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

Навчально-науковий інститут атомної та теплової енергетики

Кафедра інженерії програмного забезпечення в енергетиці

"На правах рукопису"
УДК 004.4

«До захисту допущено»
В.о. зав.кафедри
_____ Олександр КОВАЛЬ
“___”_____2022 р.

Магістерська дисертація

За освітньою програмою «Інженерія програмного забезпечення інтелектуальних кібер-фізичних систем і веб-технологій»
Спеціальності 121 Інженерія програмного забезпечення
на тему: Інформаційна система управління та аналізу замовлень телекомунікаційних компаній з використанням мікросервісної архітектури

Виконав: студент 2 курсу, групи ТВ-12мп
Комісаров Ілля Андрійович _____ _____
(прізвище, ім’я, по батькові) (підпис)

Науковий керівник к.т.н. доц. Ходаковський О.В. _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент к.т.н. доц. Волокита А. М. _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2022

Національний технічний університет України

“Київський політехнічний інститут ім. Ігоря Сікорського”

Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці
Рівень вищої освіти другий, магістерський
За освітньою програмою «Інженерія програмного забезпечення інтелектуальних кібер-фізичних систем і веб-технологій»
Спеціальності 121 “ Інженерія програмного забезпечення ”

"На правах рукопису"
УДК 004.4

«До захисту допущено»
В.о. зав.кафедри
_____ Олександр КОВАЛЬ
“___”_____2022 р.

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ ДИСЕРТАЦІЮ СТУДЕНТУ

Комісарова Іллі Андрійовича
(прізвище, ім’я, по батькові)

1. Тема дисертації:
Інформаційна система управління та аналізу замовлень телекомунікаційних компаній з використанням мікросервісної архітектури
Науковий керівник к.т.н. доц. Ходаковський Олексій Володимирович
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від “07” листопада 2022 року №4067-с
2. Строк подання студентом дисертації: 9 грудня 2022 року
3. Вихідні дані до роботи: інформаційна система автоматизації замовлень для телекомунікаційних компаній, яка базується на мікросервісній архітектурі, аналітика прибутку з замовлень, з використанням технологій машинного навчання
4. Перелік питань, які потрібно розробити: розглянути та проаналізувати існуючі системи; дослідити методи побудови систем за використанням мікросервісної архітектури; дослідити потреби телекомунікаційного бізнесу; дослідити методи машинного навчання та бізнес метрики для аналізу; розробити систему, що буде задовольняти всі потреби та покривати всі поставлені задачі.
5. Орієнтований перелік ілюстративного матеріалу: огляд існуючих систем, схеми архітектур використаних в роботі, діаграма прецедентів, діаграми класів, моделі баз даних, тестування системи.

6. Орієнтований перелік публікацій: Залевська О.В., Гагарін О.О., Комісаров І.А. Особливості структури систем обробки замовлень абонентів телекомунікаційних послуг / Прийнято до друку у “Сучасні проблеми моделювання” випуск 25.
7. Дата видачі завдання «01» жовтня 2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання магістерської дисертації	Строки виконання етапів магістерської дисертації	Примітка
1	Отримання завдання	01.10.2021	виконано
2	Дослідження предметної області	01.10.2021 - 01.11.2021	виконано
3	Постановка вимог до проєктування системи	01.11.2021 - 15.11.2021	виконано
4	Дослідження існуючих рішень	15.11.2021 - 01.12.2021	виконано
5	Підготовка публікацій	01.12.2021 - 09.12.2021	виконано
6	Розробка програмного продукту	05.03.2022 - 07.10.2022	виконано
7	Тестування	07.10.2022 - 11.11.2022	виконано
8	Захист програмного продукту	17.10-2022 – 21.10.2022	виконано
9	Підготовка магістерської дисертації	22.10.2022 – 20.11.2022	виконано
10	Передзахист	21.11.2022 - 25.11.2022	виконано
11	Захист	19.12.2022 – 23.12.2022	виконано

Студент _____ Комісаров І.А.
(підпис) (прізвище та ініціали)

Науковий керівник _____ Ходаковський О.В.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Актуальність теми. З розвитком комп`ютерних технологій та програмування все більше процесів в бізнесі та в побуті почали автоматизуватися. Сфера телекомунікацій також почала автоматизовуватись. Телекомунікаційним компаніям потрібно було зберігати дані клієнтів, налагоджувати процес замовлення продуктів, вести облік ресурсів, надавати своєчасно послуги та сервіси. Раніше всі ці задачі виконувалось повністю вручну, що вимагало багато часу та людського ресурсу. Наразі ця функціональність реалізована в системах, які зараз називаються BSS та OSS. Як правило це доволі великі та об`ємні системи, які розробляються десятки років великими корпораціями та коштують доволі дорого. Але невеликі провайдери не мають можливості придбати такі системи для своїх потреб, тому як правило замовляють такі системи на аутсорсі або розроблюють самі. Як правило такі системи все одно дорого обходяться і мають мізерну функціональність та застарілий інтерфейс користувача. Також будь-якому бізнесу потрібно підраховувати свої прибутки та витрати, проводити аналітику, прогнозувати прибутки для коригування стратегії розвитку та пропозицій для збереження та притоку клієнтів.

Мета роботи заключається в створенні системи, що забезпечує реєстрацію нових користувачів, автоматизує процес виконання замовлення, реалізує всі перевірки, веде облік ресурсів компанії та надає аналітику бізнесу, а саме – когортний аналіз, бізнес метрики та прогнозування прибутку.

Об’єктом дослідження є процес виконання замовлень для підключення абонентів до мережі.

Предметом дослідження є система, що автоматизує процес виконання замовлення та надає аналітику даних для невеликих провайдерів телекомунікаційних послуг.

Методи дослідження. В роботі використовуються сучасні технології та методи розробки систем з використанням мікросервісної архітектури, а також продвинуті підходи до аналітики та прогнозування. Насамперед, мови програмування Java, Python, TypeScript, фреймворки Spring, Angular, бібліотеки Flask, sktime, scikit-learn, СУБД PostgreSQL, MongoDB, брокер повідомлень RabbitMQ та система автоматизації збирання, розгортання, контейнеризації Docker.

Практичне значення полягає в можливості використання системи реальними провайдерами телекомунікаційних послуг, що зекономить час та кошти цільових компаній, а також дозволить більш коректно скласти стратегію по розвитку бізнесу та утриманню клієнтів.

Публікації. Залевська О.В., Гагарін О.О., Комісаров І.А. Особливості структури систем обробки замовлень абонентів телекомунікаційних послуг / Прийнято до друку у “Сучасні проблеми моделювання” випуск 25.

Структура та обсяг дисертації. Магістерська дисертація складається зі вступу, п`яти розділів, висновку, переліку посилань з 26 найменувань, містить 65 рисунків, 10 таблиць та 14 формул. Повний обсяг магістерської дисертації складає 169 сторінки, з яких перелік посилань займає 2 сторінки, додатки займають 58 сторінку.

Ключові слова: BSS, OSS, мікросервісна архітектура, телекомунікації, аналітика даних, прогнозування, розподілені системи, контейнеризація.

ABSTRACT

Significance of the topic. With the development of computer technologies and programming, more and more business processes began to be automated. The telecommunication industry also began to be automated. Telecommunications companies needed to store customer data, adjust the process of ordering products, keep records of resources, and provide services in a timely manner. Previously, all these tasks were performed completely manually, which required a lot of time and human resources. This functionality is currently implemented in systems called BSS and OSS. Usually, these are quite large and voluminous systems that have been developed for decades by large corporations and are quite expensive. But small providers do not have the opportunity to purchase such systems for their needs, so they usually order such systems from an outsourcer or develop them themselves. Usually, such systems are still expensive and have poor functionality and an outdated user interface. Also any business needs to calculate its profits and expenses, conduct analytics, forecast profits to adjust the development strategy and offers to retain and attract customers.

The purpose of the study is to create a system that ensures the registration of new users, automates the order fulfillment process, implements all checks, keeps records of company resources and provides business analytics, namely cohort analysis, business metrics and profit forecasting.

The object of study is a process of order execution for connecting subscribers to the network.

The subject of study is a system that automates the process of order execution and provides data analytics for small providers of telecommunication services.

Research methods. The work uses modern technologies and methods of system development using microservice architecture, as well as advanced approaches to analytics and forecasting. First of all, programming languages Java, Python, TypeScript, frameworks Spring, Angular, libraries Flask, sktime, scikit-learn, DBMS PostgreSQL,

MongoDB, message broker RabbitMQ and the automation system of assembly, deployment, containerization Docker.

The practical significance is in the possibility of using the system by real providers of telecommunication services, which will save time and money of the target companies, as well as allow a more correct strategy for business development and customer retention.

Publications. Zalevska O., Haharin O., Komisarov I. Peculiarities of the structure of order processing systems for subscribers of telecommunication services / Accepted for publication in "Modern Problems of Modeling", issue 25.

The structure and scope of the master's thesis. The master's thesis consists of an introduction, five chapters, a conclusion, a list of references from 26 titles, contains 65 figures, 10 tables and 14 formulas. The full volume of the master's thesis is 169 pages, of which the list of references occupies 2 pages, the appendices occupy 58 pages.

Keywords: BSS, OSS, microservice architecture, telecommunications, data analytics, forecasting, distributed systems, containerization.

ЗМІСТ

СПИСОК СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

BSS	Business Support System
OSS	Operations Support System
API	Application Programming Interface
GUI	Graphical User Interface
SOAP	Simple Object Access Protocol
REST	REpresentational State Transfer
URL	Uniform Resource Locator
URI	Uniform Resource Identifier
XML	eXtensible Markup Language
JSON	JavaScript Object Notation
HTTP	HyperText Transfer Protocol
CSS	Cascading Style Sheets
HTML	HyperText Markup Language
JVM	Java Virtual Machine
JAR	Java ARchive
JPA	Java Persistence API
JWT	JSON Web Token
AMQP	Advanced Message Queuing Protocol
SQL	Structured Query Language
SPA	Single Page Application
СУБД	Система Управління Базами Даних
NRC	Non recurring charge
MRC	Monthly recurring charge
DTV	Digital TeleVision

ВСТУП

З розвитком комп`ютерних технологій та програмування все більше процесів в бізнесі та в побуті почали автоматизуватися. Підприємства почали вкладати кошти в розробку систем, що автоматизують їхні бізнес процеси, так як вони з роками швидко окупаються. З кожним роком ця тенденція до автоматизації тільки зростає і сфера телекомунікацій також змінюється під натиском прогресу. Ще в минулому столітті телекомунікаційним компаніям потрібно було зберігати дані клієнтів, налагоджувати процес замовлення продуктів, вести облік ресурсів, надавати своєчасно послуги та сервіси, проєктувати локальній мережі, моніторити їх стан та якнайшвидше приймати рішення у випадку неполадок. Для цих потреб були створені системи, які зараз називаються BSS та OSS.

BSS – система підтримки бізнеса, що відповідає за взаємодію з користувачем та має відповідну функціональність, а саме: продажі та маркетинг, управління замовленнями, управління даними користувачів, виставлення та оплата рахунків, підтримка користувачів.

OSS – система підтримки операцій, що відповідає за технічні складові компанії, а саме: облік та управління ресурсами (кабелі, маршрутизатори, комутатори, SIM карти тощо), установку та налаштування необхідного апаратного забезпечення в рамках виконання замовлення підключення певного продукту, програмне моделювання мереж та моніторинг їхнього стану, спроби усунути неполадки у випадку їх виявлення і багато іншого.

Як правило, ці системи доволі великі та об`ємні, вони розробляються роками великими корпораціями та коштують доволі дорого. Зазвичай, вони призначені для великих та відомих провайдерів телекомунікаційних послуг, але існують і невеликі провайдери, які не мають можливості придбати такі системи для своїх потреб, тому як правило замовляють такі системи на аутсорсі або розроблюють їх самостійно. Реалізовані системи все одно дорого обходяться і мають мізерну функціональність та застарілий інтерфейс користувача. Також будь-якому бізнесу потрібно

підраховувати свої прибутки та витрати, проводити аналітику, прогнозувати прибутки для коригування стратегії розвитку та пропозицій для збереження і притоку клієнтів. Всі ці проблеми вирішуються в даній роботі. Основний акцент зосереджений на розробці функціональності BSS, так як користувач взаємодіє саме з нею, навідміну від OSS, яка оперує ресурсами.

1 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

1.1 Постановка задачі

Система має забезпечувати реєстрацію нових користувачів, аутентифікацію та авторизацію, дозволяти переглядати різні сторінки, в залежності від ролі користувача.

Для звичайних користувачів система повинна надавати можливість перегляду пропозицій всіх доступних продуктів з дисконтами та знижками, можливість переходу до корзини, вибору продуктів, введення необхідних параметрів для підключення, перегляд створених замовлень та параметрів продуктів, які підключаються або вже підключені, перегляд конкретних замовлень з потоком виконання його, тобто статусом задач, які потрібні для виконання замовлення. Також, сторінка пропозицій може бути доступна і неавторизованому користувачу.

Для користувачів з роллю Fitter система повинна надавати доступ до сторінки доставок, на якій монтажник може ввести необхідні дані для підключення клієнта та підтвердити підключення.

Для користувача з роллю Sales Manager система повинна надавати доступ до сторінки аналітики даних, де менеджер може переглянути кількість нових користувачів за певний проміжок часу, кількість замовлень певного продукту, прибуток за певний проміжок часу, різні бізнес метрики, прогноз прибутку на певний період часу.

Система повинна забезпечувати всі етапи виконання замовлення: ініціалізацію замовлення, перевірку можливості підключення послуги для користувача, проводити всі додаткові валідації, резервувати замовлені ресурси, синхронізувати доставку та підключення абонента, виконувати оплату за підключення. Також, система повинна автоматично знімати місячну плату з абонентів, і якщо коштів не вистачає призупиняти послугу.

1.2 Порівняння аналогів

На даний момент часу в світі існує доволі багато рішень таких систем. На жаль BSS/OSS - це, зазвичай, закриті комерційні проєкти, і інформації у відкритому доступі про них доволі мало. Але, так як створена система не позиціонується як конкурент таким системам, тому окрім великих ентерпрайз рішень, також, буде додано до порівняння локальні українські системи невеликих провайдерів, здебільшого, інтернет послуг.

Великі системи будуть порівнюватись в більшій мірі з відкритого сервісу gartner.com[1], який надає незалежну експертизу та порівнює існуючі системи по ключовим показникам. Взявши до уваги порівняння за 2015 рік, бачимо, що в лідерах є системи від таких розробників: Netcracker Technology, Amdocs.

На рисунках 1.1 та 1.2 зображено 4 квадранти.

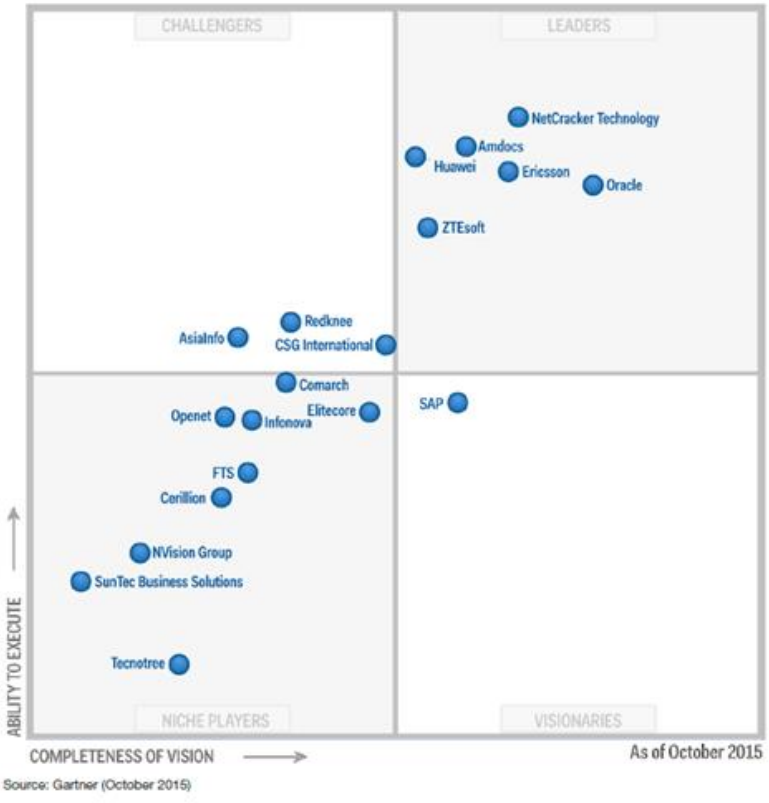


Рисунок 1.1 - Рейтинг BSS від різних компаній за 2015р. [1]

В лівому нижньому розміщені нішові компанії, які тільки починають розроблювати свої продукти та представляти їх замовникам, в правому нижньому

– компанії, що давно розробили свої продукти і довгий час знаходяться на ринку, але почали втрачати позиції останнім часом, в лівому верхньому – компанії, які недавно зайшли на ринок, але вже стали успішними завдяки широкому функціоналу та можливостям, що надають їхні системи, в правому верхньому – абсолютні лідери сфери, які вже доволі довго на ринку і мають достатньо багатофункціональні системи.

