберігання часто використовуваних тимчасових даних, з ціллю зменшення часу доступу до них і відповідно збільшення швидкості роботи з ними. Дані системи можуть також бути розподілені між серверами і датацентрами [8].

Обчислення в хмарі (Cloud computing) – це хмарні платформи, які дозволяють запускати веб-додатки на численних серверах в дата-центрах з будь якої точки світу. Наразі все частіше застосовуються як ядро і центр розміщення, зберігання та доступу до даних, завдяки необмеженим обчислюваним ресурсам, глобальній розподіленій інфраструктурі, високій надійності, та простоті у використанні. Найпопулярнішими є Amazon Web Services (AWS), Microsoft Azure та Google Cloud

Платформи для обробки великих даних (Big Data) – тільки зберігання та доступу до розподілених даних не є достатньо для забезпечення сучасних потреб бізнесу. На даний час все частіше ставиться вимога інтеграції засобів ефективної обробки та аналізу даних в реальному часі, з ціллю збору аналітики та формування рекомендаційних систем. Прикладом таких платформ є Apache Hadoop, Spark або Kafka

1.3.4 Недоліки розподілених систем

Розподілені системи містять низку недоліків, зв’язаних із складністю та потребою у координації між розрізненими компонентами. Ключовою проблемою є управління складністю, оскільки в розглянутих системах застосовується значна кількість різних серверів, баз даних і комунікаційних каналів, котрі створюють додаткові проблеми в плануванні, реалізації та підтримці інфраструктури. Вони потребують постійного моніторингу та налаштування для забезпечення стабільної роботи, що загалом є трудомістким і вимагає залучення значних ресурсів [8-9].

Також із самої концепції розподілення випливає проблема узгодженості. У випадку з інформаційними системами це узгодженість даних, які можуть зберігатися та опрацьовуватись одночасно на кількох вузлах і відповідної забезпечення консистентності являє собою нову та складну задачу. Впровадження механізмів синхронізації та обміну даними між вузлами може призводити до затримок, помилок у даних а також їхній несумісності, особливо при великих навантаженнях або в умовах мережевих збоїв.

Останньою ключовою проблемою є надійність та відмовостійкість. В деяких аспектах розподілені системи забезпечують кращу надійність, аніж їхні аналоги, однак вони все ще залишаються вразливими до критичних збоїв, наприклад збої в одному з компонентів можуть спричинити ланцюгову реакцію, що повпливає на інші частини системи. Механізм відстеження, запобігання та відновлення систем є одним з ключових викликів в сучасній мережевій інфраструктурі. На винекнення збоїв можуть повпливати також і зовнішні чинники, такі як DDoS атаки і пошук вразливостей в інфраструктурі недобросовісними конкурентами і хакерами. Ускладнення інфраструктури вимагає додаткових інвестицій в безпеку і забезпечення функціонування резервних систем [8-10].

1.4. Висновки до розділу

В даному розділі було проведено аналіз та розглянуто поняття електронної комерції, наведені альтернативні та суміжні категорії інноваційних цифрових

напрямків. Також визначені переваги, види і області застосування електронної комерції, проаналізовані ключові етапи переходу бізнесу, відображено процес розробки та підтримки порталів електронної комерції.

Окрім того було проведено дослідження ефективності використання масштабованих програмних рішень для виробничих систем, які в свою чергу мають значний вплив на процес розробки програмного продукту і охоплюють не тільки не тільки електронну комерцію, а і загалом сучасну індустрію мережевих застосунків. Розглянуто визначення та типи архітектури програмного забезпечення, чому правильний вибір архітектури є критично важливим для подальшого циклу розробки, наведені приклади застосування сервіс-орієнтованої та мікросервісної архітектури. Також розглянуті розподілені системи в контексті масштабованих програмних рішень, проаналізовані найбільш поширені реалізації розподілених систем, їхні переваги й недоліки загалом як ядра електронної комерції.

РОЗДІЛ 2
ДОСЛІДЖЕННЯ ЗАСОБІВ СТВОРЕННЯ ІННОВАЦІЙНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Технології створення програмного забезпечення

2.1.1 Проблема вибору технологій, налаштування та розгортання веб-сервісів

Вибір технологій для трансформування бізнесу в електронну комерцію є важливою та складною задачею. Вона вимагає врахування багатьох факторів (специфіки потреб бізнесу, очікуване навантаження, наявний бюджет, технічна підготовка команди розробників, тощо.). Ринок програмного забезпечення містить широкий спектр фреймворків, мов програмування та архітектурних підходів, визначитись із якими навіть підготованому спеціалісту складно. Неоптимальний вибір технологій, які не відповідають реальним потребам, породжує проблеми із масштабованістю, низькою продуктивністю, зростаючою складністю підтримки та модернізацією системи [11].

Окрім самих технологій також важлива їхня правильна конфігурація, тобто інтеграція компонентів, налаштування середовища виконання та забезпечення надійності системи. Більшість розроблюваних веб-сервісів вимагають налаштування баз даних, файлових систем, серверів із мережевою інфраструктурою та систем логування. Все перелічене створює навантаження на DevOps-інженерів [11]. Неправильна конфігурація може стати критичною для безпеки додатку, призвести до нестабільності роботи або неефективного використання ресурсів, що безпосередньо повипливає на якість створюваного сервісу.

Розгортання веб-серверів створює ще один шар ускладнення, оскільки потребується забезпечення безперебійності роботи, особливо в умовах розподілених систем. Необхідно визначитись в необхідності створення власної інфраструктури або впровадженням нових рішень, на прикладі хмарних сервісів та контейнеризації, інтеграції автоматизованих інструментів розгортання (CI/CD-пайплайнів), тобто

зменшенні ризику людського фактору, в обмін на підвищення вимог до команди розробників та вимоги в ретельному плануванні кожного процесу.

Розглянемо передові на сьогоднішній день механізми для успішного вирішення поставлених завдань, при врахуванні наявних можливостей та створенням перспектив для подальшого розвитку як бізнесу так і програмного забезпечення.

2.2 Платформа Java Spring

2.2.1 Характеристика фреймворку Spring

Spring Framework — це популярний, відкритий фреймворк для розробки корпоративних додатків на платформі Java, в якому забезпечується гнучка інфраструктура для створення масштабованих і модульних програмних рішень.

Spring побудований навколо архітектури IoC (Inversion of Control). Він являється контейнером інверсії управління та повсемісно використовує ін’єкцію залежностей. У даному фреймворку Spring Beans є основними компонентами програми, які управляються контейнером Spring. Вони представляють собою об'єкти, створені, ініціалізовані та конфігуровані за допомогою контейнера, який також керує їхнім життєвим циклом [12].

Анотації в Spring відіграють ключову роль у спрощенні конфігурації та налаштувань програми, забезпечуючи декларативний підхід до визначення поведінки компонентів. Завдяки прикладному використанню анотацій, розробники позначають класи, методи або поля для передачі їх під автоматичне управління контейнером Spring, усуваючи необхідність у створенні на протиписуванні значної кількості XML-конфігурацій. Для зразку анотації @Component, @Service, і @Repository можуть бути застосовані для автоматичної реєстрації компонентів у контейнері, @Autowired забезпечує ін'єкцію залежностей. Інші анотації, для прикладу @Transactional, @RestController або @RequestMapping, додають функціональність для управління транзакціями, створення REST API та маршрутизації запитів від користувача [12] .

З допомогою анотацій Spring підтримує більш читабельний, зручний і модульний спосіб налаштування додатків, тобто заощаджуючи час розробників та прискорюючи загалом весь процес.

Аспектно-орієнтоване програмування (АОП) також є невідємною складовою фреймворку і дозволяє впроваджувати крос-функціональну логіку окремо від основної бізнес-логіки додатку - логування, безпеку, управління транзакціями, обробку помилок.

АОП базується на понятті аспектів, які визначають, де і як крос-функціональна логіка інтегрується в додаток, використовуючи точки приєднання (join points) та вирази відповідності (pointcuts) [13]. У Spring АОП реалізується через проксі-об'єкти, які динамічно перехоплюють виклики методів і виконують визначену логіку аспекту до або після виконання методу. Це в свою чергу сприяє зменшенню дублікатів коду, покращенню його структури та забезпечує можливість централізованого управління додатковими функціями без зміни основного коду програми.

Також завдяки модульній архітектурі, Spring дозволяє розробникам обирати лише ті компоненти, які відповідають потребам проєкту, що сприяє оптимізації роботи та зменшенню складності коду.

2.2.2 Екосистема Spring

Опираючись на Spring Core, детальніше розписаному в підпункті 2.2.1, було створено значну кількість проєктів для вирішення конкретного роду завдань. Нижче розглядаються найбільш поширені із них:

Spring Boot - вирішує проблему розгортання Java проєктів і складності налаштування компонентів (серверів, баз даних, з'єднання між модулями, конфігурації додатків тощо). Все перелічене вимагає значного часу розробки і спричиняє високий ризик виникнення помилок при інтеграції розрізнених частин системи. Spring Boot дозволяє спростити даний процес завдяки вбудованим інструментам прискорюючим розробку програми і в свою чергу зосереджуючи розробників на логіці бізнес-процесів, а не на інфраструктурних питаннях [14].

Spring Data – забезпечує єдине API для роботи із сховищами даних, для прикладу з реляційними базами даних (JPA, JDBC) і нереляційними (MongoDB, Cassandra). Проєкт автоматизує створення запитів і дозволяє уникнути дублювання коду.

Spring Security – забезпечує безпеку програми шляхом інтеграції рішень для аутентифікації, авторизації, шифрування даних і захисту від атак (CSRF або SQL-ін'єкцій). Він інтегрується з іншими модулями Spring, що робить його важливою частиною розробки корпоративних додатків.

Spring Session - вирішує завдання управління сесіями в розподілених середовищах. Даний проєкт дозволяє зберігати дані сесій у зовнішніх сховищах, (Redis, JDBC, MongoDB), забезпечуючи в свою чергу безперервність сесій при роботі через балансування навантаження

Spring Cloud – створений для підтримки розподілених систем і мікросервісів у контексті хмарного сховища та пропонуючи інструменти для управління ним. (конфігурування, балансування навантаження, виявлення сервісів (Eureka), маршрутизація (Zuul, Gateway) тощо).

Spring WebFlux - забезпечує реактивне програмування, що дозволяє обробляти асинхронні запити з мінімальним споживанням ресурсів.

Spring Batch - призначений для автоматизації пакетних обчислень, таких як обробка великих обсягів даних або регулярні завдання.

Spring Integration – надає засоби для інтеграції із зовнішніми системами. Підтримує відомі патерни інтеграції підприємств

Spring AMQP – створений для роботи з повідомленнями через RabbitMQ

Spring AI – проєкт для розробки штучного інтелекту. Його мета - застосувати до області ШІ принципи проєктування екосистеми Spring, такі як портативність і модульний дизайн.

Детальніше дані проєкти описані в документації [31].

2.3 Методологія автоматизації технологічних процесів складання, налаштування та розгортання програмного забезпечення

2.3.1 Загальна характеристика DevOps розробника

DevOps-розробник являється фахівцем інтеграції розробки (Development) і експлуатації (Operations) для підтримки безперервного циклу розробки, тестування, та впровадження програмного забезпечення. Він відповідає за створення автоматизованих процесів для прискорення випуску нових версій програмного забезпечення, із збереженням стабільності та якості наявної системи. Для мінімізації ручної роботи команди та підвищення продуктивності застосовуються такі інструменти автоматизації як Jenkins, Docker, Kubernetes, Terraform, проводиться налаштування CI/CD (Continuous Integration/Continuous Deployment) конвеєрів [17-18].

DevOps інженер працює на стику кількох дисциплін, таких як програмування, адміністрування серверів і забезпечення безпеки. До його обов’язків входить моніторинг продуктивності додатків, управління хмарною інфраструктурою (AWS, Azure, Google Cloud Platform) та вирішення проблем, пов’язаних із продуктивністю, масштабованістю або стабільністю системи.

2.3.2 Алгоритм автоматизації розгортання серверної частини Java Spring додатку

Розгортання серверної сторони Java додатку, яке налаштовує DevOps, є складним і багатоступінчастим процесом, що включає автоматизацію всіх етапів - від розробки до продуктивного середовища. Алгоритм послідовності дій є наступним:

1. Підготовка середовища для розгортання – DevOps спеціалісту необхідно обрати платформу для хостингу додатку, яка в свою чергу може бути як локальним сервером, так і хмарною платформою (Amazon Web Services, Microsoft Azure або Google Cloud). Вибір платформи залежить від вимог до масштабованості, безпеки та доступності додатку. Після цього налаштовується середовище для запуску додатку,

яке включає операційну систему, необхідні бібліотеки, інструменти та утиліти для управління контейнерами та віртуалізацією.

2. Налаштування інфраструктури як коду (Infrastructure as Code, IaC) - DevOps застосовує інструменти - Terraform, Ansible або Puppet, для автоматизації розгортання інфраструктури. Що в свою чергу дозволяє описати всю інфраструктуру у вигляді конфігураційних файлів, які можна версіонувати. Це спрощує відтворення середовища на різних етапах розробки або тестування. З використанням IaC можливо забезпечити консистентність серверного середовища та швидко відновлювати його після відмов або змін у конфігураціях. Також DevOps налаштовує моніторинг та логування, використовуючи інструменти на зразок Prometheus, Grafana або ELK Stack (Elasticsearch, Logstash, Kibana) для забезпечення видимості роботи додатку в реальному часі.

3. Створення і управління контейнерами - для розгортання Java додатків застосовують контейнеризацію за допомогою Docker. DevOps спеціаліст створює Docker образи додатків, визначаючи в Dockerfile всі залежності та налаштування для їх виконання. Це дозволяє забезпечити портативність додатку між різними середовищами, а також спрощує його масштабування, оскільки кожен контейнер ізольований і має чітко визначені ресурси. Інструменти для оркестрації контейнерів, такі як Kubernetes або Docker Swarm, застосовуються в автоматизації розгортання, масштабування та управління контейнерами на кластеризованих серверах.

4. Налаштування безперервної інтеграції та доставки (CI/CD) – використовуючи інструменти Jenkins, GitLab CI або CircleCI, DevOps забезпечує автоматизовану інтеграцію та доставку нових версій додатку до тестового і продакшн середовища [18]. Процес CI/CD включає автоматичне тестування, збірку та деплоймент коду, що дозволяє значно скоротити час, необхідний для внесення змін до додатку, і знизити ризик виникнення помилок при ручному розгортанні. Це також дозволяє гарантувати, що зміни в коді не порушують стабільність роботи додатку, оскільки всі зміни тестуються в автоматичному режимі перед розгортанням у продакшн середовищі.

5. Налаштування бази даних і збереження стану додатку - DevOps спеціаліст повинен забезпечити правильне налаштування баз даних для продуктивного середовища, враховуючи вимоги до доступності, швидкості і безпеки даних. Для цього можуть використовуватись як традиційні реляційні бази даних (наприклад, PostgreSQL або MySQL), так і NoSQL бази (наприклад, MongoDB або Cassandra), залежно від архітектури додатку. Для масштабування бази даних часто використовуються технології реплікації та кластеризації. Крім того, DevOps конфігурує системи резервного копіювання і відновлення, що є критично важливим для забезпечення безпеки даних.

6. Моніторинг та підтримка роботи додатку на продакшн середовищі - після того як Java Spring додаток розгорнуто, DevOps спеціаліст займається налаштуванням постійного моніторингу за допомогою таких інструментів, як Prometheus, Datadog або New Relic, для виявлення проблем з продуктивністю, а також для оцінки навантаження на систему та виявлення потенційних точок відмов. Крім того, налаштовуються системи сповіщень, щоб своєчасно реагувати на критичні ситуації, як-от збільшення часу відповіді сервісу або збої у роботі. Для забезпечення безпеки використовуються інструменти для аналізу уразливостей та моніторингу загроз, що дозволяє підтримувати систему в актуальному та безпечному стані.

