НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ ***__Стіренко С.Г._***
(підпис)
“___”_____20__ р

Дипломний проєкт

на здобуття ступеня бакалавра

з освітньо-професійною програмою «Комп’ютерні системи та мережі»

спеціальності 123 «Комп’ютерна інженерія»

на тему: «Система обробки замовлень»

Виконав:
студент _IV_курсу, групи _ІВ-72__

___Гурняк Андрій Миколайович_____ _____
(прізвище, ім’я, по батькові) (підпис)

Керівник __асистент Сергієнко А.А._____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Консультант __н. контроль_ __проф. д.т.н. Сімоненко В.П.___ _____
(назва розділу) (вчені ступінь та звання, прізвище, ініціали) (підпис)

Рецензент _____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент __
(підпис)

Київ – 2021 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМ. ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалаврський)
Освітньо-професійна програма «Комп'ютерні системи та мережі»
Спеціальність 123 «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ ***Стіренко С.Г.***
(підпис)
"___"_____20__ р.

ЗАВДАННЯ

на дипломний проєкт студента

Гурняка Андрія Миколайовича

1. Тема проєкту «*Система обробки замовлень»*

керівник проєкту _Сергієнко Анастасія Анатоліївна_, асистент.

Затверджені наказом по університету від «*11*» *05* 2021р. №1139-с

2. Термін здачі студентом закінченої роботи *2021р.*

3. Вихідні дані до проєкту *технічне завдання, теоретичні дані.*

4. Зміст пояснювальної записки: *опис предметної області, дослідження та створення системи обробки замовлень, функціональна програма з її об'єктивним зображенням та використаний програмний код при її створенні.*

5. Консультант роботи, з вказівкою розділів роботи, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	д.т.н., проф. Сімоненко В.П.		

6. Дата видачі завдання _____ *року*

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітки
1.	*Затвердження теми роботи*	10.12.2021-15.12.2021	
2.	*Вивчення та аналіз завдання*	15.12.2021-15.03.2021	
3.	*Розробка архітектури та загальної структури систем*	15.03.2021-25.03.2021	
4.	*Розробка структур окремих підсистем*	25.03.2021-5.04.2021	
5.	*Програмна реалізація системи*	5.04.2021-15.04.2021	
6.	*Оформлення пояснювальної записки*	15.04.2021-20.05.2021	
7.	*Передзахист*	23.05.2021	
8.	*Захист*	16.06.2021	

Студент _____

(підпис)

Керівник _____

(підпис)

Анотація

В бакалаврському дипломному проєкті розроблено систему обробки замовлень.

Запропонована система обробки замовлень може бути використана для фактичного застосування та оптимізації обробки замовлень на базі інтернет-магазинів, сайтів підприємства, тощо у рамках малого бізнесу. Застосунок був написаний на мові програмування Java з використанням Spring Framework та системою збірки – Maven. Базою даних слугує MySQL. Диспетчером повідомлень виступає Apache Kafka. Дана система реалізує мікросервісний шаблон - Saga для безпосередніх транзакцій.

Annotation

An order processing system has been developed in the bachelor's degree project.

The proposed order processing system can be used for the actual use and optimization of order processing on the basis of online-stores, company websites, etc. The product was written on Java programming language using Spring Framework and Maven build system. System use MySQL Database. Apache Kafka is the message manager. Given system implements a microservice pattern - Saga for direct transactions.

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЄКТУ

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4	ІАЛЦ.467200.001 ОА	Опис альбому	1	
2	А4	ІАЛЦ.467200.002 ТЗ	Технічне завдання	4	
3	А4	ІАЛЦ.467200.003 ПЗ	Пояснювальна записка	57	
4	А4	ІАЛЦ.467200.004 Д1	Просторово-часова діаграма	1	
5	А4	ІАЛЦ.467200.005 Д2	Схема алгоритму створення замовлення	1	
6	А4	ІАЛЦ.467200.006 Д3	Структурна схема сервісів	1	
7	А4	ІАЛЦ.467200.007 Д4	Лістинг програми	18	

Зм.	Арк.	№ документа	Підпис	Дата	*ІАЛЦ.467200.001 ОА*		
Розробив		Гурняк А.М			**Система обробки замовлень** Опис альбому	Літ	Аркуш 1 / Аркушів 1
Перевірив		Сергієнко А.А.					
Н. Контр.		Сімоненко В.П.				*НТУУ «КПІ» ФІОТ*	
Затвердив.						*Група ІВ-72*	

ТЕХНІЧНЕ ЗАВДАННЯ

до дипломної роботи
освітньо-кваліфікаційного рівня бакалавр

на тему: “Система обробки замовлень”.

Київ – 2021 р.

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ 2

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ 2

3. МЕТА І ПРИЗНАЧЕННЯ РОЗРОБКИ 2

4. ДЖЕРЕЛА РОЗРОБКИ..... 2

5. ТЕХНІЧНІ ВИМОГИ..... 3

5.1. Вимоги до програмного забезпечення..... 3

5.2. Вимоги до апаратної частини 3

6. ЕТАПИ РОЗРОБКИ 4

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Найменування: система обробки замовлень.

Область застосування: практичне використання людьми при користуванні мережею інтернет.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання роботи кваліфікаційно-освітнього рівня «бакалавр комп'ютерної інженерії», затверджене кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний інститут ім. Ігоря Сікорського».

3. МЕТА І ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою даного проєкту є розробка системи обробки замовлень.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки є науково-технічна література з теорії і практики програмування, бакалаврські роботи інших студентів, публікації в Інтернеті з даних питань.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розроблюваного продукту

- Самостійність курсу – чіткі контури предмету вивчення.
- Самодостатність – курс повинен містити необхідну інформацію та дані, які дозволяють у повній мірі розкрити мету курсу.
- Коректність та актуальність інформації, які охоплює курс.

5.2. Вимоги до програмного забезпечення

- Unix-подібна ОС або Windows 7/8/10;
- Java VM 11.0;
- MySQL;
- Apache Kafka 2.8.0.

5.3. Вимоги до апаратної частини

- Серверний комп'ютер на базі процесора Intel Core i3 та вище;
- Не менше 16 Гбайт оперативної пам’яті;
- Не менше 128 Гбайт простору жорсткого диску.

6. ЕТАПИ РОЗРОБКИ

	Дата
Вивчення літератури	_____
Складання і узгодження технічного завдання	_____
Створення модулів системи, що розробляється	_____
Тестування окремих модулів системи	_____
Допрацювання, налагодження і виправлення помилок	_____
Оформлення документації дипломної роботи	_____

ПОЯСНЮВАЛЬНА ЗАПИСКА
до дипломного проєкту
на тему: «Система обробки замовлень»

Київ – 2021 р.

ЗМІСТ

ЗМІСТ 1
СПИСОК СКОРОЧЕНЬ 2
ВСТУП 3
РОЗДІЛ 1. ОГЛЯД ІСНУЮЧИХ СИСТЕМ ОБРОБКИ ЗАМОВЛЕНЬ 4
1.1 Загальні положення CRM-концепції 4
1.2 Система «Бітрікс24» 7
1.3 Система «AmoCRM» 9
1.4 Система «Мегаплан» 10
1.5 Порівняння наведених систем 12
1.6 Формулювання вимог 15
ВИСНОВКИ ДО РОЗДІЛУ 1 17
РОЗДІЛ 2. ОБГРУНТУВАННЯ ОБРАНОГО ІНСТРУМЕНТАРІЮ ДЛЯ СТВОРЕННЯ СИСТЕМИ ОБРОБКИ ЗАМОВЛЕНЬ 18
2.1 Мова програмування Java 18
2.2 Spring 22
2.3 MySQL 24
2.4 Kafka 29
2.5 Транзакційний процес на основі Saga 29
ВИСНОВКИ ДО РОЗДІЛУ 2 34
РОЗДІЛ 3. ВНУТРІШНЯ БУДОВА ТА СТРУКТУРА ЗАПРОПОНОВАНОЇ СИСТЕМИ ОБРОБКИ ЗАМОВЛЕНЬ 35
ВИСНОВКИ ДО РОЗДІЛУ 3 44
РОЗДІЛ 4. ІНСТРУКЦІЯ API-КОРИСТУВАЧА 47
ВИСНОВКИ ДО РОЗДІЛУ 4 54
ВИСНОВКИ 55
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 56

СПИСОК СКОРОЧЕНЬ

API − (англ. application programming interface) - програмний інтерфейс додатку, інтерфейс прикладного програмування.

СУБД − (англ. Database Management System, сокр. DBMS) − система керування (управління) базами даних.

MVC − (англ. Model-View-Controller) − Модель-Подання-Контролер.

JVM − (англ. Java Virtual Machine) − це абстрактна обчислювальна машина.

AMQP − (англ. Advanced Message Queuing Protocol) − вдосконалений протокол організації черг повідомлень.

HTTPS – (англ. HyperText Transfer Protocol Secure) − розширення протоколу HTTP для підтримки шифрування з метою підвищення безпеки.

HTTP − протокол для передачі гіпертекстових документів.

SQL – (англ. (Structured Query Language) − структурована мова запитів.

ВСТУП

У всьому світі простежується динаміка збільшення інтернет-користувачів. Ринок товарів та послуг відчуває явний приріст клієнтів в онлайн-сегменті. Це обумовлено різними факторами, у тому числі, наявним наразі впливом захворювання COVID-19.

Клієнта-користувача сегменту інтернет-замовлень вже мало чим здивуєш, ані різноманіттям товарів чи послуг. Останнім часом все більше клієнти обирають відповідний сервіс завдяки якості обслуговування, автоматизований та зрозумілій інтерфейс, а також те, як клієнту допомогли та вирішили його питання.

Актуальність обраної теми полягає якраз у створенні сприятливого інтерфейсу та безпосередньо системи обробки замовлень.

Проблематика даної теми зводиться до знаходження рішень (таких систем), які б відповідали сучасним вимогам ринку та користувачам.

Кінцева **мета** обраної теми – вирішити нагальні питання, щодо якості обслуговування клієнтів завдяки відповідній системі.

Для досягнення поставленої мети до уваги **пропонується** вироблення системи обробки замовлень, яка б була функціонально корисна та відрізнялась від звичайного софту.

За **результатами** була створена відповідна система обробки замовлень.

РОЗДІЛ 1
ОГЛЯД ІСНУЮЧИХ СИСТЕМ ОБРОБКИ ЗАМОВЛЕНЬ

1.1 Загальні положення CRM-концепції.

В даний час взаємини з клієнтами стали ключовою метою кожної компанії. Успішне співробітництво з клієнтом неминуче веде до подальших звернень та рекомендацій послуг компанії, а отже і до розширення клієнтської бази.

У деяких сферах конкуренція за споживача стала дуже гострою. Багато компаній вже не можуть знижувати свої тарифи для залучення покупців − залишається конкурувати за рахунок якісних показників, в тому числі і поліпшення роботи з клієнтами, поліпшення рівня сервісу. Основним завданням для компанії є утримання наявних клієнтів. А це як раз область CRM-систем.

Сьогодні недостатньо виробити товар, його треба персоналізувати, пристосувати для потреб конкретного індивідууму. Маркетинг починається з ідеї виробництва товару або задуму надання послуги, виробництво налаштовується на випуск все більш адаптивних під замовника виробів, реклама забезпечує інформованість про наявність товару, а CRM дозволяє замкнути весь цикл шляхом «правильної» роботи з клієнтом [1].

Наразі, для того щоб розвивати інтернет-діяльність з продаж товару або надання послуг, треба долучати нових клієнтів та збільшувати кількість проданих товарів необхідно налагодити усі процеси роботи з клієнтами та між працівниками сервісу. Цю проблему вирішують системи управління відносинами з клієнтами.

CRM (Customer relationship management або Управління відносинами з клієнтами) − поняття, що використовує концепції, які існують у компаніях для контролю та управління взаємовідносинами зі споживачами, включаючи збір, зберігання й аналіз інформації про споживачів, партнерів, постачальників та інформації про взаємовідносини з ними [2].

Вивчення ринку і цільових потреб клієнтів є основними задачами сучасної CRM. Використовуючи цю інформацію розробляються нові товари або послуги, дозволяючи компанії досягати поставлених цілей і покращити фінансовий показник.

Зазвичай виділяють чотири критерії класифікації:

1. Стратегічна мета − фокус на створення та підтримка клієнт орієнтованої системи. Розширення та отримання інформації про клієнтів та, на її основі, індивідуалізація взаємодії зі споживачами. Завдяки цьому з клієнтами встановлюються більш тривалі відносини. Утримати старого клієнта простіше й водночас важливіше, аніж залучити нового;

2. Оперативна – задум подібної системи інтегрувати і автоматизувати продажу, маркетинг і підтримку клієнтів. Даний функціонал реалізуються через окремі сторінки під кожного цільового клієнта компанії, дошки з оглядом на загальну діяльність компанії. Сторінки з інформацією про клієнта містять також і історію дзвінків, продажів, планованих дати клієнта і т. д. Це є інформацією з відносин між компанією і споживачем. За цією класифікацією виділяють три головні складові: блоки маркетингу (наприклад, інформування про знижки та акції), автоматизації продажів і обслуговування (наприклад, поєднання з системами повідомлень, електронною поштою та ін.);

3. Аналітична – роль аналітичних CRM систем полягає в аналізі даних про користувача, які були зібрані з різних джерел, і відображення цих даних у презентаційному вигляді, щоб бізнес-керівники приймали більш зважені рішення. Наприклад, використовуючи аналіз покупок та товарних пріоритетів окремої бази клієнтів, компанія може зрозуміти які товари чи послуги втратили популярність серед цих клієнтів за останній час. Базуючись на отриманих даних приймається рішення про запуск маркетингової кампанії, спрямовану на просування продуктів серед цієї клієнтської бази;

4. Мета співпраці – важливою метою систем CRM є об'єднання зовнішніх зацікавлених сторін: дистриб'юторів, продавців, постачальників.

Також не менш важливим є обмін інформацією про клієнтів серед цих організацій. Наприклад, служби підтримки через дзвінки збирають відгуки, потім зібрана інформація використовуються у відділі маркетингу [3].

З використанням CRM легше налагодити внутрішні комунікації. Співробітники в різних відділах можуть ефективніше працювати і бачити ситуацію компанії в цілому. І чим більша компанія або проєкт, тим більш вагомою є ця перевага. Стає можливим об'єктивно контролювати якість і ефективність роботи співробітників і відділів компанії − дані збираються на єдиному сервері (переважно − у хмарному сховищі), і доступні для обробки в будь-який момент. Багато CRM роблять акцент на візуалізацію, тому процес контролю і відстеження слабких місць в роботі стає ще простіше.

Darrell K. Rigby і Dianne Ledingham [3,4] пропонують наступний життєвий цикл процесу взаємини з клієнтами, що автоматизується CRM системою. Даний цикл складається з 5 сегментів:

• Розвиток пропозиції. Даний сегмент включає в себе дії спрямовані на розробку товарів і послуг компанії та аналізу орієнтирів компанії.

• Продаж. Даний сегмент орієнтований на управління процесом продажів.

• Покращення взаємодії. Сегмент включає в себе допоміжні процеси, що поліпшують взаємодію з клієнтами.

• Утримання та виручка. Містить процеси, що контролюють утримання клієнтів компанії і збільшення взаємної вигоди від партнерства.

• Орієнтування та маркетинг. Даний сегмент включає в себе заходи з аналізу ринку, конкурентів і цільової аудиторії.

Дані сегменти покривають всі можливі процеси, пов'язані з роботою із клієнтами. Однак це призводить до проблеми вибору процесів, які повинні бути автоматизовані. Рішення автоматизувати всі процеси циклу є невірним, з огляду на те, що одночасне використання системи в усі процеси не дасть можливості співробітникам і організації перебудовуватись і призведе тільки до уповільнення роботи і як наслідок - збитків. Darrell K. Rigby і Dianne

Ledingham відзначають, що «мудрі компанії значно звужують можливості CRM системи, акуратно вибираючи лише сегмент CRM циклу і функції, які забезпечать найбільшу вигоду» [3,4].

Зрештою, слід пам’ятати, CRM-система не універсальне рішення для бізнесу. Якщо він заснований виключно на особистих зустрічах, а лояльність клієнтів вибудовується через довгострокові договори, то варто вибрати інші інструменти вибудовування взаємин між компанією і клієнтами. В інших випадках CRM − комплексний і багатофункціональний інструмент для управління відносинами «компанія» − «клієнт / зацікавлені сторони» і автоматизації бізнес-процесів.

Нижче будуть наведені відповідні CRM-системи. Проведений аналіз ключових моментів їх структури. Та наведений перелік відповідних якостей, які допоможуть сформувати враження про те, як має виглядати створена нами система обробки замовлень.

1.2 Система «Бітрікс24»

У 2016 році аналітичне агентство «Теглайн» за підтримки CRM-агентства «ARTW» опублікувало досить великий рейтинг платформ для створення CRM-систем, які використовуються в СНД (Рис. 1.1). Топ сформований на основі анкетування (проводилося з серпня 2014 по квітень 2016 року) [5].