Рисунок 1.2 - Рейтинг OSS від різних компаній за 2015р. [1]

Зважаючи на результати, що представлені на вказаних графіках, розглянемо детальніше продукти таких компаній: Netcracker Technology, Amdocs.

1.2.1 Netcracker BSS&OSS

Компанія Netcracker розроблює BSS&OSS системи більше 25 років та вважається лідером у цій сфері для провайдерів телекомунікаційних послуг у всьому світі.

Розглянемо для початку BSS[2] рішення. Система має таку функціональність: управління маркетингом, конфігурацією цін, управлінням замовленнями, управління контрактами, управління даними абонентів, білінг, управління продажами, управління доходами, управління партнерами та управління продуктами.

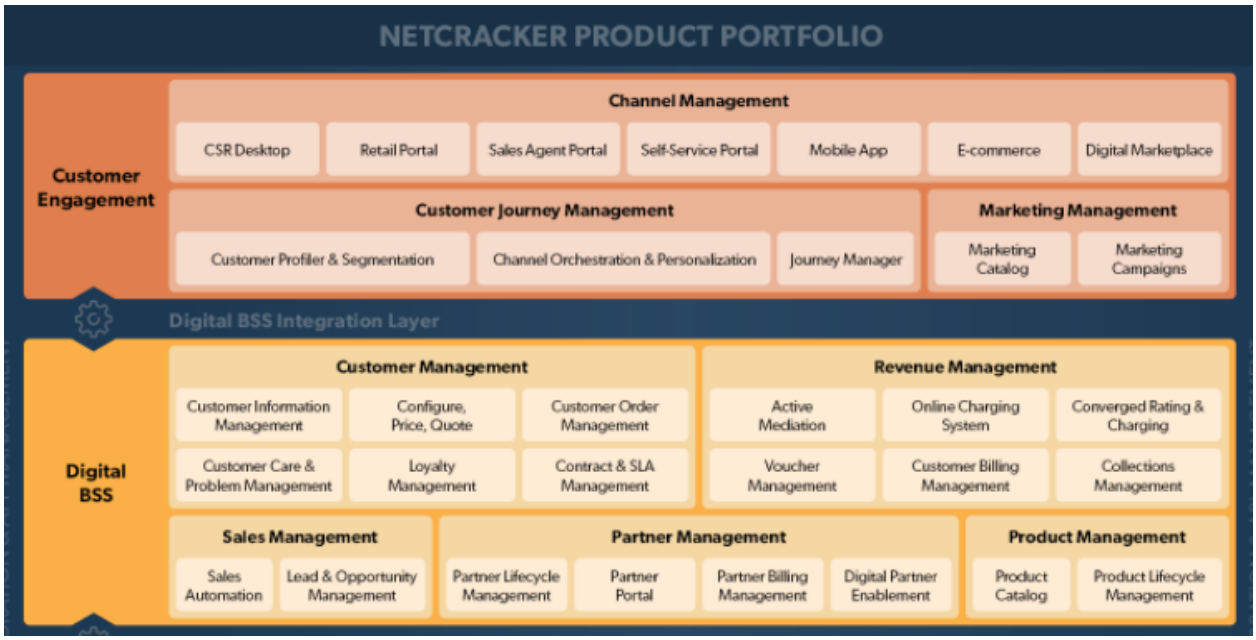


Рисунок 1.3 – BSS портфоліо Netcracker Technology [2]

Тепер розглянемо OSS рішення [3]. Схематично зображене на рисунку 1.4.

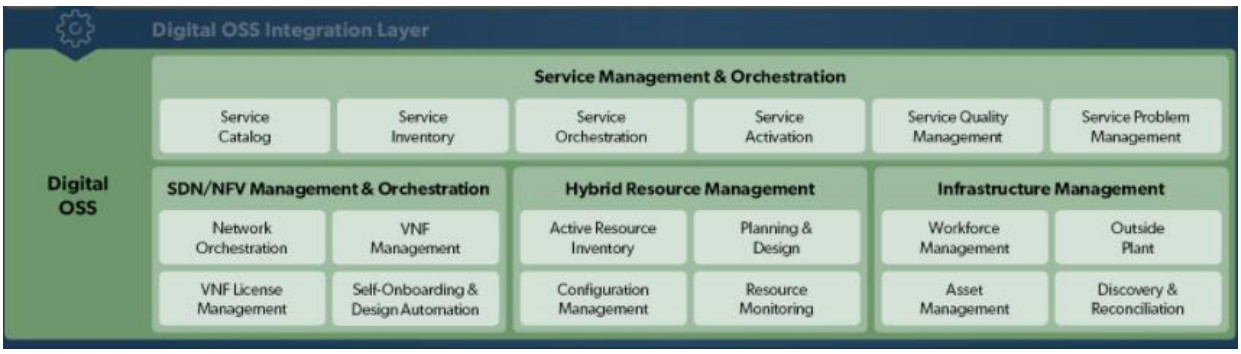


Рисунок 1.4 – OSS портфоліо Netcracker Technology [3]

Netcracker OSS використовує архітектуру на основі мікросервісів, щоб забезпечити гнучкість системи. Дана система має таку функціональність: сервіс каталог, інвентаризація ресурсів, активація сервісів, управління якістю сервісів, вирішення неполадок, що виникають в системі, оркестрація мереж SDN/NFV, гібридний менеджмент ресурсів, управління інфраструктурою.

Також, Netcracker пропонує деяку аналітику даних клієнтів: аналітика маркетингу, аналітика мережі та сервісних операцій, аналітика користувацького досвіду, оптимізація бізнес процесів. Всі ці види аналітики забезпечуються алгоритмами машинного навчання та штучним інтелектом.

1.2.2 Amdocs BSS&OSS

Компанія Amdocs в більшій мірі розробляє систему BSS[4], що реалізує білінг та взаємодію з клієнтами. Розробки в OSS[4] направлені на державний сектор.

Основні складові портфоліо Amdocs:

1) Amdocs Billing – система білінгу, яка забезпечує облік транзакцій найбільших провайдерів телекомунікаційних послуг в світі;
2) Amdocs CRM – ця система оптимізує бізнес-процеси, які тісно пов`язані з клієнтом; система вирішує проблеми, які впливають на сталість клієнтури і ефективність діяльності провайдера;
3) Amdocs Order Management – масштабована розподілена система управління замовленнями, яка реалізує найскладніші вимоги провайдерів телекомунікаційних послуг щодо цього функціоналу;
4) Amdocs Content Revenue Management – система управління потоками виконання надання контенту від провайдера до кінцевого споживача;
5) Amdocs Mediation – система, що реалізує широкомасштабне обслуговування клієнтів. Вона спрямована на вирішення найбільш складних задач щодо зменшення експлуатаційних витрат і підвищення ефективності бізнесу в цілому;
6) Amdocs Ensemble – білінгова система, яка розширює Amdocs Billing. В цю систему входять модулі: підтримка продаж, підтримка клієнтських операцій і сам білінг.

На рисунку 1.5 зображене портфоліо продуктів компанії Amdocs.

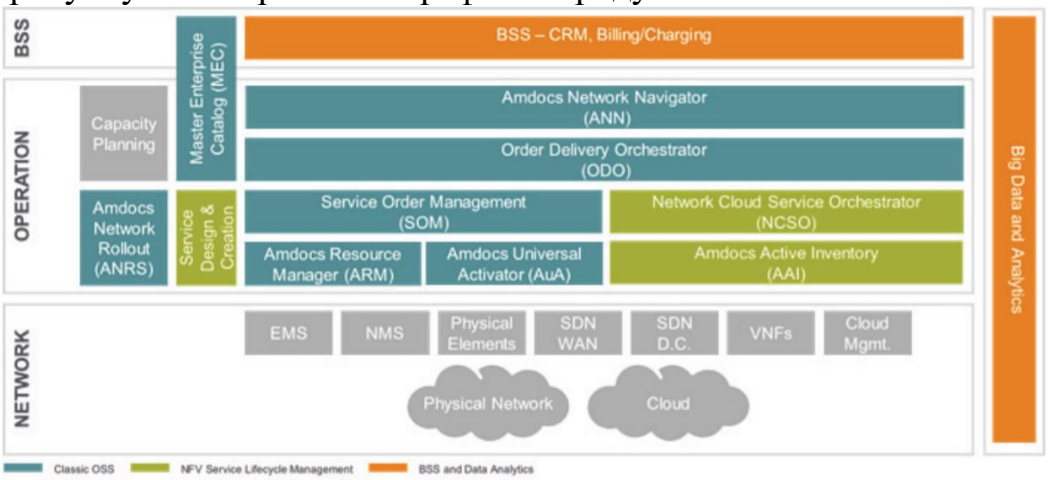


Рисунок 1.5 – BSS&OSS портфоліо Amdocs [4]

Були розглянуті великі системи від світових розробників.

1.2.3 Bilink

Перейдемо до невеликих провайдерів телекомунікаційних послуг та розглянемо інтернет провайдера Bilink[5], взявши за основу для порівняння їхній сайт та можливості, які він надає.

На рисунку 1.6 можемо бачити головну сторінку сайту, на ній ми бачимо меню зверху та рекламу знизу.

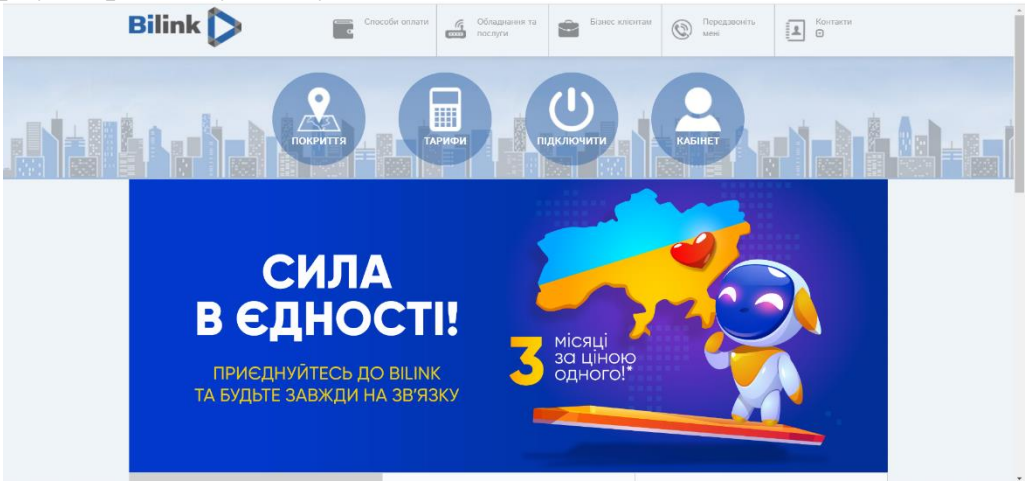


Рисунок 1.6 – Головна сторінка Bilink [6]

Користувач може перевірити чи можна підключити інтернет для нього в розділі покриття. Ця сторінка зображена на рисунку 1.7.

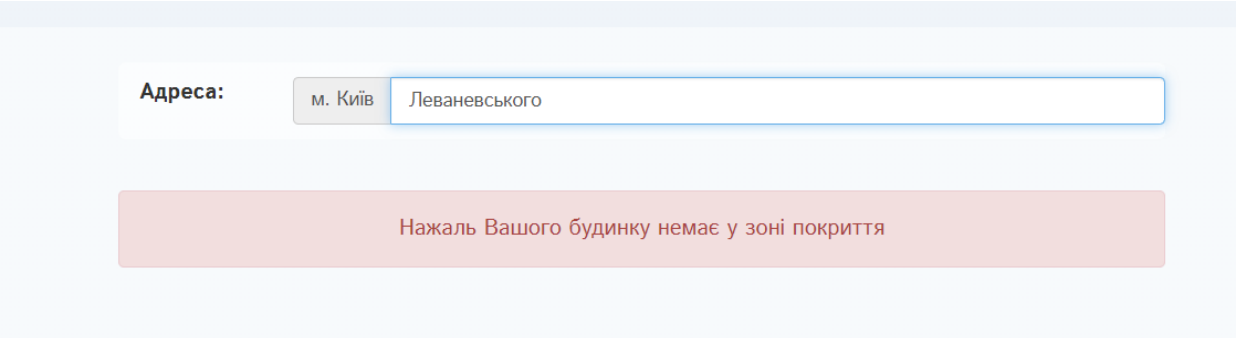


Рисунок 1.7 – Перевірка підключення Bilink

Також користувач може переглянути тарифи та пропозиції.

Інтернет		Інтернет + TV	
Назва	«150i»	«175i»	«200i»
Швидкість - Світ	100 Мбіт/с	300 Мбіт/с	500 Мбіт/с
Швидкість - Україна	100 Мбіт/с	1000 Мбіт/с	1000 Мбіт/с
Вартість підключення	~~99~~ 1 грн.	99 грн.	99 грн.
Діюча акція	знижка на 3 міс*	—	—
Абонплата	50 грн./міс*	175 грн./міс	200 грн./міс
	ПІДКЛЮЧИТИ	ПІДКЛЮЧИТИ	ПІДКЛЮЧИТИ

Рисунок 1.8 – Доступні тарифи Bilink

Підключення абонентів можливе тільки через дзвінок провайдеру, що є невеликим мінусом, бо деякі люди віддають перевагу замовленню через інтерфейс.

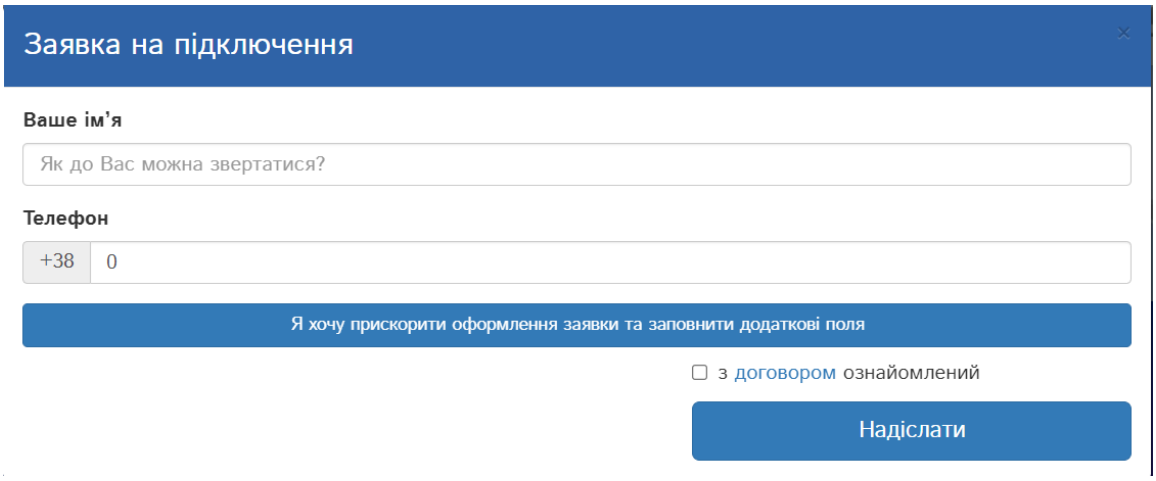


Рисунок 1.9 – Сторінка підключення послуги Bilink

На жаль при відключення послуги зайти до кабінету неможливо, тому немає змоги продемонструвати інтерфейс та можливості в кабінеті. На рисунку 1.10 зображена сторінка авторизації.