2.4 Аналіз та обробка великих обсягів даних

2.4.1 Моделі зберігання та доступу до даних

Моделі зберігання та доступу до даних ключовий елемент створення програмного забезпечення, оскільки в них визначається організація, структурування та маніпуляції даними в інформаційних системах і також задається спосіб зберігання даних (табличний, документний, графи або інша структура) і доступність для зовнішніх операцій (запис, читання, модифікація)

Вибір підходящих для проєкту моделей напряму впливає на продуктивність, масштабованість і зручність роботи з даними. Для об’єднання даних за певною структурою - стандартом виступають бази даних, а для зберігання розрізнених даних

- файлові моделі та сховища даних. Також високу актуальність мають засоби аналізу та комплексної обробки даних.

Реляційні бази даних (RDBMS) базуються на концепції зберігання даних у вигляді таблиць (реляцій), в яких рядок представляє запис, а стовпець властивість запису. В реляційних базах даних застосовується мова SQL (Structured Query Language) для проведення маніпуляцій над даними та їхньою структурою.

Головна особливість виділяюча реляційні бази даних з поміж інших є чітко визначена схема із структурою таблиць та зв'язків між ними. Це в свою чергу забезпечує цілісність даних, підтримку складних запитів і транзацій, що особливо цінується в банківській та фінансовій сфері. Основними представниками є MySQL, PostgreSQL, Oracle і Microsoft SQL Server [19].

Нереляційні бази даних (NoSQL) орієнтовані на гнучкість і масштабованість, для ефективного забезпечення обробки великих обсягів даних із розрізненною структурою. Відсутність строгої схеми, зберігання даних у вигляді документів (JSON, XML) графів, ключ-значень або колонок (в залежності від типу бази даних) робить їх зручними в якості сховища великих масивів даних з непередбачуваною структурою або для швидкого доступу до даних у реальному часі [20]. Прикладом NoSQL баз даних є MongoDB, Cassandra, Redis і DynamoDB.

Також важливим аспектом є реплікація та шардінг баз даних. Вузьким місцем великих систем можуть ставати бази даних, якщо зростає обсяг даних або кількість запитів, тоді реплікація і шардінг стають головними методами для забезпечення високої доступності та продуктивності.

Реплікація полягає в копіюванні даних з однієї бази даних на інші сервери для підвищення надійності і продуктивності. MySQL підтримує кілька моделей реплікації, найпоширенішою з яких є майстер-слейв реплікація. У даному підході один сервер бази даних (майстер) виконує всі операції запису, поки інші сервери (слейви) зберігають копію даних і обробляють операції читання. Такий підхід дозволяє розвантажити майстер-сервер і прискорити роботу додатка за рахунок розподілу операцій читання на слейви.

Основними недоліками реплакації являється:

- Затримка синхронізації: Існує затримка між моментом запису даних на майстері і моментом їх відображення на слейвах, що може призвести до несинхронізованості даних.
- Складність налаштування: Керування реплікацією та забезпечення її коректної роботи вимагає додаткових ресурсів і складних налаштувань.

Шардінг (sharding) — це процес розбиття бази даних на кілька частин (шардів) які зберігаються на окремому сервері або кластері серверів. На кожному із шарів міститься тільки частина даних (обрана й розміщена за певним критерієм, наприклад діапазоном даних ключа)

У MySQL шардінг може бути реалізований через розподіл таблиць або баз даних між кількома серверами, що дозволяє паралельно обробляти запити і значно збільшити продуктивність системи шляхом розподілу навантаження [19]. Це забезпечує горизонтальне масштабування і готовність системи загалом до великого обсягу даних.

Основними недоліками шардінгу є:

- Складність реалізації: Налаштування шардінгу вимагає складної логіки для розподілу даних та керування ними. Також важливо забезпечити балансування навантаження між шарами.
- Складність об'єднаних запитів: Запити, що охоплюють кілька шардів, можуть стати складнішими та менш ефективними, оскільки кожен шар потрібно обробляти окремо.

Поєднання реплікації та шардінгу дозволяє забезпечити високу доступність у базах даних. Реплікація надає можливість зберігати кілька копій даних, що робить систему більш надійною і дозволяє швидко відновитися після збоїв. Шардінг відповідно дозволяє працювати з великими обсягами даних, розподіляючи їх між кількома серверами, що забезпечує масштабованість [19].

2.4.2 Файлова модель зберігання данних

Окремим випадком варто розглянути контекст систем управління зберіганням та файлові системи, в котрих застосовуються моделі для фізичного або логічного

зберігання великих масивів даних, без обов’язкової інтеграції функцій характерних для баз даних (запитів, транзакцій, зв’язків).

Файлова модель - у даній моделі дані зберігаються під виглядом файлів, які організовані в ієрархічній файловій системі. Така модель є основою для операційних систем, в котрих файли можуть містити всі типи даних - текст, зображення, відео тощо. Розглянута модель ефективна для зберігання неструктурованих даних, однак відповідно мало придатна для виконання складних запитів або обробки великих масивів даних.

Об’єктна модель - застосовується для зберігання даних у вигляді об'єктів, в якій кожен об'єкт містить як дані, так і метадані, тобто дозволяючи зберігати складні структури даних разом із їхнім контекстом. Активно використовується в об’єктних сховищах (Amazon S3, OpenStack Swift). Дана модель ефективна для масштабованих хмарних систем, які обслуговують великі масиви мультимедійних файлів.

Блокова модель. – слідує концепції розбиття даних на блоки однакового розміру, коженому з яких присвоюється унікальний ідентифікатор. Дана модель широко поширена в хмарних сховищах та системах зберігання даних для швидкого доступу до великих обсягів інформації. Розглядаючи контекст файлових систем блокова є базовою для організації зберігання файлів де файли розподіляються по блоках. Прикладом даної моделі є Hadoop HDFS (Hadoop Distributed File System)

2.4.3 Алгоритми прямого пошуку даних

Базовим способом отримання даних є запити до бази даних. Однак прості запити без використання спеціалізованої під пошук логіки є не продуктивним рішенням. Для прикладу, в реляційних базах даних оператор Like у використанні в якості доступу до текстових даних може використовуватись тільки при незначних навантаженнях.

Від досягає складності пошуку O(log(n)) тільки у розі пошуку в проіндексованій таблиці і вона не є гарантована. В основних випадках складність пошуку становитиме O(n), тобто відбуватиметься повний перебір всіх значень таблиці.

Для вирішення даної проблеми були розроблені засоби прямого пошуку. MySQL та PostgreSQL містять вбудовані засоби прямого пошуку.

Прямий пошук не може відбуватись без індексації шукомих полів. Чим краще проіндексовані поля тим швидше буде знайдений результат, призначенням механізму є досягнення швидкості O(log(n)) в більшості випадків. В програмування є різні алгоритми найкоротшого знаходження даних.

В MySQL єдиним варіантом є створення індексу з атрибутом FULLTEXT. База даних сама обере алгоритм індексації. Поданий підхід є швидким, однак менш гнучким ніж в PostgreSQL

PostgreSQL пропонує такі типи індексування -

- Creates a GIN (Generalized Inverted Index)-based index. The column must be of tsvector type
- Creates a GiST (Generalized Search Tree)-based index. The column can be of tsvector or tsquery type. Optional integer parameter siglen determines signature length in bytes (see below for details).

GIN індекси містять оптимізований індексний запис (tsvector) для кожного слова (лексеми) зі стислим списком відповідних місць де його шукати.

GiST це бінарне дерево по свої структурі. tsquery містить звичайний повнотекстовий шукомий запит. Індекс GiST є індексом з втратами, тобто індекс може давати хибні збіги і для усунення таких хибних збігів необхідно перевіряти реальний рядок таблиці.

Обидві БД використовують простий запит для прямого пошуку по проіндексованим полям. Приклад типів пошуку які підтримує MySQL для запиту –

- natural language search - пошук за повною відповідністю в слові
- query expansion search - другий пошук буде включати найбільш релевантні варіанти з першого. Не рекомендується використовувати для великих обсягів інформації, оскільки в результат може потрапити дуже багато зайвого.
- boolean search type - пошук з параметрами, дозволяє виконати пошук за частковим словом, тегами, та фільтрами (“demo*” would also match “demonstration”)

Бінарний режим має наступні характеристики –

- У бінарному режимі, на відміну від інших режимів, релевантність обчислюється іншим способом - як умовна міра збігу заданого шаблону. Положення шуканого шаблону в тексті, кількість варіантів, що зустрілися, ролі не відіграють.
- У бінарному режимі дані є невпорядкованими, однак функцію впорядкування в БД бере на себе команда ORDER BY що дозволяє конфігурувати її напряму в сервісі самим користувачем. Підтримується конфігурування мінімальної та максимальної довжини шукомого слова, таблиці стоп-слів (слова які не враховуються в пошуку)
- Бінарний режим в MyISAM може використовуватись без створення індексів, однак тоді він буде надзвичайно повільним.

Бінарний режим підтримує наступні оператори –

- За замовчуванням (коли не вказано ні +, ні -), це слово є необов'язковим, але рядки, що містять його, отримують вищу оцінку. Це імітує поведінку MATCH() AGAINST() без модифікатора IN BOOLEAN MODE.
- Оператор “+” - Передній або задній знак плюс вказує на те, що це слово має бути присутнім у кожному рядку, який повертається. InnoDB підтримує лише початкові знаки плюс.
- Оператор “-” - Передній або задній знак мінус вказує на те, що це слово не повинно бути присутнім у жодному з рядків, які повертаються. InnoDB підтримує тільки передні знаки мінус.
- Оператор “ @distance ” - Даний оператор працює лише з таблицями InnoDB. Він перевіряє, чи два або більше слів починаються на вказаній відстані одне від одного, що вимірюється в словах. Вказуйте шукані слова в рядку в подвійних лапках безпосередньо перед оператором @distance, Приклад - MATCH(col1) AGAINST('"word1 word2 word3" @8' IN BOOLEAN MODE)
- Оператори “> <” - Дані два оператори використовуються для зміни внеску слова у значення релевантності, яке присвоюється рядку. Оператор > збільшує внесок, а оператор < зменшує його.

- Оператори “( )” - Дужки групують слова у під-вирази. Групи в круглих дужках можуть бути вкладеними.
- Оператор “~” - Початкова тильда діє як оператор заперечення, роблячи внесок слова в релевантність рядка від'ємним. Це корисно для позначення "шумових" слів. Рядок, що містить таке слово, оцінюється нижче, ніж інші, але не виключається повністю, як у випадку з оператором мінус
- Оператор “*” - Слугує оператором усічення (або підстановки). На відміну від інших операторів, вона додається до слова, яке буде змінено. Слова збігаються, якщо вони починаються зі слова, що стоїть перед оператором *. Наприклад, «інс* наф* і газ*» буде сприйматись як «інститут нафти і газу»
- Оператор “"” - Фраза, взята в подвійні лапки ("), відповідає лише тим рядкам, які містять цю фразу дослівно, тобто так, як вона була введена. Повнотекстовий механізм розбиває фразу на слова і виконує пошук у повнотекстовому індексі за цими словами. Символи, що не входять до складу слів, не обов'язково повинні збігатися з ними точно: Пошук за фразою вимагає лише того, щоб результати пошуку містили ті самі слова, що й фраза, і в тому самому порядку. Наприклад, "тестова фраза" відповідає "тест, фраза".

Якщо фраза не містить слів, які є в індексі, результат буде порожнім. Слова можуть бути відсутніми в індексі через комбінацію факторів: якщо вони не існують у тексті, є стоп-словами або коротші за мінімальну довжину проіндексованих слів.

Як висновок використання вбудованих пошукових засобів потребує поглибленого розуміння того, як саме організовані дані в БД і як ними маніполювати. Слід чітко знати котрі пошукові можливості потребуються в проєкті та які дані будуть задіяні. Використання засобів вбудованого прямого пошуку є значним покращенням порівняно з простими запитами та хранимими процедурами. По швидкості доступу до даних, вони вже нічим не відрізняються від корпоративних пошукових рушіїв, наприклад Google Search.

2.4.4 Пошукові та аналітичні системи.

Проблема швидкості доступу до даних успішно вирішується вбудованими засобами БД, однак з ростом кількості користувачів виникають і інші проблеми – це збір та обробка супутніх даних, які переслідують сам процес пошуку. Дана тема виходить за межі простої організації даних і в ній слід розглядати глобальні засоби маніпуляції інформацією.

Більшість пошукових API являють собою NoSQL бази даних. В них описуються процеси та алгоритми обробки розрізненої інформації.

Основні пошукові API та їх популярність представлена на рис.2.1. [21].

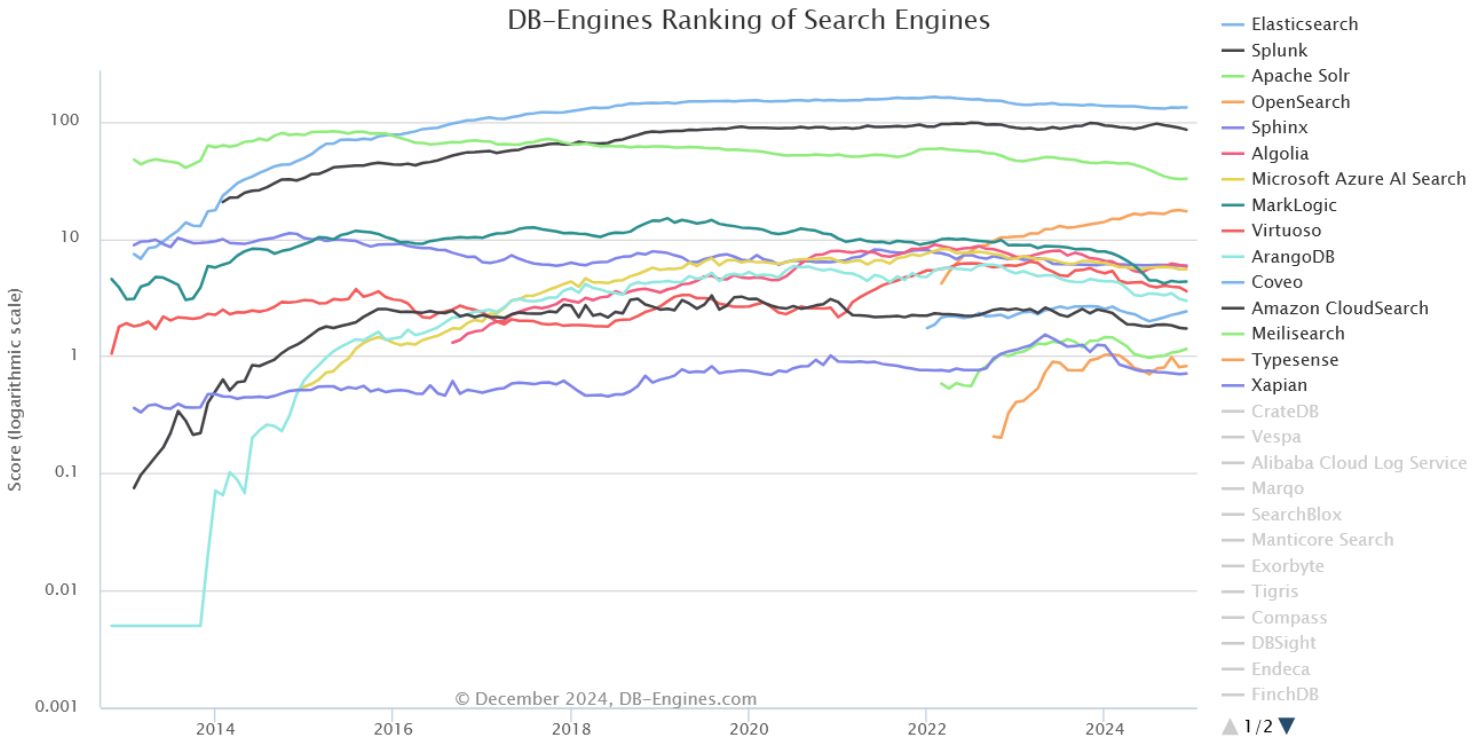

Рис. 2.1. Пошукові двигуни та їх популярність за роками

Apache Lucene - це високопродуктивна, повнофункціональна бібліотека пошукової системи, повністю написана на Java. Дана технологія може бути застосована практично для будь-якої програми, яка потребує структурованого пошуку, повнотекстового пошуку, фасетування, пошуку найближчих сусідів по векторах високої розмірності, виправлення орфографії або пропозицій щодо запитів.