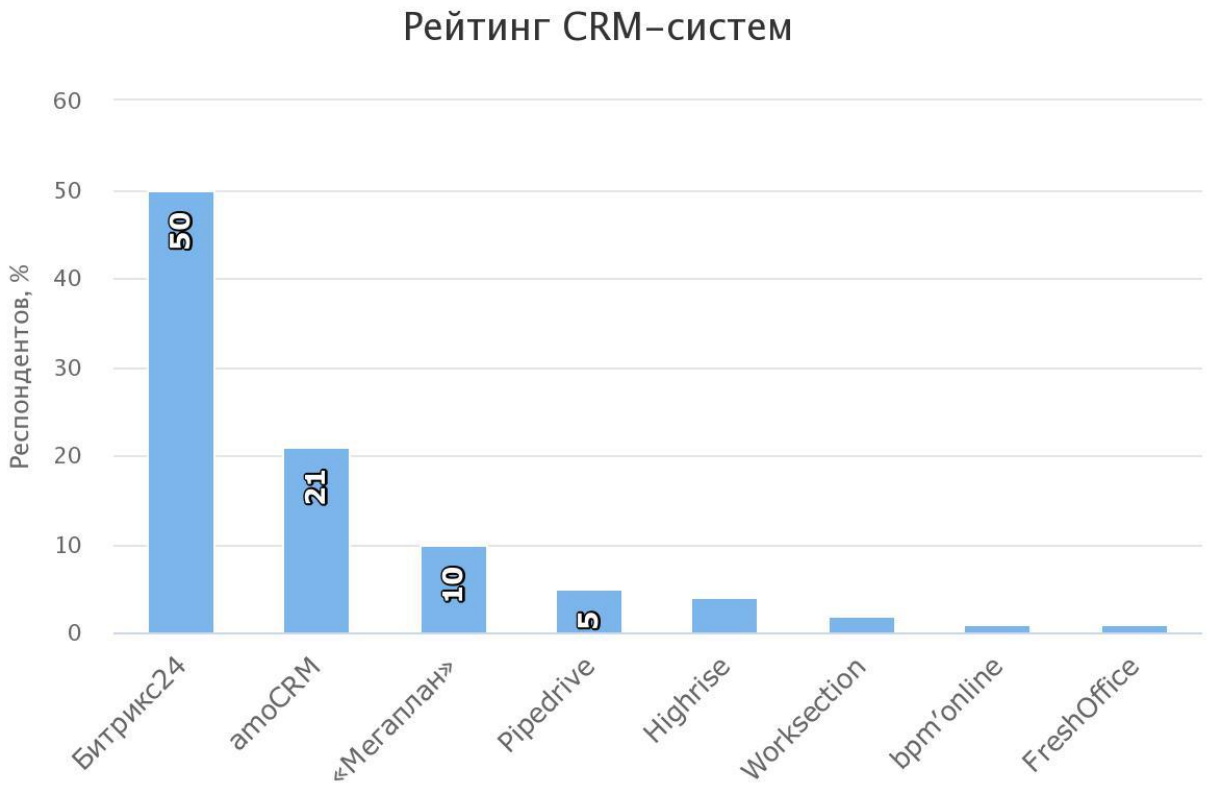


Рис. 1.1 Рейтинг CRM-систем

З наведеного вбачається те, що найбільш популярним виявився сервіс «Бітрікс24», на його частку припадає 50% респондентів. Друге місце займає система «amoCRM», її вибирають 21% опитуваних. «Мегаплан» займає третє місце з 10% голосів. Далі йдуть «Pipedrive» с 5% голосів на четвертому місці і «Highrise» с 4% − на п'ятому. На частку кожного з решти учасників рейтингу доводиться не більше 1-2%. До 10% респондентів впроваджують CRM-системи власної розробки.

Бітрікс24 − це сервіс, який об'єднує в собі величезний набір функціональних можливостей. Користувачі можуть випробувати в ньому систему управління завданнями, проєктами та документами, соціальну мережу, чат, відеодзвінки, сховище документів, календар, пошту, управління взаємовідносинами з клієнтами. Крім цього, система включає інструменти телефонії, HR, сервіс обліку робочого часу і багато іншого [6].

Платформа надає наступні можливості:

- Фіксувати всі дії, пов'язані з цим клієнтом (наприклад, дзвінки, повідомлення, зустрічі);
- Вести аналітику процесів в компанії використовуючи робочі звіти та рахунки.
- Керування завданнями та відстеження їх рішень;
- Жива стрічка змін даних всередині компанії, де також можна побачити думки співробітників.
- Можливість відправки сповіщень співробітникам;
- Доступ на мобільних пристроях;
- Інтеграція з соціальними мережами;

За відгуками Бітрікс24 відзначають дійсно більшу функціональність і зручність в освоєнні. Як мінуса можна відзначити рідкісні технічні проблеми і стабільність роботи [10].

1.3 Система «AmoCRM»

AmoCRM – системне програмне забезпечення для компанії, що надає можливість автоматизувати взаємодію з клієнтами.

Даною системою, якою можна користуватися без установки і з мобільних додатків. В системі можна заводити контакти, стежити за угодами, розподіляти зустрічі і виконувати різні завдання.

Спочатку amoCRM може нагадувати записну книжку, оскільки її інтерфейс істотно простіше аналогічних CRM-систем. Крім цього, amoCRM тісно інтегрується з сайтом і аналітикою: налаштувавши форму, встановивши її на сайті, кожне оформлення перетворюється в угоду, вона вже містить дані з Google.Analytics, далі ставляться нові завдання і тим самим вам доступна вся воронка онлайн-продажів [7].

Як і конкуренти, amoCRM пропонує наступні компоненти:

- Угоди і контакти − для бази клієнтів і управління продажами
- Воронка продажів − для звітів і збільшення продажів
- Завдання і нагадування
- Аналіз продажів -Моніторинг роботи співробітників і ефективності
- Мобільні застосунки -для Android та iOS
- Інтеграція з телефоном − облік дзвінків з АТС
- Інтеграція з сайтом − API, Web форми
- Надбудови − для збільшення функціональності
- Регулювати прав доступу

За відгуками amoCRM також відзначають зручність в освоєнні і приємний дизайн, так само і інтеграція з соц-мережами. Як мінус відзначають не стабільну роботу і неефективну підтримку [10].

1.4 Система «Мегаплан»

Мегаплан − це не тільки CRM система, але і система управління бізнесом у цілому. Неважливо, великий бізнес або малий і в чому його суть. Важливо, що бізнес працює, і ним треба керувати. «Мегаплан» допомагає роздавати доручення і контролювати співробітників. Отримувати звіти по виконаним завданням можна поза офісом. Сервіс зберігає історію відносин з клієнтами: записи дзвінків, листування, рахунки. Завжди можна повернутися назад і дізнатися, чому зірвалася угода і хто до цього причетний [8].

Керівнику:

- Перенесення завдань в «Мегаплан», призначивши відповідальних.
- Система повідомлень і нагадувань не дасть підлеглим забути доручення.
- Налагодження системи автоматичних звітів і прозора робота співробітників.

Співробітникові:

- У співробітника єдиний список завдань. Нові завдання ставляться і виконуються в «Мегаплані».
- Чіткі терміни виконання. Співробітник звітує про виконання завдання в коментарях.
- Менеджерам доступна актуальна інформація і статуси по клієнтам.
- «Мегаплан» нагадає про заплановані дзвінки і запросить на зустріч.
- Менеджери можуть виставити рахунок, а «Мегаплан» синхронізує дані з 1С.

Хоч «Мегаплан» менш поширений, відгуки про нього досить лояльні. Гарна базова функціональність, але все ж таки не досить гнучка. Не особливо зручний інтерфейс [10].

Таким чином, використання подібних сервісів оптимізують роботу і підвищують вигоду.

Bryan Foss і ін. поділяють вигоду на вигоду для підприємця, клієнта і співробітника [11]. Підприємницька вигода в свою чергу ділиться на збільшення прибутку і зниження витрат. Збільшення прибутку випливає з «поліпшення якості продажів і сервісів, що надаються і управління проблемами клієнтів (інциденти і запит) ...» [12]. Зниження витрат є причиною «збільшення загальної ефективності роботи клієнтських підрозділів, автоматизації процесу продажів, сервісних та маркетингових кампаній, скорочення витрат на менеджмент клієнтських процесів і автоматизація рутинної роботи».

Вигода клієнта полягає в можливості «отримати пропозицію в необхідній формі, коли необхідно і в необхідному обсязі, бути якісно обслужених, бути вчасно поінформованим, і вибрати спосіб надання сервісу».

Вигода співробітника полягає в «доброзичливості системи, можливості виконувати більшу кількість функцій, і як наслідок отримувати більшу заробітну функцію».

В результаті, не дивлячись на невеликі складності при впровадженні CRM систем, в разі успішного впровадження «CRM система дозволяє компаніям швидко збирати інформацію, виявляти найбільш цінних клієнтів і збільшувати клієнтську лояльність шляхом надання відповідних продуктів і сервісів. Вона також дозволяє скоротити витрати на обслуговування клієнтів і спрощує процес залучення клієнтів зі схожими інтересами» [11].

1.5 Порівняння наведених систем

Таблиця 1.1 – Набір інструментів для роботи з клієнтами

Назва системи / Компонент	**«Бітрікс24»**	**«AmoCRM»**	**«Мегаплан»**
Звіти	+	+	+
Управління доступом	+	+	+
Сповіщення співробітників	+	+	-
Сповіщення клієнтів	+	-	+

Таблиця 1.2 – Набір інструментів для спільної роботи

Назва системи / Компонент	**«Бітрікс24»**	**«AmoCRM»**	**«Мегаплан»**
Робота з електронною поштою	+	+	+
Обмін повідомленнями	+	-	+
Звіти	+	+	+

Кінець таблиці 1.2

Тайм менеджмент	+	+	+
Розклади	+	-	+
Список задач	+	+	+
Управління ресурсами	+	-	-

Таблиця 1.3 – Набір інструментів для управління проектом

Назва системи / Компонент	**«Бітрікс24»**	**«AmoCRM»**	**«Мегаплан»**
Діаграма Ганта	+	+	+
Розклади	+	+	+
Оцінка та облік витрат	+	+	+
Звіти	+	-	+
Шаблони проектів	+	+	+
Пріоритети	-	+	+
Повідомлення	+	+	-
Добавлення гостьових користувачів	+	-	-
Коментарі до завдань	+	+	+
Вкладення файлів до завдань	+	+	+
Фільтри	+	+	-
Повтори завдань	+	-	+
Делегування завдань	-	-	+
Налаштування доступу	+	-	+

Таблиці 1.1-1.3 демонструють порівняльні характеристики для кожної з наведених систем обробки замовлень, а саме: роботу з клієнтами, інструменти для спільної роботи та управління проектом.

Загальні недоліки вищеописаних CRM систем:

1. Вартість – подані системи є доступними лише за платною підпискою. Винятком є лише Бітрікс24, але його безплатна версія сильно обмежена у функціоналі;
2. Закритість – немає можливості модифікувати код цих програмних продуктів у сторону більш конкретної бізнес логіки;
3. Інтегрованість – складне і платне поєднання із сторонніми системами;
4. Постачальники послуг – дані програмні продукти розроблені, підтримуються та працюють на Російської Федерації. Вважається необхідним створити вітчизняний аналог;
5. Орієнтованість на компанію – більша увага приділяється правилам та процесам у середині компанії, аніж на клієнті. Малий бізнес, у складі якого не більше 10 людей не має потреби у контролі працівників. Головна ціль тут – зосередженість на роботі зі споживачами, підтримка зв’язку з ними та надання їм найзручніших умов співпраці;
6. Керування системою – подані програмні застосунки мають складний функціонал і передбачають наявність додаткового працівника, який би управляв та підтримував систему.

Спираючись на наведені системи, можливо перейняти відповідні якості з цих систем, та можливо викоренити недоліки під час написання власної системи обробки замовлень із урахуванням потреб малого бізнесу.

1.6 Формулювання вимог

Оскільки малий бізнес в основному полягає у продажі товарів та доставці їх клієнтам необхідності створити універсальний інструмент для обробки таких замовлень. Основними вимогами для такого програмного забезпечення є:

1) Універсальність – система повинна бути доступна на будь-якій операційній системі, що має доступ до мережі Інтернет і не залежати від операційної системи. Це правило стосується як і клієнтів, так і поставників. Саме тому підхід із Http-клієнтами є найбільш універсальними;
2) Доступність – кожен бажаючий може модифікувати програму. Недоліком тут є те, що доступність ця полягає у тому, що користувач зі сторони бізнесу повинен мати певні технічні навички для налаштування, встановлення, запуску цього програмного продукту;
3) Масштабованість – у кожній системі на сьогоднішній час цінується можливість розширятися завдяки збільшенню кількості апаратури, а не завдяки нарощенню потужностей. Це стає можливим завдяки використанню мікросервісного архітектурного підходу. Тобто отримаємо можливість, наприклад для великих навантажень використовувати по одному комп'ютеру на кожен сервіс. А при зворотній ситуації навпаки запустити систему на одному відносно потужному комп'ютері;
4) Інтегрованість – сьогодні маємо багато продуктів, які дають змогу спростити взаємодію з користувачами. Наприклад, для більш зручного спілкування можна використовувати популярні месенджери , що дозволяють за допомогою публічного API дають можливість створювати ботів. Ще одним популярним рішенням є сервіси по доставці замовлень. Ними можуть бути як і кур'єри таких служб як Glovo, так і поштові служби доставки;

5) Орієнтованість на клієнтів – головним об’єктом ведення бізнесу є потреби клієнта, а також прозорість у наданні послуг. Це означає, що кожен клієнт повинен мати змогу у будь-який момент часу дізнатися стан замовлення, історію замовлень, модифікувати створене замовлення тощо.

ВИСНОВКИ ДО РОЗДІЛУ 1

В першому розділі для огляду було наведені CRM-системи з їх відповідним функціоналом, а також позитивними та негативними якостями.

Здебільшого, перелічені програми так чи інакше стають в нагоді бізнесу. Деякий софт більше підходить до клієнтського аналізу, інший більше пристосований виробничих правовідносин роботодавця та працівника.

До того ж, наведені програми прямо чи опосередковано надають інформацію про товари чи послуги в автономному режимі, допомагають при виборі та замовлені, тощо.

Таким чином, спираючись на наведену інформацію створили вимоги, яким система повинна відповідати, та які проблеми вона може вирішити.

РОЗДІЛ 2 ОБГРУНТУВАННЯ ОБРАНОГО ІНСТРУМЕНТАРІЮ ДЛЯ СТВОРЕННЯ СИСТЕМИ ОБРОБКИ ЗАМОВЛЕНЬ

2.1 Мова програмування Java

Для виконанні даної роботи використовувалася мова високого рівня Java. Вона є високорівнева мова програмування і платформу обчислень, яка була вперше випущена Sun Microsystems в 1995 р. Була вибрана саме ця мова програмування тому, що платформа Java має велику кількість інструментів, бібліотек та фреймворків, що були створені для вирішення широкого спектру прикладних задач. Не менш важливим є її популярність – Java більше десяти років входить у п’ятірку найпопулярніших мов, а це у свою чергу означає наявність великої кількості програмістів, які б змогли підтримувати та розвивати продукти, написані на Java. Зі сторони програміста, окрім величезної кількості допоміжних інструментів, плюсом також буде чималої бази так званих «best practices», тобто правил та підходів, які полегшують розробку.

Java є об'єктно-орієнтованою мовою. Має підтримку всіх принципів ООП:

1) Абстракція – програміст працює з об’єктами, які мають певні властивості на відміну від інших імперативних мов, де найвищим рівнем є дані. Це дозволяє простіше оперувати великою кількістю даних, а також створювати гранульовані шари абстракції, де з кожним вищим рівнем концепція стає все більш загальною [13];
2) Поліморфізм – дозволяє описати контракт і в базуючись на певних умовах підставляти різні реалізації;
3) Успадкування – поведінку об’єктів можна змінювати або розширювати не міняючи їх реалізації.;
4) Інкапсуляція – програміст сам вирішує, які дані та у якому вигляді вони будуть доступні у інших точках програми.

Архітектура Java забезпечує кроссплатформенність і апаратну переносимість програм написаних на цій мові програмування. Саме завдяки наявності віртуальної машини програми без перекомпіляції можуть виконуватися на різних операційних системах − Linux, Mac OS, Andriod і т.д. Для кожної з платформ створена своя конкретна реалізація JVM (Java Virtual Machine), але кожна з них може виконувати один і той же код. Це є ключовою особливістю мови Java, її код спочатку транслюється в байт-код, який є зрозумілим для всіх віртуальних машин, навіть для попередніх версій. Тобто код написаний, а потім скомпільований на Java 11, зможе виконуватися на машині восьмої версії. Далі цей байт-код виконується JVM (рис. 2.1).

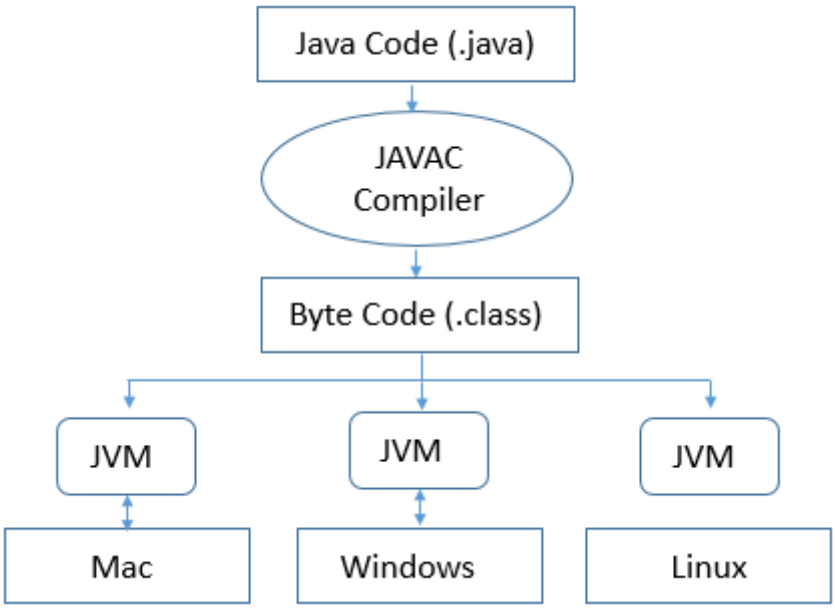


Рис. 2.1 Загальне уявлення роботи JVM

Java Virtual Machine є наріжним каменем платформи Java. Це компонент технології відповідальний за її незалежність від апаратного забезпечення та операційної системи, невеликий розмір скомпільованого коду та здатність захищати користувачів від більшості шкідливих програм. Віртуальна машина Java − це абстрактна обчислювальна машина, яка має

заданий набір інструкцій та маніпулює різними областями оперативної пам'яті та регістрами процесора під час роботи [13,14].

Java-програмісту не потрібно стежити за розподілом та звільненням пам'яті, так як збирач сміття автоматично керує пам'яттю.

Heap

JVM має Heap (купа), яка є спільною для всіх потоків в межах однієї віртуальної машини Java. Heap − це область даних часу виконання, з якої виділяється пам’ять для всіх екземплярів класу та масивів.

Heap створюється під час запуску віртуальної машини. Зберігання купи об’єктів відновлюється автоматичною системою управління сховищем (відомою як сміттєзбирач); об'єкти ніколи не звільняються явно. Віртуальна машина Java не передбачає жодного конкретного типу автоматичної системи управління пам’яттю, і техніка управління пам’яттю може бути обрана відповідно до системних вимог реалізатора. Купа може мати фіксований розмір або може бути розширена, якщо цього вимагає обчислення, і може бути зменшена, якщо більша купа стає непотрібною. Розмір купи можна задати параметрами при запуску програми. Пам'ять для купи не повинна бути суміжною [14].

Stack

Кожен потік JVM має приватний стек віртуальної машини Java, створений одночасно з потоком.