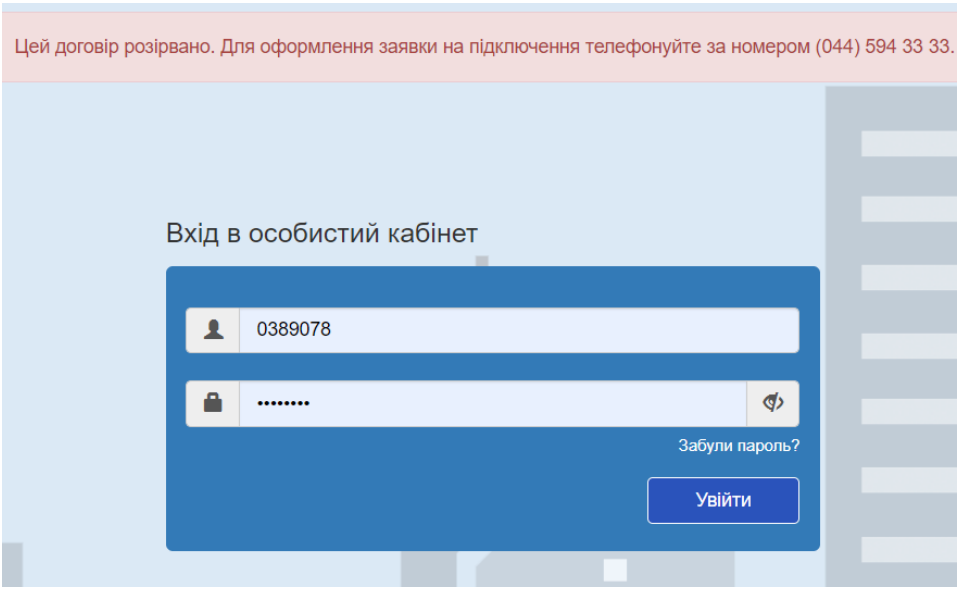


Рисунок 1.10 – Сторінка входу до кабінету Bilink

Була детально розглянута система від провайдера Bilink, а точніше їхній офіційний сайт.

1.2.4 Softlink

Також розглянемо ще одного невеликого провайдера під назвою Softlink[6] та можливості, які надаються на сайті. На рисунку 1.11 ми бачимо головну сторінку сайту softlink. Інтерфейс користувача попростіше ніж у Bilink. Знизу ми знаходиться меню, в якому можна переглянути тарифи інтернету та цифрового телебачення, можливість авторизації в кабінеті, доступні акції, техпідтримка та контакти.

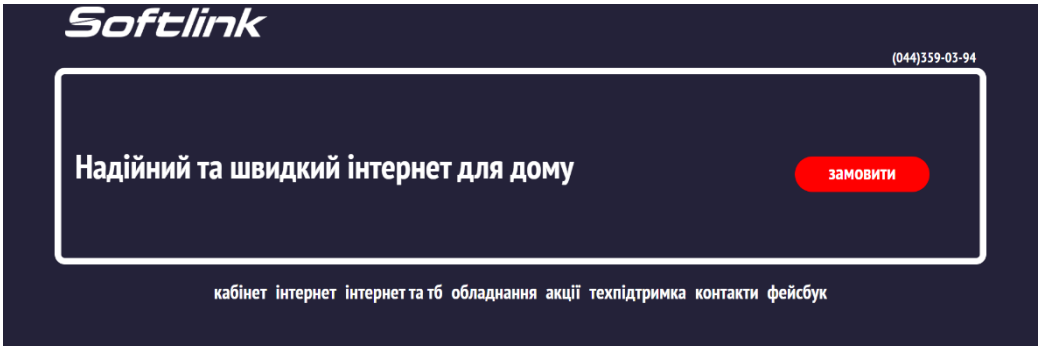


Рисунок 1.11 – Головна сторінка Softlink

На рисунку 1.12 зображені тарифи інтернету доступні в Softlink.

Рисунок 1.12 – Тарифи інтернету від Softlink

На рисунку 1.13 зображені доступні пристрої та обладнання від Softlink.

Рисунок 1.13 – Доступні пристрої від Softlink

Як і з сайтом Bilink, сайт Softlink не має можливості створити замовлення через інтерфейс на сайті – замовити послугу чи продукт можна тільки по номеру телефону. На рисунку 1.14 зображена сторінка кабінету користувача.

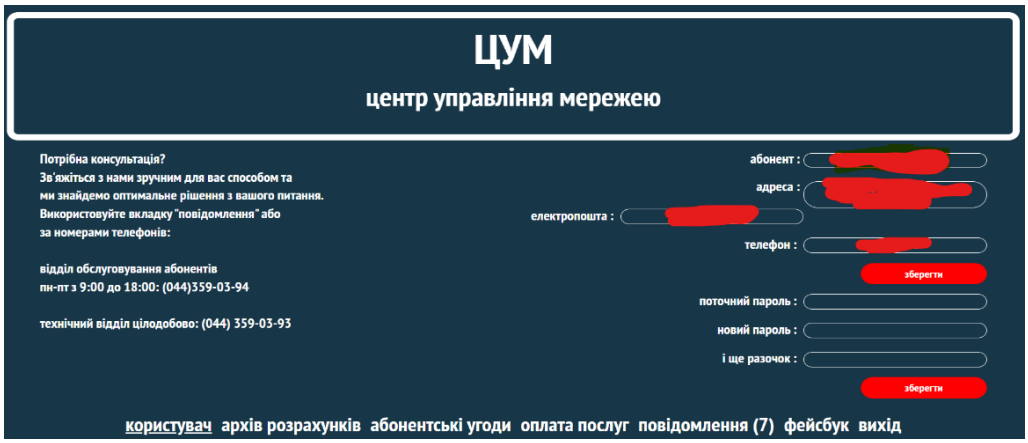


Рисунок 1.14 – Сторінка кабінету Softlink

Як бачимо інтерфейс користувача не дуже гарний. Можливості меню в кабінеті: переглянути інформацію користувача, архів платежів, угоди, оплатити послуги, переглянути повідомлення.

Висновки до розділу 1

В цьому розділі були розглянуті існуючі багатофункціональні системи BSS та OSS від основних компаній-розробників таких систем. Так як системи закриті, тому інформація доступна тільки на офіційних сайтах в вигляді портфоліо. Тому, інформації для об’єктивного порівняння недостатньо. Але можна сказати, що всі ці системи доволі схожі і мають фактично однакову функціональність. Замовники таких систем звертають увагу більше на конкретні модулі, тобто функціональність, які їм потрібні, і також на їхню ціну. Тому не можна сказати, що якась система краща за інші, це залежить від багатьох факторів.

Також були розглянуті сайти невеликих українських інтернет провайдерів, так як саме з ними конкурує розроблювана система. Можна сказати, що функціональність в них схожа, в Bilink трохи краще інтерфейс користувача. Але в обох системах не можна замовити підключення інтернету через інтерфейс, потрібно дзвонити оператору. Також в цих системах відсутня детальна інформація про замовлення, та доволі мало інформації про сам продукт, який підключений.

2 АПАРАТ ВИРІШЕННЯ ПОСТАВЛЕНОЇ ЗАДАЧІ

2.1 Аналіз архітектур

В рамках даної роботи були використані основні архітектури та принципи для проєктування комплексних систем, а саме - клієнт-серверна архітектура, багатошарова архітектура, архітектурний шаблон REST та мікросервісна архітектура. Усі ці архітектури розроблені для різних задач, тому вони можуть бути скомбінованими між собою і з іншими архітектурами, що не суперечать принципам цих архітектур. Розглянемо ці архітектури більш детально.

2.1.1 Клієнт-серверна архітектура

Клієнт-серверна архітектура[7] складається з сервера – фізичний комп`ютер, на якому запущене програмне забезпечення серверу, яке приймає запити та повертає відповідь, та клієнта – браузер або інший сервер, який відправляє запити на сервер. На рисунку 2.1 ілюстрована схема клієнт-серверної архітектури.

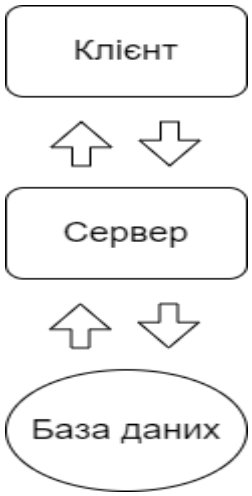


Рисунок 2.1 – Схема клієнт-серверної архітектури

Як бачимо, на схемі зображений також блок бази даних. Вона не обов’язкова, якщо не потрібно зберігати дані після обробки запиту. Але, здебільшого, вона

присутня, так як без бази даних, або іншого сховища даних, неможливо спроєктувати багатофункціональну систему.

Розглянемо кожен блок більш детально:

1) Клієнт – це, як правило, кінцевий користувач, який в браузері виконує певні дії і скрипти на JavaScript відправляють HTTP запити на сервер, щоб отримати дані або змінити існуючі. Також клієнтом може бути мобільний додаток на Android або iOS, який також може відправляти запити на сервер. Окрім програмного забезпечення, яким користується кінцевий споживач, клієнтом може бути деякий сервіс, який відравляє запити та оброблює відповіді для своїх потреб.
2) Сервером є як програмне забезпечення так і комп'ютер, на якому запущене це програмне забезпечення. Сервер приймає HTTP запити та повертає клієнту відповіді на ці запити. Програмне забезпечення, яке запущене на сервері, може бути написане на високорівневих мовах, таких як Python, Java, C#, PHP, Go тощо. Також сервер, як правило, має доступ до бази даних або декількох баз даних. На запит клієнта сервер зчитує дані з бази даних, виконує над ними певні операції та повертає клієнту в JSON або в XML форматі. Також, сервер може змінювати дані на запит клієнта.
3) База даних є сховищем структурованих даних, як правило, в текстовому форматі. Бази даних функціонують під управлінням Система Управління Базами Даних (СУБД). СУБД надає можливість створення бази даних, структур для зберігання даних (таблиці, документи, пар «ключ»: «значення»), роботи з цими структурами – редагування, читання і багато іншого. Також вони забезпечують цілісність, надійність та відмовостійкість бази даних. Найбільш популярними СУБД є PostgreSQL, Microsoft SQL Server , Oracle, MySQL, MongoDB.

В даному підрозділі було розглянуто основні принципи та терміни клієнт-серверної архітектури. Ця архітектура є доволі абстрактною та гнучкою, тому її можна змінювати під потреби системи.

2.1.2 Багаторівнева архітектура

Багаторівнева архітектура[8] призначена для розділення зон відповідальності логіки системи на декілька рівнів. Як правило, виділяють 3 рівні – рівень представлення, рівень бізнес логіки, рівень доступу до даних. Ці рівні допомагають організувати роботу команд, так як можна розділити зони відповідальності кожної команди. Таким чином, команда, що займається розробкою бізнес логіки не впливає на діяльність команди, що відповідальна за доступ до бази даних, так як кожен рівень надає певний інтерфейс іншому рівню для використання його функціоналу. Такий підхід сприяє швидкому темпу розробки систем та полегшує її підтримку в довготривалій перспективі.

На рисунку 2.2 зображена схема, що складається з трьох рівнів. Розглянемо їх більш детально:

1) Рівень представлення – це інтерфейс, з яким взаємодіє користувач, або API, яке надається сервером. Як правило, він знаходиться на стороні клієнта, але може знаходитися як на стороні сервера так і на стороні клієнта.
2) Рівень бізнес логіки – це основна частина система, яка знаходиться на сервері. Вона відповідає, власне, за бізнес логіку, тобто ключова функціональність системи. На цьому рівні мінімізовані технічні операції, такі як виклики сторонніх сервісів, звертання до бази даних, брокеру повідомлень тощо.
3) Рівень доступу до даних відповідає за роботою з базою даних або іншим сховищем даних. Надає інтерфейс для читання, редагування даних рівню

бізнес логіки, також може містити валідацію даних або обробку для подальшої роботи з ними на рівні бізнес логіки. Рівень доступу до даних, розміщується на сервері з рівнем бізнес логіки. Цей рівень ніяк не повинен взаємодіяти з рівнем представлення, тільки з рівнем бізнес логіки.

Рисунок 2.2 – Схема багаторівневої архітектури

Ця архітектура також є доволі абстрактною і гнучкою, може деяким чином бути змінена, в залежності від потреб проєкту.

2.1.3 Архітектурний шаблон REST

REpresentational State Transfer (скорочено REST) [9] – архітектурний принцип/шаблон/стиль, який отримав широку популярність останнім часом та використовується при розробці широкого спектру систем. REST має ряд вимог та принципів, дотримуючись яких, можна покращити надійність, стабільність та швидкодію системи. Системи, що реалізують всі вимоги REST називаються RESTful. Цей архітектурний шаблон був запропонований австрійським фахівцем Роєм Філдингом в його науковій роботі. Розглянемо всі вимоги/принципи, які він надає в своїй роботі:

1) Модель клієнт-сервер. Цей пункт був детально розглянутий в підрозділі 2.1.1. Він потрібен для гарної масштабованості системи;
2) Відсутність стану. Сервер не повинен мати та зберігати стан, тобто один і той самий запит повинен мати одну і ту саму відповідь в різні проміжки часу (якщо дані не змінювалися в базі даних). Цей пункт потрібен для запобігання різних конфліктів на стороні клієнта, а також допомагає тримати систему в консистентному (цілісному) стані.
3) Кешування. Кешування потрібно для покращення швидкодії системи, так як клієнт може не відправляти запит на сервер, а мати дані на своїй стороні, або сервер може кешувати дані і не зчитувати їх з бази даних. Але потрібно мати засоби для уникнення неконсистентності даних в кеші та обновляти кеш при зміні даних в базі даних. З цієї причини, кешувати потрібно дані, які рідко змінюються;
4) Єдиний інтерфейс. Сервер повинен мати єдиний інтерфейс для клієнта для уніфікації. Ця вимога має 4 підпункти:
    - Ідентифікатор ресурсу. Шаблон REST оперує, в першу чергу, ресурсами, а не операціями, як SOAP. Ресурсом є будь-яка сутність, яка може мати явний ідентифікатор, по якому можна отримати інформацію по цьому ресурсу. Наприклад замовлення, користувач,

продукт, доставка тощо. Цей ідентифікатор називається URI. Приклад URI: */api/v1/users/5* – користувач з id, що дорівнює 5.

- Операції над ресурсами через представлення. Цей підпункт означає, що ресурси можна змінювати через представлення. Представлення – це форма, що відображає стан деякої сутності. Це може бути JSON, HTML або XML формат. Змінюючи представлення змінюється ресурс.
- Самоописуюче повідомлення. Цей підпункт означає, що на запит клієнта сервер повинен повертати об’єкт ресурсу, над яким можна виконувати операції. Тобто клієнт з сервером повинні обмінюватись повідомленнями с корисним навантаженням.
- HATEOAS (англ. «hypermedia as the engine of application state»). Цей підпункт означає, що параметри повинні передаватись у посиланні з ідентифікатором ресурсу, а стан ресурсу передається в тілі запиту.

5) Шари системи. Цей пункт аналогічний до багаторівневої архітектури. Він детально був описаний в розділі 2.1.2;
6) Код на вимогу. Цей пункт не є обов’язковим, ним можна нехтувати за потребою. Він означає, що клієнт може у сервера запитувати якийсь виконуваний код для виконання його у себе на стороні. Прикладом є Java-аплети або JavaScript скрипти.

Для комунікації клієнта та сервера в шаблоні REST, як правило, використовується протокол HTTP. HTTP використовує такі методи для указання цілі запиту: POST, GET, PUT, DELETE, PATCH, HEAD, OPTION, TRACE, CONNECT. Зазвичай, використовуються перші 4 методи.

2.1.4 Мікросервісна архітектура

Мікросервісна архітектура[10] – це принцип розробки систем, в якому система розбивається на декілька (можливо десятків або сотень) модулів, які ще називаються мікросервіси, кожен з яких відповідає за конкретну функціональну задачу та взаємодіє з іншими мікросервісами шляхом HTTP запитів. Ці мікросервіси існують, як правило, незалежно один від одного і можуть бути реалізовані на різних мовах та технологіях, які більш підходящі для тієї чи іншої функціональноті. Також, як правило, мікросервіси використовують різні бази даних.

Ця архітектура активно почала використовуватися з 2010 року. Першими почали її використовувати великі американські компанії, такі як Netflix та Amazon. Потім її стали частіше використовувати й інші компанії по всьому світу. Її популярність зростає з кожним роком, так як вона має доволі багато переваг перед стандартним підходом.

Детально розглянемо ключові властивості мікросервісної архітектури:

1) Логіка всієї системи функціонально розбита на невеликі незалежні шматки з чіткими рамками відповідальності;
2) Комунікація і передача даних між мікросервісами, як правило, відбувається через HTTP протокол;
3) Мікросервіси можуть бути написані на різних мовах та використовувати різні технології. Наприклад, основна бізнес логіка може бути написана на Java, аналітика даних на Python, так як на цій мові багато бібліотек для роботи з даними, машинного навчання та нейронних мереж, критичні по швидкодії сервіси на Go та сервіс, якому потрібно взаємодіяти з апаратним забезпеченням, може бути написаним на C або Rust;

4) Так як доменні області різні, то для кожного мікросервіса можуть бути використані різні бази даних, що покращить швидкодію та масштабованість;
5) Так як мікросервіси є незалежними та автономними, то вони можуть розроблюватися різними командами, але потрібно гарно організувати роботу цих команд.

Мікросервісна архітектура була створена як конкурент класичній архітектурі, яка ще називається монолітною, коли вся система написана на одній мові програмування, використовує одну базу даних та розміщена на одному сервері. Зобразимо схематично ці архітектури для кращого розуміння. На рисунку 2.3 зображена схема монолітної архітектури.