Її форками (розширеннями) є найбільш відомі Elasticsearch та Apache Solr пошукові API

Elasticsearch - Це дистрибутивна open-source пошукова і аналітична система. Вона може використовуватись для будь-яких типів даних. Написана на Java, вона підтримує усі операційні системи, що мають віртуальні java машини. Ця система масштабована і швидка, що робить її простим і оперативним інструментом для аналітики даних, їх обробки та візуалізації.

Досі залишається найпопулярнішою в світі, однак почала втрачати позиції після комерцізації та переходу на платну ліцензію. Такий крок став сильним підривом репутації як open-source API.

Перевага пошукової системи Elasticsearch полягає у індексування документів за певними правилами. Індексування у цьому контексті означає, що значення полів зводяться до документів або просто рядків для більш швидкого пошуку. Замість того, щоб здійснювати пошук по всьому документу за заданим значенням, система може знайти це значення в її внутрішньому індексі і миттєво знати, які документи чи рядки містять це значення [22].

Apache Solr - Платформа повнотекстового пошуку з відкритим вихідним кодом, заснована на проєкті Apache Lucene. Її основні можливості: повнотекстовий пошук, підсвічування результатів, фасетний пошук, динамічна кластеризація, інтеграція з базами даних, обробка документів зі складним форматом (наприклад, Word, PDF). Оскільки в Solr є можливість розподіленого пошуку і реплікації, Solr добре масштабується.

Відмінності Solr від Elasticsearch:

Solr

- (перевага) OpenSource, ліцензія вільного доступу та використання
- (недолік) Відсутність автоматичного масштабування
- (недолік) DevOps. Потребує додаткових програм, для кращого моніторингу. Потребує сторонніх тулзів для встановлення
- Analythics Engine більш продуктивний
- (перевага) Має графічний інтерфейс

Elasticsearch

- (недолік) Не OpenSource. в 2021 перевели на подвійну ліцензію. Основна ліцензія накладає обмеження на використання. Умови можуть змінюватись в будь який момент часу, доходячи до погодинної оплати за використання тощо. Всі додаткові сервіси є платними.
- (перевага) Зручне автоматичне масштабування
- (перевага) DevOps. Має хороші API для моніторингу «З коробки».
- (перевага) Встановлюється одною командою
- Analythics Engine менш продуктивний, але більш гнучкий
- (перевага) Ecosystem. ES є більш поширеним та має більшу підтримку

У всьому іншому дані API є майже ідентичними і при відповідних навичках можна реалізовувати тіж самі задачі не залежно від вибору.

Застосування будь яких інших пошукових API потребує чіткого розуміння проблем які вони можуть вирішити. Як правило їх використовують в дуже вузьких галузях або розглядають як альтернативу для переходу вже існуючих проєктів. На рис 2.1 можна побачити багато нереляційних баз даних, їх також застосовують для досягнення певної впорядкованості даних і швидкого доступу до них.

Splunk – заснований на архітектурі реального часу. Splunk здійснює збір, пошук, моніторинг та аналіз за різними і досить великими (сотні Тб даних на день) обсягами даних у режимі реального часу в межах однієї системи.

Дозволяє автоматично архівувати дані. Может зменшити обсях даних в десяток разів порівняно із іншими рішеннями.

Налаштування Splunk простіше і вимагає менше технічних знань, тоді як Elasticsearch потребує більш глибокого розуміння розподілених систем.

Користувацькі інтерфейси Splunk і Elastic Stack створені та служать різним цілям і надають різну функціональність. Splunk зосереджений на аналітиці на основі пошуку, тоді як Elasticsearch орієнтований на виявлення даних і дослідження.

В цілому по вартості Elastic Stack буде набагато дешевше Splunk. З об'ємом обробки даних близько 1.5-2Гб на день Splunk коштуватиме додаткові 3-4 тисячі $ на рік відповідно до їхніх тарифних планів

MarkLogic - Operational and transactional Enterprise NoSQL database. Документоорієнтована база даних. Одна з провідних NoSQL баз даних. Найкраще XML та JSON сховище даних в світі.

Використовує розподілену архітектуру, яка може обробляти сотні мільярдів документів і сотні терабайт даних.

Sphinx - Раніше перспективний, а на даний момент мертвий пошуковий API. Його розробка була зупинена в 2016 році.

З переваг - наявність лематизаторів ua, en, du, велика кількість стеммерів індексації та пошуку: full-text, фасетний, geo. Немає нічого зайвого, лише пошуковий рушій зі швидкою індексацією і своїм індексером. За правильної конфігурації RT індексів можна досягти (real time indexing). На відміну від Elasticsearch спостерігається помірне використання пам'яті. Фундаментально він є швидчим за Elasticsearch

Якщо розглядати недоліки, то головний з них - необхідність самостійно прораховувати всю структуру індексів, а отже, масштабування проходить не так просто. Також у системи мізерний API, немає можливості для візуалізації, відсутній fuzzy-пошук за замовчуванням (але можна реалізувати власний) і майже відсутній колектив користувачів, тобто складно отримати поради в мережі інтернет. Щоразу викликає БД, тоді як інші системи можуть самостійно зберігати дані (NoSQL Data Storage).

Manticore Search - форк Sphinx. Open-source проєкт, який на даний момент розвивається погодинно. Роки застою в розвитку Sphinx породили багато ентузіастів, які самоорганізувавшись створили власний форк і тепер активно привносять нововведення в індустрію [23].

Ком'юніті активно працює над вирішенням всіх проблем Sphinx. На відміну від конкурентів кожне твердження закріплене проведеним тестуванням.

Основні переваги які прописані на сторінці GitHub із наведеними тестами -

Висока швидкість і тому вища економічна ефективність, порівняно з альтернативами [24]:

- 182x швидший ніж MySQL для малих даних

- 29x швидший ніж Elasticsearch для логування аналітики
- 15x швидший нід Elasticsearch для малих датасетів
- 5x швидший нід Elasticsearch для даних середнього обсягу
- 4x швидший ніж Elasticsearch для великих даних (big data)
- до 2x разів вища максимальна пропускна здатність для завантаження даних у порівнянні з Elasticsearch на одному сервері.

Завдяки сучасній багатопотоковій архітектурі та ефективному розпаралелюванню запитів, Manticore здатна повністю задіяти всі ядра процесора для досягнення найшвидшого часу відгуку.

Manticore є SQL-орієнтованою, використовуючи SQL як власний синтаксис, і пропонує сумісність з протоколом MySQL, що дозволяє використовувати звичний в застосуванні клієнт MySQL.

Побудований на C++, Manticore Search швидко запускається і використовує мінімум оперативної пам'яті, а низькорівневі оптимізації сприяють його вражаючій продуктивності.

Також розробники планують добавити підтримку інших open-source сервісів, які і принесли славу Elasticksearch, і через які його часто називають скороченням ELK - Elasticsearch, Logstash, Kibana. В перспективі ввібравши в себе все найкраще з Elasticsearch даний проєкт може стати хорошую альтернативою і найголовніше безкоштовною та open-source.

Недоліки всіх пошукових API

- Якщо для прикладу є база даних розміром 500гб, то при підключенні API прийдеться загрузити в неї всі дані і фактично збільшити використання пам'яті до 1тб. Відповідно нагрузка на сервер також зрозте, оскільки ці всі дані необхідно обслуговувати.
- В разі виникнення збою в ланцюгу «База даних <-> пошуковий API» буде порушена консистентність даних. (пркл. дані видалені, але користувач їх бачить). Не кожний пошуковий API гарантує дотримання ACID.

- Сторонній API потребує підтримки та обслуговування DevOps спеціалістом. Це складний окремий напрямок в програмуванні, який значно збільшить вартість підтримки проєкту

В підсумку із наведених вище порівнянь можна підкреслити високу вартість переходу до використання пошукових API та складність у підтримці, однак безпосередньо варто відзначити що основний прибуток великим корпораціям наразі приносить аналітика та успішний маркетинг. Досягнути схожих результатів без використання спеціалізованих засобів майже неможливо.

2.5 Висновки до розділу

В даному розділі виконано огляд проблеми вибору технологій, налаштування та розгортання веб-сервісів, які в свою чергу є необхідними для функціонування електронної комерції. Розглянуто Java Spring фреймворк в якості основної платформи для побудови цільового рішення, наведено характеристику найбільш популярних елементів його екосистеми.

Також наведена характеристика DevOps розробника, як ключового спеціаліста для забезпечення автоматизації технологічних процесів складання, налаштування та розгортання програмного забезпечення. Досліджено алгоритм автоматизації розгортання серверної частини Java Spring додатку.

Окремо приділена увага аналізу та обробці великих обсягів даних. У зв’язку з тим, що в сучасних умовах різні типи даних часто пов’язані між собою, було розглянуто ефективні моделі зберігання та доступу до них. Досліджено алгоритм прямого пошуку даних, завдяки якому можливо зменшити складність обчислювальних процесів від лінійної O(n) до логарифмічної O(log n). Також проаналізовано та порівняно пошукові та аналітичні системи, які використовують для обробки великих даних.

РОЗДІЛ 3

РОЗРОБКА ПРОГРАМНОЇ МОДЕЛІ ТА АЛГОРИТМІВ ДЛЯ ЕФЕКТИВНИХ РІШЕНЬ В ЕЛЕКТРОННІЙ КОМЕРЦІЇ

3.1 Формування вимог та архітектури програмного рішення

3.1.1 Основні вимоги до програмного рішення

В якості зразка переходу до електронної комерції обрано готельний бізнес. Для забезпечення ефективного управління всіма аспектами операційної діяльності готелю, розроблюване рішення має включати в себе низку наступніх вимог:

- систему бронювання
- систему управління номерами (житлом)
- обробку платежів
- організацію взаємодії з клієнтами через онлайн-канали

Врахувати всі можливі вимоги одразу неможливо, з цієї причини програмний продукт мусить бути стандартизованим і масштабованим. Сучасною практикою є створення базового функціонального каркасу і його подальша адаптація до зростання бізнесу, при уже введеній в експлуатацію системі, яка приносить власнику прибуток. Тобто обрані технології повинні бути досить гнучкими для подальшої інтеграції з існуючими системами готелю, такими як бухгалтерія, облік персоналу, платіжні системи.

На додачу до функціональності, розроблюване програмне забезпечення повинне мати зручний інтерфейс для персоналу і клієнтів, гарантувати високу швидкість і точність обробки запитів, а також надавати доступ до даних у реальному часі для швидкого реагування на зміни та запити клієнтів. Для забезпечення безпеки даних, програмне рішення має відповідати стандартам захисту персональних даних і реалізовувати надійну систему автентифікації, для запобігання несанкціонованому доступу. Очікується що цільова система буде стабільною, здатною витримувати високі навантаження, особливо в пікові періоди, а також інтегруватися з мобільними

додатками та веб-платформами для зручності онлайн-бронювання і взаємодії з клієнтами.

3.1.2 Архітектура рішення запропонованої програмної системи

Враховуючи подані вимоги, було обрано окремий випадок клієнт-серверної архітектури - RESTful Web Service, в якій в якості серверної частини застосовується Java Spring фреймворк, а клієнтської веб-браузерні представлення (html, css, js). Вони наведені на рис.3.1. В якості бази даних обрано реляційну MySQL.

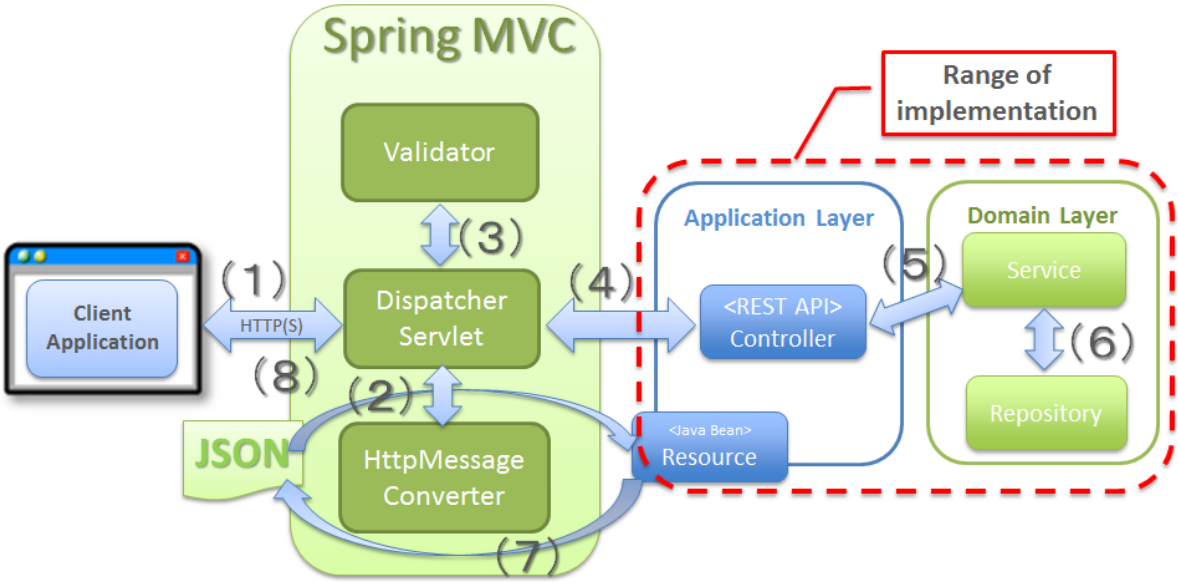

Рис. 3.1. Архітектура RESTful Web Service

Java Spring Boot є одним з найпродуктивніших рішень на ринку інноваційного програмного забезпечення. Завдяки вбудованій підтримці високої паралельності та асинхронної обробки запитів, він забезпечує ефективне розподілення наявних машинних ресурсів та надає необмежені можливості для масштабування, на відміну від Wordpress або PHP, які досить швидко досягають ліміту продуктивності, хоча і є більш дешевшими рішеннями.

У зв’язку з відсутністю необхідності створення значної кількості сервісів, застосовується монолітна архітектура, що в свою чергу позбавляє необхідності вирішення проблем узгодженості і цілістності даних, і значною мірою зменшує

вартість продукту та час розробки. При успішній інтеграції в бізнес, зростанню фінансування і наявності потреби у введенні багатьох нових сервісів, проєкт потрібно перенести на мікросервісну архітектуру.

Враховуючи описану специфіку Java Spring фреймворку, під час розробки застосовуються наступні патерни проєктування –

Патерн MVC (Model-View-Controller, Модель–представлення–контролер) є архітектурним шаблоном, для забезпечення структурованої організації програмного забезпечення через поділ на три основні компоненти: Модель, Представлення та Контролер. Модель (Model) відповідає за управління даними, логікою та правилами сервісів. На модельному рівні зберігається стан програми і забезпечується доступ до даних, їх обробка та оновлення. Представлення (View) відповідають за відображення інформації на стороні клаєнта (браузери, android, ios системи), вони забезпечують інтерфейс для взаємодії із сервісами. Контролер (Controller) забезпечує взаємодію між представленнями та моделью. На поданому рівні відбується обробка запитів зі сторони користувача або іншого сервера, формуються набори даних та передаються у відповідь [25].

Розглянутий патерн дозволяє досягати високої модульності та спрощує підтримку і масштабування системи. Поділ відповідальності між компонентами сприяє ізоляції змін: зміни в логіці бізнесу впливають тільки на модель, зміни в інтерфейсі тільки на представлення, контролер залишається відносно стабільним. Такий підхід також підтримує можливість багаторазового використання компонентів, оскільки представлення та моделі можна застосовувати незалежно в різних частинах програми або експортувати в іншу програму. MVC є основою для багатьох сучасних веб-фреймворків.