Стек Java віртуальної машини є аналогом стеку звичайної мови, такої як C: він містить локальні змінні та часткові результати та відіграє роль у виклику метода та поверненні результату із нього. Оскільки стеком віртуальної машини Java ніколи не маніпулюють безпосередньо, за винятком push та pop фреймів, блоки можуть бути виділені купою. Пам'ять для стеку віртуальної машини Java не повинна бути суцільною [14].

Збирач сміття

Збирач сміття (Garbage Collector) є невід'ємною частиною JVM і виконує лише дві функції – шукає сміття та очищує його. Сміттям вважаються об'єкти та дані, які більше не потрібні програмі.

При пошуку сміття використовуються два підходи. При першому кожен об'єкт, що знаходиться в купі має спеціальний лічильник, який означає кількість посилань на цей об'єкт. Переважно, після того, як об'єкт був використаний, то посилання на нього видаляється, зменшуючи таким чином значення лічильника. Коли ж воно досягає нуля, об'єкт позначається на подальшого видалення. Недоліком є те, що об'єкти можуть містити посилання один на одного у циклічній залежності, яку важко відстежити при такому підході. Для того, щоб це вирішити існує другий підхід, ідея якого полягає у проходженні із завідома «живого» об'єкту по всіх його посиланнях. Об'єкти, які є недосяжними позначаються на видалення.

Видалення об'єктів вимагає зупинки роботи програми. Очевидно, що чим ретельніша очистка, тим довша пауза. Відомі декілька підходів:

- Копіюючий збирач – створення двох рівноцінних областей, одна з яких пуста. У цю область будуть перенесені «живі» об'єкти. В кінці, перша область очищується і об'єднується із другою;
- «Mark-and-sweep» – використовується один сегмент пам'яті. Виконання програми призупиняється. Об'єкти видаляються одразу ж після позначення. Вільні ділянки пам'яті зберігаються як «вільному списку»;
- Збір сміття поколінь – використовуються різні підходи для різних поколінь купи;
- Послідовний збирач сміття – є один із найперших збирачів. В молодому поколінні очищення є швидким і частим, а для старого покоління – рідке та ретельне.

Тип збирача може бути заданий при запуску програми.

Maven

Для складання проєкту була використана система збирання Maven. Система збирання − це програмне забезпечення, що забезпечує автоматизацію складання проєкту.

Одна з головних особливостей даного фреймворка − декларативний опис проєкту. Це означає, що розробнику не потрібно попередньо налаштовувати систему − всі необхідні параметри, що виставлені за замовчуванням дозволяють компілювати та запускати програму. Зміни вносяться лише в при необхідності.

Ще одна перевага проєкту − гнучке управління залежностями. Maven завантажує у локальний репозиторій сторонні та власні бібліотеки. Таким чином завжди є доступ до необхідних бібліотек, навіть без доступу до мережі Інтернет. Задавши конкретну версію залежності можна бути впевненим, що вона ж буде завантажена на будь-який інший комп'ютер. Maven до цього ж дозволяє вирішувати конфліктні залежності – коли різні бібліотеки посилаються на спільну, але з різними версіями. Додаткові кроки життєвого циклу збірки можна виконувати за допомогою плагінів [15].

Основні цілі Maven:

- Полегшити процес складання
- Забезпечення єдиної системи збирання
- Надання якісної інформації про проєкт
- Заохочення більш ведення практики розвитку

2.2. Spring

При розробці був використаний Spring Framework. Це набір універсальних фреймворків з відкритим вихідним кодом для Java-платформи. Фреймворком називається бібліотека інструментів, що певним чином полегшують розробки програм, переважно приховуючи шаблонний код та дають змогу розробнику зосередитись на реалізації логіки. Він став широко

поширеним серед Java-розробників головним чином як альтернатива і заміна складної моделі Enterprise JavaBeans.

Ключовим елементом Spring є інфраструктурна підтримка на рівні додатків: Spring фокусується на структурі корпоративних додатків, щоб команди могли зосередитись на бізнес-логіці рівня додатків, без зайвих прив'язок до конкретних середовищ розгортання [16].

Мабуть, найважчою частиною пояснення технології Spring є точна її класифікація. Зазвичай Spring описується як легкий каркас для побудови додатків на Java, але з цим формулюванням пов'язані два моменти.

По-перше, Spring можна застосовувати для написання будь-якої програми на мові Java (наприклад, корпоративних (JEE) додатків), на відміну від багатьох інших каркасів і, зокрема, від каркаса Apache Struts, призначеного тільки для створення веб-додатків [17].

По-друге, легкий характер Spring насправді означає не кількість класів або розміри дистрибутива, а головний принцип усієї філософії Spring − мінімальний вплив.

Spring є легковажним каркасом в тому сенсі, що для використання всіх переваг ядра Spring вам доведеться внести лише мінімальні (якщо взагалі якісь) зміни в свій прикладної код, а якщо в якийсь момент ви вирішите не користуватися Spring, то і це зробити буде дуже просто. Поки мова йде тільки про ядро Spring – адже багато додаткових компонентів Spring, в тому числі для доступу до даних, вимагають більш тісної прив'язки до Spring Framework. Але користь від такої прив'язки цілком очевидна.

В Spring надається своя підтримка віддаленої обробки (через механізм виклику Spring HTTP), а також таких технологій як RMI, EJB, JMS, Hessian, Burlap1, JAX-RPC, JAX-WS і JAX-RS. Нижче розглянуті технології, що найчастіше вживаються.

- Механізм виклику Spring HTTP. Якщо додатки, які повинні взаємодіяти, засновані на Spring, то механізм виклику Spring HTTP надає

простий і ефективний спосіб звернення до служб, які надаються іншими додатками.

• Служби JMS в Spring. Служба обміну повідомленнями в Java (Java Messaging Service − JMS) надає ще один асинхронний і слабо зв'язний спосіб обміну повідомленнями між додатками.

• Веб-служби REST в Spring. Веб-служби REST, спеціально призначені для мережевого протоколу HTTP, відносяться до технології, що найчастіше вживається для віддаленої підтримки програм, а також призначених для користувача інтерфейсів веб-додатків, в яких застосовується технологія Ajax.

• Протокол AMQP в Spring. У родинному проєкті Spring AMQP надається типова для Spring абстракція протоколу AMQP (Advanced Message Queuing Protocol − вдосконалений протокол організації черг повідомлень) поряд з реалізацією системи обміну повідомленнями RabbltMQ. В даному проєкті надається великий ряд функціональних можливостей, таким його можливостям, як віддалена обробка через віддалений виклик процедур (RPC).

2.3 MySQL

MySQL − це система управління базами даних (СУБД), яка поширюється як вільне програмне забезпечення. СУБД – сукупність програм, що дають змогу створити базу даних (БД) і маніпулювати даними (вставляти, оновлювати, видаляти і вибирати). Система забезпечує надійність зберігання і цілісність даних, а також надає засоби для адміністрування БД [19].

Розробка та підтримка MySQL здійснюється корпорацією Oracle.

Основний принцип роботи:

• Система створює базу даних для зберігання інформації (її сортування, ідентифікації і т.п.).

• Клієнти (інші комп'ютери в мережі) подають запити до бази за допомогою специфічних для SQL команд.

• Серверний додаток обробляє запит і видає відповідь клієнту (повертає запитувані дані).

Для вибірки записів з бази даних розроблено спеціалізований мова − SQL (Structured Query Language - структурована мова запитів). За допомогою цієї мови можна створювати та модифікувати схеми та таблиці, маніпулювати даними та отримувати їх [20].

Служба баз даних MySQL використовує шифрування для забезпечення безпеки даних.

Шифрування в стані спокою: Служба баз даних MySQL використовує Block Volume для зберігання всіх даних. Обсяги блоків і резервного копіювання завжди шифруються.

Шифрування при передачі: Служба баз даних MySQL підтримує зашифровані з'єднання між клієнтами і сервером за допомогою Transport Layer Security (TLS). За замовчуванням додатка MySQL намагаються підключитися за допомогою шифрування. Тим не менш, використання шифрування для даного користувача може бути налаштоване як факультативне або обов'язкове [19].

Важливою особливістю MySQL є така архітектура підсистеми зберігання, в якій обробка запитів і інші серверні завдання відокремлені від зберігання та вилучення даних. Подібне розділення завдань дозволяє вибирати спосіб зберігання даних, а також налаштовувати продуктивність, ключові характеристики та ін. (Рис. 2.2)

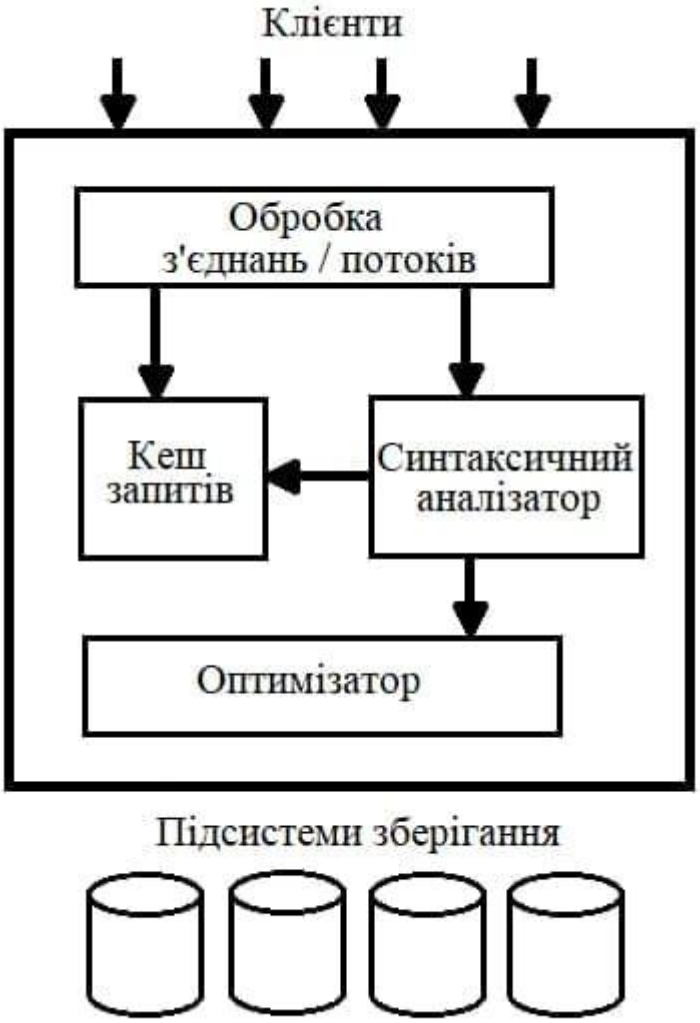


Рис. 2.2 Логічний вид архітектури сервера MySQL

На верхньому рівні розташовуються служби, які не є унікальними компонентами MySQL. Вони необхідні більшості мережевих клієнт-серверних інструментів або серверів: для обслуговування з'єднань, автентифікації, забезпечення безпеки і т. п.

Другий рівень значно цікавіший. Адже тут знаходиться велика частина «мізків» MySQL: код для обробки, аналізу, оптимізації та кешування запитів, а також всі вбудовані функції (наприклад, функції дати / часу, математичні і функції шифрування). Тут також розташовані інструменти, що використовувані в підсистемах зберігання, наприклад збережені процедури, тригери та відображення.

Третій рівень містить підсистеми зберігання даних. Вони відповідають за збереження даних в MySQL та їх вилучення. Подібно різних файлових систем, доступних для GNU / Linux, кожна така підсистема має як сильні, так і слабкі сторони. Сервер взаємодіє з ними через API підсистеми зберігання даних. Цей інтерфейс приховує відмінності між такими підсистемами і робить їх практично прозорими на рівні запитів. API містить пару десятків низькорівневих функцій, які виконують операції типу «розпочати транзакцію» або «витягти рядок з таким первинним ключем». Підсистеми зберігання не аналізують запити SQL і не взаємодіють один з одним, вони просто відповідають на вихідні від сервера запити.

Транзакція являє собою групу запитів SQL, оброблюваних атомарному, тобто як єдине ціле. Якщо підсистема бази даних може виконати всю групу запитів, вона робить це, але якщо який-небудь запит не може бути виконаний в результаті збою або з іншої причини, жоден запит групи не буде виконано. Все або нічого.

2.4 Liquibase

Liquibase – це бібліотека з відкритим кодом, до використовується для ініціалізації схем у базах даних, відслідковування змін у цих схемах та міграції невеликої кількості даних, наприклад для збереження тестових значень одразу ж після створення пустої схеми. Найбільшою перевагою даного інструменту перед іншими є те, що він не залежить від системи управління базами даних. Це досягається за допомогою загального формату опису схем – за допомогою XML, JSON, YAML форматів. Є можливість застосування нативних SQL скриптів, але тоді втрачається універсальність [18].

Найближчим конкурентом для liquibase є flyway. У таблиця 2.1 порівнюються ці інструменти.

Табл. 2.1 - Порівняння Liquibase та Flyway

Особливість	Liquibase	Flyway
Інкрементальна міграція	+	+
Формати для міграції	XML, SQL, YAML, JSON	SQL, Java
Стратегія визначення порядку виконання міграції	Ручна	Автоматична
Відкат міграції	+	-
Повторюване виконання	+	-
Виконання, що залежить від контексту виконання	+	-

Для того, щоб даний інструмент для міграції зміг працювати з іншою СУБД достатньо підключити та вказати у файлі конфігурацій відповідний JDBC-драйвер.

2.5 Swagger

Swagger – стандарт, який дозволяє описувати RESTful API [23]. На даний момент найактуальнішою версією є OpenAPI 3. Завдяки цьому інструменту можна описувати API-контракти. Опис контракту включає:

- Версія сервісу, що на даний момент обслуговує клієнта
- Точки входу та методи за якими до них можна доступитися
- Параметри запиту
- Опис JSON-тіл запиту та відповіді
- Опис правил валідації для параметрів, тіл та властивостей. Наприклад, чи є поле або параметр обов’язковим, регулярний вираз за яким перевіряється значення та інші параметри.

В даному проєкті використовується реалізація даного стандарту, що називається Springdoc [24]. Ця бібліотека автоматично генерує контракт під час компіляції, а також надає доступ до браузерної http-сторінки, де у зручному для користувача вигляді зображений вищезгаданий контракт. А також можна добавляти власну інформацію про кінцеву точку та про сервіс в цілому, і є можливість перезатирати згенеровані значення.

2.6 Kafka

Apache Kafka − розподілене програмний брокер повідомлень, проєкт з відкритим вихідним кодом, що розробляється в рамках фонду Apache. Написаний на мовах програмування Java і Scala.

Kafka − це розподілена система, що складається з серверів і клієнтів, які спілкуються через високопродуктивний мережевий протокол TCP. Він може бути розгорнутий на обладнанні без операційної системи, віртуальних машинах, так і в хмарних середовищах.

Сервери: Kafka працює як кластер з одного або декількох серверів, які можуть охоплювати кілька центрів обробки даних або хмарних областей. Деякі з цих серверів утворюють шар зберігання даних, званий брокерами. Кластер Kafka є високо масштабованим і стійким до збоїв: якщо який-небудь з його серверів виходить з ладу, інші сервери беруть на себе їх роботу, щоб забезпечити безперервну роботу без втрати даних.

Також є доволі зручним у використанні як і для високонавантажених систем, що слугують великим підприємствам, так і невеликих систем. У відкритому доступі є багато прикладів для роботи із даним інструментом і також детальна документація, завдяки які можна налаштувати брокер під будь-які потреби.

Клієнти: Можна писати розподілені додатки і мікрослужби, які зчитують, пишуть і обробляють потоки подій паралельно, і навіть у разі мережевих проблем або збоїв машини [21].

Для програм, що написані на мові Java є можливість використовувати бібліотеку–клієнта, яка надана та підтримується самими розробниками Kafka. А також для сервісів, що працюють із Spring Framework також створена обгортка для кращого інтегрування із фреймворком.

2.7 Транзакційний процес на основі Saga

Шаблон Saga − це архітектурний шаблон мікрослужб для реалізації транзакції, яка охоплює кілька служб. Так звана послідовність локальних транзакцій. Кожна служба в сазі виконує свою власну транзакцію і публікує подія. Інші служби слухають цю подію і виконують наступну локальну транзакцію. Якщо з якої-небудь причини одна транзакція не виконується, сага також виконує компенсуючі транзакції, щоб усунути вплив попередніх транзакцій.

Транзакція - це єдина одиниця логіки або роботи, яка іноді складається з декількох операцій. В рамках транзакції подія - це зміна стану, яке відбувається з сутністю, а команда інкапсулює всю інформацію, необхідну для виконання дії або запуску наступної події [22].

Транзакції повинні бути атомарними, постійними, ізольованими і стійкими (ACID). Абревіатура ACID розшифровується як atomicity, consistency, isolation і durabllity (атомарність, узгодженість, ізольованість і стійкість). Це тісно пов'язані критерії, яким повинна відповідати правильно функціонуюча система обробки транзакцій.

• Атомарність. Транзакція має функціонувати як єдина неподільна робоча одиниця таким чином, щоб вся вона була або виконана, або скасована. Для атомарних транзакцій не існує такого поняття, як часткове виконання: все або нічого.

• Узгодженість. База даних завжди повинна переходити з одного узгодженого стану в інше.

• Ізольованість. Результати транзакції зазвичай невидимі іншим транзакцій, поки вона не підтверджена.

• Стійкість. Після підтвердження внесення в ході транзакції зміни стають постійними. Це означає, що вони повинні бути записані так, щоб дані не загубилися при збої системи.

Модель «база даних на мікрослужбах» надає безліч переваг для архітектур мікрослужб. Інкапсуляція даних домену дозволяє кожній службі використовувати оптимальний тип сховища даних і схему, масштабувати своє сховище даних у міру необхідності і ізолювати від збоїв інших служб.

Шаблон Saga забезпечує управління транзакціями за допомогою послідовності локальних транзакцій. Локальна транзакція - це атомарна робоча діяльність, виконувана учасником Saga. Кожна локальна транзакція оновлює базу даних і публікує повідомлення або подію, щоб активувати наступну локальну транзакцію в Saga. У разі збою локальної транзакції Saga виконує ряд компенсуючих транзакцій, які скасовують зміни, внесені попередніми локальними транзакціями [22].

Існує два поширених підходу до реалізації Saga, хореографія та оркестрація.

Хореографія − це спосіб координації Saga, в якому учасники обмінюються подіями без централізованої точки управління. При використанні хореографії кожна локальна транзакція публікує події домену, які активують локальні транзакції в інших службах (Рис. 2.3).