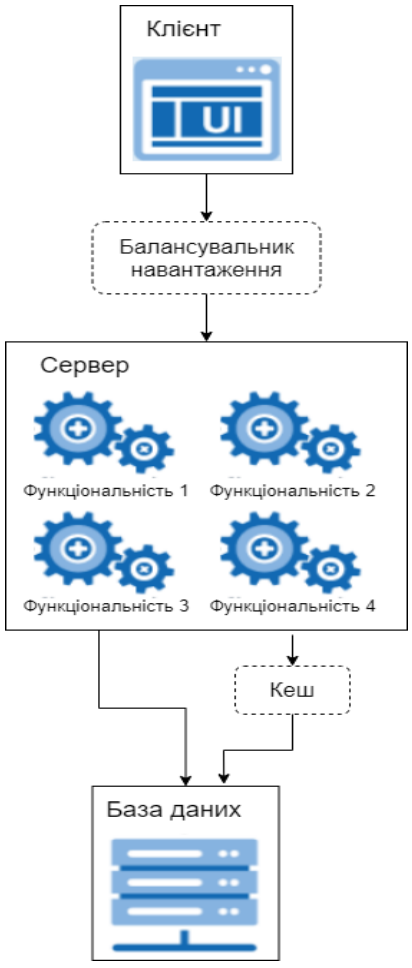


Рисунок 2.3 – Схема монолітної архітектури

В той же час на рисунку 2.4 зображена схема мікросервісної архітектури.

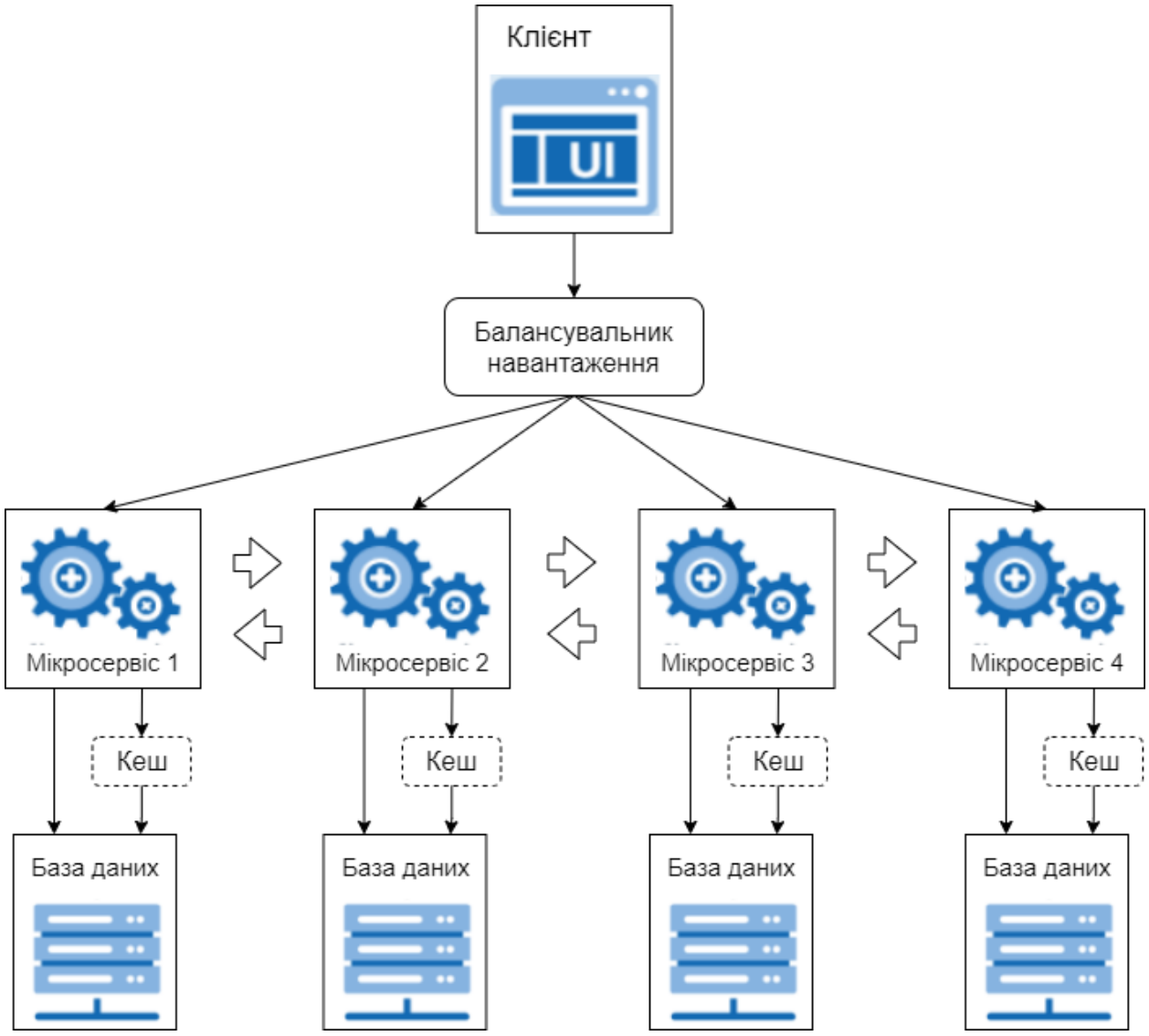


Рисунок 2.4 – Схема мікросервісної архітектури

Монолітний підхід до побудови систем використовувався дуже довгий час. Загалом він реалізує всі потреби, які накладаються на систему і всіх він влаштовував, але в нього є декілька недоліків, а саме:

1) Якщо на початку проєктування архітектури системи були допущені помилки і проєкт розроблюється доволі довго (2 роки+), то в такому випадку з часом для продовження розробки нової функціональності потрібно постійно переписувати старий код і тести, це скорочує швидкість розробки проєкту і також провокує створення нових дефектів, на які витрачається час. В сучасному світі такі затримки коштують дуже дорого.

2) Обмежена масштабованість. Монолітна система може масштабуватися тільки повністю, тобто додати фізичних серверів для збільшення швидкодії – горизонтальна масштабованість, або додати обчислювальних потужностей на один сервер – вертикальна масштабованість. Але, що роботи якщо потрібно збільшити швидкодію для одної функціональності? Наприклад, пошуком користуються 3 млн. користувачів в день і 3 фізичних сервери не справляються з таким навантаженням і потрібно масштабувати систему, але функціональністю створення замовлень користуються близько 20 тисяч користувачів в день і з таким навантаженням 3 фізичних сервери справляються без затримок. У такому випадку додавання нових серверів або обчислювальних потужностей є неефективним, так як ми масштабуємо всю систему, а потрібно покращити швидкодіє тільки для пошуку. Це проблема неефективного використання ресурсів.
3) Відсутність чітких зон відповідальності та ізоляції в системі. Так як вся функціональність реалізована в одному модулі, то зміни в одній частині можуть вплинути на логіку роботи в іншому місті. В такому випадку виникають проблеми описані в першому пункті. Також, коли нема чіткої структури та рамок у кожної функціональноті, то код швидко перетворюється в неструктуровану кашу, яку важко підтримувати.
4) Обмеження одним стеком технологій. При проєктуванні системи архітектори обирають стек технологій для реалізації всієї системи. Тобто обирають мову, основний фреймворк, СУБД, брокер повідомлень, систему збірки проєкту тощо. Але для одних задач краще підходять одні технології, а для інших – інші. Таким чином вибір технологій завжди буде компромісом. Для маленьких систем з простим та обмеженим функціоналом це не проблема, а для великих систем – проблема.
5) При будь-якій зміні в будь-якій частині коду потрібно перезбирати весь проєкт, що для великих систем може зайняти декілька годин.

Перелічені недоліки і вирішує мікросервісна архітектура. Але так як нічого ідеального не буває, тому мікросервісна архітектура також має недоліки:

1) Розробка системи на базі мікросервісної архітектури складніше на початку проєктування, так як потрібно продумати зони відповідальності кожного мікросервісу, взаємодію між мікросервісами, розбити домен для баз даних тощо. Ця задача повинна виконвуватися досвідченими спеціалістами.
2) Проблеми з написанням тестів. Так як потрібно враховувати комунікацію між мікросервісами, то ускладнюються інтеграційне та E2E тестування.
3) Чим більше мікросервісів тим важче підтримувати комунікацію між ними. В великих системах кількість мікросервісів може перевищувати 100 – очевидно, що тримати в голові всі зв`язки практично нереально.

В таблиці 2.1 наведене порівняння цих архітектур.

Таблиця 2.1 – Порівняння та мікросервісної та монолітної архітектур

Архітектура / Характеристика	Мікросервісна архітектура	Монолітна архітектура
Швидкість розробки	Простіше після 1-2 років розробки	Простіше на початку
Можливість використання різних технологій для задач	+	-
Перезбирання системи при змінах в проєкті	Потребує менше часу	Потребує більше часу
Масштабованість	Можлива і для окремих компонентів	Тільки всієї системи
Рівень кваліфікації розробників	Більш кваліфіковані	Менш кваліфіковані
Тестування	Простіше для модульного, складніше для E2E	Простіше для E2E, складніше для модульного
Зони відповідальності по функціональності	Чіткі/явні	Розмиті/неявні

В висновку, можна заключити, що мікросервісна архітектура підходить якщо: розроблювана система є складна и об’ємна, розробка триватиме більше 2 років, навантаження на різні модулі системи неоднорідне, система має різні функціональності під які підходять різні технології, розробка проходить швидко та динамічно і часто доводиться змінювати код. Для всіх інших випадків достатньо монолітної архітектури.

2.2 Технології серверної частини

Розглянемо детально технології (мови програмування, фреймворки, бібліотеки, СУБД тощо), які використовуються на серверній частині проєкту.

2.2.1 Мова програмування Java

Java[11] – високорівнева об’єктно-орієнтована мова програмування з статичною строгою типізацією, що розроблена в 1995 році компанією Sun Microsystems. Але зараз права на ліцензію належать компанії Oracle. При створенні мови головною ідеєю мови була можливість запускати один і той самий код на будь-якій платформі. Ця можливість реалізована за допомогою JVM (англ. Java Virtual Machine). JVM – віртуальна машина, що виконує байт-код, який є результатом компіляції вихідного коду на Java.

Java є однією з найбільш популярних мов програмування, за версіями Spectrum, TIOBE та IEEE вона займає 2 місце в рейтингу популярності (після JavaScript). Приваблює розробників і компанії вона своєю надійністю, передбачуваністю та оберненою сумісністю. Через обернену сумісність Java є трохи застарілою та багатослівною в порівнянні з іншими мовами програмування, але це ключова функція завдяки якій її обирають. Зазвичай, на Java пишуть складні

enterprise системи, веб-додатки, Android додатки, IoT системи, ігри, також використовується в Big Data для обробки великих об’ємів даних.

Java використовує статичну та строгу типізацію і є об’єктно-орієнтованою мовою, тобто ключовими поняттями є клас та об’єкт. Клас описує, які властивості та атрибути може мати об’єкт цього класу.

Ця мова була вибрана для розроблюваної системи, так як подібні системи написані, як правило, на Java і вона дає широкі можливості для інтеграції з іншими фреймворками та бібліотеками, наприклад Spring або Hibernate.

2.2.2 Мова програмування Python

Python[12] – високорівнева об’єктно-орієнтована інтерпретована мова програмування з динамічною типізацією та простим синтаксисом, спрямованим на швидкість розробки. Розроблений Гвідо ван Россумом ще в 1991 році, але з тих пір сильно змінився. Як і Java є одним з найбільш популярних мов програмування в світі. В різних рейтингах він знаходиться або на 3-му місці або на 2-му. Python має декілька особливостей, що виділяють його від інших мов. Такі як:

1) Інтерпретованість. Python є інтерпретованою мовою програмування, завдяки цьому код можна виконувати порядково та легко налагоджувати програми. Але ціною за це є швидкість виконання програм, що менше ніж в інших мовах програмування.
2) Динамічна типізація. Тип змінних визначається від час виконання програми, а не під час трансляції, як в інших мовах. Це дозволяє писати лаконічний простий код, але якщо виникнуть конфлікти типів, то виключення буде викинуто під час виконання програми, а не під час трансляції. Це доволі суттєвий недолік. Також динамічна типізація впливає на швидкодію виконання.

3) Простий синтаксис. Python є однією з найпростіших мов програмування завдяки його синтаксису. Код написаний на Python мало відрізнається від звичайного тексту. Також особливістю є відсутність { } для блоків коду.
4) Велика стандартна бібліотека. Python «з коробки» надає багато можливостей, таким чином ще більше скорочується написаний код.

Мова програмування Python вибрана для роботи, так як Python широковикористовується в аналітиці даних та має дуже зручні бібліотеки для роботи з даними та машинного навчання. Завдяки мікросервісній архітектурі мікросервіс аналітики даних може бути написаний на більш зручній мові для цієї задачі.

2.2.3 Фреймворк Spring

Spring Framework[13] – найбільш популярний веб-фреймворк для Java, був створений 2002 року Родом Джонсоном для полегшення розробки великих систем на Java. Надає рішення для широкого спектру задач та є фактично стандартом для сучасних enterprise систем, що розробляються на мові Java. Також, розробників приваблює велика кількість ресурсів для вивчення фреймворку, проста і зрозуміла документація.

Spring Framework, є саме набором фреймворків, які призначаються для різних задач. Всі фреймворки є доволі об’ємними і завдяки тому, що вони є незалежними розробники можуть брати тільки ті, що потрібні їм для розробки. Детально розглянемо компоненти, що використовувались в роботі:

1) Spring IoC (Inversion of Control) контейнер є ядром Spring Framework. Це рішення займається створенням об’єктів, конфігурацією та збиранням їхніх залежностей, а також керуванням всім життєвим циклом об’єкта, які називаються бінами. Контейнер використовує ін’єкцію залежностей (англ.

Dependency Injection) для керування бінами, які складають додаток. Він отримує інформацію про об’єкти з файлу конфігурації, коду Java або анотацій. Так як керування об’єктами Java або бінами та їхнім життєвим циклом здійснюється не розробниками, тому і назва компоненти - Inversion Of Control. Існує 2 типи контейнерів IoC:

- BeanFactory
- ApplicationContext

BeanFactory є найпростішою версією контейнерів інверсії контролю, а ApplicationContext розширює можливості BeanFactory. Інверсія контролю надає такі можливості і переваги: створення об'єктів, управління об’єктами, налаштування системи.

2) Spring AOP або аспектно-орієнтоване програмування в Spring полягає в додаванні «наскрізної» функціональності, тобто функціональність, яка стосується декількох модулів, але вона не має прямого відношення до бізнес-логіки, і її добре б винести в окреме місце. Прикладом наскрізної функціональності є проксі – об’єкт-обгортка над існуючою функціональністю для додавання технічної, або тієї, що не стосується бізнес логіки тощо.
3) Spring Boot дозволяє легко додати можливості Spring в програму, так як Spring Framework має дуже багато функціональності, в якій можна заплутатись, тому розробники випустили Spring Boot, який має мінімум, який потрібен більшості розробників ПО. Ключовими особливостями Spring Boot є спрощення створення Spring додатків, інтеграція з серверами Tomcat та Jetty, позбавлення необхідності в конфігурації через XML та генерації коду.
4) Spring Data надає зручний та простий інтерфейс для роботи з різними базами даних. Завдяки Spring Data можна легко сконфігурувати data source з URL доступу до бази даних та працювати з нею. Spring Data складається з багатьох модулів, наприклад: Spring Data JPA, Spring Data JDBC, Spring Data MongoDB, Spring Data LDAP тощо. Spring Data JPA є однією з реалізацій

специфікації JPA. JPA – специфікація, яка задає інтерфейс для мапінгу таблиць бази даних до класів Java та роботи з базою даних через простий інтерфейс. Spring Data JPA базується на Hibernate та доповнює його можливості. Spring Data MongoDB має такий самий інтерфейс, як і Spring Data JPA, але призначена для роботи з нереляційною СУБД MongoDB.