Патерн DAO (Data Access Object) - структурний шаблон програмування для забезпечення абстагування доступу до бази даних. Досягається це в патерні шляхом розділення логіки бізнеси і логіки доступу до даних, тобто створення проміжного рівня між програмою і джерелом даних. DAO реалізовують шляхом створення об'єкту або класу, який відповідає за за виконання CRUD-операцій (створення, читання, оновлення, видалення) з базою даних або іншим сховищем даних, тобто інкапсуляції

комплексних операцій з базою даних, надаючи зрозумілий інтерфейс для взаємодії [26].

Застосування даного патерну забезпечує незалежність коду бізнес-логіки від конкретного механізму зберігання даних, що спрощує підтримку і масштабування системи. Також розглянутий підхід підтримує принципи розробки, такі як розділення обов'язків (Separation of Concerns) і залежність від інтерфейсів (Dependency Inversion Principle), в наслідок чого досягається підвищення гнучкості і тестованості коду.

Наступній патерн Repository (Репозиторій) в свою чергу розширює концепцію DAO, привносячи більш високий рівень абстракції для управління даними. Repository являє собою коллекцію об'єктів домену і виконує роль адаптера між бізнес-логікою і шаром доступу до даних. На відміну від DAO, який зосереджується на виконанні CRUD-операцій для конкретних таблиць або сутностей бази даних, Repository працює на рівні доменних об'єктів забезпечуючи операції які відповідають логіці бізнесу. Даний патерн дозволяє приховати деталі взаємодії з базою даних і підтримувати чисту архітектуру, де бізнес-логіка не залежить від конкретного способу збереження даних. Repository дозволяє інтегрувати множину джерел даних (бази даних, API, файлові системи) об'єднуючи їх у єдиному інтерфейсі. У порівнянні з DAO, Repository є більш орієнтованим на концепцію доменно-орієнтованого проєктування (DDD), дозволяючи працювати з доменними моделями, зберігаючи їхню цілісність і узгодженість. Розглянутий патерн часто використовується із ORM-фреймворками для зручної роботи із об’єктами [27].

3.2 Розробка алгоритмів роботи системи

3.2.1 Алгоритм аутентифікації користувача в системі

Шляхом інтеграції модулю Spring Security забезпечується добавлення в проєкт готових реалізацій протоколів та механізмів забезпечення безпеки. Базовий алгоритм функціонування процесу автентифікації користувача виглядає наступнім чином на рис.3.2. [28] –

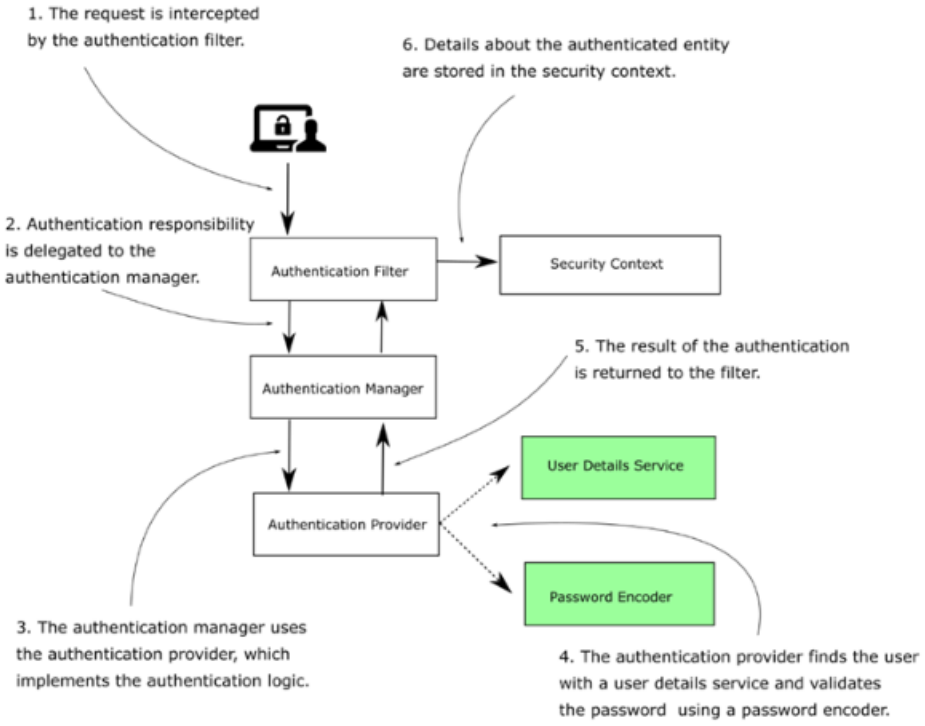

Рис. 3.2. Процесс автентифікації користувача

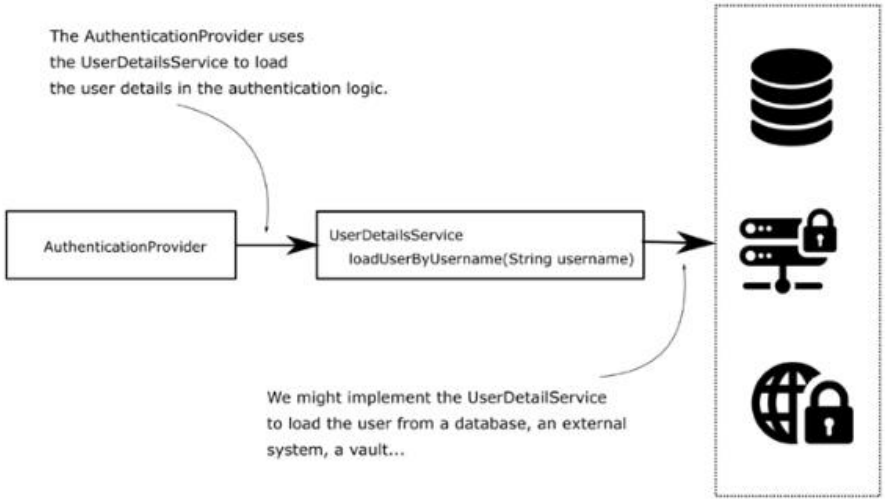

Рис. 3.3. Використання провайдером сервісу UserDetailsService для отримання інформації про користувача (із зовнішнього джерела даних)

1. Authentication Filter - Запит до сервера перехоплюється фільтром автентифікації

2. AuthenticationManager (Менеджер автентифікації) - отримує запит з фільтру і делегує конкретному AuthenticationProvider. Може маніпулювати кількома

провайдерами (по аналогії із дверьми, які мають два замка – один під ключ, інший під картку). Менеджер автентифікації застосовує відповідні провайдери які імплементують логіку автентифікації користувача

3. AuthenticationProvider (провайдер автентифікації) імлементує логіку автентифікації, тобто приймає рішення автентифікувати користувача чи ні (для прикладу за паролем, смс, відпитком пальця, тощо).

4. Провайдер знаходить користувача застосовуючи відповідний сервіс (імплементацію контракту UserDetailsService) представлену на рис.3.3. Також він валідує конфіденційні дані застосовуючи крифтографічну реалізацію контракту PasswordEncoder

5. Результат валідації повертається в фільтр

6. Деталі отримані про користувача, його сутність (Entity) зберігаються в загальному контексті захисту (security context) – який в подальшому доступний у всій системі для проведення операцій з користувачем.

Вказані рівні абстракції необхідні для забезпечення модульності та гнучкості в підтримці актуальності безпеки даних. Криптографічні алгоритми стрімко застарівають й потребують постійного оновлення. Квантові комп’ютери мають потенціал обійти сучасні криптографічні стандарти через використання алгоритмів, які ефективно розв'язують математичні задачі, що лежать в основі цих стандартів. Наприклад, алгоритм Шора дозволяє квантовому комп’ютеру розкладати великі числа на прості множники в експоненційно швидший спосіб, ніж класичні комп’ютери, що підриває безпеку RSA, DSA і схожих схем шифрування. Також завдяки алгоритму Гровера квантові комп'ютери можуть суттєво прискорити перебір ключів у симетричних криптосистемах. Поява нових загроз змушує криптографічну спільноту постійно вдосконалювати алгоритми, а архітектура Spring Security передбачає простоту в їх подальшій інтеграції, без суттєвого втручання в роботу основних сервісів.

Окрім основного механізму автентифікації, Spring Security містить велику кількість засобів для забезпечення додаткового захисту системи. Ось деякі із них:

- Захист від XSS/CSRF/CORS атак

- Конфігурування сесії користувача (Session)
- Формування та менеджмент JSON Web Token (JWT) (веб токенів).
- Конфігурування та розробка власних крифтографічних механізмів
- Конфігурування системи для забезпечення захисту від DDoS атак
- Двохфакторна аутентифікація (2FA)

Дані засоби детальніше описані в наступніх джерелах [28-29].

3.2.3 Алгоритм відкладених подій

В цільовій системі необхідно передбачити реалізацію наступного алгоритму дій –

1. Бронювання користувачем житла.
2. Виставлення рахунку користувачу.
3. Створення тимчасового бронювання на час очікування оплати.
4. Перевірка статусу оплати через проміжок часу продиктований політикою компанії.
5. Остаточне закріплення бронювання за користувачем або його скасування у зв’язку з неоплатою.

Для його реалізації необхідно застосувати планувальний подій (Scheduling). В Spring існує вбудований планувальник подій, який гарантує що задача не виконається більше ніж один раз в один і той самий проміжок часу. Він заснований на механізмах блокування потоків. Однак, його ключовими проблемами є масштабування і мікросервіси, а також відсутнє збереження стану при зупинці роботи сервера.

Альтернативою є фреймворк Quartz, заснований на механізмі збереження подій в базі даних, однак він є архаїчним та багатослівним.

Зберігаючи основний принцип роботи Quartz, було обрано планувальник завдань на стороні самої бази даних (Event Scheduler) [30].

3.3 Програмна реалізація e-commerce рішення для готельного бізнесу

3.3.1 Створення сутностей

Програмні сутності мають наступний вигляд (додаток А). Для цільової програми було створено наступній перелік сутностей –

- Account – аккаунти користувачів та адміністрації
- AccountRole – (enum тип) перелік доступних ролей для користувачів
- Apartment – номери, апартаменти та інші типи житла
- Booking – бронювання користувача
- Order – замовлення користувача
- ResponseToOrder – відповідь адміністрації на замовлення користувача, рекомендації користувачеві

3.3.2 Створення Model рівня додатку

Представлені в п. 3.3.1 сутності також входять до модельного рівня патерну MVC. Для розробки наступньої частини модельного рівня, був використаний патерн Repository. Відповідна його реалізація приведена в додатках (додаток В)

Наслідування інтерфейсів JpaRepository дозволяє отримати готову CRUD логіку, а наслідування інтерфейсу QuerydslPredicateExecutor розширити її, таким чином значно зменшуючи дублювання коду [32].

Останнім пунктом в формуванні Model рівня є Services (сервіси). Сервісний підрівень складається з набору Java-класів, які реалізують бізнес-логіку (отримання даних, оновлення, видалення і так далі) через один або кілька методів високого рівня. Іншими словами, сервісний рівень керує робочим процесом. Наприклад, коли користувач натискає кнопку "ОК" на сторінці, сервер повинен виконати задану серію кроків для виконання бізнес-функцій. Сервісний рівень контролює, як ці кроки виконуються.

Для цільового додатку було розроблені наступні інтерфейси сервісного рівня рис.3.4.

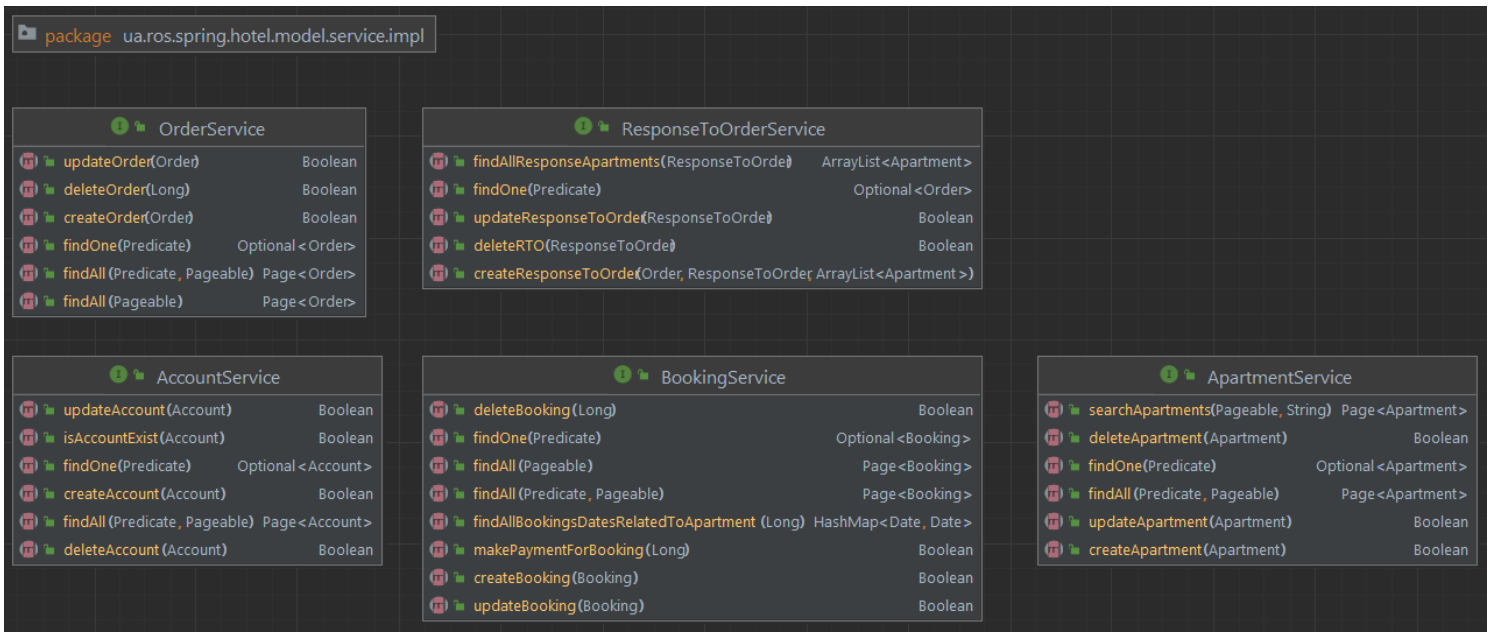

Рис. 3.4. UML Діаграма класів інтерфейсів сервісного рівня

Реалізація інтерфейсів сервісного рівня представлена на рис.3.5.