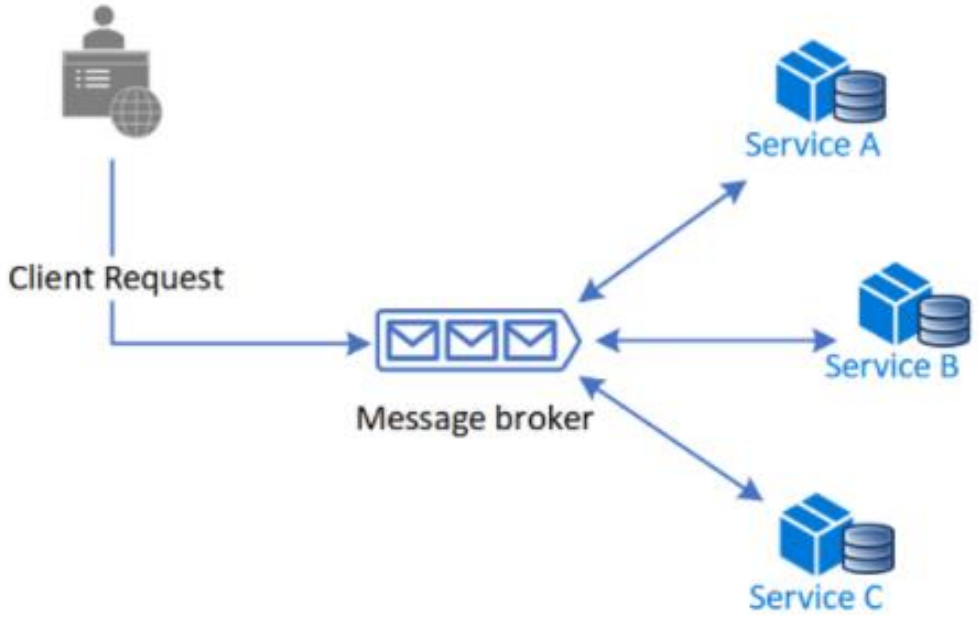


Рис. 2.3 Схема роботи Хореографії

Оркестрація - це спосіб координації Saga, коли централізований контролер повідомляє учасникам Saga, які локальні транзакції слід виконати. Saga оркестрація обробляє всі транзакції і повідомляє учасникам, яка операція повинна виконуватися на основі подій. Оркестрація виконує запити Saga, зберігає та інтерпретує стану кожного завдання, а також виконує відновлення після збою за допомогою компенсуючих транзакцій (Рис. 2.4).

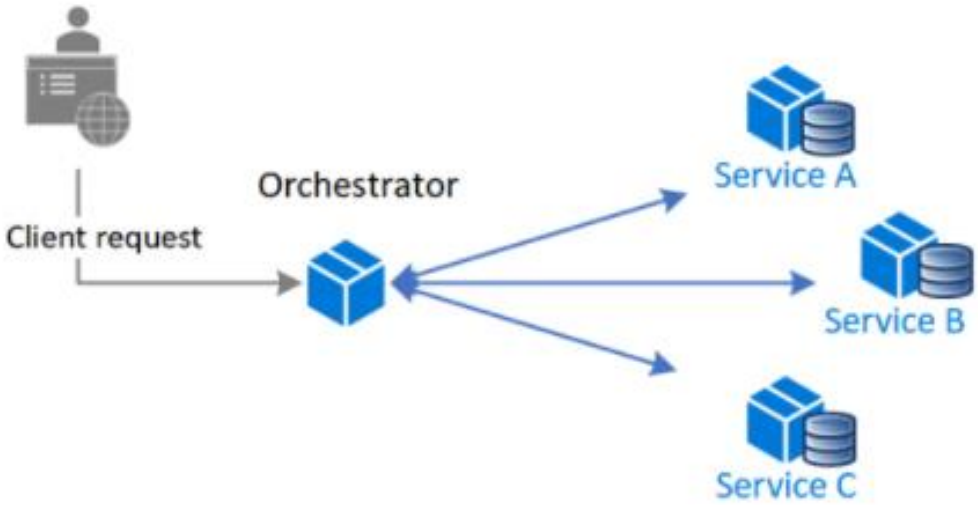


Рис. 2.4 Схема роботи Оркестрації

В даній системі будемо використовувати саме підхід з Оркестрацією, так як він дозволить більш інтуїтивним чином робити відкочування даних та скасування операцій. Оркестратором виступатиме сервіс, що відповідає за замовлення в загальному.

ВИСНОВКИ ДО РОЗДІЛУ 2

В наведеному вище розділі було стисло оглянуто інструментарій, який використовується для створення системи обробки замовлень.

Коротко схарактеризовано кожен із наведених інструментів, проаналізовані позитивні та негативні якості.

Важливо виокремити те, що обрані інструменти підходять та доповнюють один одного.

До того ж, обрані мови та фреймворки прямо відповідають сучасним вимогам програмістів надаючи якісне програмне забезпечення для розробки програмної системи.

РОЗДІЛ 3. ВНУТРІШНЯ БУДОВА ТА СТРУКТУРА ЗАПРОПОНОВАНОЇ СИСТЕМИ ОБРОБКИ ЗАМОВЛЕНЬ

Для того щоб перейти до безпосереднього розгляду запропонованої системи обробки замовлень слід розглянути її структурну складову, виділити взаємозв'язки процесів, та базові елементи, які приймають участь під час роботи зазначеної системи.

Для того щоб отримати замовлення слід пройти декілька етапів (Рис. 3.6). Система обробки замовлень розпочинає свою роботу із клієнтом (користувачем) з фактичної ідентифікації (Рис. 3.1).

Ідентифікація

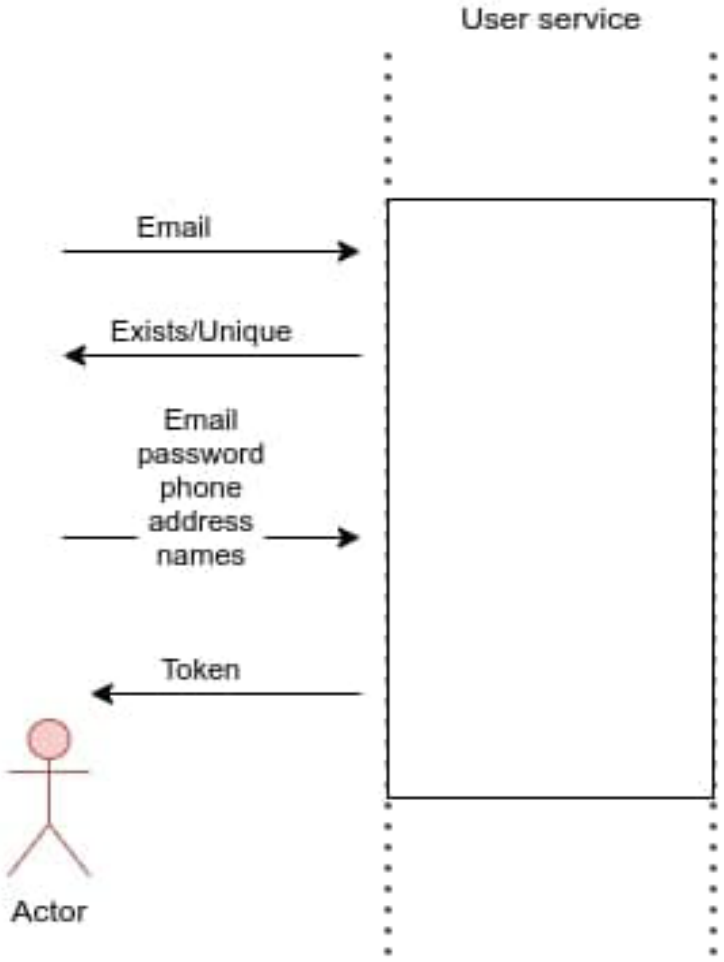


Рис 3.1 Процес створення облікового запису

На Рис 3.1 схематично зображений процес створення облікового запису для нового користувача.

Вихідними даними на цьому етапі будуть: електрона адреса користувача, його індивідуальний пароль, особистий номер телефону, його адреса та ім’я користувача.

Сервіс створення облікового запису акумулює цю інформація та вносить у єдину базу для подальшого звернення до такої інформації.

У свою чергу користувач після внесення відповідної інформації для проходження реєстрації його облікового запису отримує відповідний Token для подальшого використання такого запису.

Процес звернення до наявної у єдиній базі інформації зображений на Рис. 3.2 та Рис. 3.3:

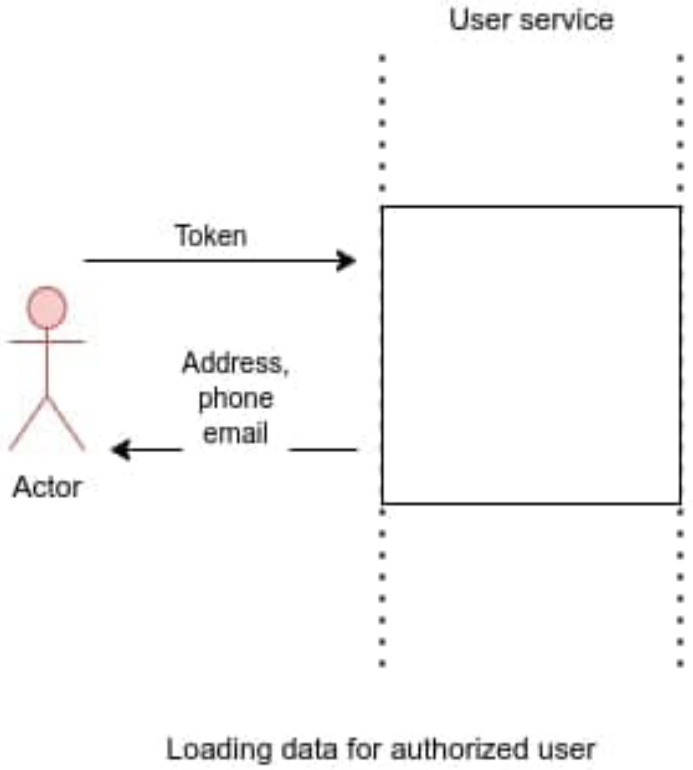


Рис 3.2 Процес віднаходження наявної у базі інформації

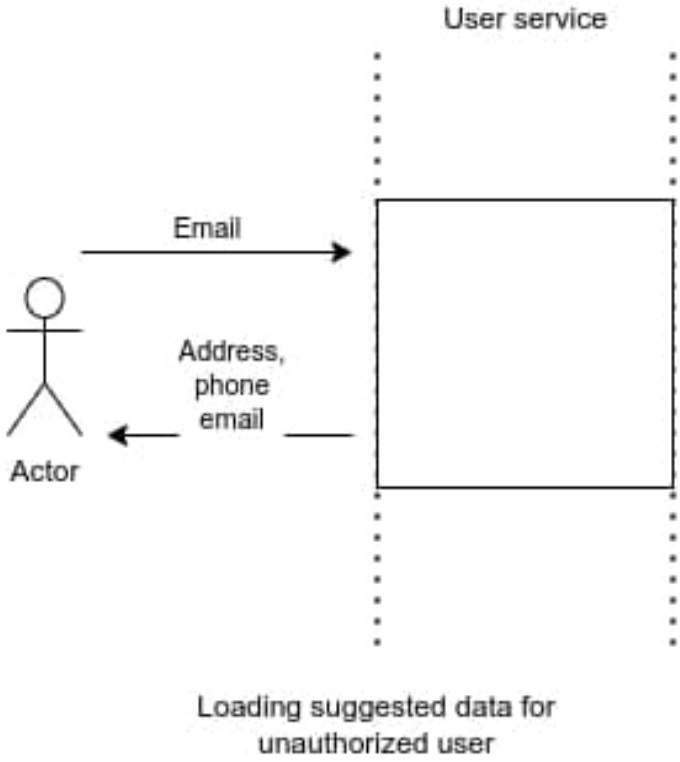


Рис 3.3 Зворотна відповідь для клієнта

У разі коли облікового запису або наявної у єдиній базі інформації немає, сервіс сповістить клієнта про це, та запропонує створити новий обліковий запис або відновити доступ до вже існуючого.

Після створення облікового запису користування ним буде здійснюватися за допомогою Enter-системи.

Для входу в особовий кабінет здебільшого використовується вказана клієнтом адреса електронної пошти та обраний індивідуальний пароль.

На Рис. 3.4 зображений процес Enter-системи:

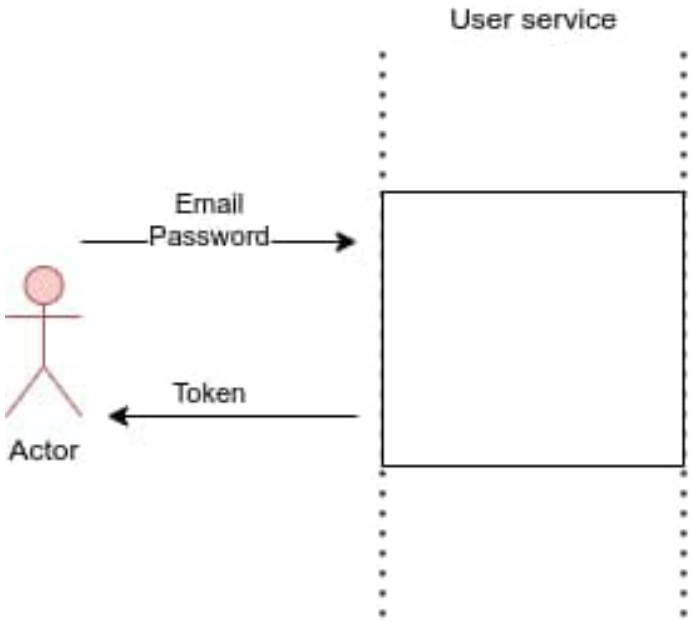


Рис 3.4 Процес Enter-системи

Вказаний етап ідентифікації грає важливу роль під час формування клієнтського заказу. Сервіс формування заказу звертається до наявної в обліковому записі інформації для різних потреб: надсилання інформативних повідомлень, здійснення дзвінків, перевірки адреси, тощо (рис. 3.5):

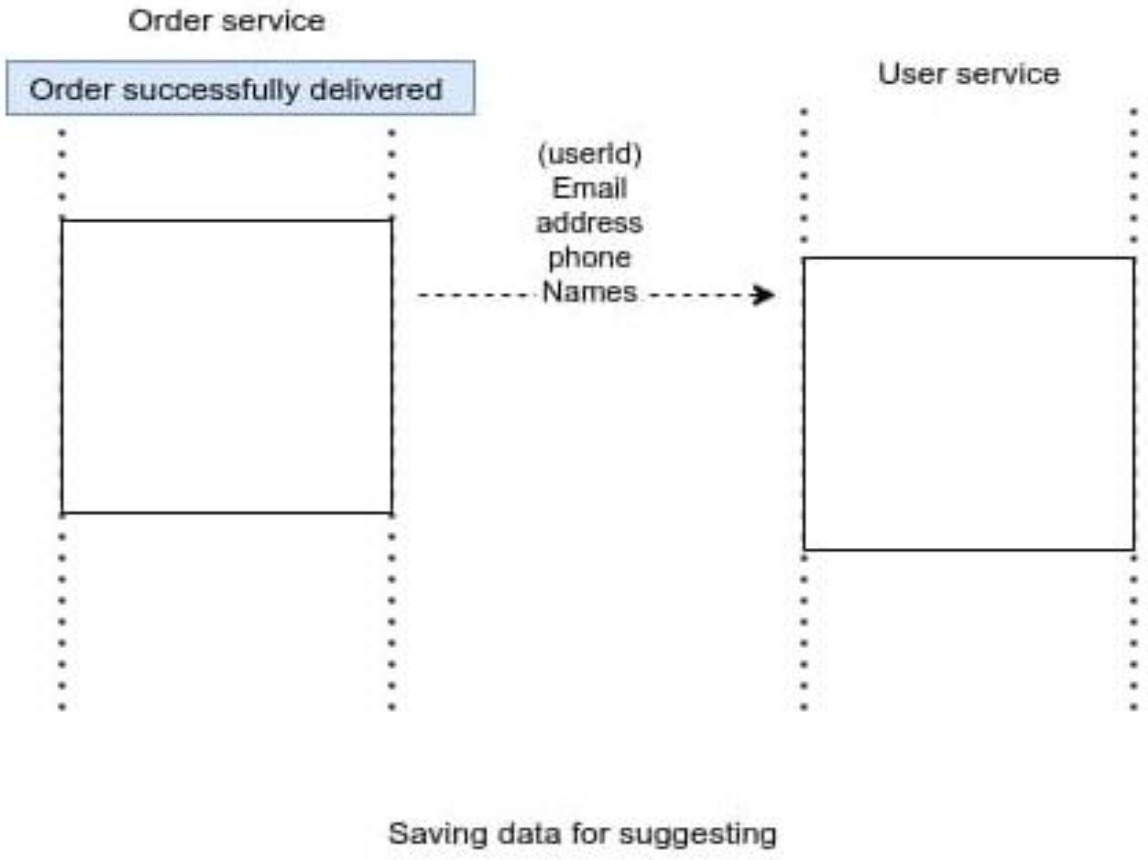


Рис 3.5 Спілкування сервісів для перевірки інформації

Повний процес роботи системи обробки замовлень

У Додатку 1 схематично визначено повний цикл системи обробки замовлень.

Запропонована схема розподілена не декілька етапів (за номерами від 1 до 5), кожний етап є невід'ємною складовою загального процесу, етапи виконують окремі процеси у середині кожного, проте усе разом за загальною структурою.

Нижче коротко наведемо пояснення для кожної окремої області:

«Область 1»: Після того як користувач вже пройшов авторизацію в системі він може перейти до вибору відповідних товарів.

Order Service (або сервіс заказу) одразу зв'язується із Delivery service (сервіс доставки) для отримання інформації стосовно того куди саме слід доставити обрані товари.

Звісно різниця для доставки обчислюється у одиницях виміру, і чим ближче буде доставити від точці створення товару, тим дешевше буде кінцевий price (ціна). Так само діаметрально навпаки.

«Область 2»: Надалі, під час створення замовлення сервіс заказу звертається до Goods service (сервіс товарів), таким чином сервіс заказу отримує інформацію про обрані товари із наявних у базі, ціну, знижки, тощо.

Після того як система підтверджує інформацію про товари, замовлення зберігається та присвоюється статус – «NEW».

«Область 3»: Наступний крок – оплата.

Сервіс заказу із вже сформованим заказом звертається до Сервісу оплати (Payment Service) для подальшого вирахування кінцевої ціни заказу, яка також окремо присвоюється із статусом – «NEW».

Далі клієнт отримує відповідний номер для сплати за своє замовлення.