5) Spring Security. Spring Security є фреймворком, що вирішує задачі ідентифікації, аутентифікації та авторизації для Spring додатків. Spring Security реалізує багато видів аутентифікації та авторизації, розробнику залишається тільки сконфігурувати роботу фреймворку. Найбільш популярним методом авторизації на даний момент є авторизація з використанням JWT. JWT – специфікація для створення токенів авторизації в JSON форматі. Токен створюється на сервері, заповнюється корисними даними, хешується та відправляється клієнту для подальшого використання. На рисунку 2.5 зображений процес авторизації з використанням JWT:

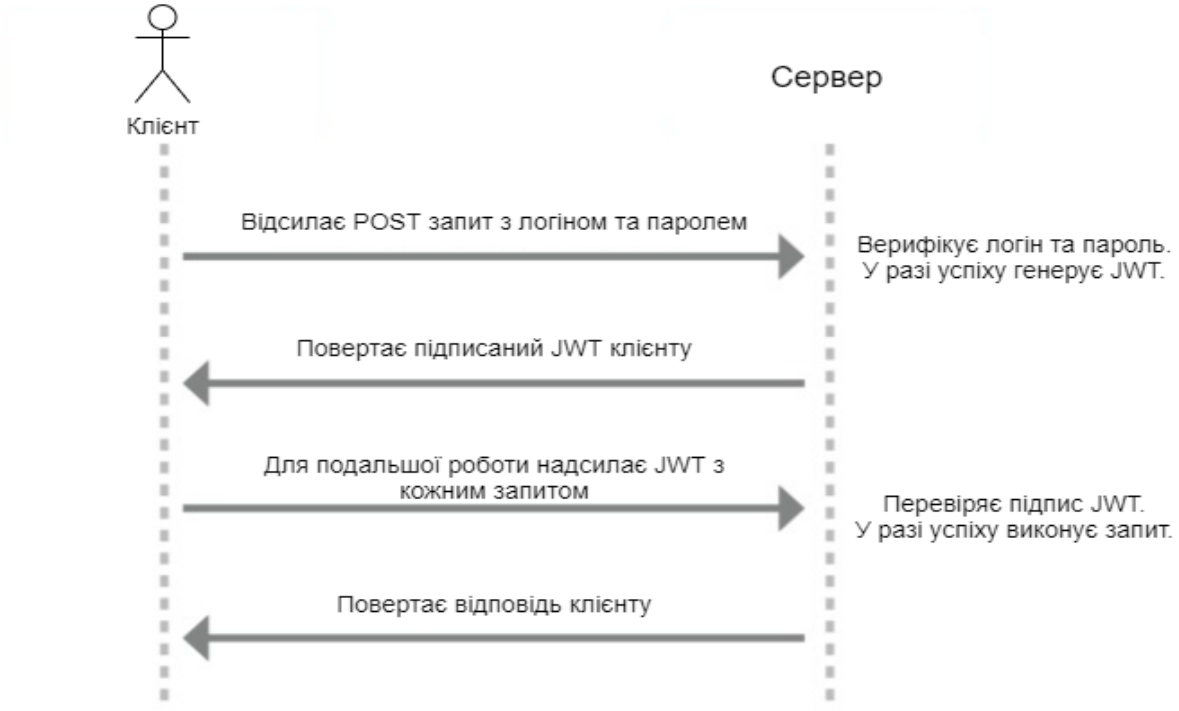


Рисунок 2.5 – Схема аутентифікації та авторизації з JWT

6) Spring Cloud включає в себе багато модулів, для розробки систем на базі мікросервісної архітектури, також містить в собі модулі для інтеграції з

Netflix OSS, яка, в свою чергу, має засоби для роботи з мікросервісами. Розглянемо детально модулі з Spring Cloud та Netflix OSS, які були використані в роботі:

- Spring Cloud Netflix Eureka. Netflix Eureka займається реєстрацією мікросервісів в системі. Для цього достатньо додати відповідну залежність до проєкту та в конфігурації описати назву сервісу та дозволити Eureka реєструвати цей сервіс.
- Spring Cloud Netflix Zuul. Zuul сервер займається маршрутизацією запитів від клієнта до конкретного мікросервіса. Тобто, коли клієнт має отримати відповідь від певного мікросервіса він має знати його URL, щоб відправити запит, а якщо мікросервісів багато, то це не дуже зручно. Цю проблему вирішує Zuul сервер – він дістає з реєстру Eureka серверу всі мікросервіси та надає можливість доступу до мікросервісу шляхом вказання його назви. Тепер клієнту потрібно знати тільки URL Zuul сервера, а він в свою чергу перенаправить запит на потрібний мікросервіс. Також Zuul сервер може займатися балансуванням навантаження, якщо додати задежність Netflix Ribbon, яка реалізує алгоритм планування Round Ribbon, таким чином покращити швидкодію на ефективність системи.
- Spring Cloud OpenFeign надає дуже простий інтерфейс для взаємодії одного мікросервісу з іншими. Для цього потрібно тільки об’явити інтерфейс з методами доступу до ресурсів іншого мікросервісу та вказати URI. Він прийшов на заміну громіздкому Rest Template.

Як бачимо, Spring Cloud міть дуже багато корисних модулів для розробки великих навантажених систем, надає багато засобів для спрощення розробки, тому і був використаний як основний фреймворк в роботі.

2.2.4 Фреймворк Flask

Flask[14] — фреймворк для створення веб-застосунків на мові програмування Python. Flask належить до категорії мікрофреймворків, тобто мінімалістичних каркасів, які надають лише базові можливості для розробки веб-застосунків.

Flask надає «з коробки» не дуже багато можливостей, тому потрібно використовувати багато сторонніх бібліотек. На щастя, спільнота розроблює багато бібліотек для зручної інтеграції з Flask, наприклад, flask-sqlalchemy в ролі ORM та для роботи з базами даних, flask-cors для настроювання CORS запитів, flask-restful для роботи з REST. В порівнянні зі Spring, який надає дуже багато можливостей, інструменти «з коробки» для будь-яких задач, Flask дуже легковісний, та надає тільки базовий функціонал для розробки веб-застосунків, такі як сервер розробки та налагоджувач, підтримку модульного тестування, підтримка шаблонізатор Jinjia2, використання WSGI, зручний простий інтерфейс для сторення endpoint-ів.

У висновку можна сказати, що Flask ідеально підходить для невеликих мікросервісів, так як є легким та простим, тому і був використаний в роботі.

2.2.5 Бібліотеки для аналізу даних

Для мови програмування Python існує велика кількість зручних бібліотек та інструментів для роботи з даними, аналітики, машинного навчання, нейронних мереж, штучного інтелекту тощо. Така ситуація склалась через простоту мови Python[15].

Розглянемо основні бібліотеки для аналітики даних:

1) NumPy (Numerical Python) – бібліотека для роботи з великими масивами даних та матрицями для мови програмування Python, написана на C та Fortran. Ця бібліотека була створена, бо робота з великими масивами даних на чистому Python занадто повільна, тому було написано numpy на

C та Fortran, завдяки яким швидкість виконання операцій над масивами та матрицями виконується на порядок швидше. Також окрім швидкості numpy надає широкі можливості по індексації, зручний інтерфейс по зміні даних в масиві. Часто numpy порівнюють з MATLAB, так як вони виконують одні й ті самі задачі, але MATLAB має вбудовані інструменти для роботи та багато пакетів для роботи, а numpy більш спеціалізований інструмент.

2) Pandas – бібліотека для обробки даних та їх аналізу на мові програмування Python. Pandas є надбудовою над Numpy, але написана на Python, тому швидкість операцій набагато менша ніж в numpy. Pandas надає спеціальну структуру даних для роботи – DataFrame, що представляє собою таблицю з рядками та колонами, як в реляційних базах даних. Завдяки цій структурі можна легко оперувати даними, виконувати на них різні функції над числовими даними або часовими рядами. Pandas надає такий функціонал: зручна індексація даних, імпорт та експорт даних в різні текстові формати, методи обробки порожніх даних (NaN), групування по колонці, вставка та видалення, поєднання двох таблиць, робота з часовими рядами.
3) scikit-learn – бібліотека для машинного навчання для мови програмування Python. Надає можливості для створення, навчання та тестування моделей для вирішення задач машинного навчання, таких як регресія, класифікація, кластеризація. Scikit-learn написана на Python та CPython для покращення швидкодії. Scikit-learn має реалізації багатьох алгоритмів, таких як лінійна регресія, Lasso, градієнт буст, random forest, k-середні та багато інших. Також вона надає дуже простий інтерфейс для роботи з цими моделями, достатньо створити об’єкт та викликати метод fit() для навчання моделі та метод predict() для отримання результатів, або fit_predict(). Повністю інтегрована з бібліотеками numpy та pandas.
4) sktime – сучасна бібліотека для прогнозування даних в часових рядах. Надає такий самий інтерфейс як і scikit-learn для прогнозування даних.

Була створена для спрощення задачі прогнозування даних в часі. Має реалізації популярних алгоритмів для прогнозування – Prophet, ARIMA, TBAS. Повністю інтегрована з бібліотеками scikit-learn, numpy та pandas. Також може використовувати регресійні моделі з scikit-learn.

Перелічені бібліотеки надають широкі можливості та значно полегшують задачу аналізу даних, тому і були використані в роботі.

2.2.6 Бази даних

Наразі неможливо уявити систему, яка не має бази даних для збереження корисної інформації та зручної роботи з ними. Бази даних управляються Системами Управління Базами Даних або СУБД. Існує багато різних СУБД. Вони діляться на реляційні та нереляційні.

Реляційна база даних – база даних, що зберігає дані структуровано в таблицях, в яких колонки – атрибути сутності, що зберігається в таблиці, а рядки – корисне навантаження або об’єкти зберігаємих сутностей. Реляційні БД так називаються, бо таблиці можуть бути пов’язані та посилатися на записи з інших таблиць. Для доступу до даних реляційні бази даних використовують SQL (англ. Structured Query Language) – структуровану мову запитів, за допомогою якої можна створювати, читати, модифікувати, видаляти дані. Прикладами реляційних СУБД є PostgreSQL, Oracle, Microsoft SQL Server, MySQL[16].

Розглянемо СУБД PostgreSQL. Від інших СУБД вона виділяється об’єктно-реляційною моделлю, тобто вона може оперувати об’єктами, створювати власні типи даних тощо, підтримкою великих об’ємів даних, одночасна модифікація даних, реалізація ACID, великий набір вбудованих функцій для обробки даних різних типів.

Нереляційна база даних – база даних, що зберігає дані в неструкторованому форматі, наприклад JSON, або в парами <ключ>: <значення>. Інша популярна назва для нереляційних баз даних - NoSQL (англ. Not only SQL). Прикладами нереляційних СУБД є MongoDB, Apache Cassandra, Redis, Neo4j[17].

Розглянемо особливості СУБД MongoDB. Вона зберігає дані в документах, використовуючи власний формат - Binary JavaScript Object Notation (BSON) як альтернативу JSON. Дані зберігаються без заданої структури, що дозволяє зберігати будь-які об’єкти в документах. Також СУБД має гарно швидкодію та зручний графічний інтерфейс.

В роботі використовуються як реляційна СУБД PostgreSQL, так і нереляційна СУБД MongoDB. Так як ці СУБД доволі популярні, надають широкі можливості для зберігання та обробки даних та мають гарну швидкодію.

2.2.7 Брокер повідомлень RabbitMQ

Брокер повідомлень – програмне забезпечення, що займається пересиланням повідомлень між відправником та споживачем. Використовується для асинхронної передачі даних між різними модулями або системами. Брокери повідомлень оперують наступними термінами:

- Відправник (англ. Producer) – компонента, що створює та відправляє повідомлення в брокер.
- Черга (англ. Queue) – структура даних, яка зберігає повідомлення до відправки отримувачу. Гарним порівнянням є порівняння з «поштовим ящиком».
- Отримувач (англ. Consumer) – компонента, що дістає повідомлення з черги.

Існує багато брокерів повідомлень. Найбільш популярні - Apache ActiveMQ, Apache Kafka, RabbitMQ. Розглянемо брокер повідомлень RabbitMQ детальніше.

Він використовує протокол AMQP для пересилки повідомлень. На рисунку 2.6 зображена схема роботи брокера повідомлень RabbitMQ[18].

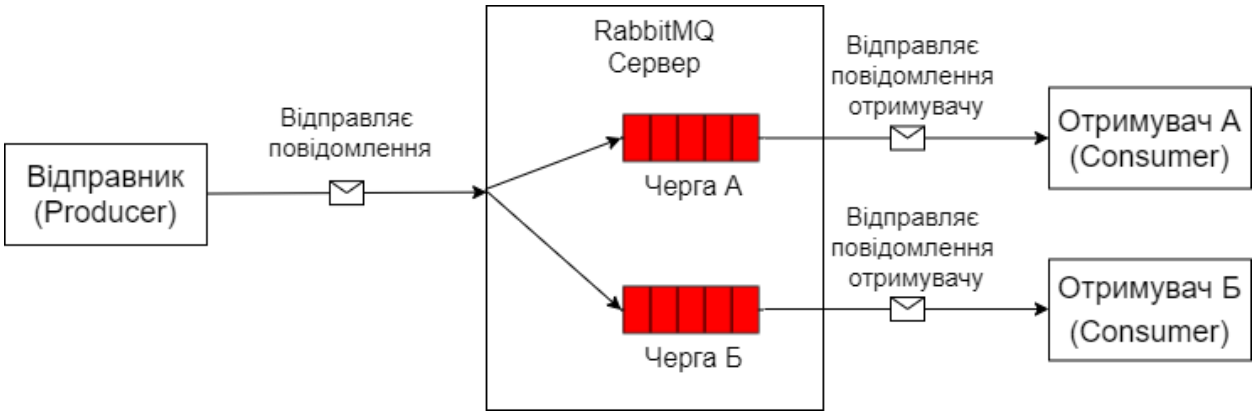


Рисунок 2.6 – Схема роботи брокера повідомлень RabbitMQ

Apache Kafka, є доволі потужним інструментом і часто використовується для високонавантажених систем, бо може забезпечити гарну відмовостійкість та витримує більші навантаження ніж RabbitMQ. Але Apache Kafka довше потрібно конфігурувати для коректної роботи, і з тієї причини, що RabbitMQ є більш простим інструментом та покриває всі задачі даної роботи, то був обраний саме він.

2.2.8 Засіб контейнеризації Docker

Docker[19] – це програмна платформ, що автоматизує збірку, тестування та розгортання програм. Docker упаковує програми у стандартизовані модулі, які називаються образами (англ. image). Кожен образ включає все необхідне для розгоратння: бібліотеки, системні інструменти, код і середовище виконання. Щоб створити образ потрібно в папці проєкту створити файл Dockerfile, з командами, завдяки яким буде створений образ для розгортування. На рисунку 2.7 зображений приклад Dockerfile.

```
# syntax=docker/dockerfile:1
FROM node:12-alpine
RUN apk add --no-cache python2 g++ make
WORKDIR /app
COPY . .
RUN yarn install --production
CMD ["node", "src/index.js"]
EXPOSE 3000
```

Рисунок 2.7 – Приклад Dockerfile для створення образу

Після створення образу, на його основі можна запустити контейнер – робоча програма, яка виконується поверх двигуна Docker (англ. Docker Engine). Контейнери виконуються в контейнеризованому середовищі. Контейнеризація — це метод віртуалізації, при якому єдиний «простір користувача» в ядрі операційної системи поділяється на декілька незалежних логічних розділів. Контейнеризація дозволяє розгортати програми на будь-якій платформі, якщо на ній встановлений Docker (або інші засоби контейнеризації) та покращує відмовостійкість. Контейнери легко масштабуються і вони незалежні один від одного.

Головною відмінністю від віртуалізації є те, що віртуальна машина створює віртуальне представлення апаратного забезпечення сервера, а при контейнеризації створюються віртуальне уявлення серверної операційної системи. На рисунку 2.8 схематично зображена відмінність контейнеризації та віртуалізації.

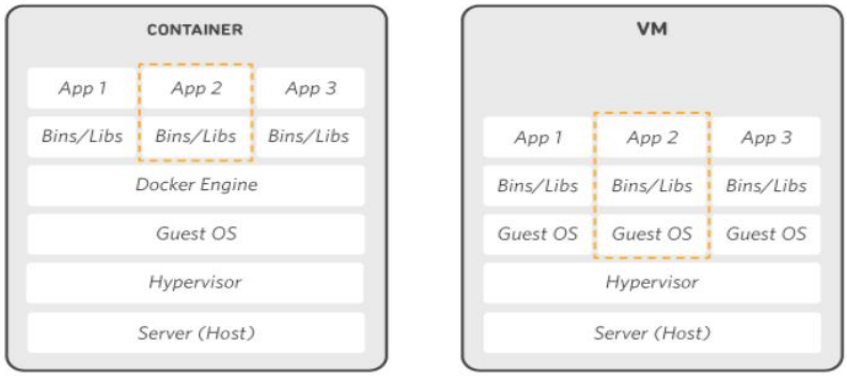


Рисунок 2.8 – Відмінність контейнеризації та віртуалізації

Наведемо переваги використання Docker:

1) Швидке розгортування програмного забезпечення. Так як Docker надає єдиний інтерфейс для збірки та розгортування програм, то швидкість розгортування зростає.
2) Стандартизація операцій. Доставка додатків в контейнерах спрощує процес розгортання, виявлення проблем та усунення до попередніх версій.
3) Легке переміщення. Контейнери Docker легко переміщувати між різними машинами.
4) Інтеграція з багатьма Cloud-сервісами.