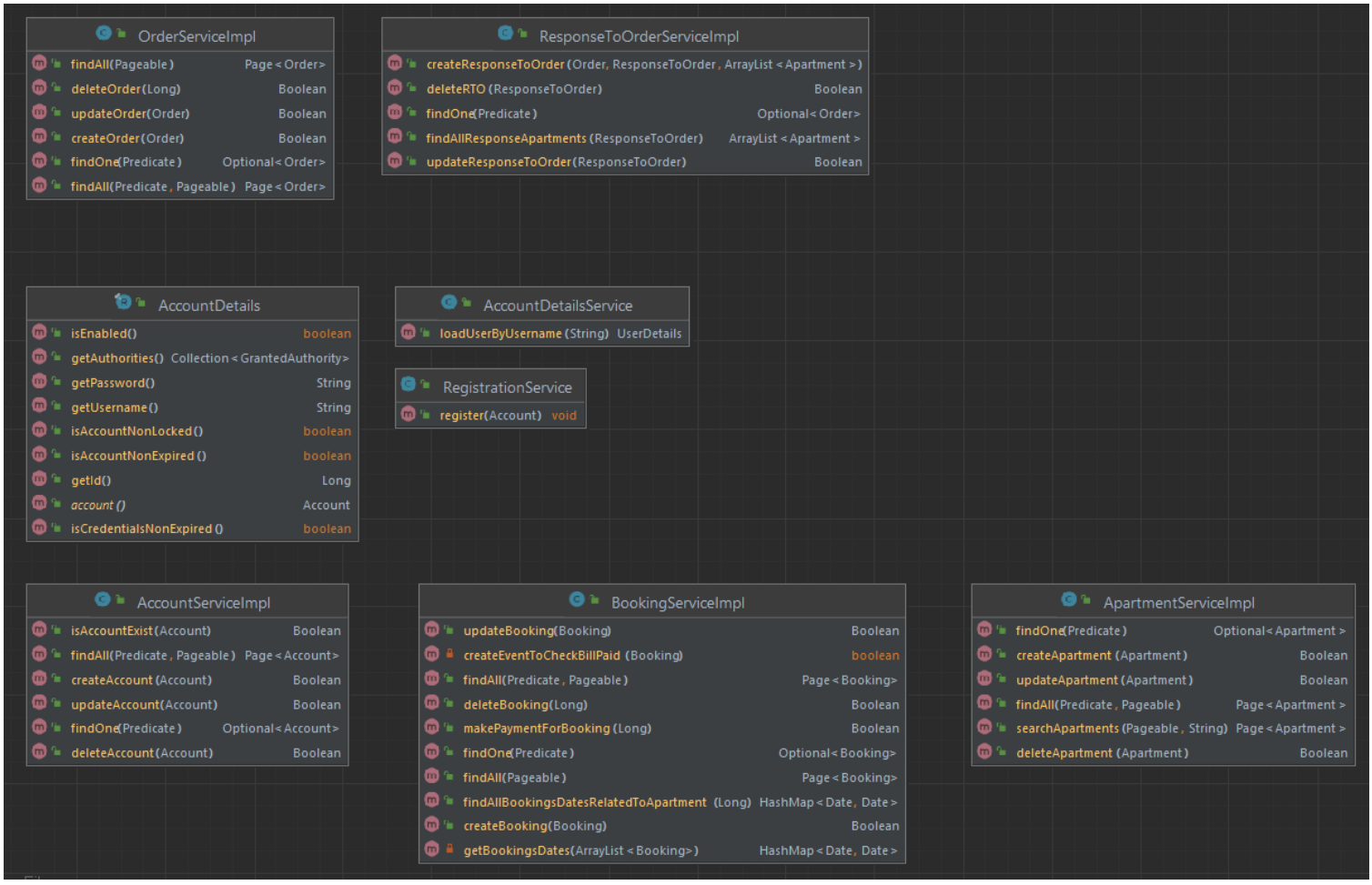

Рис. 3.5. UML Діаграма класів реалізації інтерфейсів сервісного рівня

Сервісний рівень є точкою входу до середнього рівня і відокремлює рівень представлення від рівня бази даних. Як правило, методи, що розкриваються класами сервісного рівня, приймають та/або повертають бізнес-об'єкти. Сервісний рівень інкапсулює бізнес-логіку, викликаючи бізнес-об'єкти та рівень доступу до даних, таким чином роблячи код більш зручним для обслуговування.

На сервісному рівні також розроблені та імплементовані наступні класи з інтерфейсами для забезпечення роботи алгоритму автентифікації?

- AccountDetails. - Імплементує інтерфейс UserDetails. Реалізує логіку взаємодії із сутністю Account
- AccountDetailsService - Імплементує інтерфейс UserDetailsService. Реалізує логіку отримання даних користувача із репозиторію AccountRepository
- RegistrationService – Реалізує логіку реєстрації нового користувача.

Тобто вони формують всю необхідну логіку для функціонування модулю Spring Security. Подальша конфігурація відбувається на рівні контролерів.

3.3.3 Реалізація алгоритму рекомендацій та прямого пошуку в базі даних

В лістингу 3.1. відображено метод інтерфейсу ApartmentRepository в якому через анотацію @Query описується SQL скрипт для прямого пошуку в MySQL базі даних. Засобами MySQL визначається релевантність відповідно до вказаних параметрів.

Лістинг 3.1.

```
@Query(value =
"SELECT SQL_CALC_FOUND_ROWS *, " +
"MATCH(`title`, `description`, `address`, `apartment_class`)
AGAINST(:search_value IN BOOLEAN MODE) AS REL " +

"FROM apartment " +

"WHERE MATCH(`title`, `description`, `address`, `apartment_class`)
AGAINST(:search_value IN BOOLEAN MODE) " +

"ORDER BY REL DESC ",
        countQuery = " SELECT FOUND_ROWS() AS count ",
        nativeQuery = true)
```

```
Page<Apartment>
      searchApartments(Pageable pageable, String search_value);
```

Результат повертається в пагінованому вигляді (реалізація інтерфейсу Pageable) та застосовується подальшому в сервісах програми на модельному рівні.

3.3.4 Реалізація алгоритму відкладених подій

Для реалізації поданого алгоритму в лістингу 3.2. створений відповідний метод, із аннотацією @Query і прописаним SQL кодом, мовою JPQL в інтерфейсі BookingRepository.

Лістинг 3.2

```
@Modifying
@Query(value=
"CREATE EVENT :event_name " +
"ON SCHEDULE AT CURRENT_TIMESTAMP + INTERVAL :days_time_interval  MINUTE
" +
"DO " +
"DELETE FROM " + DB_NAME_AND_BOOKING_TABLE_NAME +
"WHERE id = :id_value AND is_paid_for_reservation = 0 ",
                                                nativeQuery=true)
int createEventIsBillPaid(
                @Param("event_name") String eventName,
                @Param("days_time_interval") String daysTimeInterval,
                @Param("id_value")  Long id);
```

Однак специфікація JPA містить наступні критичні обмеження –

- єдиними операторами, дозволеними JPA, є SELECT, UPDATE та DELETE

Тобто застосування оператору CREATE EVENT є неможливе. Рішенням є застосування ORM засобів, поза специфікаєю і написання власного механізму реалізації логіки звернення до бази даних.

Для цього був створений відповідний метод в сервісі BookingServiceImpl на лістринг 3.3. який відправляє запит напряму в об’єкт персистентності (Persistence provider)

Лістинг 3.3.

```
private boolean createEventToCheckBillPaid(Booking booking) {
    log.info("Create event to check paid for booking");
    String eventName = " IsBill_"+ booking.getId()+ "_Paid ";

    String createEvent =
          "CREATE EVENT " + eventName +
          "ON SCHEDULE AT CURRENT_TIMESTAMP + INTERVAL
          :days_time_interval DAY " +
          "DO " +
          "DELETE FROM `" + databaseName + "`.`booking`" +
          "WHERE id = :id_value AND is_paid_for_reservation = 0 ";

     Query query = entityManager.createNativeQuery(createEvent)
            .setParameter("days_time_interval", timeToPayBill)
            .setParameter("id_value", booking.getId());

    query.executeUpdate();
    return true;
}
```

В результаті методом створюється подія в базі даних, яка через вказаний проміжок часу перевіряє, чи оплачено користувачем заброньоване житло. Дану подію створює сервіс бронювання, автоматично при бронюванні користувачем житла.

3.3.5 Створення Controller рівня додатку

Для взаємодії з фронт-ендом (користувачем) використовуються REST контролери які оперують даними і методами отриманими із Model рівня

Також на даному рівні описуються

- Мережеві фільтри, сервіси валідації,
- Конфігурація системи,
- Механізми маршрутизації (routing), для визначення порядку виклику контролерів у відповідь на запити,
- Централізована обробка винятків.

Процес парсингу в Json формат та обміну даними відображено на рис.3.6.

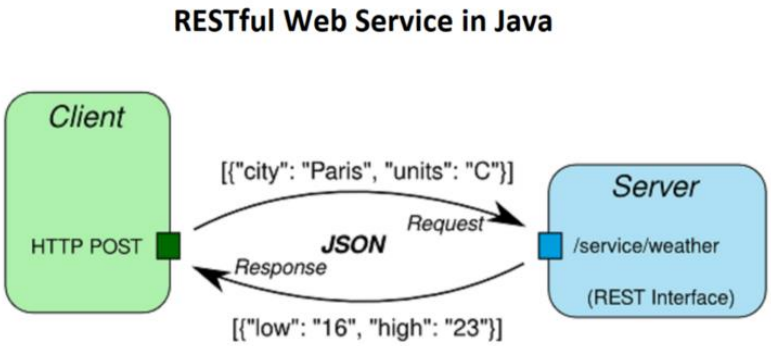

Рис. 3.6. Спрощена версія представлення взаємодії з клієнтом

Структура контролерів розроблених для цільового додатку представлена на рис.3.7.

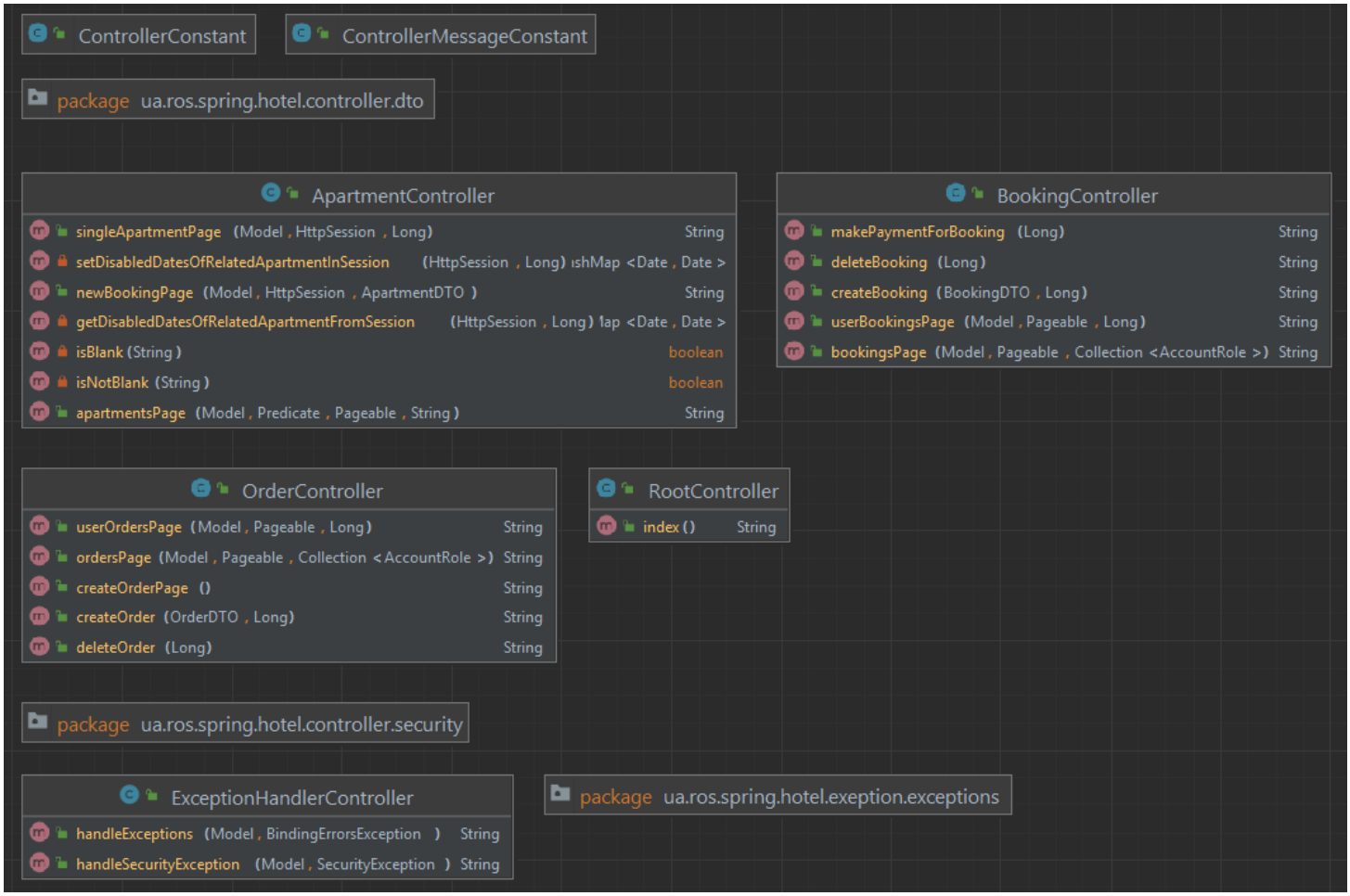

Рис. 3.7. UML Діаграма класів реалізації рівня Controller

Для забезпечення захисту додатку, шифрування паролів, генерацію криптографічних ключів тощо, додатково вводиться конфігурація Spring Security і реалізовуються необхідні для його функціонування механізми.

3.3.6 Створення View рівня додатку

View (представлення) додатку реалізоване засобами мов програмування та розмітки - HTML, CSS, JavaScript .

Для сучасного та динамічного відображення сторінки на стороні користувача, при її побудові використовується фреймворк Bootstrap [33].

Також механізм локалізації в Thymeleaf забезпечує динамічне відображення текстів та інших елементів інтерфейсу у відповідності до мовних або регіональних налаштувань користувача. В його основі закладені ресурсні файли (resource bundles) в форматі .properties. В даних файлах містяться ключі й відповідні переклади кожної підтримуваної мови. Thymeleaf використовує стандартну Java-інфраструктуру для локалізації, тому розширення доступних мов потребує лишень добавлення нового ресурсного файлу і перекладу його ключів [34].

Для цільового додатку були розроблені наступні сторінки та скрипти, що показані на рис.3.8.

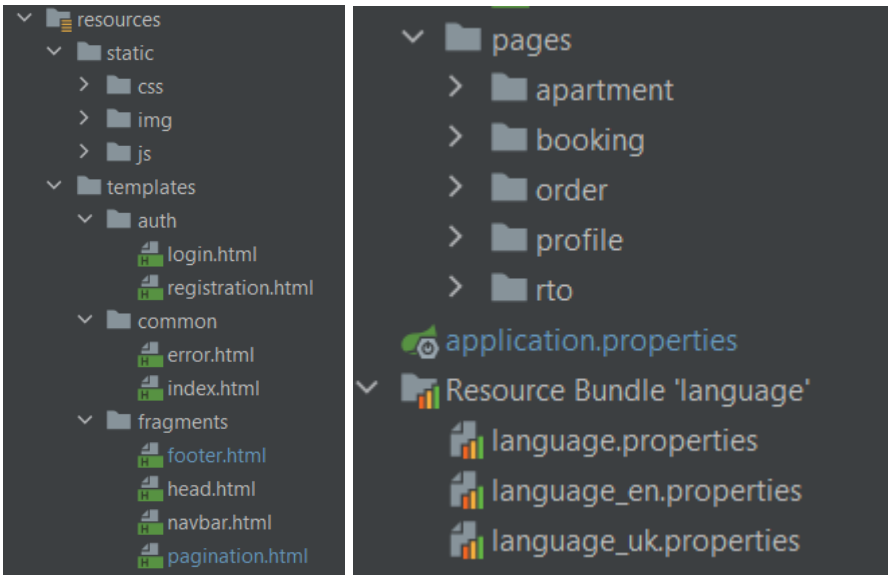

Рис. 3.8. Структура View рівня

Відповідно для функціонування представлень, застосовуються уже готові бібліотеки, які довантажуються користувачем індивідуально і не зберігаються на сервері рис.3.9.