«Область 4»: Із отриманням розрахункового номеру клієнт продовжує здійснювати кінцеву сплату шляхом надання відповідної інформації банківської карти.

Кінцевим результатом такої сплати буде відповідні повідомлення про «Сплачено», «Не сплачено», «Помилка», і тд.

«Область 5»: У разі проведення успішної оплати клієнту видається відповідне сповіщення шляхом звернення до Supplier Service (Сервісу підтвердження)

У свою чергу Сервіс підтвердження працює в парі разом із Сервісу доставки задля того аби клієнт міг отримати актуальну інформацію щодо статусу його заказу. Крім цього, інформацію щодо доставки, виникаючих помилок, або що.

Структурні блоки

При проходженні етапів виконання замовлення, кожен сервіс виробляє перевірку успішності його внутрішніх транзакцій, і обмін даних з іншими сервісами для підтвердження або скасування подальших етапів.

Зареєстрований замовлення має перелік даних для розрахунку і перевірки його статусу. Реквізити замовника, перелік обраних позицій, їх ціна і кількість (Рис. 3.6).

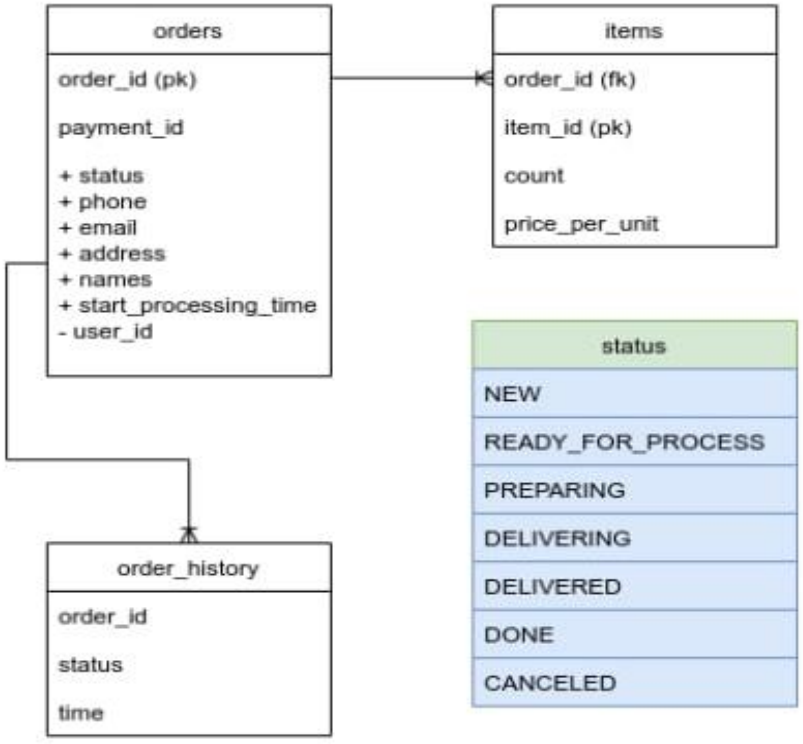


Рис. 3.6 Orders service

При створенні платежу обробляються сума до оплати, дані про карту. Формується payment_id. На кожному з етапів присуджується певний статус. Замовнику надається кілька можливих варіантів для оплати замовлення: передоплата або оплата після отримання замовлення (Рис. 3.7).

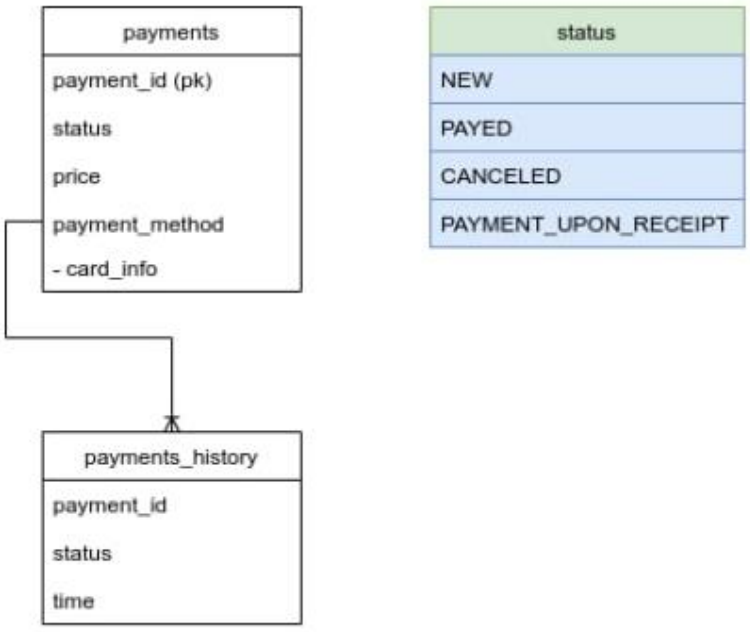


Рис. 3.7 Payments service

Для кожної позиції виконується перевірка ціни і кількості товару. Якщо є діюча знижка - виконується перерахунок ціни на оновлену (Рис. 3.8).

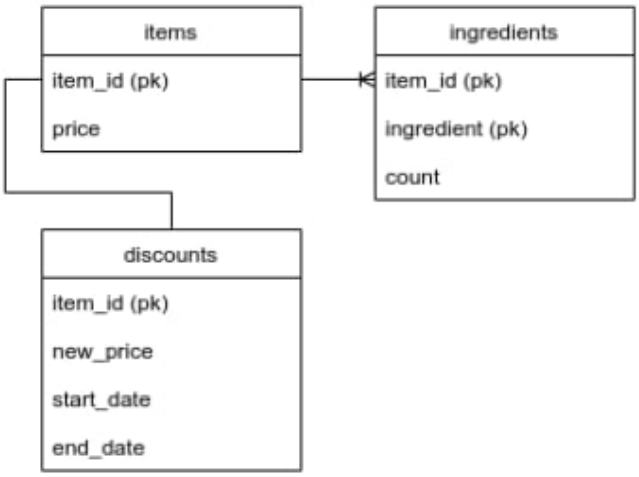


Рис. 3.8 Items service

При успішній оплаті постачальник інформує замовника про статус замовлення, часу на виконання і доставку. Даний алгоритм зображений у Додатку 2. Служба доставки отримує дані за замовленням і проводить розрахунок часу доставки (Рис. 3.9).

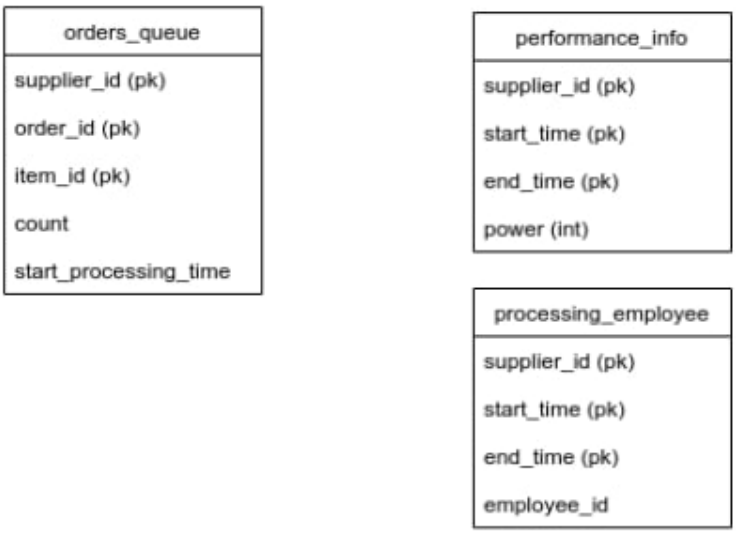


Рис. 3.9 Supplier service

Після установки адреси замовника та перевірки статусу готовності замовлення Служба доставки призначає працівника (кур'єра) і транспорту для відправки замовлення (Рис. 3.10).

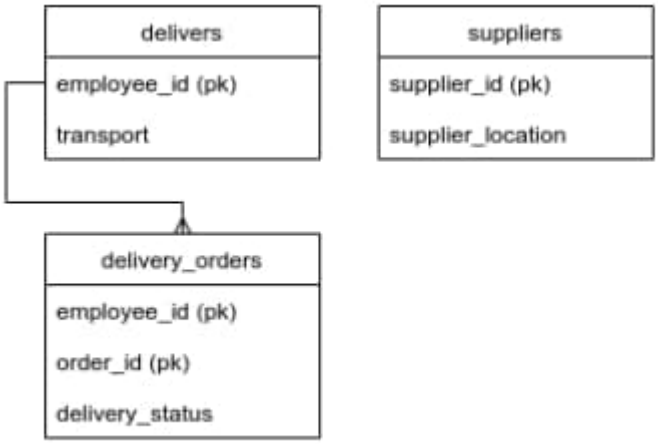


Рис. 3.10 Delivery service

За наслідками проведення усіх цих процесів, з урахуванням позитивних результатів на всіх окремих етапах проведення замовлення, клієнт – отримує замовлення, сервіс (працівник) – відповідну оплату праці.

Завдяки тому, система використовує систему повідомлень Kafka є можливість інтеграції із сторонніми системами такими як Telegram та Viber. Достатньо лише підписатися на відповідні теми брокера повідомлень.

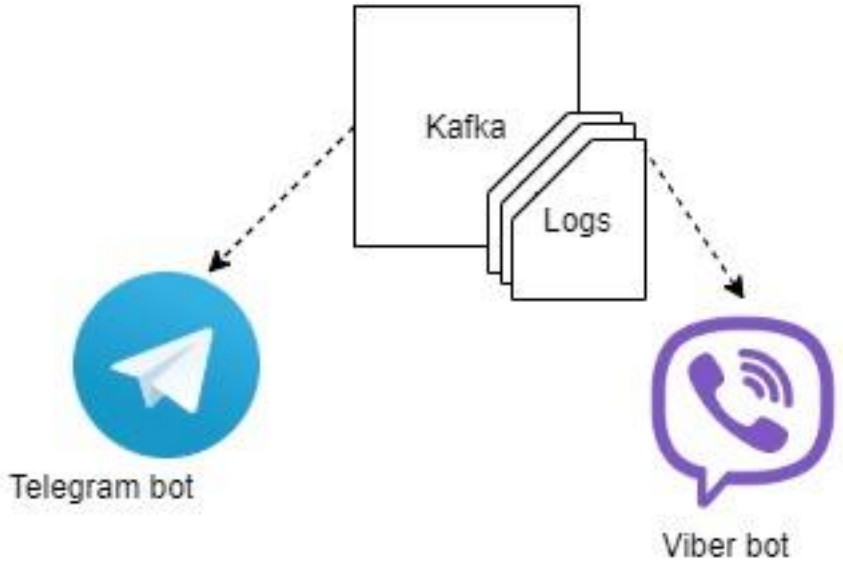


Рис. 3.11 Інтеграція із сторонніми застосунками

Даний прийом використовується у більшості сучасних систем, що взаємодіють з людьми. Схожий підхід реалізований банками, сервісами для продажі білетів та інші.

Структура схема сервісів, API-клієнтів та споживачів повідомлень зображено у Додатку 3.

ВИСНОВКИ ДО РОЗДІЛУ 3

В цьому розділі була представлена схематична будова запропонованої системи обробки замовлень.

Розглянуті внутрішні процеси, а також окремі його складові. Схематично наведений аналіз кожного структурного блоку який бере участь під час роботи запропонованій системі обробки замовлень.

Наведені схеми наддадуть змогу більш чітко зрозуміти програмний код, який буде наведений у наступному розділі.

РОЗДІЛ 4. ІНСТРУКЦІЯ API-КОРИСТУВАЧА

Кожен сервіс надає API-контракти (Рис 4.1). Вони є доступними для усіх користувачів за наступники посиланнями:

- http://<*HOST*>/public/users/swagger-ui.html – User API
- http://<*HOST*>/public/suppliers/swagger-ui.html – Supplier API
- http://<*HOST*>/public/deliveries/swagger-ui.html – Delivery API
- http://<*HOST*>/public/orders/swagger-ui.html – Orders API

, де <*HOST*> - це хост на за яким доступні сервіси. Наприклад, для локального запуску значення буде localhost:80XX.

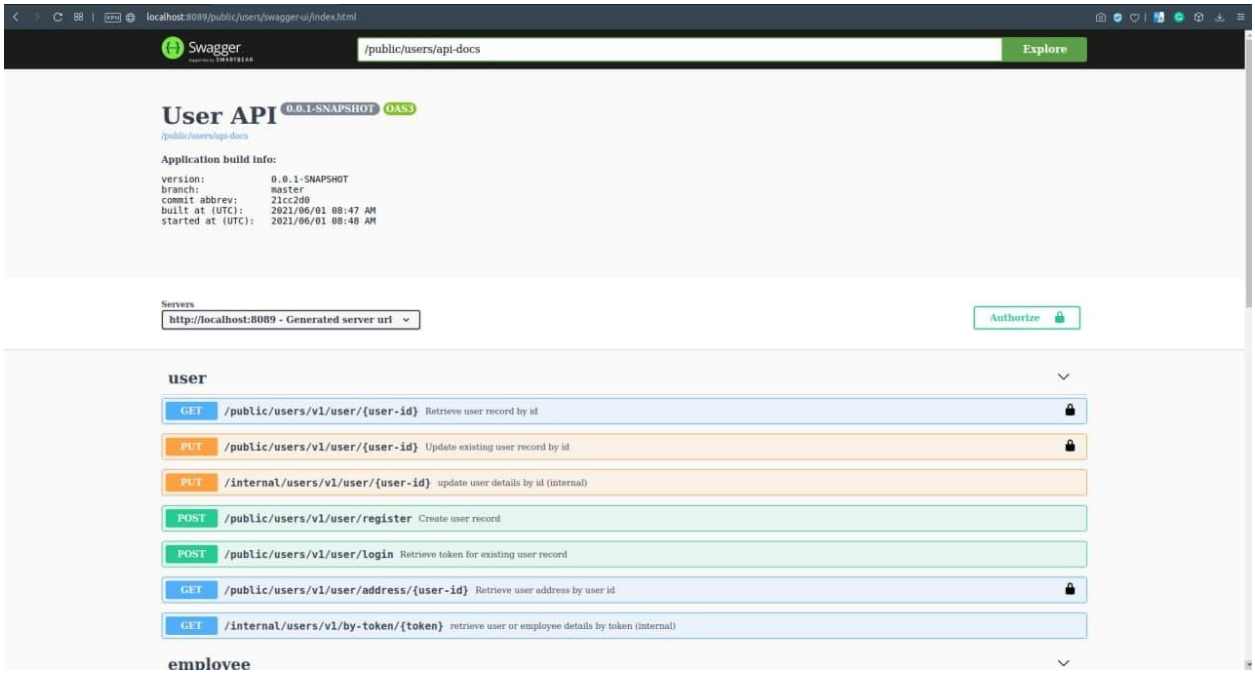


Рис. 4.1 Swagger-сторінка для User API сервісу

1) Авторизація користувача

Для того, щоб створити нового непривілейованого користувача потрібно зробити аналогічний запит (Рис. 4.2):

```
curl -X POST "http://localhost:8089/public/users/v1/user/register"
-H  "accept: */*"
-H  "Content-Type: application/json"
-d "{\"email\":\"user1@mail.com\",\"phone\":\"0123456789\",\"name\":\"User1\","
 + "\"password\":\"password123\",\"address\":\"City, street name\"}"
```

Рис. 4.2 Запит на створення нового користувача

У випадку коли вхідні дані задовольняють усі правила валідації отримаємо відповідь (Рис 4.3):


```
{
  "id": "438e3989-5f96-4503-b63e-66ab0f905d4f",
  "name": "User1",
  "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzUxMiJ9.eyJyb2xlIjpbIlVTRVIiXSwia
    XNzIjoiNDM4ZTM5ODktNWY5Ni00NTAzLWI2M2UtNjZhYjBmOTA1ZDRmIiwibmFtZSI6IlZhc
    3lhIiwiZXhwIjoxNjIyNTM4NjY3LCJpYXQiOjE2MjI1Mzc3Njd9.WCdeuGCH1mrmEb2ZyGmd
    QUnDjLTm0xen8VwuK45ubEYy1eyFPi3OSk3AjzTsE4iAg-g02ZlT82Y8FyDpzBUt-Q"
}
```


Рис 4.3 Успішна відповідь на створення нового користувача

Якщо було знайдено невірний формат даних або ж порушення консистентності даних отримаємо схожу відповідь зі статусом 400 (Bad request) (рис. 4.4):


```
{
  "timestamp": "2021-06-01T09:00:59.531+00:00",
  "status": 400,
  "message": "User with such email already exist",
  "path": "/public/users/v1/user/register",
  "source": "email"
}
```


Рис 4.4 Негативна відповідь на створення користувача

2) Отримання списку товарів

Для виконання даної операції авторизація не потрібна (рис. 4.5)

```
curl -X 'GET' \
  'http://localhost:8090/public/items/v1/item/items?page=0&size=10' \
  -H 'accept: */*'
```

Рис. 4.5 Запит на отримання списку товарів

Результат (рис. 4.6)


```
{
  "total_pages": 1,
  "total_elements": 1,
  "size": 10,
  "content": [
    {
      "id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
      "price": {
        "count": 150,
        "currency": "USD"
      },
      "discount": {
        "price": {
          "count": 145,
          "currency": "USD"
        },
        "start_time": "2021-06-01T12:00:00Z",
        "end_time": "2021-08-01T12:00:00Z"
      },
      "info": "Item#1",
      "deleted": false
    }
  ],
  "pageable": {
    "offset": 0,
    "sort": {
      "sorted": false,
      "unsorted": true,
      "empty": false
    },
    "page_number": 0,
    "page_size": 10,
    "paged": false,
    "unpaged": true
  },
  "first": true,
  "last": true,
  "number_of_elements": 1,
  "empty": false
}
```


Рис 4.6 Успішний результат на отримання списку товарів

3) Створення замовлення

Даний запит виконується лише непривілейованим авторизованим користувачем (рис. 4.7):

```
curl -X 'POST' \
  'http://localhost:8090/public/orders/v1/order' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzUxMiJ9.eyJyb2xlIjpbI
      lVTRVIiXSwiaXNzIjoiNDM4ZTM5ODktNWY5Ni00NTAzLWI2M2UtNjZhYjBmOTA1ZDRmIiw
      ibmFtZSI6IlZhc3lhIiwiZXhwIjoxNjIyNTM4NjY3LCJpYXQiOjE2MjI1Mzc3Njd9.WCde
      uGCH1mrmEb2ZyGmdQUnDjLTm0xen8VwuK45ubEYy1eyFPi3OSk3AjzTsE4iAg-g02ZlT82
      Y8FyDpzBUt-Q'
  -d '{
  "supplier_id": "594ff7a0-54a3-4402-9e0f-a370509d4645",
  "order_items": [
    {
      "item_id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
      "count": 4
    }
  ]
}'
```