Також варто згадати про docker-compose. Цей інструмент дуже корисний в системах з мікросервісною архітектурою. Він дозволяє конфігурувати створення образів та запускати декілька контейнерів за раз та оркеструвати їх. Для цього достатньо створити docker-compose.yml файл, в якому потрібно прописати всі сервіси, які мають бути запущені в рамках системи. На рисунку 2.9 зображений приклад такого файлу.

```yaml
version: "3.7"

services:
  app:
    image: node:12-alpine
    command: sh -c "yarn install && yarn run dev"
    ports:
      - 3000:3000
```

Рисунок 2.9 – Приклад docker-compose.yml файлу

Можна заключити, що Docker є дуже корисним інструментом для розробки систем на базі мікросервісної архітектури. Багато компаній та розробників використовують його для розробки своїх продуктів не тільки в мікросервісах але і в монолітних системах. З цієї причини Docker був використаний в даній роботі.

2.2.9 Unit-тестування

Юніт-тест (unit test)[20] - це програма, яка перевіряє роботу невеликої частини коду. Розробники регулярно оновлюють сайти та програми, розширюють функціональність, налагоджують код і вносять правки, а потім перевіряють, як все працює. І так як тестувати систему повністю після кожного оновлення доволі неефективно, тому були розроблені unit-тести, які перевіряють правильність виконання оновлених або виправлених частин коду.

Окрім unit-тестів існує багато інших видів тестування, наприклад, інтеграційне, E2E, регресійне і т.д.. Інтеграційне тестування перевіряє взаємодію декількох систем, E2E покриває весь сценарій використання програми з точки зору користувача, тобто перевірка авторизації, створення замовлення від початку до кінця тощо.

Unit-тести дуже важливі в розробці великих систем в довгостроковій перспективі. Так як з часом існуючий код змінюється, то витрачається багато часу на тестування однієї і тієї ж функціональності. Цю проблему і вирішують unit-тести – з ними вартість розробки, навпаки буде зменшуватись, так як менше часу буде витрачатись на тестування існуючої функціональності. На рисунку 2.10 зображені залежності вартості розробки від часу з unit-тестами та без.

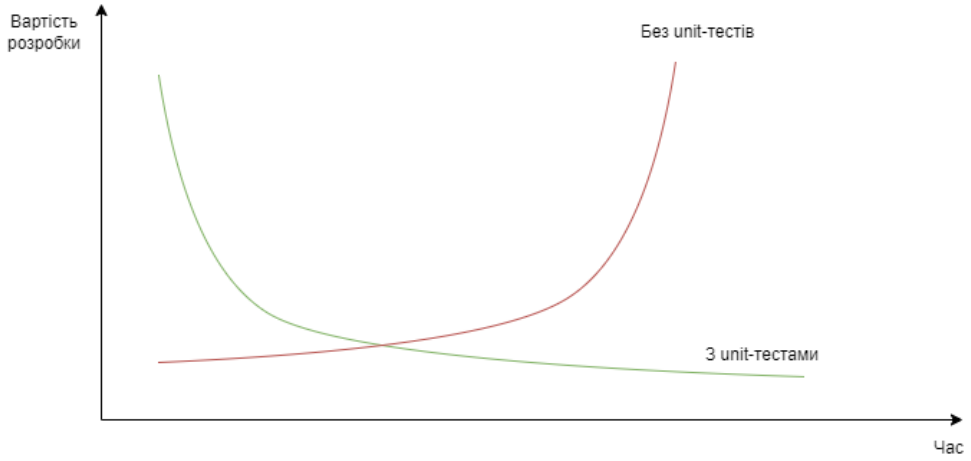


Рисунок 2.10 – Залежності вартості розробки від часу розробки

Для написання unit-тестів на Java гарно зарекомендував себе фреймворк Junit. Він може бути легко інтегрований в Spring застосунки, або використовуватися

окремо, має підтримку більшості інтеграційних середовищ розробки (IDE). Наведемо переваги JUnit:

1) Проста структура для написання автоматизованих тестів із самоперевіркою на Java;
2) Підтримка тестових затверджень (assertions);
3) Можливість розробки набору тестів (test suite);
4) Наявність звітності про тестування.

2.3 Технології клієнтської частини

Детально розглянемо технології, які використовуються на клієнтській частині проєкту.

2.3.1 Мова програмування TypeScript

TypeScript[21] – мова програмування з строгою явною статичною типізацією, розроблена 2012 року компанією Microsoft як альтернатива JavaScript для розробки веб-застосунків. TypeScript є тільки розширенням мови JavaScript, тобто вона базується на JavaScript та компілюється в JavaScript код. TypeScript був створений через слабку неявну динамічну типізація, яка не подобається багатьом розробникам, так як через неї виникає непередбачувана логіка при міксуванні типів. Також код написаний на мові з такою типізацією важче підтримувати. Мова TypeScript як раз і вирішує ці проблеми. І так як TypeScript – це надбудова над JavaScript, то розробники можуть використовувати бібліотеки, що написані для JavaScript, яких існує велика кількість. Також варто зазначити, що навідміну від JavaScript, в якому ООП реалізоване частково в TypeScript воно реалізовано повноцінно. В таблиці 2.2 наведено порівняння цих мов програмування.

Таблиця 2.2 – Порівняння JavaScript та TypeScript

Мова / Характеристика	JavaScript	TypeScript
Типізація	Неявна, слабка, динамічна	Явна, строга, статична
ООП	Частково реалізоване	Реалізовані всі принципи
Складність вивчення	Легкий для вивчення	Вимагає більше часу
Модульність	Не підтримується	Підтримується
Необхідність компіляції	Нема	Є
Спільнота розробників	Дуже велика	Менша, мова тільки розвивається

У висновку можна зазначити, що TypeScript більше підходить для розробки великих систем, а JavaScript для написання простих скриптів та недовгої розробки невеликих веб-додатків. Також, варто сказати, що TypeScript є мовою програмування за замовчування для фреймворку Angular. З цих причин вона і була обрана для роботи.

2.3.2 Мова розмітки гіпертексту HTML

HTML – мова розмітки гіпертексту для веб-сторінок, що розроблена Консорціумом Всесвітньої Павутини та підтримується й розширюється і зараз. Файли HTML відображаються в браузері на веб-сторінці у тому вигляді, в залежності від тексту html файлу. Основним поняттям в HTML є тег – це певна структурна одиниця, яка вказує, як буде виглядати зміст тегу. Існує багато тегів для різних задач. HTML майже завжди використовується з CSS для задання стилів.

2.3.3 Мова стилю CSS

CSS – мова описання стилю елементів HTML, в залежності від стилів відображених елементів буде відрізнятись. CSS дозволяє змінювати шрифти, колір тексту, форму кнопок, розташування елементів і багато іншого. Для застосування стилю CSS в документі HTML достатньо вказати шлях до файлу .css в тезі <style>. Деякі фреймворки або бібліотеки підставляють стилі автоматично.

2.3.4 Веб-фреймворк Angular

Angular[22] – фреймворк для створення веб-застосунків, розроблений 2016 року компанією Google. Angular повністю написаний на мові TypeScript та використовує її як основну мову, але дозволяється використовувати JavaScript.

Angular додатки складаються к компонентів. Компонента представляє собою папку з 4 файлами: виконуваний TypeScript файл, HTML документ, CSS стиль, файл з тестами для даної компоненти. HTML документ є представленням (view), яке змінює код в TypeScript файл, а CSS файл накладає стилі на відповідні елементи. Одна компонента може описувати всю сторінку або частину певної сторінки. Таким чином можна групувати компоненти на одній сторінці та не дублювати код, наприклад, щоб панель меню була однакова при відкритті інших сторінок.

Також Angular «з коробки» надає багато бібліотек та інструментів для роботи. Наприклад, власний http клієнт, засоби маршрутизації по сторінкам, підтримки сесії авторизації, для управління формами і багато іншого. Це великий плюс, так як не потрібно використовувати сторонні бібліотеки, які можуть викликати конфлікти при взаємодіїї міє собою. Також, Angular задає шаблон коду проєкту, таким чином легко розроблювати великі системи, так як зберігається єдина структура коду. З цієї причини Angular легко вивчається новачками, так як

вони розроблюють систему строго по вказаному шаблону, не задумуючись як треба правильно і так далі.

Також існують інші фреймворки та бібліотеки для розробки веб-застосунків. Розглянемо існуючі альтернативи, такі як React та Vue.

React - популярна бібліотека, широко використовується розробниками по всьому світу. Вважається доволі гнучкою, так як надає тільки каркас та базову функціональність. Це є і перевагою і недоліком, так як з однієї сторони можна швидко розроблювати веб-застосунки. З іншої сторони недостатність інструментів «з коробки» змушує використовувати сторонні інструменти, які викликають конфлікти версій тощо. Також, легко заплутатись в великих проєктах, так як немає єдиної структури.

Vue є найбільш сучасним та молодим з цієї трійки, тому є найменш популярним, але він доволі простий в освоєнні, а також не такий об’ємний, як Angular, тому доволі швидко набирає популярність. Він є гарною альтернативою React та Angular, так як є чимось середнім, має гарну документацію та швидко набирає спільноту розробників, що прискорює швидкість вивчення фреймворку та написання додатків.

Для даної роботи був обраний саме Angular, так як він має задану структуру проєкту та надає багато інструментів «з коробки».

2.3.5 Бібліотека візуалізації даних Plotly

Plotly[23] – сучасна бібліотека для візуалізації даних, побудови графіків, анімацій та інтерактивних діаграм. Існує версія для мови Python і для мов JavaScript та TypeScript. Розглянемо версію для JavaScript та TypeScript, яка називається Plotly.js.

Розглянемо основні переваги цієї бібліотеки:

1. За допомогою Plotly.js можна легко створювати інтерактивні діаграми. Будь-яка діаграма, оснащена такими функціями, як масштабування, панорамування, автомасштабування тощо. Також можна налаштувати певні дії на діаграмі при спрацюванні певної події та взаємодіяти з графіком.
2. Висока продуктивність при побудові великої кількості графіків, точок робить Plotly.js чудовим вибором, коли потрібно намалювати великий обсяг даних. Оскільки більшість бібліотек використовують SVG (англ. Scalable Vector Graphics), яка надає сумісність між браузерами та можливість експортувати високоякісні зображення будь-якого графіка. Однак залучення великої кількості елементів SVG у DOM може негативно позначитися на продуктивності. Plotly.js використовує stack.gl для створення високопродуктивних 2D та 3D-діаграм.
3. Всі створені 3D-діаграми за допомогою WebGL повною мірою використовуватимуть усі можливості, пропоновані графічним процесором.
4. Усі графіки Plotly.js легко налаштовуються. Все, від кольорів та міток до ліній сітки та легенд, можна налаштувати за допомогою набору атрибутів JSON. Ви дізнаєтесь, як налаштувати різні типи діаграм у наступних трьох частинах серії.
5. Гарний візуал створених графіків. Окрім того, що графіки можна зконфігурувати як завгодно, навіть з базовою конфігурацією вони виглядають сучасно.
6. Легко інтегрується в фреймворки Angular, Vue та бібліотеку React.

В роботі використовується бібліотека Plotly.js, так як вона проста в використанні, легко інтегрується з фреймворком Angular та має гарний візуал. Також при необхідності можна легко перенести конфігурацію графіків для мови програмування Python.

2.4 Методи аналітики даних

Розглянемо основні засоби, алгоритми та метрики для аналізу даних телекомунікаційних компаній, що були використані в роботі.

2.4.1 Когортний аналіз

Когортний аналіз[24] - це метод маркетингового дослідження поведінки користувачів, об'єднаних у когорти на підставі дій, за певний період. Когорта – група користувачів, об’єднаних певною ознакою. Такий підхід дозволяє:

- проаналізувати вплив прийнятих маркетингових рішень на рівень утримання;
- оцінити результативність рекламних кампаній;
- знайти найефективніші канали залучення покупців;
- знайти шляхи підвищення залучення користувачів з низькою активністю та мінімізувати відтік клієнтів;
- оцінити вплив сезонності до рівня продажів;
- побачити необхідність підключення нового каналу просування;
- ефективніше розподілити маркетинговий бюджет;
- краще пізнати цільову аудиторію.

Для більшого ефективності когортного аналізу варто також збирати деякі дані та розраховувати різні бізнес метрики, наприклад:

- ARPU кожного учасника когорти;
- перегляди сторінок сайту;
- статистика відвідувань сайту;
- повторні покупки;
- канали залучення користувачів;

- ROI;
- LTV/CLV;
- CAC;
- відтік клієнтів;
- дату покупки;
- кількість залучених покупців за певний період.

Процес підготовки когортного аналізу складається із 3 основних етапів. Розглянемо їх покроково:

- Визначити ознаки формування когорти. Прикладами ознак може бути відвідування сайту, купівлю товару, реєстрацію, куплений тип продукту, локація, сезон покупки, частота покупок, кількість покупок за певний час тощо.
- Встановити часовий інтервал, протягом якого користувачі потраплятимуть до однієї групи. Наприклад, день, тиждень, місяць, рік, квартал.
- Встановити ключові метрики. На основі цілей потрібно обрати необхідний маркетинговий показник. Наприклад, LTV, ROI, CAC, коефіцієнт утримання, час перебування на сайті, кількість транзакцій, середній прибуток та інше.

Після збору даних та розрахування метрик можна порівняти їх в різних когортах та зробити висновки.

У висновку можна сказати, що когортний аналіз це ефективний інструмент для аналітики бізнесу, але для його застосування потрібно бути кваліфікованим бізнес аналітиком та мати значний досвід. Також потрібно мати багато різних даних та розраховувати бізнес метрики, базуючись на них, це іноді важко реалізувати.

2.4.2 Бізнес метрики

Розглянемо основні бізнес метрики, формули для їх розрахунку та навіщо їх взагалі розраховувати та брати до уваги.

Бізнес-метрики - показники, за допомогою яких вимірюють ефективність розвитку бізнесу, розраховують показники витрати та прибутку. Без них неможливо зрозуміти, скільки грошей заробляє компанія, чи має бізнес фінансові проблеми чи навпаки — можна збільшити бюджет на розвиток. Щоб отримати точний результат дані поділяють на періоди для простеження динаміки.

Користь від розрахування бізнес метрик така сама як від когортного аналізу. А саме – оцінка результативності маркетингової компанії, мінімізація відтоку клієнтів, максимізація притоку клієнтів, визначення акценту для збільшення ефективності бізнесу тощо. Також, варто зазначити, що бізнес метрики є інструментом когортного аналізу – це взаємопов'язані сутності. Виділяють такі бізнес метрики[25]:

1. CPO (англ. Cost per Order) – вартість залучення одного замовлення. Показує, скільки, в середньому, коштів витратила компанія на залучення замовлення від одного покупця. Розрахування CPO для кожого рекламого каналу або оголошення допоможе правильно планувати витрати на рекламу в майбутньому. А також знаходити та відключати канал, які приносять більше витрат, ніж виручки. CPO розраховується за формулою:

$$CPO = C_M / N_O \tag{2.1}$$

де C_M – загальний бюджет на рекламу,

N_O – кількість замовлень на період часу.

2. CAC (англ. Customer Acquisition Cost) – вартість залучення клієнта, тобто сума, яку компанія платить для залучення нового клієнта за певний період. Майже те саме, що CPO, тільки розраховується для клієнтів, а не замовлень.

CAC розраховується за формулою:

$$CAC = C_M / N_U \quad (2.2)$$

де C_M — загальний бюджет на рекламу для залучення клієнта,

N_U — кількість залучений клієнтів на період часу.

3. ROI (англ. Return on Investment) – окупаємість інвестицій, показує відсоток окупності інвестицій компанії. Допомагає розрахувати рентабельність бізнесу загалом, з урахуванням усіх витрат, наприклад, витрат за логістику, зарплатню співробітників і т.д.. ROI розраховується за формулою:

$$ROI = (C_P - C_C)/C_C \quad (2.3)$$

де C_P — прибуток компанії за певний період часу,

C_C — витрати компанії за певний період часу.

Якщо показник ROI більше нуля, означає, що бізнес приносить прибуток. Нижче за нуль — бізнес збитковий. А якщо ROI = 0, то компанія досягла точки беззбитковості.

Зазвичай підрахунок повернення інвестицій спрощують і рахують лише витрати на маркетинг, не враховуючи, наприклад, витрати на оренду офісу або роботу кур'єра.

4. AOV (англ. Average Order Value) – середня вартість замовлення, тобто сума, яку компанія в середньому заробляє з одного замовлення за певний період. Також її ще називають «середній чек». Знаючи середній чек, компанія може прогнозувати прибуток.