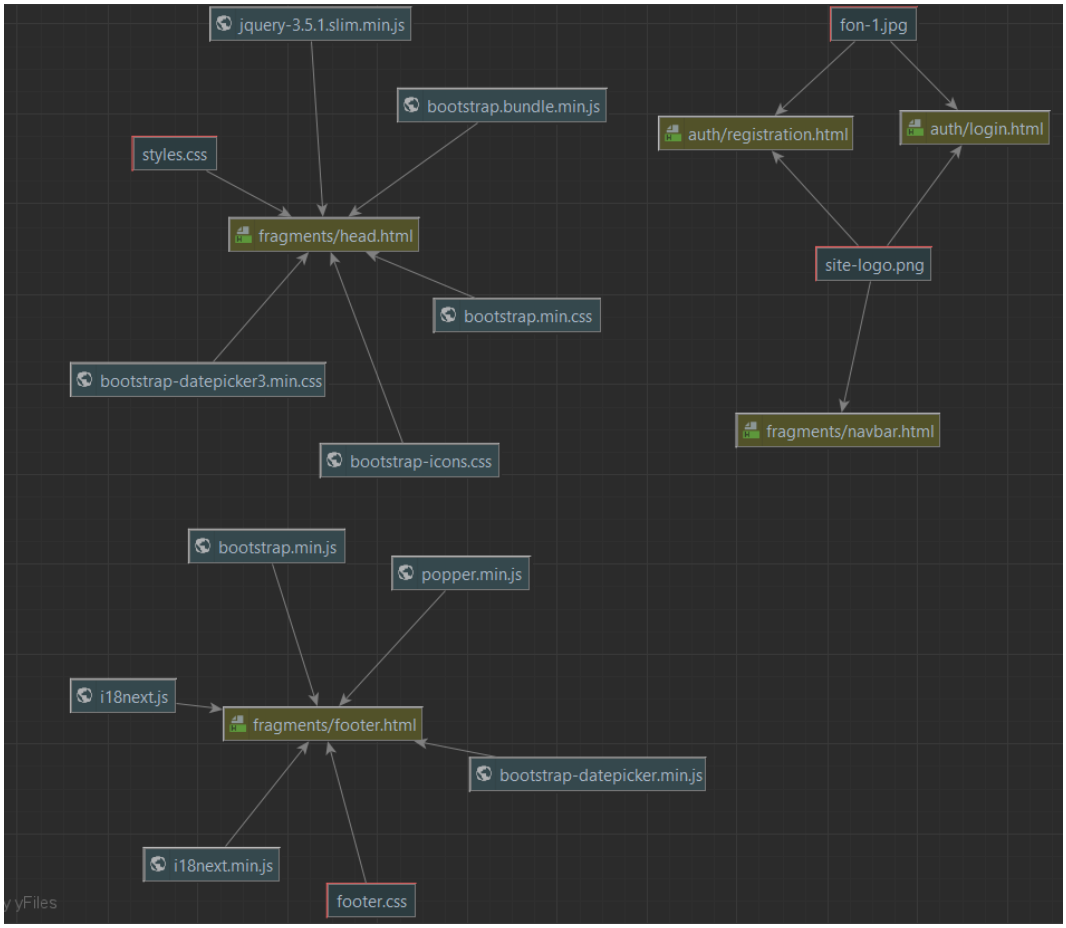

Рис. 3.9. Імпортовані бібліотеки та статичні засоби View рівня

В результаті користувач отримує гнучкий, масштабований, локалізований інтерфейс для взаємодії із сервісами додатку.

3.4 Тестування розробленого програмного продукту на основі тестових даних

Для забезпечення тестування були розроблені і застосовані Unit та Integrational тести.

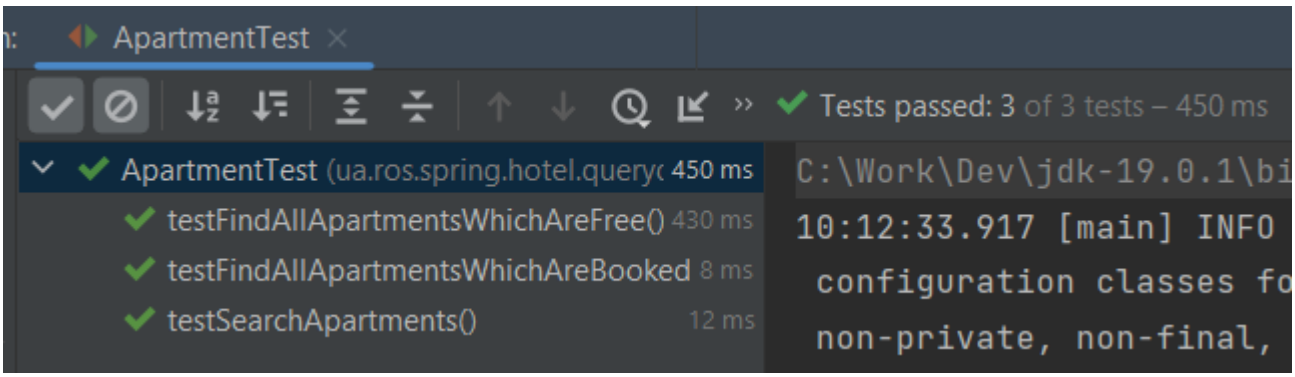

Рис. 3.10. Тестування додатку

Найважливішою частиною тестування є перевірка стабільності та швидкодії. Дана перевірка була проведена засобами Spring Boot Test (Junit) та з логером Sl4j. Отриманий результат продемонстровано на рис.3.10.

Час виконання тестового запиту становить 450 мілісекунд, з урахуванням відкриття нового підключення та першого запуску самої серверної частини. Якщо розглядати безпосередню роботу додатку, тобто запити поступають уже після формування кешу та розгортання підключень, тоді середній час відгуку становить від 8 до 50 мілісекунд.

Тестування проведене внутрішніми засобами Inteliji IDE із відображенням часу виконання кожного із тестів та з використанням засобів логування

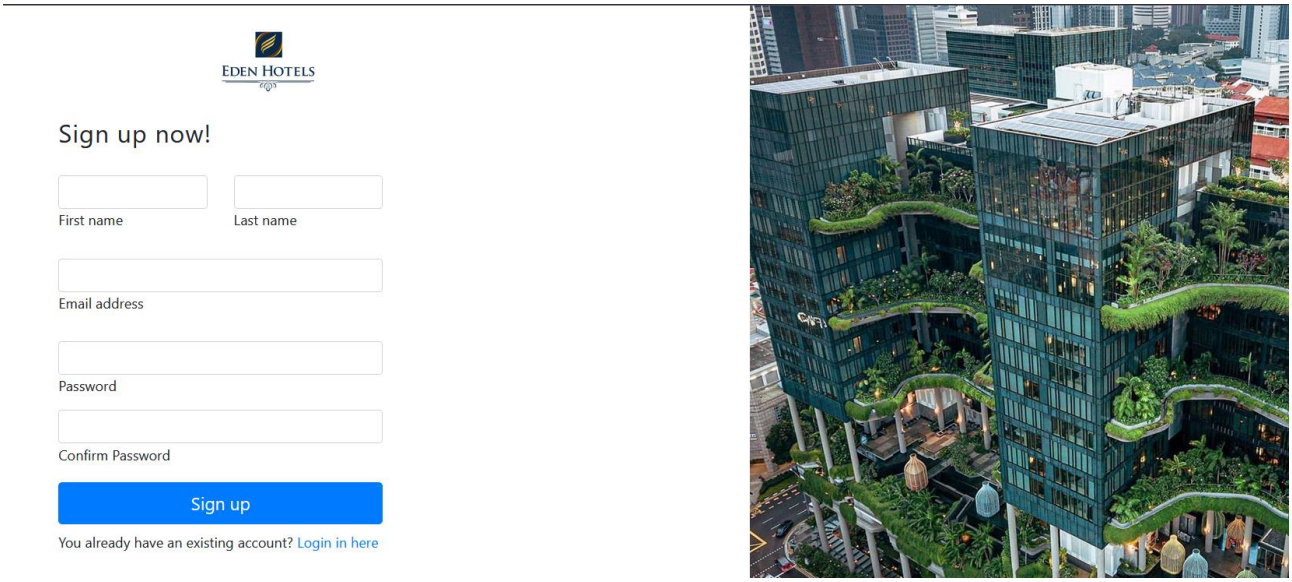

Рис. 3.11. Меню реєстрації в додатку

Наступнім кроком відбувається тестування кінцевого продукту з точки зору користувача. Найперше було проведено тестування на операційній системі Windows 10 із браузерного режиму та повноекранним розширенням. Меню реєстрації відображено на рис. 3.11.

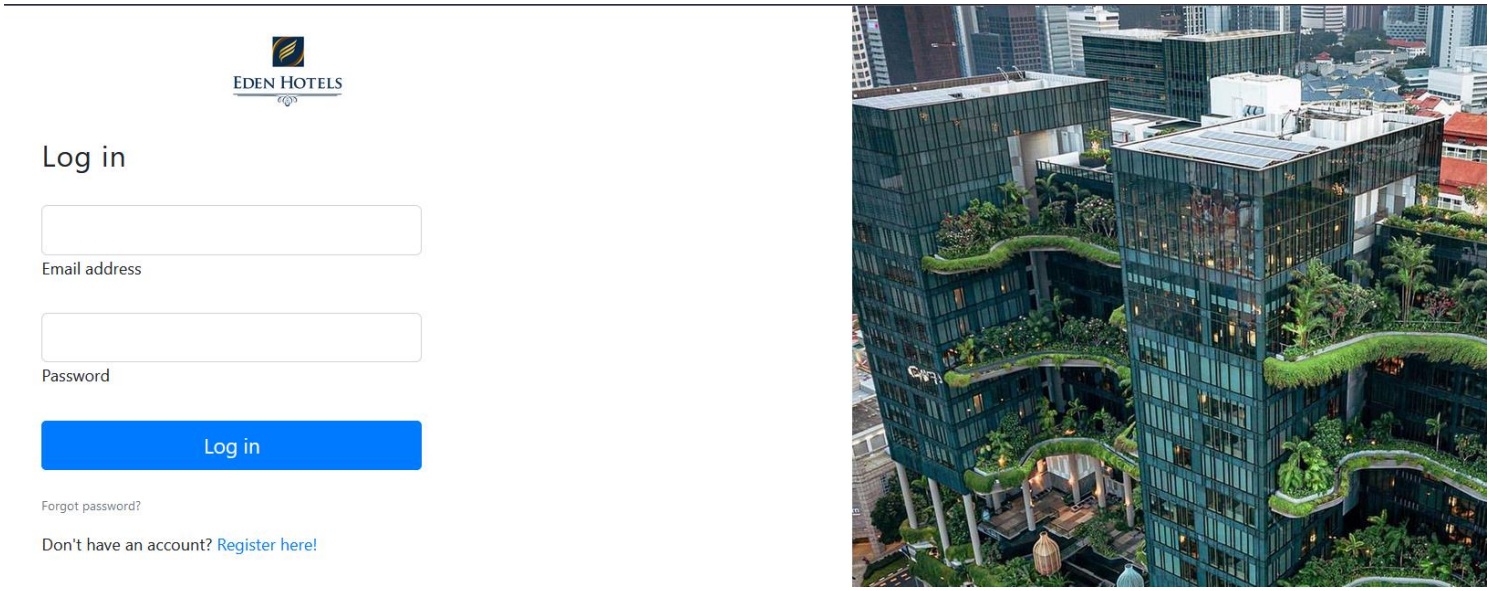

Рис. 3.12 Меню входу в додаток

У разі успішної реєстрації користувач буде перенаправлений на меню входу, яке відображено на рис.3.12.

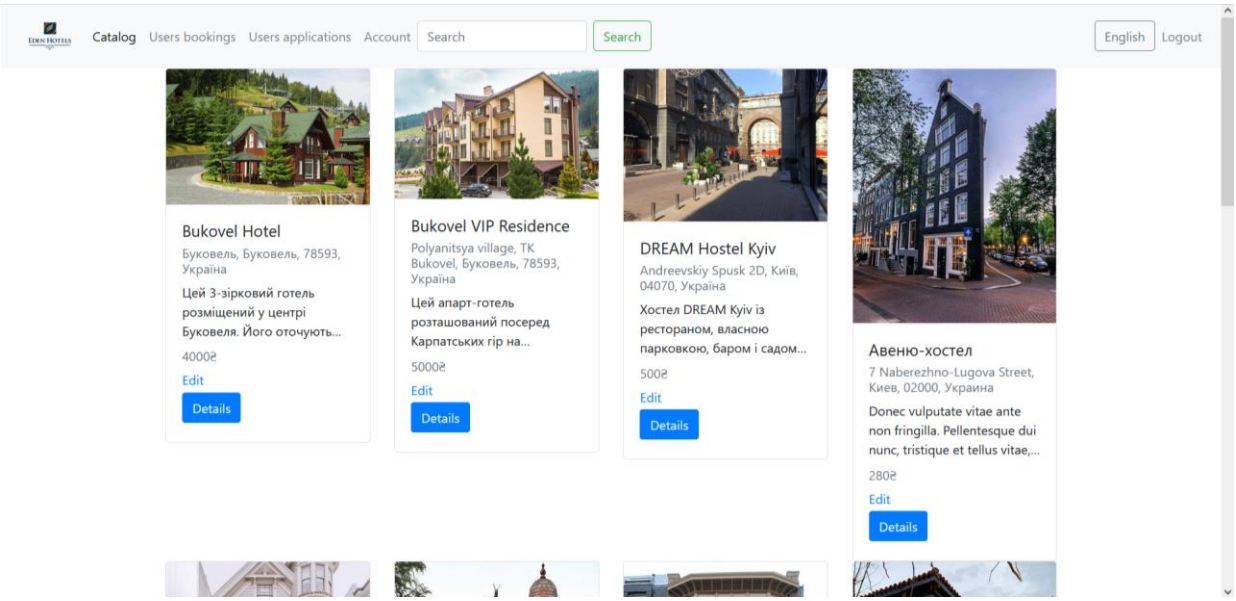

Рис. 3.13. Сторінка наявних апартаментів

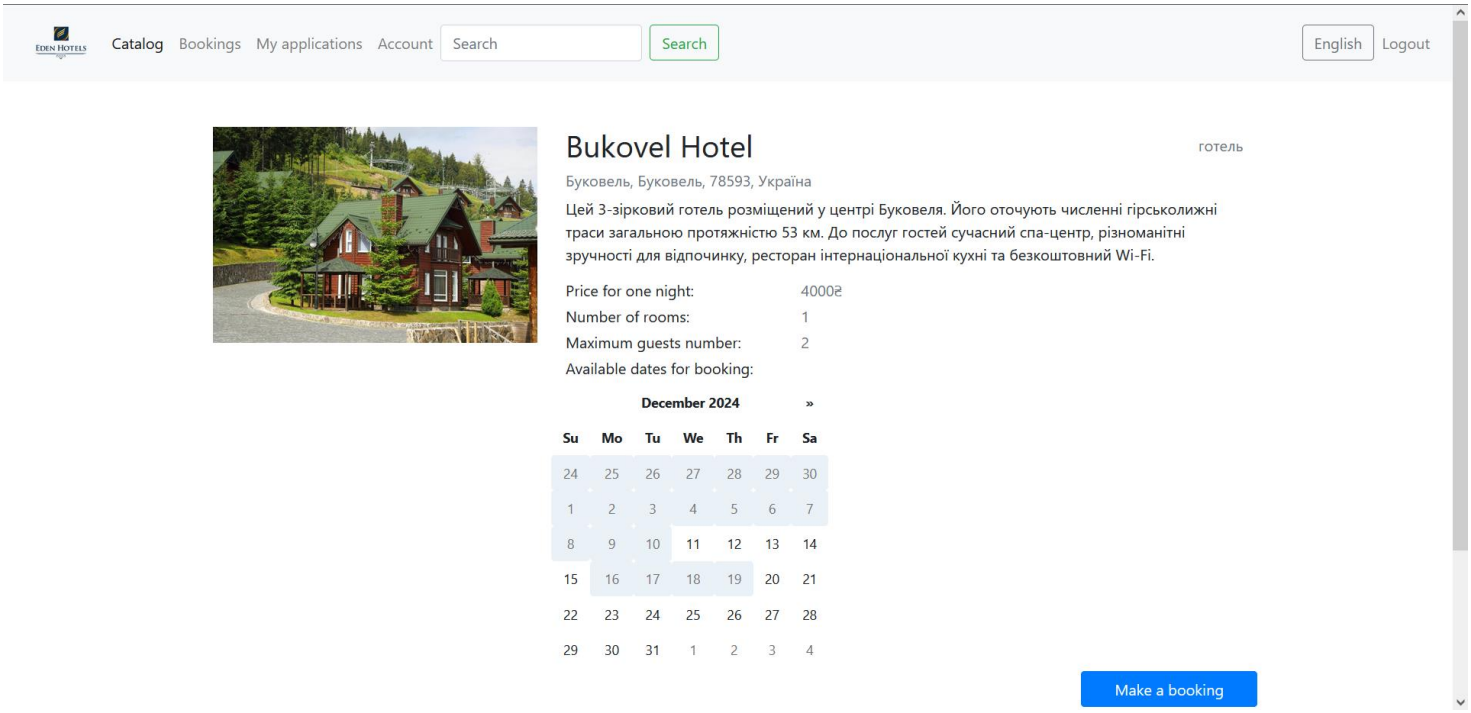

Рис. 3.14. Сторінка детальної інформації про апартаменти

Користувач може оглянути доступні апартаменти і прийняти рішення про бронювання, обравши доступну із списку дату. Заброньовані іншими користувачами дати відображені синім кольором і є неклікабельні. Вигляд сторінок представлений на рис.3.13. - рис.3.15.