Рис 4.7 Авторизований запит на створення замовлення

У результаті отримаємо унікальний ідентифікатор замовлення та загальну вартість (рис. 4.8):


```
{
  "order_id": "550f7ea8-099a-4dc5-8a96-b99896044f3d",
  "supplier_id": "594ff7a0-54a3-4402-9e0f-a370509d4645",
  "items": [
    {
      "item": {
        "id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
        "price": {
          "count": 150,
          "currency": "USD"
        },
        "discount": {
          "price": {
            "count": 145,
            "currency": "USD"
          },
          "start_time": "2021-06-01T12:00:00Z",
          "end_time": "2021-08-01T12:00:00Z"
        },
        "info": "Item#1",
        "deleted": false
      },
      "count": 4
    }
  ],
  "price": {
    "count": 580,
    "currency": "USD"
  },
  "order_status": "NEW",
  "creation_time": "2021-06-01T12:25:04.190Z"
}
```


Рис 4.8 Результат створення замовлення

4) Отримання списку замовлень користувача (рис 4.9)

```
curl -X 'GET' \
  'http://localhost:8090/public/orders/v1/order/user/
      438e3989-5f96-4503-b63e-66ab0f905d4f' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzUxMiJ9.eyJyb2xlIjpbI
      lVTRVIiXSwiaXNzIjoiNDM4ZTM5ODktNWY5Ni00NTAzLWI2M2UtNjZhYjBmOTA1ZDRmIiw
      ibmFtZSI6IlZhc3lhIiwiZXhwIjoxNjIyNTM4NjY3LCJpYXQiOjE2MjI1Mzc3Njd9.WCde
      uGCH1mrmEb2ZyGmdQUnDjLTm0xen8VwuK45ubEYy1eyFPi3OSk3AjzTsE4iAg-g02ZlT82
      Y8FyDpzBUt-Q'
```

Рис. 4.9 Авторизований запит на отримання списку замовлень

Результат (рис. 4.10)


```
{
  "user_id": "438e3989-5f96-4503-b63e-66ab0f905d4f",
  "orders": [
    {
      "order_id": "550f7ea8-099a-4dc5-8a96-b99896044f3d",
      "creation_time": "2021-06-01T12:25:04.190Z",
      "order_status": "NEW"
    },
        {
      "order_id": "a73abcf2-7b87-4b43-9869-5fbc579e8b5e",
      "creation_time": "2021-05-29T13:05:48.334Z",
      "order_status": "COMPLETED"
    }
    ]
}
```


Рис. 4.10 Результат на отримання списку замовлень користувача

5) Підписка на події

Сервіс оплати публікує події про зміну статусу щодо платежу у тему під назвою *payment_status* (рис. 4.11):


```
{
    "paymentId": "7960733233",
    "status": "NEW"
}
```


Рис. 4.11 Тіло повідомлення про зміну статусу платежа

За темою *delivery_status* публікуються події про зміну статусу в доставці товару (рис. 4.12):


```
{
    "deliveryId": "398977bd-6bd9-4f60-93cf-f6cd89f1c702",
    "status": "NEW"
}
```


Рис. 4.12 Тіло повідомлення про зміну статусу доставки

Загальна інформацію про замовлення публікуються у темі *order_status* (рис. 4.13):


```
{
    "orderId": "550f7ea8-099a-4dc5-8a96-b99896044f3d",
    "status": "READY_TO_PROCESS"
}
```


Рис. 4.13 Тіло повідомлення про зміну статусу замовлення

ВИСНОВКИ ДО РОЗДІЛУ 4

У цьому розділі була описана «Інструкція API-користувача» для реалізованої системи та продемонстрована її робота. Були пройдені кроки від реєстрації до створення замовлення. А також показані події, що надсилаються різними сервісами.

Дана інструкція націлена на те, щоб дозволити користувачі швидко розібратися з основним сценарієм використання представленого застосунку.

ВИСНОВКИ

Під час виконання дипломного проєкту були розглянуті наявні аналоги систем, що дають можливість взаємодіяти з клієнтами, в тому числі обробляти замовлення. На основі отриманої інформації була розроблена серверна частина програми, яка орієнтована саме на обробку та доставлення замовлень.

Перевагою такої системи є її масштабованість, можливість інтеграції з сторонніми сервісами та вузьконапрямленість на потреби малого бізнесу. Також беззаперечним плюсом є те, що система знаходиться у відкритому доступі.

Програмна система була перевірена на працездатність. Одержані результати дають підстави вважати, що завдання реалізоване, мета досягнута.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ:

1. Пэйн Э., «Руководство по CRM: путь к совершенствованию менеджмента клиентов» / Э.Пэйн − Гревцов-паблишер, 2007.
2. CRM-системи // [Режим доступу – електронне джерело: https://cutt.ly/bbDpMH1].
3. Darrell K. Rigby Dianne Ledingham, «CRM done right Harvard business review», 2004.
4. Frederick F. Reichheld, Phil Schefter, and Darrell K. Rigby, « Avoid the Four Perils of CRM», 2002 // [Режим доступу – електронне джерело: https://hbr.org/2002/02/avoid-the-four-perils-of-crm].
5. Рейтинг CRM-систем (управління відносинами з клієнтами) // [Режим доступу – електронне джерело: https://tagline.ru/crm-systems-rating/].
6. Офіційний сайт Bitrix24 // [Режим доступу – електронне джерело: https://www.bitrix24.ru/].
7. Офіційний сайт AmoCRM // [Режим доступу – електронне джерело: https://www.amocrm.ru/].
8. Офіційний сайт «Мегаплан» // [Режим доступу – електронне джерело: https://megaplan.ru/].
9.
10. Офіційний сайт // [Режим доступу – електронне джерело: https://startpack.ru/].
11. Bryan Foss, Merlin Stone and Yuksel Ekinci, «Journal of Database Marketing & Customer Strategy Management», 2008.
12. Вартан Х., «Рост по затухающей» / Общенациональный деловой журнал «Эксперт» № 13 (845), 2013 // [Режим доступу – електронне джерело: http://expert.ru/expert/2013/13/rost-po-zatuhayuschej/].
13. Documentation oracle // [Режим доступу – електронне джерело: https://docs.oracle.com/].

14. Tim Lindholm, Frank Yellin, Gilad Bracha, Alex Buckley, Daniel Smith, «The Java Virtual Machine Specification Java SE 16 Edition» // [Режим доступу – електронне джерело: https://docs.oracle.com/javase/specs/jvms/se16/jvms16.pdf].

15. Офіційний сайт Maven // [Режим доступу – електронне джерело: https://maven.apache.org/].

16. Козмина, Юлиана, Харроп, Роб, Шефер, Крис, Хо, Кларенс, К59 «Spring 5 для профессионалов»: Пер. с англ. - СПб.: ООО "Диалектика", 2019. - 1120 с. : ил. - Парал. тит. Англ // ISBN 978-5-907114-07-4 (рус.)

17. Introduction to Spring Framework // [Режим доступу – електронне джерело: https://docs.spring.io/springframework/docs/3.0.0.M4/reference/html/ch01.html].

18. Офіційний сайт // [Режим доступу – електронне джерело: https://www.liquibase.org].

19. Система управління базою даних // [Режим доступу – електронне джерело: https://bigenc.ru/technology_and_technique/text/3666497].

20. Офіційний сайт // [Режим доступу – електронне джерело: https://www.mysql.com/].

21. Офіційний сайт // [Режим доступу – електронне джерело: https://kafka.apache.org/].

22. Saga distributed transactions // [Режим доступу – електронне джерело: https://docs.microsoft.com/enus/azure/architecture/referencearchitectures/saga/saga].

23. Офіційний сайт // [Режим доступу – електронне джерело: https://swagger.io].

24. Офіційний сайт // [Режим доступу – електронне джерело: https://springdoc.org].

ДОДАТОК 1

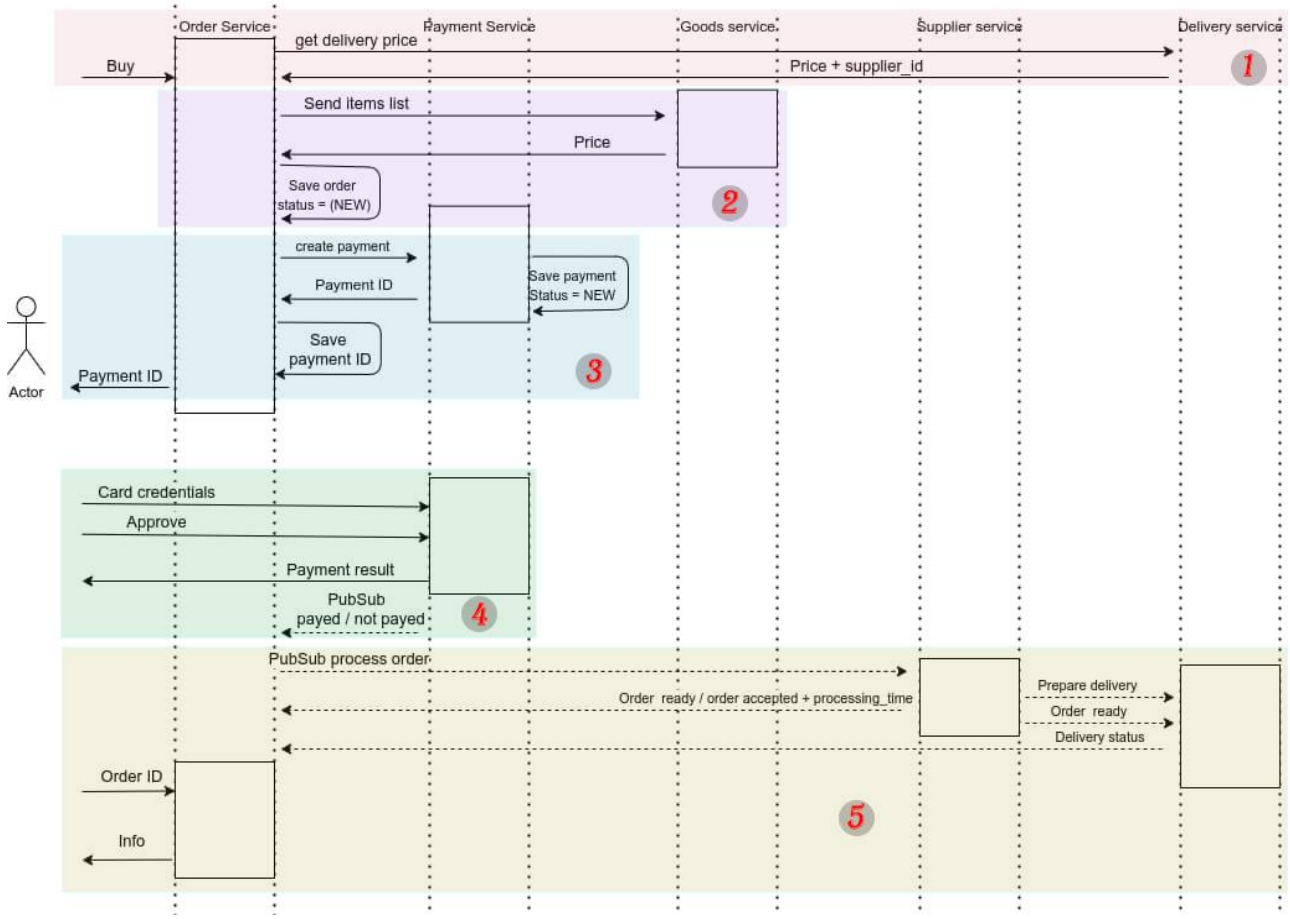
Система обробки замовлень

Просторово-часова діаграма

ІАЛЦ.467200 Д1

Листів 1

Київ – 2021 р.



					ІАЛЦ.467200.004 Д1	
Зм.	*Арк.*	*№ докум.*	*Підпис*	*Дата*		
Розробив		Гурняк А.М			**Система обробки замовлень** Додаток 1	*Літ.* / *Арк.* / *Аркушів*: 1 / 1
Перевірив		Сергієнко А.А.				*НТУУ «КПІ» ФІОТ*
Н. Контр.		Сімоненко В.П.				*Група ІВ-72*
Затвердив						

ДОДАТОК 2

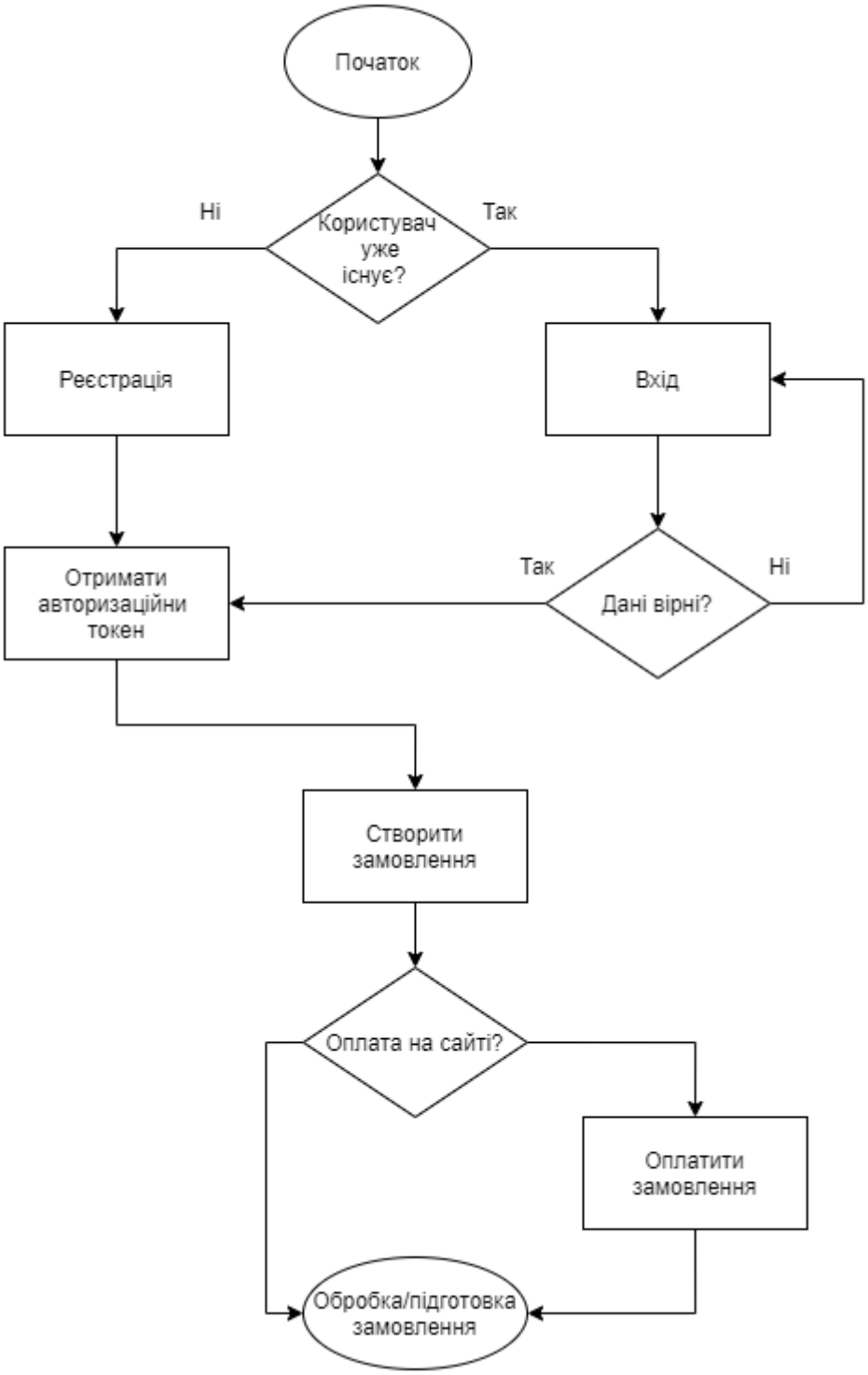
Система обробки замовлень

Схема алгоритму створення замовлення

ІАЛЦ.467200 Д2

Листів 1

Київ – 2021 р.



					ІАЛЦ.467200.005 Д2	
Зм.	*Арк.*	*№ докум.*	*Підпис*	*Дата*		
Розробив		Гурняк А.М			**Система обробки замовлень** Додаток 2	*Літ.* / *Арк.* 1 / *Аркушів* 1
Перевірив		Сергієнко А.А.				
						НТУУ «КПІ» ФІОТ
Н. Контр.		Сімоненко В.П.				*Група ІВ-72*
Затвердив						

ДОДАТОК З

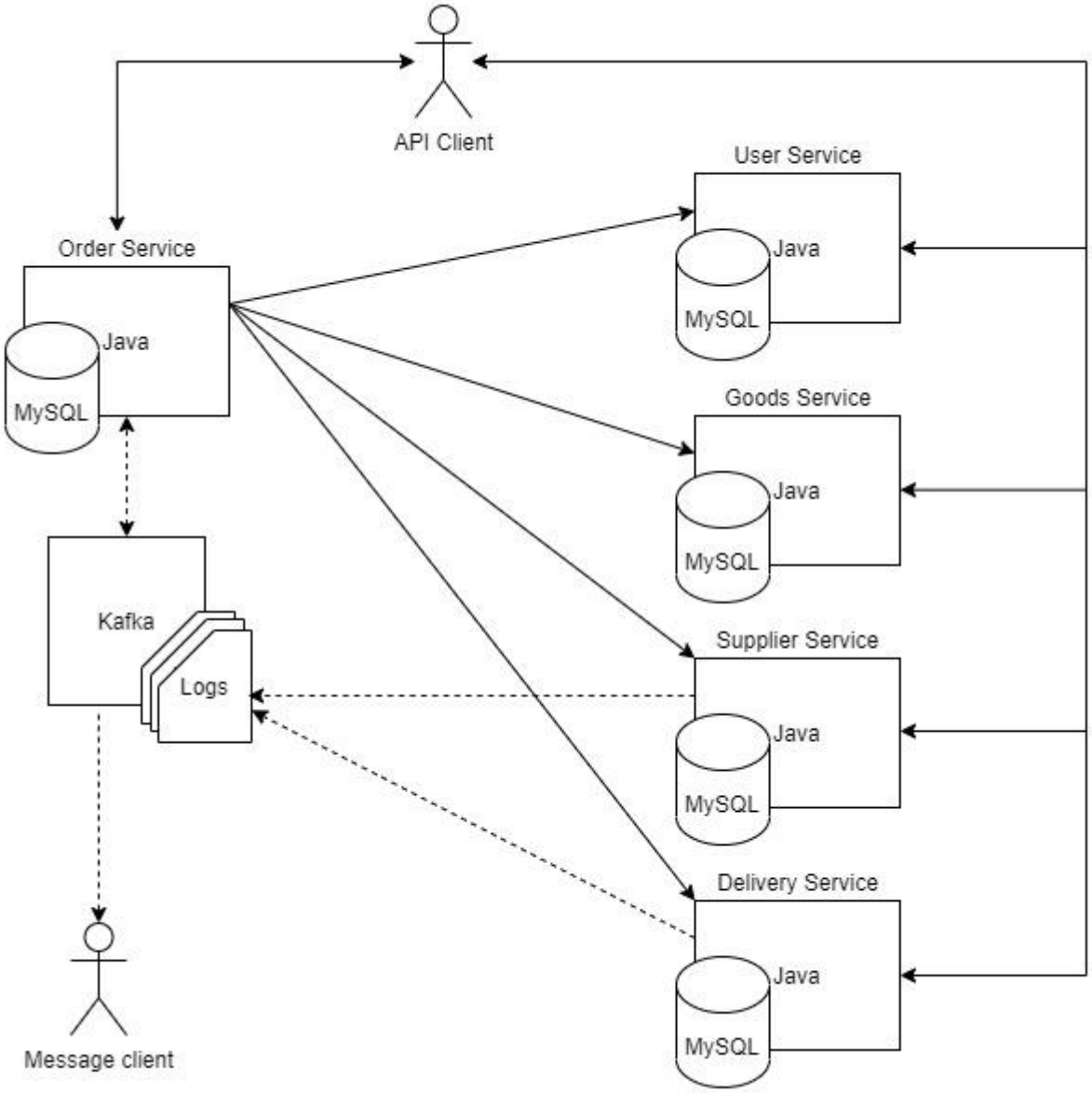
Система обробки замовлень

Структурна схема сервісів

ІАЛЦ.467200 ДЗ

Листів 1

Київ – 2021 р.