Показник AOV можна збільшувати за рахунок продажу супутніх товарів. Наприклад, при продажі великогабаритної техніки пропонувати клієнтам купити страховку, при продажі масляних фарб — пензлики. Стимулювати зростання середнього чека для компанії завжди вигідніше, ніж залучати нових клієнтів. Готовому до покупки клієнту не потрібно ще раз знайомитися

з компанією, читати відгуки інших покупців чи шукати переваги товару цього бренду. AOV розраховується за формулою:

$$AOV = C_P / N_O \quad (2.4)$$

де C_P — загальний прибуток компанії за певний проміжок часу,

N_O — кількість замовлень за певний проміжок часу.

5. GMV (англ. Gross Merchandise Volume) – загальна вартість проданого товару за певний проміжок часу. Цю метрику можна рахувати по різному, в залежності від зберігаємих даних.
6. ARPU (англ. Average Revenue per User) - середній дохід з користувача, показує прибуток, який компанія в середньому отримує з одного користувача за конкретний період часу. Наприклад, один підписник сервісу музики в середньому платить 5$ на місяць. Метрика ARPU допомагає порівняти середній виторг з одного користувача за різні періоди часу та виміряти реакцію клієнтів на зміну ціни на продукт. ARPU розраховується за формулою:

$$ARPU = C_P / N_U \quad (2.5)$$

де C_P — загальний прибуток компанії за певний проміжок часу,

N_U — кількість унікальних користувачів за певний проміжок часу.

7. LTV (англ. LifeTime Value) або CLV (англ. Customer Lifetime Value) – життєва цінність клієнта, показує прибуток, яку клієнт принесе компанії за весь час. CLV допомагає визначити, скільки витрачати на залучення клієнта - витрати на залучення клієнта можуть перевищувати дохід з першої покупки, тому слід розуміти, чи продовжить клієнт приносити прибуток в довгостроковій перспективі. Також CLV дозволяє сегментувати клієнтів на основі цінності, тобто можна роботи VIP-клієнтам подарунки та спецпропозиції, щоб підвищити шанси на утримання, або залучати нових покупців зі схожими характеристиками.

Загально прийнято, що CLV розраховується за формулою:

$$CLV = AOV * K_F * C_{GM} * 1/K_{Ch} \quad (2.6)$$

де K_F – коефіцієнт частоти покупок, що розраховується за формулою 2.7,

C_{GM} – валова марж,

K_{Ch} - коефіцієнт відтоку клієнтів.

$$K_F = N_O / N_U \quad (2.7)$$

Підставивши в формулу 2.6 формули 2.4 та 2.7 маємо:

$$CLV = \frac{C_P}{N_O} * \frac{N_O}{N_U} * C_{GM} * 1/K_{Ch} \quad (2.8)$$

Як бачимо кількість замовлень скорочується і ми маємо формулу 3.5 для розрохунку ARPU. Після скорочень і підстановок маємо кінцеву формулу:

$$CLV = ARPU * C_{GM} * 1/K_{Ch} \quad (2.9)$$

Були розглянуті найбільш популярні та корисні бізнес метрики. Бізнес метрики часто використовуються бізнес аналітиками для просування бізнесу. В даній роботі розраховуються GMV, AOV, ARPU, CLV в виду специфіки зберігаємих даних.

2.4.3 Алгоритми прогнозування

Задача прогнозування є однієї з основних задач в машинному навчанні. Популярний метод вирішення є регресія, але існують і інші методи і алгоритми для прогнозування даних. Прогнозувати можна будь-які дані в часових рядах, наприклад, прибуток, витрати, кількість покупців тощо. Існує багато алгоритмів, але розглянемо регресію Lasso, Gradient Boosting, Decision Tree та Prophet[26].

1. Lasso регресія

Регресія - це метод, для прогнозування даних на основі вибірки даних, або з математичної точки зору – це метод вибору функції з певного їх сімейства, яка мінімізує функцію втрат. Інсує багато видів регресій, такі як лінійна, логістична, Lasso, Ridge. Lasso (англ. Least Absolute Shrinkage and Selection Operator) регресія базується на лінійній регресії. Лінійну регресію виражається формулою:

$$f(x) = a + b * x \tag{2.10}$$

де f(x) ─ функція регресії або функція відгуку,

x ─ незалежна змінна,

a, b - коефіцієнтами регресії оціночної лінії.

Ціллю регресії є знаходження коефіцієнтів a та b. Для цього використовується, як правило, метод найменших квадратів, суть якого є мінімізація функції втрат:

$$MSE = \sum_i (y_i - f(x_i))^2 \tag{2.11}$$

де MSE ─ середня квадратична помилка,

y_i ─ значення вибірки,

$f(x_i)$ – значення функції регресії в точці x_i.

Lasso регресія відрізняється тільки функцією втрат, яка виглядає:

$$MSE = \sum_i (y_i - f(x_i))^2 + \lambda * |\omega| \tag{2.12}$$

де λ - гіперпараметр, який налаштовується вручну. Чим більше лямбда, тим сильніше модель штрафується за величину коефіцієнтів,

$|\omega|$ - модуль коефіцієнтів.

Таким чином отримуємо покращену модель лінійнії регресії, яка може ефективно виконувати поставлені задачі.

2. Gradient Boosting

Gradient Boosting – це метод машинного навчання, що використовується у задачах регресії та класифікації. Він дає модель прогнозування у формі ансамблю слабких моделей, які є деревами рішень, як правило. Модель дерев із посиленням градієнта будується поетапно і узагальнює інші методи, дозволяючи оптимізувати довільну диференційовану функцію втрат. Математична формула алгоритма є:

$$F(x) = argmin \sum_i L(y_i, \gamma) \quad (2.13)$$

де F(x) – кінцева функція Gradient Boosting,

$L(y_i, \gamma)$ – функція втрат вкладеної моделі.

Для функції втрат, як правило використовується середньоквадартична помилка.

3. Decision Tree

Decision Tree – це деревоподібний класифікатор із 3 типами вузлів. Кореневий вузол — це початковий вузол, від якого відходять внутрішні вузли. Внутрішні вузли представляють умови прийняття рішень. Листові вузли представляють результат, тобто певне вихідне значення. На рисунку 2.11 зображена структура вузлів дерева рішень.

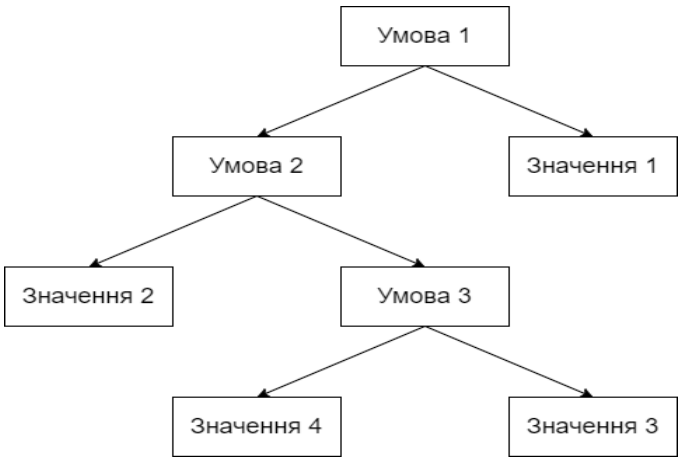


Рисунок 2.11 – Приклад побудови дерева рішень

Цей алгоритм використовується для вирішення проблем, пов’язаних із прийняттям рішень. Він використовується як для задачі регресії так і для задачі класифікації.

4. Prophet

   Prophet – сучасна модель для прогнозування даних, розроблена компанією Facebook для своїх задач. Виділяється від інших моделей врахуванням сезонності, історичності, канікул для прогнозування даних та трендів не лінійних даних. Математична модель алгоритму виражається формулою:

$$y(t) = g(t) + s(t) + h(t) \quad (2.14)$$

   де y(t) – вихідне значення моделі,

   g(t) – функція, що відображає темп загального росту,

   s(t) – функція, що враховує сезонність даних,

   h(t) – функція, що враховує канікули.

   Функція росту може мати лінійну регресію в основі або логістичну. Функція сезонності базується на перетворенні Фур'є. Функція канікул враховує канікули тої чи іншох країни, також їх можна задати вручну.

Ці алгоритми використовуються в роботі, так як вони показують гарні результати, представляють різні методи вирішення задачі прогнозування, що є корисним для порівняння їхньої роботи для знаходження найбільш ефективного в роботі.

Висновки до розділу 2

Існує дуже багато архітектур, технологій та методів для розробки систем та аналізу даних. Правильний вибір технологій відіграє важливу роль в успіху проєкту, так як через непривильний вибір ускладнюється розробка та витрачається більше ресурсів ніж могло б бути. Тому був проведений детальний аналіз існуючих архітектур, мов програмування, фреймворків, методів бізнес аналізу, алгоритмів прогнозування та інших технічних засобів для вирішення поставленої задачі. В даному розділі були детально описані обрані засоби розробки та основні причини їх вибору. Заданий стек технологій є ефективним набором для розробки проєкту.

3 РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Загальна структура системи

На рисунку 3.1 зображена загальна архітектура (структура) системи:

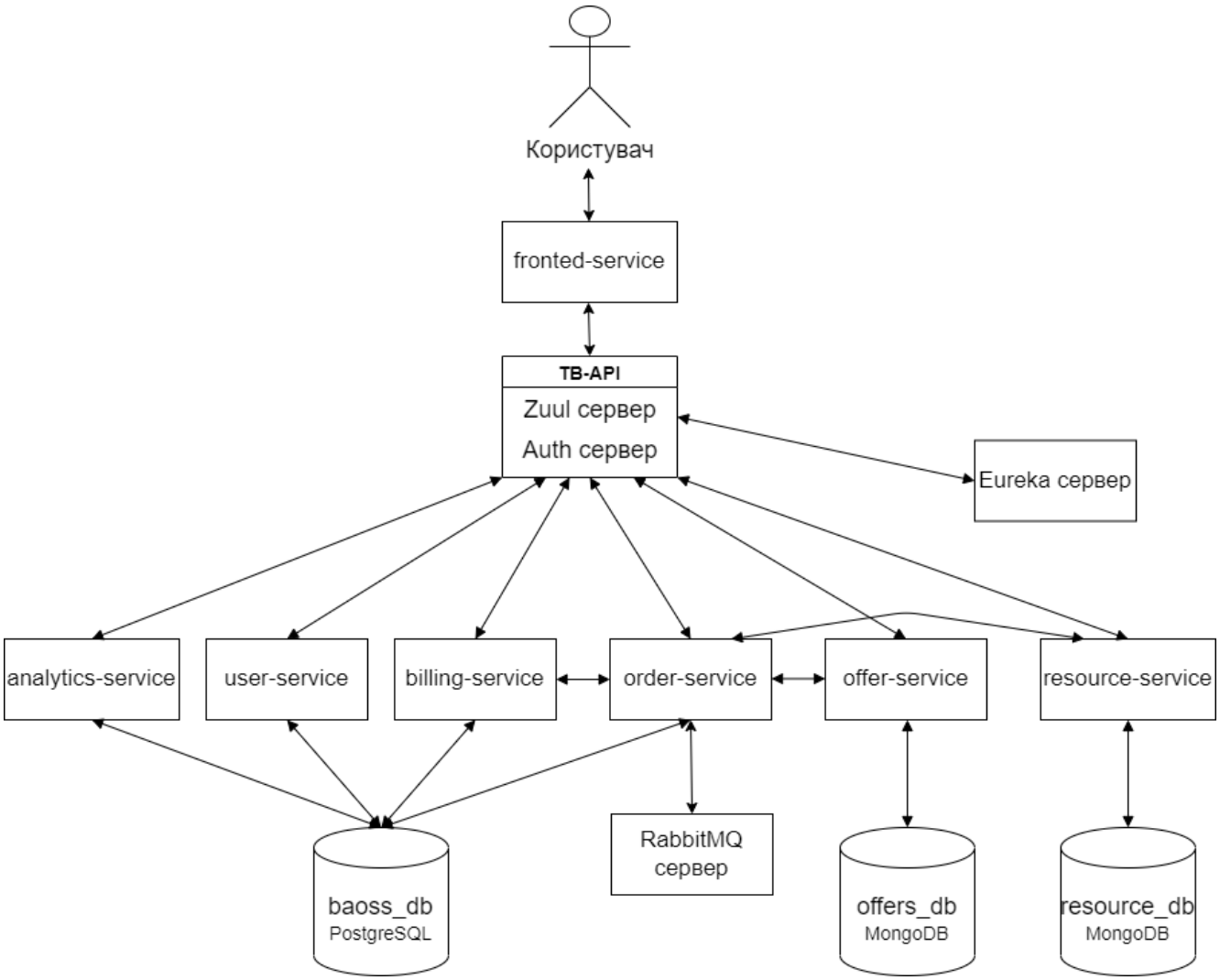


Рисунок 3.1 – Схема загальної структури системи

Система складається з 7 мікросервісів, шлюзу маршрутизації (TB-API), серверу реєстрації мікросервісів (Eureka сервер), 3 баз даних та серверу брокеру повідомлень RabbitMQ. Користувач працює в браузері напряму з frontend-service, який звертається до шлюзу маршрутизації TB-API, який реалізований за допомогою Spring Cloud Netflix Zuul компоненти. Також цей сервіс реалізує

авторизацію, щоб не виносити її в окремий сервіс. Eureka сервер використовується для реєстрації мікросервісів та надання ТВ-API IP адрес відповідних мікросервісів. Мікросервіси analytics-service, user-service, billing-service, order-service використовують одну реляційну базу даних baoss_db, яка підтримується СУБД PostgreSQL. Мікросервіс offer-service використовує нереляційну базу даних offers_db, яка підтримується NoSQL СУБД MongoDB. Мікросервіс resource-service використовує також нереляційну базу даних resource_db, яка підтримується СУБД MongoDB.

3.2 Моделі баз даних

Детально розглянемо схеми баз даних, які використовюються в системі.

3.2.1 База даних baoss_db

На рисунку 3.2 зображена ER діаграма бази даних baoss_db. Детально опишемо таблиці:

- users – таблиця з даними користувача. Використовується переважно мікросервісами user-service та analitycs-service.
- payments – таблиця з транзакціями, тобто платежами за послуги. Використовується тільки мікросервісами billing-service та analitycs-service.
- instances – таблиця екземплярів продуктів користувачів. Тобто, коли користувач має пікдлючений Інтернет, відображається запис в цій таблиці, що для користувача активний екземпляр Інтернет продукту. У користувача може бути декілька активних продуктів, тому таблиця має зовнішній ключ на таблиц users.

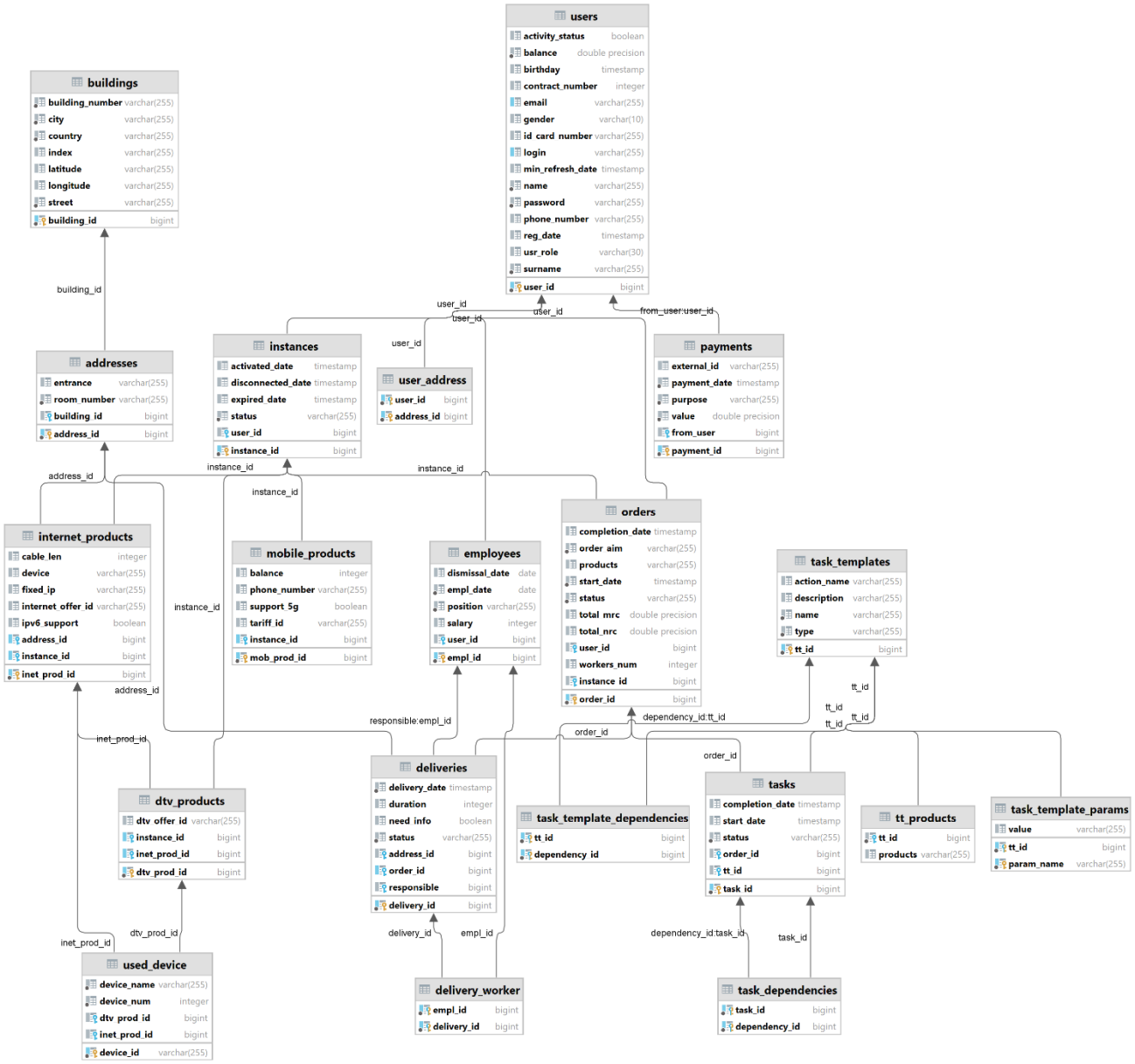


Рисунок 3.2 – ER діаграма бази даних baoss_db

- orders – таблиця замовлень. Так як у користувача може бути декілька замовлень, тому таблиця має зовнішній ключ на таблицю users. Також має зовнішній ключ на таблицю instances, так як одне замовлення активує певний продукт.
- tasks – таблиця задач, які повинні виконатися для виконання замовлення. Так як у замовлення може бути багато задач, тому має зовнішній ключ на таблицю orders. Задачі створюються з шаблону, який зберігається в task_templates.