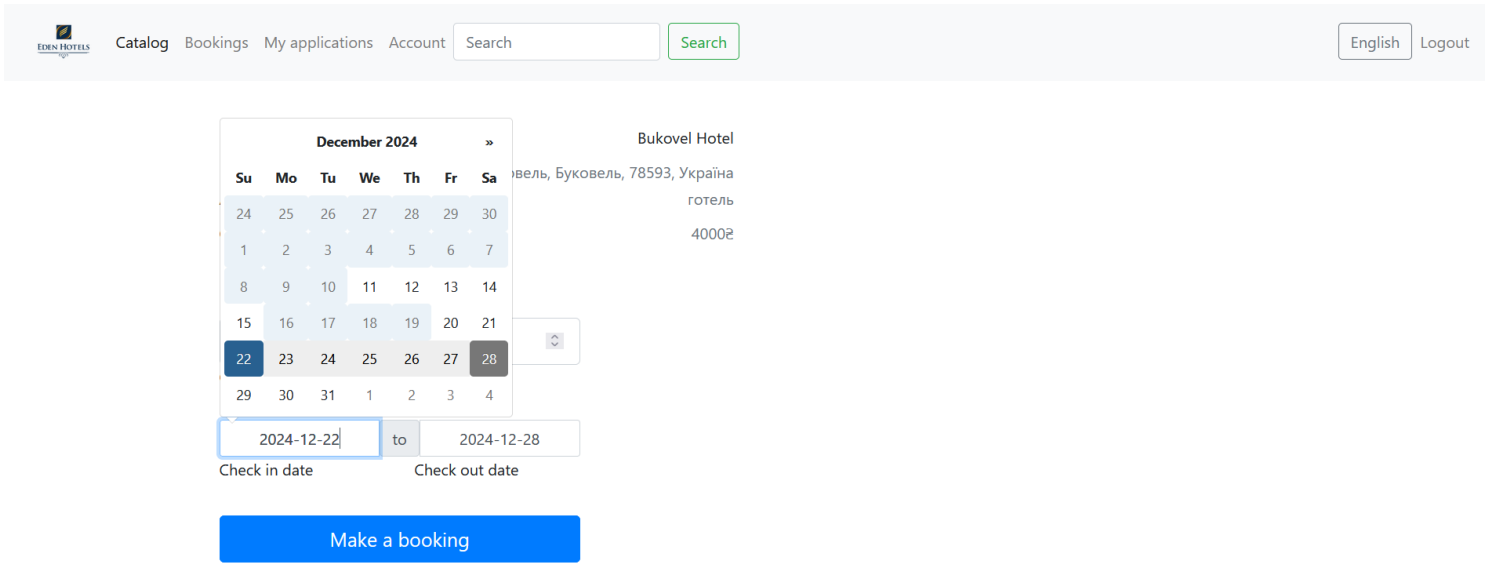

Рис. 3.15. Сторінка оформлення бронювання

Створене бронювання відобразиться в списку бронювань користувача та потребуватиме оплати (відображається жовтим кольором і відповідно зазначене). У разі несплати бронювання буде видалене через певний проміжок часу. Сплачені бронювання відображені зеленим кольором на рис.3.16.

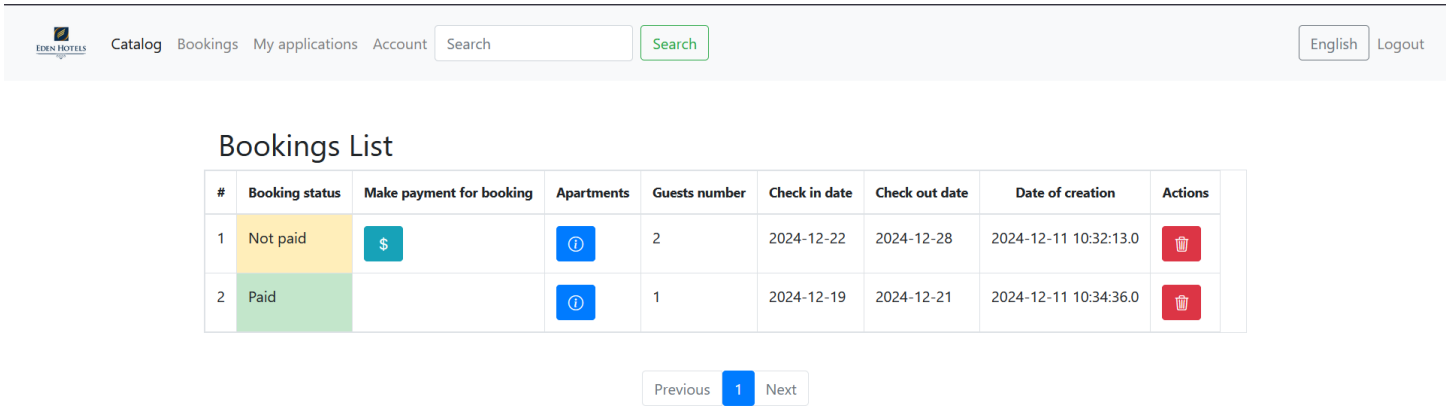

Рис. 3.16. Сторінка активних бронювань користувача

Для емуляції роботи з під мобільного пристрою був зменшений розмір браузерного вікна. Щоб повторно пройти основні кроки був виконаний вихід з аккаунту. Користувач автоматично був перенесений на сторінку входу відображену на рис.3.17.

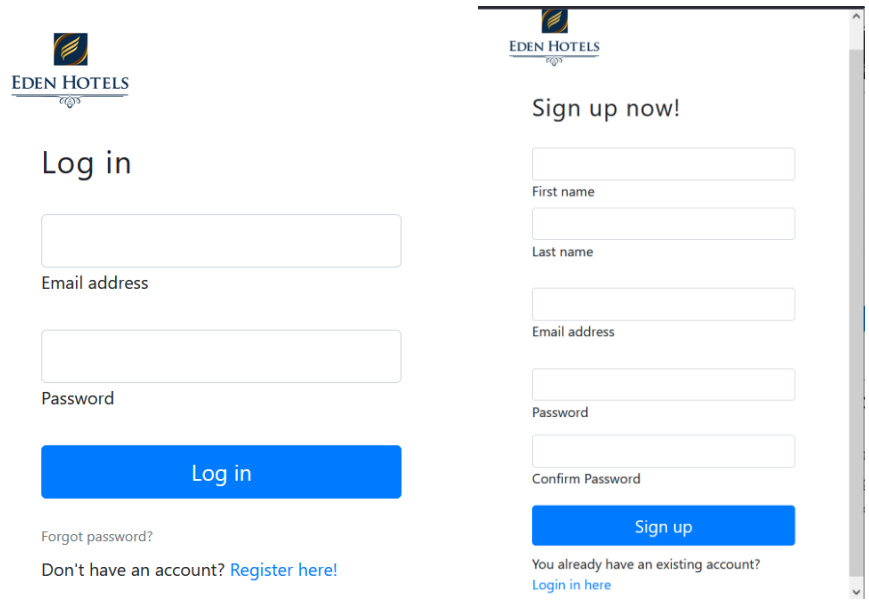

Рис. 3.17. Вигляд сторінки входу та реєстрації з мобільного пристрою

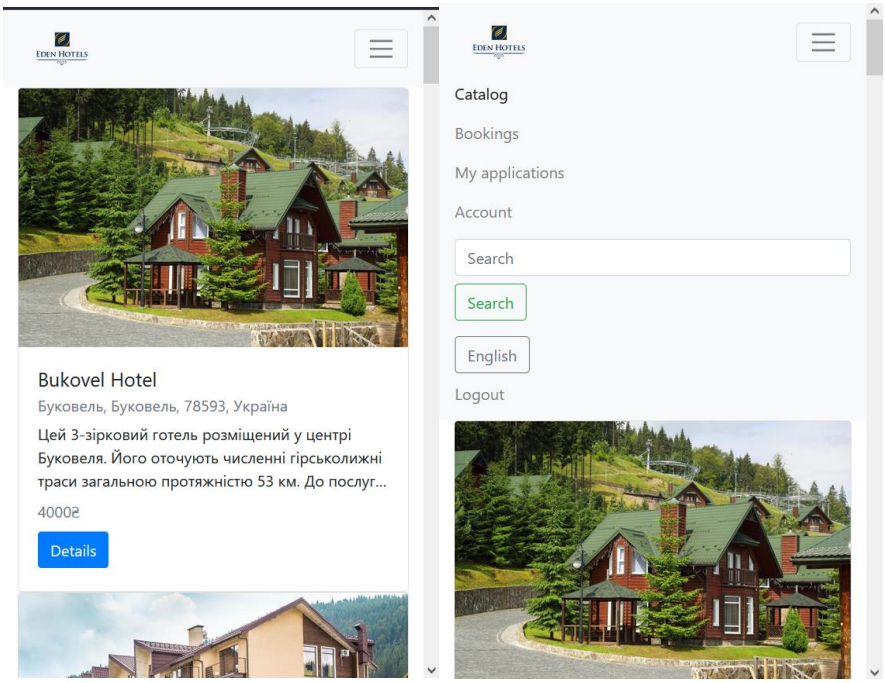

Рис. 3.18. Вигляд сторінки із апартаментами та навігаційного меню

Після повторного проходження автентифікації користувач переходить до головного меню відображеного на рис.3.18. Процес продовження використання сайту представлено на рис.3.19. - рис.3.20.

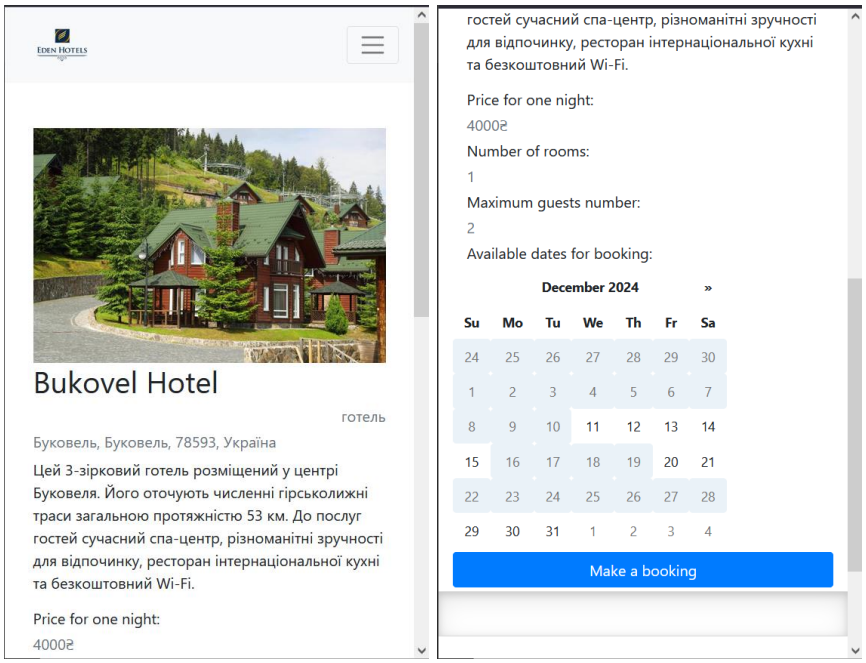

Рис. 3.19. Вигляд сторінки із апартаментами №2 та сторінки детальної інформації про апартаменти

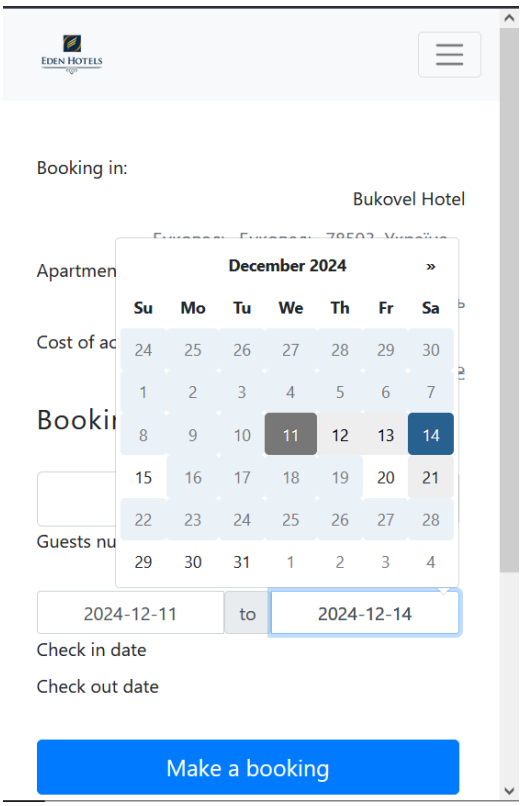

Рис. 3.20. Вигляд сторінки бронювання житла

Останнім етапом є тестування технології прямого пошуку. В поле для пошуку було введено вираз «апар*» та виконано пошук. Результат відображено на рис.3.21.

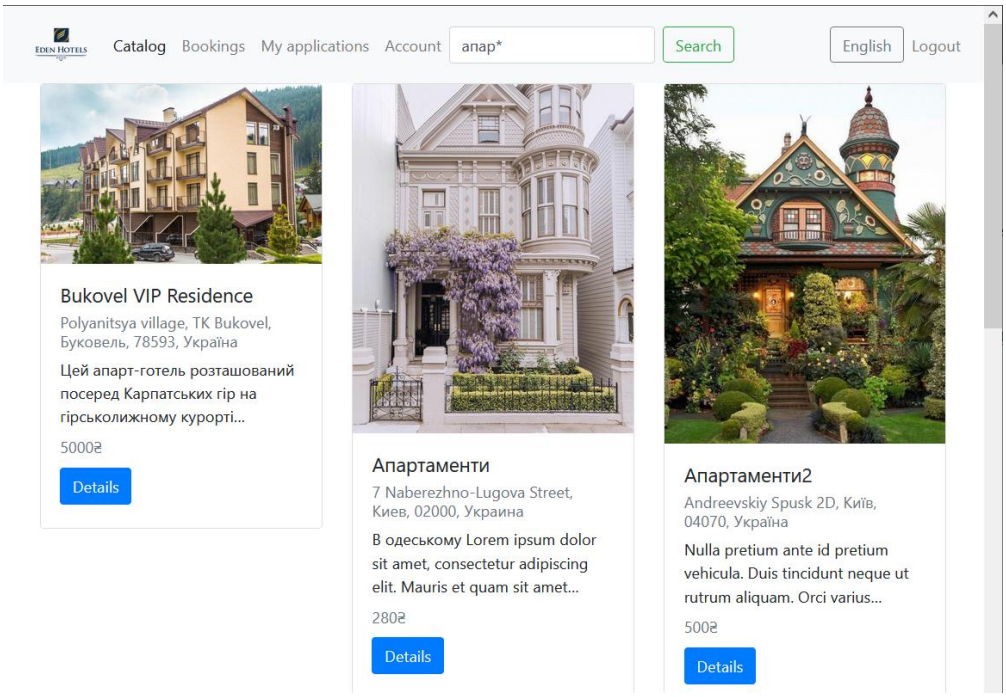

Рис. 3.21. Тестування прямого пошуку із неправильним виразом

Тобто тестується ситуація коли користувач не повністю ввів слово «апартаменти». В результаті було виведено список житла в назві та описі яких зустрічається вираз апар*. Під дану категорію також попав опис «апарт-готель». Більшість сучасних веб-сайтів не вміє працювати із частковим пошуком, а також в них не правильно реалізована система релевантності, яка побудована на шаблонах заданих адміністратором, а не відштовхується від запиту користувача. В розробленій системі пошук функціонує у відповідності до очікувань користувача

В результаті на вказаних вище рисунках продемонстровано повну роботоздатність додатку на різних платформах, завдяки масштабованому інтерфейсу й використанню незалежних від користувача серверних технологій

3.5 Висновки до розділу

В даному розділі сформульовані основні вимоги до програмного рішення в сфері готельного бізнесу, для забезпечення його успішного переходу до електронної комерції. Було запропоновано та обґрунтовано використання архітектурних підходів та патернів, згідно яким систему було розділено на самостійні компоненти з чіткими обов’язками й взаємодією. Також було розроблено і реалізовано ключові алгоритми необхідні для функціонування системи, забезпечення безпеки додатку і конфіденційності персональних даних користувачів. Після завершення розробки здійснено тестування кожного компоненту, перевірено продуктивність і роботоздатність визначених функцій. В результаті отримано програмний продукт який відповідає сучасним викликам, є гнучким, масштабованим, високопродуктивним і захищеним.