					ІАЛЦ.467200.006 Д3	
Зм.	*Арк.*	*№ докум.*	*Підпис*	*Дата*		
Розробив		Гурняк А.М			**Система обробки замовлень** Додаток 3	*Літ.* | *Арк.* 1 | *Аркушів* 1
Перевірив		Сергієнко А.А.				
						НТУУ «КПІ» ФІОТ
Н. Контр.		Сімоненко В.П.				*Група ІВ-72*
Затвердив						

ДОДАТОК 4

Система обробки замовлень

Лістинг програми

ІАЛЦ.467200 Д4

Листів 18

Київ – 2021 р.

```java
package com.kpi.hurniak.userapi.controller;

import com.kpi.hurniak.userapi.dto.AddressDto;
import com.kpi.hurniak.userapi.dto.JwtDto;
import com.kpi.hurniak.userapi.dto.user.UserCreateDto;
import com.kpi.hurniak.userapi.dto.user.UserDto;
import com.kpi.hurniak.userapi.dto.user.UserLoginDto;
import com.kpi.hurniak.userapi.dto.user.UserUpdateDto;
import com.kpi.hurniak.userapi.mapper.UserMapper;
import com.kpi.hurniak.userapi.openapi.user.LoginUserOperation;
import com.kpi.hurniak.userapi.openapi.user.RegisterUserOperation;
import com.kpi.hurniak.userapi.openapi.user.RetrieveUserAddressOperation;
import com.kpi.hurniak.userapi.openapi.user.RetrieveUserOperation;
import com.kpi.hurniak.userapi.openapi.user.UpdateUserOperation;
import com.kpi.hurniak.userapi.service.UserService;
import com.kpi.hurniak.userapi.validation.PermissionValidator;
import java.util.UUID;
import javax.validation.Valid;
import lombok.RequiredArgsConstructor;
import lombok.extern.log4j.Log4j2;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@Log4j2
@RestController
@RequestMapping(UserPublicController.ENDPOINT)
@RequiredArgsConstructor
public class UserPublicController {

  static final String ENDPOINT = "public/users/v1/user";

  private final UserService userService;
  private final UserMapper userMapper;
  private final PermissionValidator permissionValidator;

  @RegisterUserOperation
  @PostMapping("register")
  public JwtDto createUser(@Valid @RequestBody UserCreateDto userDto) {
    return userService.createUser(userDto);
  }

  @LoginUserOperation
  @PostMapping("login")
  public JwtDto loginUser(@Valid @RequestBody UserLoginDto userLoginDto) {
    return userService.loginUser(userLoginDto);
  }

  @RetrieveUserAddressOperation
  @GetMapping("address/{user-id}")
  public AddressDto suggestAddress(@PathVariable("user-id") UUID userId) {
    permissionValidator.validateAccessAllowed(userId);

    UserDto user = userService.findUser(userId);

    return userMapper.toAddress(user);
  }

  @RetrieveUserOperation
  @GetMapping("{user-id}")
```

```
  public UserDto retrieveUserDetails(@PathVariable("user-id") UUID userId) {
    permissionValidator.validateAccessAllowed(userId);

    return userService.findUser(userId);
  }

  @UpdateUserOperation
  @PutMapping("{user-id}")
  public UserDto updateUserDetails(
      @PathVariable("user-id") UUID userId,
      @Valid @RequestBody UserUpdateDto userUpdateDto
  ) {
    permissionValidator.validateAccessAllowed(userId);

    return userService.updateUser(userId, userUpdateDto);
  }
}
```

```
package com.kpi.hurniak.userapi.service;

import com.kpi.hurniak.userapi.dto.JwtDto;
import com.kpi.hurniak.userapi.dto.user.UserCreateDto;
import com.kpi.hurniak.userapi.dto.user.UserDto;
import com.kpi.hurniak.userapi.dto.user.UserLoginDto;
import com.kpi.hurniak.userapi.dto.user.UserUpdateDto;
import java.util.UUID;

public interface UserService {

  JwtDto createUser(UserCreateDto userCreateDto);

  JwtDto loginUser(UserLoginDto userLoginDto);

  UserDto findUser(UUID userId);

  UserDto updateUser(UUID userId, UserUpdateDto userUpdateDto);
}
```

```
package com.kpi.hurniak.userapi.service.impl;

import static com.kpi.hurniak.userapi.utils.MaskUtils.maskEmail;

import com.kpi.hurniak.userapi.config.exception.BusinessException;
import com.kpi.hurniak.userapi.dto.JwtDto;
import com.kpi.hurniak.userapi.dto.user.UserCreateDto;
import com.kpi.hurniak.userapi.dto.user.UserDto;
import com.kpi.hurniak.userapi.dto.user.UserLoginDto;
import com.kpi.hurniak.userapi.dto.user.UserUpdateDto;
import com.kpi.hurniak.userapi.entity.User;
import com.kpi.hurniak.userapi.mapper.UserMapper;
import com.kpi.hurniak.userapi.repository.UserRepository;
import com.kpi.hurniak.userapi.security.encryption.HashingComponent;
import com.kpi.hurniak.userapi.security.jwt.JwtComponent;
import com.kpi.hurniak.userapi.service.UserService;
import java.util.Optional;
import java.util.UUID;
import lombok.RequiredArgsConstructor;
import lombok.extern.log4j.Log4j2;
import org.apache.commons.lang3.StringUtils;
import org.springframework.http.HttpStatus;
import org.springframework.stereotype.Service;
```

```
@Log4j2
@Service
@RequiredArgsConstructor
public class UserServiceImpl implements UserService {

  private final UserRepository userRepository;
  private final UserMapper userMapper;
  private final HashingComponent hashingComponent;
  private final JwtComponent jwtComponent;

  @Override
  public JwtDto createUser(UserCreateDto userLoginDto) {
    checkNotExists(userLoginDto);

    var user = userMapper.from(userLoginDto);
    user.setPassword(hashingComponent.hash(user.getPassword()));

    var savedUser = userRepository.save(user);
    log.info("Successfully create user with id '{}'", savedUser.getId());

    return jwtComponent.createToken(savedUser);
  }

  private void checkNotExists(UserCreateDto userDto) {
    String email = userDto.getEmail();
    if (StringUtils.isNotBlank(email) && userRepository.existsByEmail(email))
{
      log.info("User with email '{}' already exists", maskEmail(email));
      throw new BusinessException(
          "User with such email already exist",
          "email",
          HttpStatus.BAD_REQUEST
      );
    }
  }

  @Override
  public JwtDto loginUser(UserLoginDto userLoginDto) {
    return findUser(userLoginDto)
        .filter(dbUser -> hashingComponent.match(userLoginDto.getPassword(),
dbUser.getPassword()))
        .map(jwtComponent::createToken)
        .orElseThrow(this::createInvalidCredentialsException);
  }

  private Optional<User> findUser(UserLoginDto userLoginDto) {
    String email = userLoginDto.getEmail();
    String phone = userLoginDto.getPhone();

    if (StringUtils.isAllBlank(email, phone)) {
      return Optional.empty();
    }

    if (StringUtils.isNotBlank(email)) {
      return userRepository.findByEmail(email);
    }
    return userRepository.findByPhone(phone);
  }

  @Override
  public UserDto findUser(UUID userId) {
    return userRepository.findById(userId)
        .map(userMapper::to)
        .orElseThrow(this::createUserNotFoundException);
```

```
  }

  @Override
  public UserDto updateUser(UUID userId, UserUpdateDto userUpdateDto) {
    var updatedUser = userRepository.findById(userId)
        .map(oldUser -> userMapper.merge(userUpdateDto, oldUser))
        .orElseThrow(this::createUserNotFoundException);

    var savedUser = userRepository.save(updatedUser);

    return userMapper.to(savedUser);
  }

  private BusinessException createInvalidCredentialsException() {
    return new BusinessException(
        "Credentials are invalid. Cannot authorize user",
        "credentials",
        HttpStatus.UNAUTHORIZED
    );
  }

  private BusinessException createUserNotFoundException() {
    return new BusinessException(
        "Cannot find user with such id",
        "user",
        HttpStatus.NOT_FOUND
    );
  }
}
package com.kpi.hurniak.userapi.entity;

import
com.kpi.hurniak.userapi.security.encryption.EncryptionAttributeConverter;
import java.time.Instant;
import java.util.UUID;
import javax.persistence.Column;
import javax.persistence.Convert;
import javax.persistence.Entity;
import javax.persistence.EntityListeners;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;
import lombok.Data;
import lombok.experimental.Accessors;
import org.hibernate.annotations.GenericGenerator;
import org.hibernate.annotations.Type;
import org.springframework.data.annotation.CreatedDate;
import org.springframework.data.annotation.LastModifiedDate;
import org.springframework.data.jpa.domain.support.AuditingEntityListener;

@Data
@Accessors(chain = true)
@Entity
@EntityListeners(AuditingEntityListener.class)
@Table(name = "users")
public class User {

  @Id
  @Type(type = "uuid-char")
  @GeneratedValue(strategy = GenerationType.AUTO)
  @GenericGenerator(name = "UUID", strategy =
"org.hibernate.id.UUIDGenerator")
  private UUID id;
```

```
  @Convert(converter = EncryptionAttributeConverter.class)
  private String name;

  @Convert(converter = EncryptionAttributeConverter.class)
  private String phone;

  @Convert(converter = EncryptionAttributeConverter.class)
  private String email;

  @Convert(converter = EncryptionAttributeConverter.class)
  private String address;

  private String password;

  @Column(name = "created_date", nullable = false, updatable = false)
  @CreatedDate
  private Instant createdDate;

  @Column(name = "modified_date")
  @LastModifiedDate
  private Instant modifiedDate;

}
```

```
package com.kpi.hurniak.userapi.security.encryption;

import com.kpi.hurniak.userapi.config.properties.SecurityProperties;
import
com.kpi.hurniak.userapi.config.properties.SecurityProperties.EncryptionProper
ties;
import java.util.Base64;
import javax.crypto.SecretKey;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import org.springframework.stereotype.Component;

@Component
public class EncryptionComponentImpl implements EncryptionComponent {

  private final IvParameterSpec iv;
  private final SecretKey secretKey;
  private final String encryptionAlgorithm;

  public EncryptionComponentImpl(SecurityProperties securityProperties) {
    EncryptionProperties encryptionProperties =
securityProperties.getEncryption();

    byte[] ivBytes =
Base64.getDecoder().decode(encryptionProperties.getIv());
    this.iv = new IvParameterSpec(ivBytes);

    byte[] keyBytes =
Base64.getDecoder().decode(encryptionProperties.getKey());
    this.secretKey = new SecretKeySpec(keyBytes,
encryptionProperties.getKeyAlgorithm());

    this.encryptionAlgorithm = encryptionProperties.getAlgorithm();
  }

  @Override
  public String encrypt(String plainValue) {
    return plainValue == null
        ? null
```

```
        : EncryptionUtils.encrypt(plainValue, encryptionAlgorithm, secretKey,
iv);
  }

  @Override
  public String decrypt(String encryptedValue) {
    return encryptedValue == null
        ? null
        : EncryptionUtils.decrypt(encryptedValue, encryptionAlgorithm,
secretKey, iv);
  }
}

package com.kpi.hurniak.userapi.security.encryption;

import com.kpi.hurniak.userapi.config.properties.SecurityProperties;
import
com.kpi.hurniak.userapi.config.properties.SecurityProperties.HashingPropertie
s;
import java.util.Base64;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.PBEKeySpec;
import lombok.SneakyThrows;
import org.apache.commons.lang3.StringUtils;
import org.springframework.stereotype.Component;

@Component
public class HashingComponentImpl implements HashingComponent {

  private final SecretKeyFactory secretKeyFactory;
  private final byte[] salt;
  private final int strength;
  private final int keyLength;

  public HashingComponentImpl(SecurityProperties securityProperties) throws
Exception {
    HashingProperties hashingProperties = securityProperties.getHashing();

    this.secretKeyFactory =
SecretKeyFactory.getInstance(hashingProperties.getAlgorithm());
    this.salt = Base64.getDecoder().decode(hashingProperties.getSalt());
    this.strength = hashingProperties.getStrength();
    this.keyLength = hashingProperties.getKeyLength();
  }

  @Override
  @SneakyThrows
  public String hash(String plainValue) {
    if (plainValue == null) {
      return null;
    }
    PBEKeySpec keySpec = new PBEKeySpec(plainValue.toCharArray(), salt,
strength, keyLength);
    byte[] hash = secretKeyFactory.generateSecret(keySpec).getEncoded();

    return Base64.getEncoder().encodeToString(hash);
  }

  @Override
  public boolean match(String plainValue, String encryptedValue) {
    return StringUtils.equals(hash(plainValue), encryptedValue);
  }
}
```

```
package com.kpi.hurniak.userapi.openapi.user;

import io.swagger.v3.oas.annotations.Operation;
import java.lang.annotation.Inherited;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;

@Inherited
@Retention(RetentionPolicy.RUNTIME)
@Operation(
    tags = "user",
    summary = "Create user record"
)
public @interface RegisterUserOperation {

}
```

```
package com.kpi.hurniak.userapi.openapi.user;

import com.kpi.hurniak.userapi.config.OpenApiConfig;
import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.security.SecurityRequirement;
import java.lang.annotation.Inherited;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;

@Inherited
@Retention(RetentionPolicy.RUNTIME)
@Operation(
    tags = "user",
    security = @SecurityRequirement(name = OpenApiConfig.SECURITY_SCHEME),
    summary = "Retrieve user record by id"
)
public @interface RetrieveUserOperation {

}
```

```
package com.kpi.hurniak.userapi.mapper;

import com.kpi.hurniak.userapi.dto.AddressDto;
import com.kpi.hurniak.userapi.dto.user.UserCreateDto;
import com.kpi.hurniak.userapi.dto.user.UserDto;
import com.kpi.hurniak.userapi.dto.user.UserUpdateDto;
import com.kpi.hurniak.userapi.entity.User;
import org.mapstruct.Mapper;
import org.mapstruct.Mapping;
import org.mapstruct.MappingTarget;

@Mapper(config = CommonMapper.class)
public interface UserMapper {

  @Mapping(target = "id", ignore = true)
  @Mapping(target = "createdDate", ignore = true)
  @Mapping(target = "modifiedDate", ignore = true)
  User from(UserCreateDto source);

  UserDto to(User source);

  AddressDto toAddress(UserDto source);

  @Mapping(target = "id", ignore = true)
  @Mapping(target = "password", ignore = true)
  @Mapping(target = "createdDate", ignore = true)
```

```
  @Mapping(target = "modifiedDate", ignore = true)
  User merge(UserUpdateDto userUpdateDto, @MappingTarget User oldUser);

}
```

```
package com.kpi.hurniak.userapi.dto.user;

import static
com.kpi.hurniak.userapi.validation.RegexContainer.ADDRESS_REGEX;
import static com.kpi.hurniak.userapi.validation.RegexContainer.NAME_REGEX;
import static com.kpi.hurniak.userapi.validation.RegexContainer.PHONE_REGEX;

import com.kpi.hurniak.userapi.validation.annotation.EmailOrPhone;
import javax.validation.constraints.Email;
import javax.validation.constraints.NotBlank;
import javax.validation.constraints.Pattern;
import javax.validation.constraints.Size;
import lombok.Data;
import lombok.experimental.Accessors;

@Data
@Accessors(chain = true)
@EmailOrPhone
public class UserCreateDto {

  @Email
  private String email;

  @Pattern(regexp = PHONE_REGEX)
  private String phone;

  @Pattern(regexp = NAME_REGEX)
  private String name;

  @NotBlank
  @Size(min = 8, max = 128)
  private String password;

  @Pattern(regexp = ADDRESS_REGEX)
  private String address;

}
```

```
package com.kpi.hurniak.userapi.validation.validator;

import com.kpi.hurniak.userapi.validation.annotation.EmailOrPhone;
import java.text.MessageFormat;
import javax.validation.ConstraintValidator;
import javax.validation.ConstraintValidatorContext;
import org.apache.commons.lang3.StringUtils;
import org.apache.commons.lang3.reflect.FieldUtils;

public class EmailOrPhoneValidator implements
ConstraintValidator<EmailOrPhone, Object> {

  @Override
  public boolean isValid(Object obj, ConstraintValidatorContext
constraintValidatorContext) {
    String phone = getValue(obj, "phone", String.class);
    String email = getValue(obj, "email", String.class);

    return !StringUtils.isAllBlank(phone, email);
  }
```

```
  @SuppressWarnings("unchecked")
  private static <T> T getValue(Object obj, String fieldName, Class<T>
resultType) {
    Object result;
    try {
      result = FieldUtils.getField(obj.getClass(), fieldName, true).get(obj);
    } catch (IllegalAccessException e) {
      e.printStackTrace();
      throw new IllegalArgumentException("Could not retrieve " + fieldName +
" field");
    }

    if (result == null) {
      return null;
    }
    if (result.getClass().isAssignableFrom(resultType)) {
      return (T) result;
    }

    throw new RuntimeException(MessageFormat.format("Cannot cast {} of type
{} to {}",
        fieldName, result.getClass(), resultType));
  }
}
```

```
package com.kpi.hurniak.userapi.security.jwt;

import com.auth0.jwt.JWT;
import com.auth0.jwt.JWTVerifier;
import com.auth0.jwt.algorithms.Algorithm;
import com.auth0.jwt.interfaces.Claim;
import com.auth0.jwt.interfaces.DecodedJWT;
import com.kpi.hurniak.userapi.config.properties.SecurityProperties;
import com.kpi.hurniak.userapi.dto.JwtDto;
import com.kpi.hurniak.userapi.entity.Employee;
import com.kpi.hurniak.userapi.entity.User;
import com.kpi.hurniak.userapi.mapper.JwtMapper;
import com.kpi.hurniak.userapi.provider.TimeProvider;
import com.kpi.hurniak.userapi.utils.UuidUtils;
import java.time.Duration;
import java.util.Collections;
import java.util.Date;
import java.util.List;
import java.util.Objects;
import java.util.UUID;
import org.apache.commons.lang3.StringUtils;
import org.springframework.stereotype.Component;

@Component
public class JwtComponentAuth0Impl implements JwtComponent {

  private static final String ROLE_CLAIM = "role";
  private static final String NAME_CLAIM = "name";
  private static final String TOKEN_PREFIX = "Bearer ";

  private final TimeProvider timeProvider;
  private final JwtMapper jwtMapper;

  private final Algorithm algorithm;
  private final Duration tokenLifeDuration;

  private final JWTVerifier verifier;

  public JwtComponentAuth0Impl(
```

```
      TimeProvider timeProvider,
      JwtMapper jwtMapper,
      SecurityProperties securityProperties
  ) {
    this.timeProvider = timeProvider;
    this.jwtMapper = jwtMapper;

    var jwtProperties = securityProperties.getJwt();

    this.algorithm = Algorithm.HMAC512(jwtProperties.getSecret());
    this.tokenLifeDuration = jwtProperties.getDuration();

    this.verifier = JWT.require(algorithm).build();
  }

  @Override
  public JwtDto createToken(User user) {
    String token = createToken(user.getId(), user.getName(), USER_ROLE);

    return jwtMapper.toJwt(user, token);
  }

  @Override
  public JwtDto createToken(Employee employee) {
    String token = createToken(employee.getId(), employee.getFirstName(),
EMPLOYEE_ROLE);

    return jwtMapper.toJwt(employee, token);
  }

  private String createToken(UUID id, String name, String... roles) {
    return JWT.create()
        .withIssuer(Objects.toString(id))
        .withIssuedAt(getIssuedAt())
        .withExpiresAt(getExpiredAt())
        .withArrayClaim(ROLE_CLAIM, roles)
        .withClaim(NAME_CLAIM, name)
        .sign(algorithm);
  }

  private Date getIssuedAt() {
    return Date.from(timeProvider.now());
  }

  private Date getExpiredAt() {
    return Date.from(
        timeProvider.now()
            .plus(tokenLifeDuration)
    );
  }

  @Override
  public void verifyToken(String token) {
    Objects.requireNonNull(token, "token must not be null");
    //TODO add correct exception handling (handle 500)
    verifier.verify(cleanup(token));
  }

  @Override
  public TokenClaims parseToken(String token) {
    Objects.requireNonNull(token, "token must not be null");
    DecodedJWT decodedJWT = JWT.decode(cleanup(token));

    TokenClaims tokenClaims = new TokenClaims();
```

```
    String id = decodedJWT.getIssuer();
    tokenClaims.setId(UuidUtils.parse(id));

    Claim rolesClaim = decodedJWT.getClaim(ROLE_CLAIM);
    List<String> roles = rolesClaim.asList(String.class);
    tokenClaims.setRoles(Objects.requireNonNullElse(roles,
Collections.emptyList()));

    Claim nameClaim = decodedJWT.getClaim(NAME_CLAIM);
    tokenClaims.setName(nameClaim.asString());

    return tokenClaims;
  }

  private String cleanup(String token) {
    return StringUtils.replaceIgnoreCase(token.trim(), TOKEN_PREFIX, "");
  }
}
```

```
server.port: 8089

spring:
  datasource:
    username: USER2
    password: 1234
    url: jdbc:mysql://localhost:3306/users?serverTimezone=UTC

  jpa:
#    hibernate.ddl-auto: update
    properties.hibernate.dialect: org.hibernate.dialect.MySQL5InnoDBDialect
    open-in-view: true

  jackson:
    property-naming-strategy: SNAKE_CASE
    default-property-inclusion: non_null

  kafka:
    bootstrap-servers: localhost:9092

springdoc:
  api-docs.path: /public/users/api-docs
  swagger-ui.path: /public/users/swagger-ui.html

security:
  jwt:
    secret: super_secret_string
    duration: PT15M
  encryption:
    iv: qd7EsZwXEYHb7W8gnpLRzg==
    key: Z3NzgT8LnZLDgwITEnLlZQ==
    key-algorithm: AES
    algorithm: AES/CBC/PKCS5Padding
  hashing:
    salt: 5Vj33bD3si2VryRUmxXX0A==
    strength: 65536
    key-length: 128
    algorithm: PBKDF2WithHmacSHA1
```

```
package com.kpi.hurniak.paymentapi.service.impl;

import
com.kpi.hurniak.paymentapi.component.event.publisher.PaymentEventsPublisher;
```

```
import com.kpi.hurniak.paymentapi.dto.PaymentCreateDto;
import com.kpi.hurniak.paymentapi.dto.internal.PaymentCompleteDto;
import com.kpi.hurniak.paymentapi.entity.Payment;
import com.kpi.hurniak.paymentapi.entity.PaymentHistory;
import com.kpi.hurniak.paymentapi.enums.PaymentMethod;
import com.kpi.hurniak.paymentapi.enums.PaymentStatus;
import com.kpi.hurniak.paymentapi.exception.BusinessException;
import com.kpi.hurniak.paymentapi.mapper.PaymentMapper;
import com.kpi.hurniak.paymentapi.repository.PaymentRepository;
import com.kpi.hurniak.paymentapi.service.CardService;
import com.kpi.hurniak.paymentapi.service.PaymentHistoryService;
import com.kpi.hurniak.paymentapi.service.PaymentService;
import java.util.List;
import lombok.RequiredArgsConstructor;
import lombok.extern.log4j.Log4j2;
import org.springframework.http.HttpStatus;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

@Log4j2
@Service
@RequiredArgsConstructor
public class PaymentServiceImpl implements PaymentService {

  private final PaymentRepository paymentRepository;
  private final PaymentHistoryService paymentHistoryService;
  private final CardService cardService;

  private final PaymentEventsPublisher paymentEventsPublisher;

  private final PaymentMapper paymentMapper;

  @Override
  @Transactional
  public Payment createNewPayment(PaymentCreateDto paymentCreateDto) {
    Payment newPayment = paymentMapper.from(paymentCreateDto);

    Payment savedPayment = paymentRepository.save(newPayment);
    log.info("Created payment with id '{}'", savedPayment.getId());

    paymentHistoryService.saveHistory(savedPayment);
    paymentEventsPublisher.sendPaymentStatusChangeEvent(savedPayment);

    return savedPayment;
  }

  @Override
  @Transactional
  public Payment completePayment(PaymentCompleteDto paymentCompleteDto) {
    Long paymentId = paymentCompleteDto.getPaymentId();
    Payment oldPayment = getPayment(paymentId);

    if (needReturnMoney(paymentCompleteDto, oldPayment)) {
      cardService.returnMoney(oldPayment);
    }

    Payment completedPayment = paymentMapper.from(paymentCompleteDto,
oldPayment);
    Payment savedPayment = paymentRepository.save(completedPayment);
    log.info("Complete payment '{}' with status '{}'", paymentId,
oldPayment.getStatus());
    paymentHistoryService.saveHistory(savedPayment);
    paymentEventsPublisher.sendPaymentStatusChangeEvent(savedPayment);
```

```
    return savedPayment;
  }

  private boolean needReturnMoney(PaymentCompleteDto paymentCompleteDto,
Payment oldPayment) {
    return oldPayment.getStatus() == PaymentStatus.PAYED
        && oldPayment.getPaymentMethod() == PaymentMethod.CARD
        && paymentCompleteDto.getStatus() == PaymentStatus.CANCELED;
  }

  @Override
  public Payment getPayment(long paymentId) {
    return paymentRepository.findById(paymentId)
        .orElseThrow(() -> createNotFoundException(paymentId));
  }

  @Override
  public List<PaymentHistory> getPaymentHistory(long paymentId) {
    checkPaymentExists(paymentId);

    return paymentHistoryService.getAllPaymentHistory(paymentId);
  }

  @Override
  public void checkPaymentExists(long paymentId) {
    if (!paymentRepository.existsById(paymentId)) {
      throw createNotFoundException(paymentId);
    }
  }

  private BusinessException createNotFoundException(long paymentId) {
    log.error("Could not find payment with id '{}'", paymentId);

    return new BusinessException(
        "Payment not found",
        HttpStatus.NOT_FOUND
    );
  }
}

package com.kpi.hurniak.paymentapi.entity;


import com.kpi.hurniak.paymentapi.enums.PaymentMethod;
import com.kpi.hurniak.paymentapi.enums.PaymentStatus;
import java.math.BigDecimal;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.EnumType;
import javax.persistence.Enumerated;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;
import lombok.Data;
import lombok.experimental.Accessors;

@Data
@Accessors(chain = true)
@Entity
@Table(name = "payments")
public class Payment {

  @Id
```

```
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  @Column(name = "payment_id")
  private Long id;

  @Column(name = "price")
  private BigDecimal price;

  @Enumerated(EnumType.STRING)
  @Column(name = "payment_method")
  private PaymentMethod paymentMethod;

  @Enumerated(EnumType.STRING)
  @Column(name = "status")
  private PaymentStatus status;

  @Column(name = "card_info")
  private String cardInfo;
}
```

```
package com.kpi.hurniak.supplierapi.service.impl;

import com.kpi.hurniak.supplierapi.component.PriceComponent;
import com.kpi.hurniak.supplierapi.config.exception.BusinessException;
import com.kpi.hurniak.supplierapi.dto.order.OrderCreateDto;
import
com.kpi.hurniak.supplierapi.dto.order.OrderCreateDto.OrderItemCreateDto;
import com.kpi.hurniak.supplierapi.dto.order.OrderDto;
import com.kpi.hurniak.supplierapi.dto.order.OrderPageableListDto;
import com.kpi.hurniak.supplierapi.entity.order.Order;
import com.kpi.hurniak.supplierapi.entity.order.OrderItem;
import com.kpi.hurniak.supplierapi.entity.order.OrderItem.OrderItemId;
import com.kpi.hurniak.supplierapi.enums.OrderStatus;
import com.kpi.hurniak.supplierapi.mapper.OrderMapper;
import com.kpi.hurniak.supplierapi.repository.OrderItemRepository;
import com.kpi.hurniak.supplierapi.repository.OrderRepository;
import com.kpi.hurniak.supplierapi.service.ItemService;
import com.kpi.hurniak.supplierapi.service.OrderService;
import com.kpi.hurniak.supplierapi.service.SupplierService;
import java.util.List;
import java.util.Set;
import java.util.UUID;
import java.util.stream.Collectors;
import javax.transaction.Transactional;
import lombok.RequiredArgsConstructor;
import lombok.extern.log4j.Log4j2;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Pageable;
import org.springframework.data.domain.Sort;
import org.springframework.data.domain.Sort.Direction;
import org.springframework.http.HttpStatus;
import org.springframework.stereotype.Service;

@Log4j2
@Service
@RequiredArgsConstructor
public class OrderServiceImpl implements OrderService {

  private final OrderRepository orderRepository;
  private final OrderItemRepository orderItemRepository;
  private final OrderMapper orderMapper;
  private final PriceComponent priceComponent;

  private final SupplierService supplierService;
  private final ItemService itemService;
```

```
  @Override
  @Transactional
  public OrderDto createOrder(OrderCreateDto orderCreateDto) {
    var supplier =
supplierService.getSupplierById(orderCreateDto.getSupplierId());
    Set<OrderItem> items = orderCreateDto.getOrderItems()
        .stream()
        .map(this::findOrderItem)
        .collect(Collectors.toSet());

    var newOrder = orderMapper.to(orderCreateDto);
    newOrder.setItems(items);
    newOrder.setSupplier(supplier);

    var savedOrder = orderRepository.save(newOrder);

    //TODO beautify
    items.forEach(item -> {
      item.setOrder(savedOrder);
      item.setId(
          new
OrderItemId().setOrderId(savedOrder.getId()).setItemId(item.getItem().getId()
));
    });
    orderItemRepository.saveAll(items);
    log.info("Successfully saved order with id '{}'", savedOrder.getId());

    return orderMapper.from(savedOrder,
priceComponent.calculatePrice(savedOrder));
  }

  private OrderItem findOrderItem(OrderItemCreateDto orderItemCreateDto) {
    var item = itemService.getItemById(orderItemCreateDto.getItemId());

    return orderMapper.to(orderItemCreateDto, item);
  }

  @Override
  public OrderDto getOrderDetails(UUID orderId) {
    var order = orderRepository.findById(orderId)
        .orElseThrow(() -> createOrderNotExistsException(orderId));

    var price = priceComponent.calculatePrice(order);

    return orderMapper.from(order, price);
  }

  @Override
  @Transactional
  public void updateOrderStatus(UUID orderId, OrderStatus orderStatus) {
    verifyOrderExists(orderId);

    orderRepository.updateOrderStatus(orderId, orderStatus);
    //TODO publish event

    log.info("Successfully changes status of order with id '{}'", orderId);
  }

  @Override
  public OrderPageableListDto getOrdersByStatus(
      UUID supplierId, OrderStatus orderStatus, int page, int size
  ) {
```

```
    PageRequest pageRequest = PageRequest.of(page, size, Direction.ASC,
"creationTime");
    List<Order> orders =
orderRepository.findPageableBySupplierIdAndOrderStatus(
        supplierId, orderStatus, pageRequest);

    List<OrderDto> orderDtos = orders.stream()
        .map(this::toOrderDto)
        .collect(Collectors.toList());

    return new OrderPageableListDto()
        .setOrders(orderDtos)
        .setPageNumber(page)
        .setPageSize(size);
  }

  private OrderDto toOrderDto(Order order) {
    var price = priceComponent.calculatePrice(order);

    return orderMapper.from(order, price);
  }

  void verifyOrderExists(UUID orderId) {
    if (!orderRepository.existsById(orderId)) {
      throw createOrderNotExistsException(orderId);
    }
  }

  private BusinessException createOrderNotExistsException(UUID orderId) {
    log.error("Order with id '{}' does not exist", orderId);

    return new BusinessException(
        "Order with such id does not exist",
        "order",
        HttpStatus.NOT_FOUND
    );
  }
}
package com.kpi.hurniak.supplierapi.entity.order;

import com.kpi.hurniak.supplierapi.entity.supplier.Supplier;
import com.kpi.hurniak.supplierapi.enums.OrderStatus;
import java.time.LocalDateTime;
import java.util.Set;
import java.util.UUID;
import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.EntityListeners;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Index;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToMany;
import javax.persistence.ManyToOne;
import javax.persistence.OneToMany;
import javax.persistence.Table;
import lombok.Data;
import lombok.experimental.Accessors;
import org.hibernate.annotations.GenericGenerator;
import org.hibernate.annotations.Type;
import org.springframework.data.annotation.CreatedDate;
```

```
import org.springframework.data.jpa.domain.support.AuditingEntityListener;

@Data
@Accessors(chain = true)
@Entity
@Table(
    name = "orders",
    indexes = @Index(name = "queue_index", columnList = "supplier_id,
creation_time, order_status")
)
@EntityListeners(AuditingEntityListener.class)
public class Order {

  @Id
  @Type(type = "uuid-char")
  @GeneratedValue(strategy = GenerationType.AUTO)
  @GenericGenerator(name = "UUID", strategy =
"org.hibernate.id.UUIDGenerator")
  private UUID id;

  @ManyToOne(fetch = FetchType.LAZY, cascade = CascadeType.ALL)
  private Supplier supplier;

  @OneToMany(mappedBy = "order")
  private Set<OrderItem> items;

  @CreatedDate
  @Column(name = "creation_time")
  private LocalDateTime creationTime;

  @Column(name = "start_processing_time")
  private LocalDateTime startProcessingTime;

  @Column(name = "end_processing_time")
  private LocalDateTime endProcessingTime;

  @Column(name = "order_status")
  private OrderStatus orderStatus;

}
package com.kpi.hurniak.supplierapi.entity.order;

import com.kpi.hurniak.supplierapi.entity.item.Item;
import java.io.Serializable;
import java.util.UUID;
import javax.persistence.Column;
import javax.persistence.Embeddable;
import javax.persistence.EmbeddedId;
import javax.persistence.Entity;
import javax.persistence.ManyToOne;
import javax.persistence.MapsId;
import javax.persistence.Table;
import javax.validation.constraints.Min;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.EqualsAndHashCode;
import lombok.NoArgsConstructor;
import lombok.experimental.Accessors;
import org.hibernate.annotations.Type;

@Data
@Accessors(chain = true)
@Entity
@Table(name = "order_items")
```

```
public class OrderItem {

  @EmbeddedId
  private OrderItemId id;

  @SuppressWarnings("JpaModelReferenceInspection")
  @MapsId("order_id")
  @ManyToOne
  @EqualsAndHashCode.Exclude
  private Order order;

  @SuppressWarnings("JpaModelReferenceInspection")
  @MapsId("item_id")
  @ManyToOne
  @EqualsAndHashCode.Exclude
  private Item item;

  @Min(1)
  private int count;

  @Embeddable
  @Data
  @Accessors(chain = true)
  @NoArgsConstructor
  public static class OrderItemId implements Serializable {

    @Column(name = "order_id")
    @Type(type = "uuid-char")
    private UUID orderId;

    @Column(name = "item_id")
    @Type(type = "uuid-char")
    private UUID itemId;
  }
}
```