ВИСНОВКИ

В даній магістерській роботі було розглянуто проблему створення інноваційного програмного забезпечення в сфері e-commerce (електронної комерції), оскільки впровадження електронної комерції дозволяє підприємствам своєчасно реагувати на зміни в ринковому середовищі, швидко адаптуватися до нових вимог споживачів і постійно вдосконалювати власні продукти та послуги, що значно підвищує їх конкурентоздатність на ринку. Відповідно до цього було описано класифікацію інноваційних рішень, види та переваги електронної комерції, перспективи розвитку та основні етапи переходу бізнесу. Також розглянуто доступні на ринку рішення.

В процесі дослідження проаналізовано архітектури, які являють собою стандарт незалежно від вибору технологій. Наведені переваги і недоліки найбільш поширених розподілених систем. Розподіленість є важливою, оскільки для прикладу при застосуванні хмарних обчислень, в яких дані і ресурси зберігаються на численних серверах, навіть при виході з ладу одного з них система продовжить працювати завдяки автоматичному перенаправленню запитів на інші сервери. Це робить розподілені системи необхідним рішенням для бізнесів, які потребують безперебійної роботи, тобто розподіленість є особливо актуальною для електронної комерції. Проблемою розглянутих технологій є складність вибору та компонування їх між собою, і як наслідок налаштування та розгортання розробленого продукту

Для вирішення цієї проблеми було розглянуто Java Spring фреймворк в якості основної платформи. Проаналізовано та наведено характеристику найбільш популярних елементів його екосистеми. Java Spring пропонує гнучкість і масштабованість завдяки модульному підходу до розробки додатків. Вбудовані механізми управління залежностями даного фреймворку спрощують конфігурацію та сприяють створенню масштабованих і легко підтримуваних систем. Забезпечення підтримки інтеграції із більшістю сучасних технологій робить Spring одним із найкращих виборів для побудови динамічних, продуктивних і надійних рішень.

Не менш важливим аспектом є експлуатація та впровадження технологій. Для забезпечення вирішення цих проблем була наведена характеристика DevOps розробника і досліджений алгоритм автоматизації розгортання серверної частини Java Spring додатку.

Оскільки сучасна електронна комерція концентрується не тільки на надання послуг, а і на аналіз та обробку даних, відповідно було проведене дослідження та запропоновані моделі і алгоритми оптимізації роботи з даними.

Для демонстрації ефективності запропонованих рішень, було створено прототип програмного забезпечення, на прикладі готельного бізнесу. Відповідно до сформованих вимог розроблено ключові алгоритми та реалізовані всі необхідні для функціонування програми сервіси, ретельно описано кожен із рівнів і їхні обов’язки. Також було проведене комплексне тестування, яке підтвердило роботоздатність, ефективність і відповідність створеного прототипу поставленим вимогам. Результати роботи дозволяють підвищити точність вибору технологій для створення високоефективних рішень, здатних забезпечити адаптацію бізнесу до цифрових умов із мінімальними витратами та високим рівнем конкурентоздатності.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Priyaka K. Recent prospects and challenges in E-Commerce [Електронний ресурс] / K. Priyaka, K. Preeti. – 2023. – Режим доступу до ресурсу: http://dx.doi.org/10.13140/RG.2.2.25756.50565.
2. Traver C. E-Commerce 2020: Business, Technology, Society / Kenneth C. Laudon, Carol Guercio Traver., 2020. – (16).
3. Whiteley D. E-Commerce: Strategy, Technologies and Applications / David Whiteley., 2001.
4. Sangeet P. C. Platform Revolution: How Networked Markets Are Transforming the Economy /Geoffrey G. Parker, Marshall W. Van Alstyne, Sangeet Paul Choudary, 2016.
5. Kingsnorth S. Digital Marketing Strategy: An Integrated Approach to Online Marketing / Simon Kingsnorth., 2016.
6. Kleppmann M. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems / Martin Kleppmann., 2017.
7. Bass, L. Software Architecture in Practic / Bass, L., Clements, P., & Kazman, R, ., 2003 – (2).
8. Coulouris, G. Distributed systems: concepts and design / Coulouris, G., Dollimore, J., & Kindberg, T., 2012. – (5).
9. Tanenbaum, A. Distributed systems: principles and paradigms / Tanenbaum, A. S., & Van Steen, M., 2007. – (2).
10. Lynch, N. A. Distributed algorithms: an intuitive approach / Lynch, N. A., 1996.
11. Allspaw J. Web Operations: Keeping the Data On Time / John Allspaw, Jesse Robbins., 2010.
12. Johnson R. Spring Framework Reference Manual / Rod Johnson, Juergen Hoeller., 2011. - (3).
13. Walls C. Spring in Action / Craig Walls., 2018. - (5).
14. Mak K. Pro Spring 5 / Iuliana Cosmina, Rob Harrop, Chris Schaefer, Clarence Ho., 2017. - (4).

15. Sharma M. Beginning Spring Boot 2 / K. Siva Prasad Reddy, Shagun Bakliwal., 2019. - (3).
16. Gutierrez F. Spring Microservices in Action / John Carnell., 2017.
17. Kim G. The DevOps Handbook: How to Create World-Class Agility, Reliability, & Security in Technology Organizations / Gene Kim, Patrick Debois, John Willis, Jez Humble., 2016. - (2).
18. Huttermann M. DevOps for Developers / Michael Huttermann., 2012.
19. Elmasri R. Fundamentals of Database Systems / Ramez Elmasri, Shamkant B. Navathe., 2016. - (7).
20. Gajda P. NoSQL Databases: A Step-by-Step Guide / Pascal Gajda., 2016.
21. DB-Engines Ranking - Trend of Search Engines Popularity [Електронний ресурс] – Режим доступу до ресурсу: https://db-engines.com/en/ranking_trend/search+engine.
22. Elastic Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://www.elastic.co/docs.
23. Manticore Search [Електронний ресурс] – Режим доступу до ресурсу: https://manticoresearch.com/.
24. Manticore Search GitHub. Alternative to Elasticsearch [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/manticoresoftware/manticoresearch.
25. Model–view–controller pattern [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller.
26. Data access object [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Data_access_object.
27. Software repository [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Software_repository.
28. Ngyen L. Spring Security in Action MEAP V06 / Laurentiu Spilca., 2020. - (6).
29. Winch R. Pro Spring Security / Robert Winch, Carlo Scarioni., 2019. - (3).

30. Using the Event Scheduler [Електронний ресурс] – Режим доступу до ресурсу: https://dev.mysql.com/doc/refman/8.4/en/event-scheduler.html.
31. Документація Spring [Електронний ресурс] – Режим доступу до ресурсу: https://spring.io/projects/spring-boot.
32. Іванишин Р. М. Підвищення ефективності розробки програмного забезпечення на базі Java Spring Data JPA з інтеграцією предметно-орієнтованої мови (DSL) / Р. М. Іванишин. // Матеріали всеукраїнської науково-практичної конференції молодих учених і студентів "Інформаційні технології в освіті, техніці та промисловості", Івано-Франківськ, 10 жовтня 2024 р. - с. 236–238.
33. Bootstrap Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://getbootstrap.com/docs/5.3/getting-started/introduction/
34. Thymeleaf Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://www.thymeleaf.org/documentation.html.
35. Enterprise Integration Patterns Enterprise Integration Patterns [Електронний ресурс] – Режим доступу до ресурсу: https://www.enterpriseintegrationpatterns.com/.
36. "What is Cloud Computing?". Amazon Web Services. 2013-03-19. Retrieved 2013-03-20.
37. "Cloud Computing: Clash of the clouds". The Economist. 2009-10-15. Retrieved 2009-11-03.
38. "Gartner Says Cloud Computing Will Be As Influential As E-business". Gartner. Retrieved 2010-08-22.
39. Gruman, Galen (2008-04-07). "What cloud computing really means". InfoWorld. Retrieved 2009-06-02.
40. "Announcing Amazon Elastic Compute Cloud (Amazon EC2) - beta". Amazon.com. 24 August 2006. Retrieved 31 May 2014.
41. Rivest, R.; Shamir, A.; Adleman, L. (February 1978). "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems" (PDF). Communications of the ACM. 21 (2): 120–126. doi:10.1145/359340.359342.

42. Diffie, W.; Hellman, M.E. (November 1976). "New directions in cryptography". IEEE Transactions on Information Theory. 22 (6): 644–654. doi:10.1109/TIT.1976.1055638. ISSN 0018-9448.
43. Rivest, Ronald. "The Early Days of RSA -- History and Lessons" (PDF).
44. R. Szeliski, Computer Vision, algorithms and applications, Springer
45. Azure Електронний ресурс. - Режим доступу: https://azure.microsoft.com/en-us/

ДОДАТКИ

Додаток А

Програмна сутність (Entity)

```
@Entity
@Table(name = "account")
@NoArgsConstructor
@AllArgsConstructor
@Getter
@Setter
@Builder(toBuilder = true)
@ToString
public class Account implements Serializable {

    @Serial
    private static final long serialVersionUID = 12994784029922283L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Enumerated(EnumType.STRING)
    @Column(name = "role")
    private AccountRole role;
    @Column(name = "first_name")
    private String firstName;
    @Column(name = "last_name")
    private String lastName;
    @NaturalId
    @Column(name = "email")
    private String email;
    @ToString.Exclude
    @Column(name = "password")
    private String password;
    //indicate state of the Account (false - blocked/ true - available)
```

```
    @Column(name = "state")
    private Boolean state = true;

    //Foreign keys
    @ToString.Exclude
    @OneToMany(mappedBy = "account", cascade = CascadeType.ALL)
    private List<Booking> bookings = new ArrayList<>();

    @ToString.Exclude
    @OneToMany(mappedBy = "account", cascade = CascadeType.ALL)
    private List<Order> orders = new ArrayList<>();

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;

        Account account = (Account) o;

        return email.equals(account.email);
    }

    @Override
    public int hashCode() {
        return email.hashCode();
    }
}
```

Додаток В

Реалізація патерну Repository модельного рівня додатку

```
      @Repository
public interface AccountRepository extends JpaRepository<Account,
Long>, QuerydslPredicateExecutor<QAccount> {

    Optional<Account> findByEmail(String email);
    @Modifying(flushAutomatically = true)
    @Query(value = "DELETE FROM account WHERE id = ?1", nativeQuery =
true)
    int customDeleteById(Long id);

}
      @Repository
public interface ApartmentRepository extends JpaRepository<Apartment,
Long>, QuerydslPredicateExecutor<QApartment> {


    @Query(value = "SELECT SQL_CALC_FOUND_ROWS * FROM apartment "+
                    "INNER JOIN (SELECT DISTINCT apartment.`id` as
inner_id FROM apartment " +
                                    "INNER JOIN booking ON
booking.`apartment_id` = apartment.`id` " +
                                    "WHERE `booking`.`check_in_date` >=
CURRENT_DATE() " +
                                          "OR `booking`.`check_out_date`
>= CURRENT_DATE() " +
                    ") AS a2 " +
                    "ON apartment.`id` = a2.inner_id ",
            countQuery = " SELECT FOUND_ROWS() AS count ",
           nativeQuery = true)
    Page<Apartment> findAllApartmentsWhichAreBooked(Pageable
pageable);


     @Query(value = "SELECT SQL_CALC_FOUND_ROWS * FROM apartment "+
             "WHERE apartment.`id` NOT IN (SELECT DISTINCT
`apartment`.`id` as inner_id FROM apartment " +
             "INNER JOIN `booking` booking ON booking.`apartment_id` =
      `apartment`.`id` " +
             "WHERE `booking`.`check_in_date` >= CURRENT_DATE() OR
`booking`.`check_out_date` >= CURRENT_DATE() " +
             ") AND apartment.state = 1 ",
             countQuery = " SELECT FOUND_ROWS() AS count ",
             nativeQuery = true)
    Page<Apartment>        findAllApartmentsWhichAreFree(Pageable
pageable);

    Page<QApartment> findAll(Predicate predicate, Pageable pageable);
```

Продовження додатку А

```
    @Query(value = "SELECT SQL_CALC_FOUND_ROWS *, " +
            "MATCH(`title`, `description`, `address`,
`apartment_class`) AGAINST(:search_value IN BOOLEAN MODE) AS REL " +
            "FROM apartment " +
            "WHERE MATCH(`title`, `description`, `address`,
`apartment_class`) AGAINST(:search_value IN BOOLEAN MODE) " +
            "ORDER BY REL DESC ",
            countQuery = " SELECT FOUND_ROWS() AS count ",
            nativeQuery = true)
    Page<Apartment> searchApartments(Pageable pageable, String
search_value);

    @Modifying(flushAutomatically = true)
    @Query(value = "DELETE FROM apartment WHERE id = ?1", nativeQuery
= true)
    int customDeleteById(Long id);
}

      @Repository
public interface BookingRepository extends JpaRepository<Booking,
Long>,

QuerydslPredicateExecutor<QBooking> {

    @Query("SELECT b FROM Booking b " +
            "WHERE b.apartment.id = ?1 " +
            "AND (b.checkInDate >= CURRENT_DATE " +
            "OR b.checkOutDate >= CURRENT_DATE)")
    Optional<ArrayList<Booking>>


findAllBookingsRelatedToApartment(Long apartmentId);


    @Modifying(flushAutomatically = true)
    @Query(value = "DELETE FROM booking WHERE id = ?1", nativeQuery =
true)
    int customDeleteById(Long id);
}

      @Repository
public interface OrderRepository extends JpaRepository<Order, Long>,
QuerydslPredicateExecutor<QOrder> {
    @Query("SELECT Order FROM Order WHERE ?1 LIKE ?2")
    Optional<Order> findByField(String field, Object value);

    @Modifying(flushAutomatically = true)
    @Query(value = "DELETE FROM `order` WHERE id = ?1", nativeQuery =
```

Продовження додатку А

```
      true)
    int customDeleteById(Long id);
}
      @Repository
public interface ResponseToOrderRepository extends
JpaRepository<ResponseToOrder, Long>,
QuerydslPredicateExecutor<QResponseToOrder> {

    @Query("SELECT ResponseToOrder FROM ResponseToOrder WHERE ?1 LIKE
?2")
    Optional<ResponseToOrder> findByField(String field, Object value);

    @Modifying(flushAutomatically = true)
    @Query(value = "DELETE FROM response_to_order WHERE id = ?1",
nativeQuery = true)
    int customDeleteById(Long id);

    /** Delete all attached to response-to-order(RTO) apartments by
RTO id.
     *  <br> Delete only references to apartments
     * @param id response to order id
     * @return Boolean operation result
     */

    @Modifying(flushAutomatically = true)
    @Query(value = "DELETE FROM `response_to_order_has_apartment`
WHERE response_to_order_id = ?1", nativeQuery = true)
    boolean deleteResponseApartments(Long id);


}
```