- task_dependencies – таблиця, що зберігає залежності між задачами. Адже певна задача не може бути виконана до виконання задачі, від якої вона залежить.
- task_templates – таблиця шаблону задач. В ній зберігаються назви та описання для всіх задач, які можуть бути виконані для будь-якого замовлення.
- tt_products – таблиця, яка використовується для вказання, для активації яких продуктів повинна бути створенна задача.
- task_template_params – таблиця, яка зберігає параметри шаблону задачі.
- task_template_dependencies – таблиця, що зберігає залежності між шаблонами задачами.
- buildings – таблиця, що зберігає інформацію про адреса будинків користувачів, а також ідентифікатори записів зберігаються в resource_db для підключених будинків.
- addresses – таблиця адрес користувачів, містить тільки поля під'їзду та квартири. Також містить зовнішнє поле на таблицю buildings, так як у будинка може бути багато квартир.
- user_address – таблиця для зв'яку Many to Many для таблиць users та addresses, так як один користувач може мати декілька адрес проживання і за однією адресою може проживати декілька людей.
- mobile_products – таблиця, що зберігає дані мобільного продукту користувачів. Зберігає id тарифу, інформація про який зберігається в offers_db.
- internet_products – таблиця, що зберігає дані інтернет продукту користувачів. Має зовнішні поля на таблицю addresses для зберігання адреси до якої підключений, та id пропозиції, інформація про яку зберігається в offers_db.

- dtv_products – таблиця, що зберігає дані продукту цифрового телебачення користувачів. Має зовнішній ключ на таблицю internet_products, так як цей продукт не може бути піключений без Інтернету. Також має id пропозиції кількості каналів, інформація про яку зберігається в offers_db.
- employees – таблиця даних співробітників компанії. Має зовнішній ключ на таблицю users, так як співробітник також може користуватися системою.
- deliveries – таблиця, що зберігає інформацію про час, місце та монтажників, які виконують певне замовлення. Має зовнішні ключі на таблиці addresses, orders та employees, останній, щоб вказати головного монтажника.
- delivery_worker – таблиця для зв'язку Many to Many між таблицями deliveries та employees, так як підключенням інтернету та DTV займаються декілька монтажників, а також, очевидно, що у одного робітника може бути багато доставок.

3.2.2 База даних offers_db

Детально розглянемо колекції нереляційної бази даних offers_db, яка підтримується СУБД MongoDB та використовується мікросервісом offer-service. Вона зберігає дані про пропозиції та тарифи, а також ціни на інші послуги та дисконти. На рисунку 3.3 схематично зображені всі колекції цієї бази даних. Розглянемо кожну колекцію:

- internet_price_list – колекція, яка зберігає пропозиції для Інтернет продукту. Має поля суми щомісячного платежу та швидкості Інтернету.
- dtv_price_list – колекція, яка зберігає пропозиції для DTV продукту. Має поля суми щомісячного платежу та кількості доступних каналів.

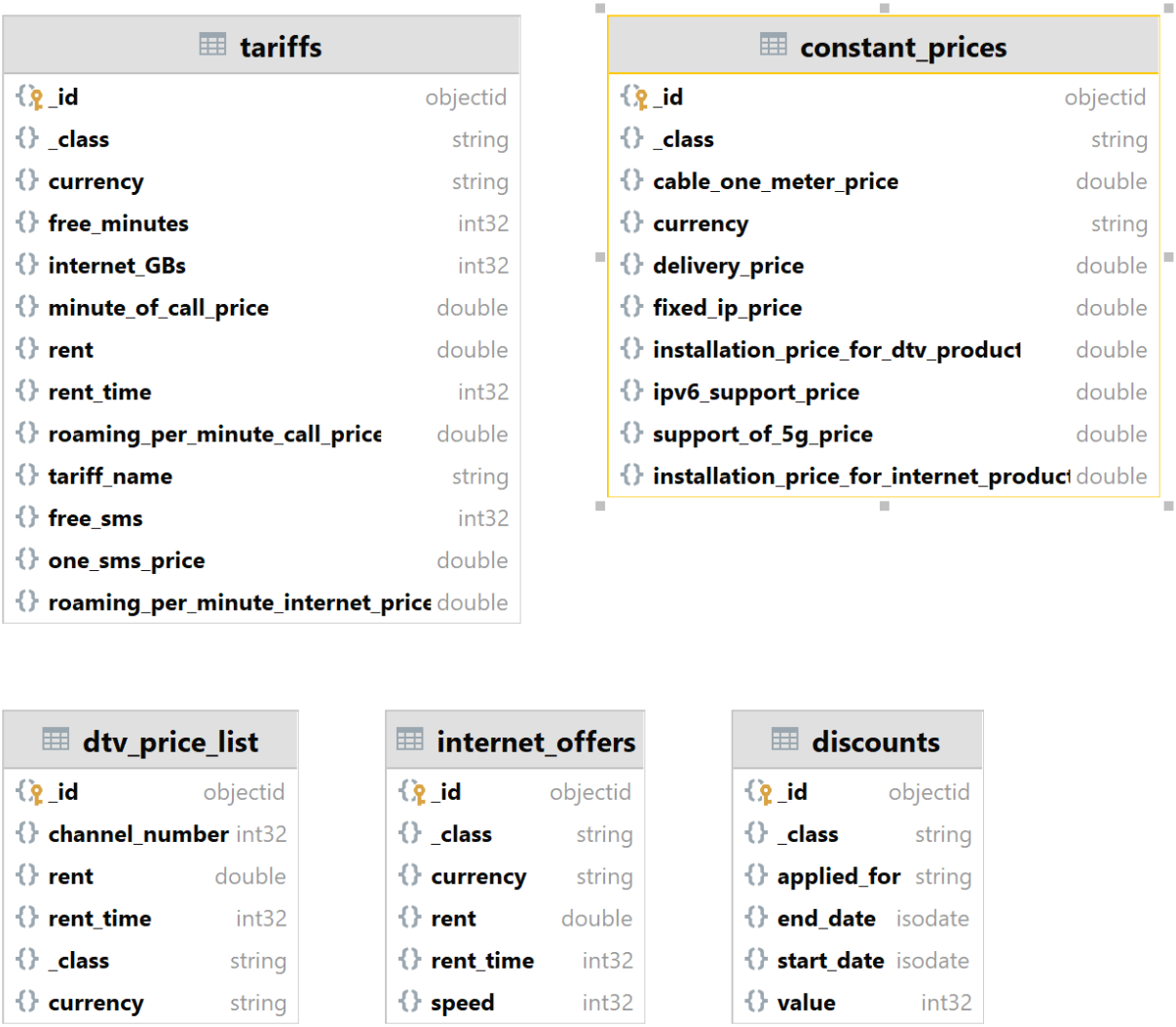


Рисунок 3.3 – Колекції бази даних offers_db

- tariffs – колекція, яка зберігає тарифи. Має поля назви тарифу, суми щомісячного платежу, кількості вільних гігабайтів інтернету, кількості вільних хвилин тощо.
- discounts – колекція, яка зберігає акції та дисконти. Має поля знижки, дати початку дії знижки, дати закінчення дії знижки та списку назв продуктів для яких діє знижка.
- constant_prices – колекція, яка зберігає ціни на єдиноразові послуги, наприклад, доставка, ціна фіксованої IP адреси, підключення Інтернету/DTV, ціна за 1 метр кабелю тощо.

3.2.3 База даних resource_db

Детально розглянемо колекції нереляційної бази даних resource_db, яка підтримується СУБД MongoDB та використовується мікросервісом resource-service. База даних зберігає дані про ресурси, якими оперує компанія. Наприклад, SIM карти, комутатори, маршрутизатори, кабелі, підключені будинки до мережі, зарезервовані фіксовані IP адреси. На рисунку 3.4 схематично зображені всі колекції цієї бази даних.

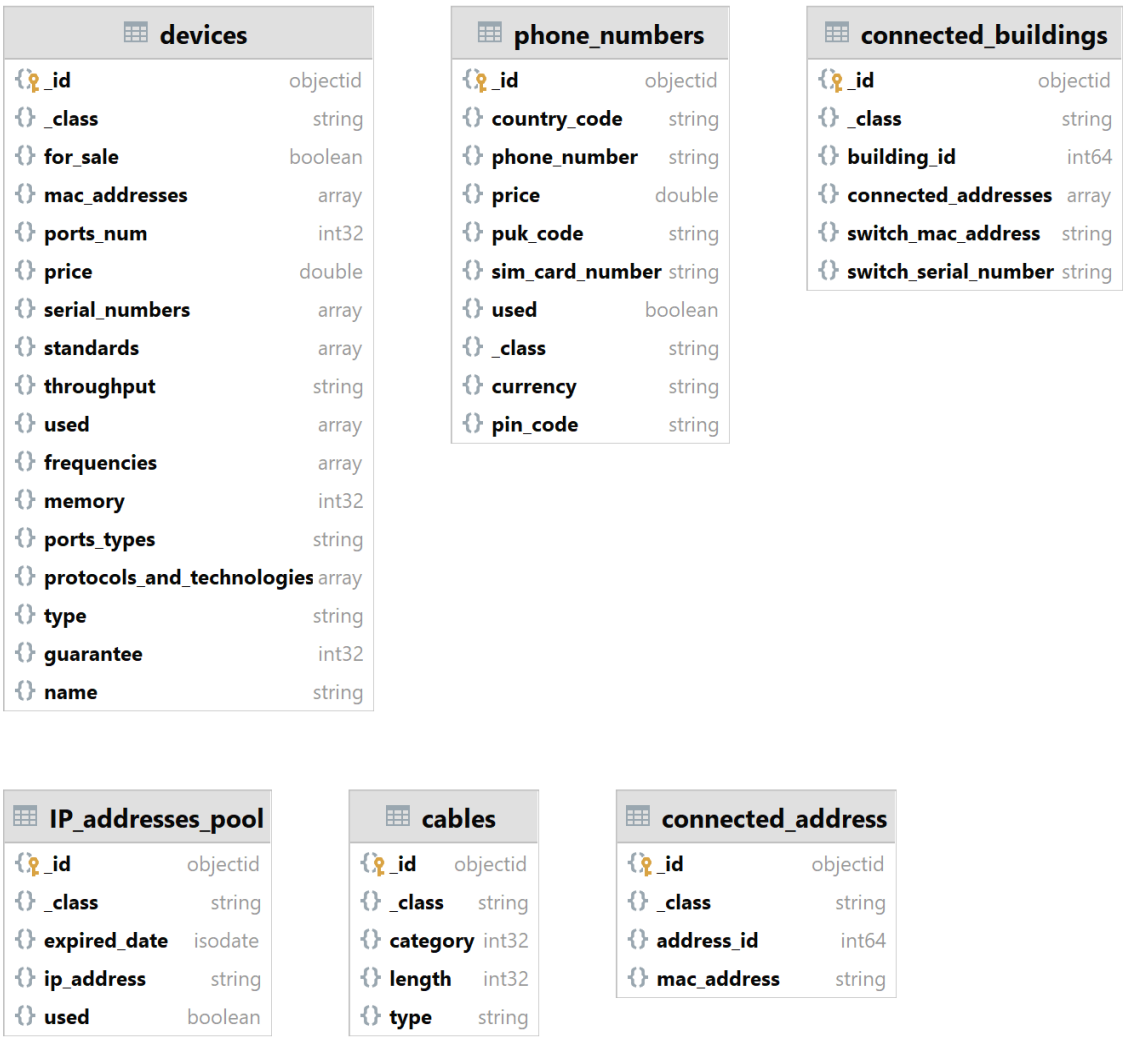


Рисунок 3.4 – Колекції бази даних resource_db

Опишемо кожну колекцію:

- phone_numbers – колекція, яка зберігає дані про SIM карт. Містить поля телефонного номеру, серійного номеру SIM карти, ціни телефонного номеру, коду країни та флажку використання даної SIM карти.
- IP_addresses_pool – колекція, яка зберігає зарезервовані фіксовані IP адреси. Містить поля IP адреси, флажку використання певним користувачем, дати закінчення резерву IP адреси.
- devices – колекція, яка містить інформацію про пристрої провайдера, наприклад, комутатори та марушрутизатори. Містить поля назви пристрою, типу пристрою, ціни, технічної інформації про пристрій, масив MAK-адрес, масив серійних номерів, масив флажку використання певним користувачем.
- cable – колекція, яка зберігає інформацію про кабелі, які використовує компанія. Містить поля типу (вита пара, коаксіальний, оптоволоконний тощо). та довжини кабелю
- connected_buildings – колекція, яка зберігає інформацію про будинки, які підключені до мережі провайдера. Містить поля списку підключених адрес, що предсавляє собою список записів колекції connected_address, MAC адреси комутатора, що установлений в будинку та його серійного номеру.
- connected_address – колекція, що зберігає інформацію про підключені до мережі адреси користувачів. Містить поле id адреси користувача та MAK адреси пристрою користувача (роутера чи комп’ютера).

3.3 Реалізація серверної частини

Серверна частина складається з 6 функціональних мікросервісів та 2 допоміжних технічних серверів (TB-API, Eureka). Розглянемо детально функціональні мікросервіси.

3.3.1 Мікросервіс user-service

Цей мікросервіс відповідає за роботу з даними користувача, а саме додавання та видалення адрес, пошук користувача. Також цей мікросервіс використовує сервер авторизації. Мікросервіс надає такий API:

1) GET */api/auth/user?login* – шукає користувача за переданим логіном.
2) PUT */api/auth/min-refresh-date?login&date* – змінює дату останньої операції користувача в системі за логіном користувача. Потрібен для оновлення токена.
3) POST */api/auth/user* – реєструє нового користувача, приймає об’єкт користувача в тілі запиту.
4) GET */api/users* – повертає список всіх користувачів, що зареєстровані в системі.
5) POST */api/users/{userId}/address* – додає адресу для користувача з ID, який передається в URL, тіло запиту містить об’єкт адреси, яку потрібно додати.
6) DELETE */api/users/{userId}/address/{addressId}* – видаляє адресу для користувача.
7) PUT */api/users/user* – оновлює дані користувача, тіло запиту містить об’єкт користувача.

«Кінцеві точки» (англ. endpoints) 1, 3 доступні, будь-якому користувачу, тобто авторизованому і неавторизованому, решта – тільки авторизованим.

Розглянемо структуру мікросервісу, яка представлена у вигляді UML діаграми компонентів системи на рисунку 3.5. На ній ми наявно бачимо реалізацію багаторівневої архітектури. Рівень доступу до даних преставлений класами AddressRepository, BuildingRepository, UserRepository, які оперують відповідними сутностями. Рівень бізнес логіки представлений класми ValidationService та UserService. ValidationService, в свою чергу, відповідає за перевірку даних користувача при реєстрації, чи всі поля заповнені згідно з правилами тощо.

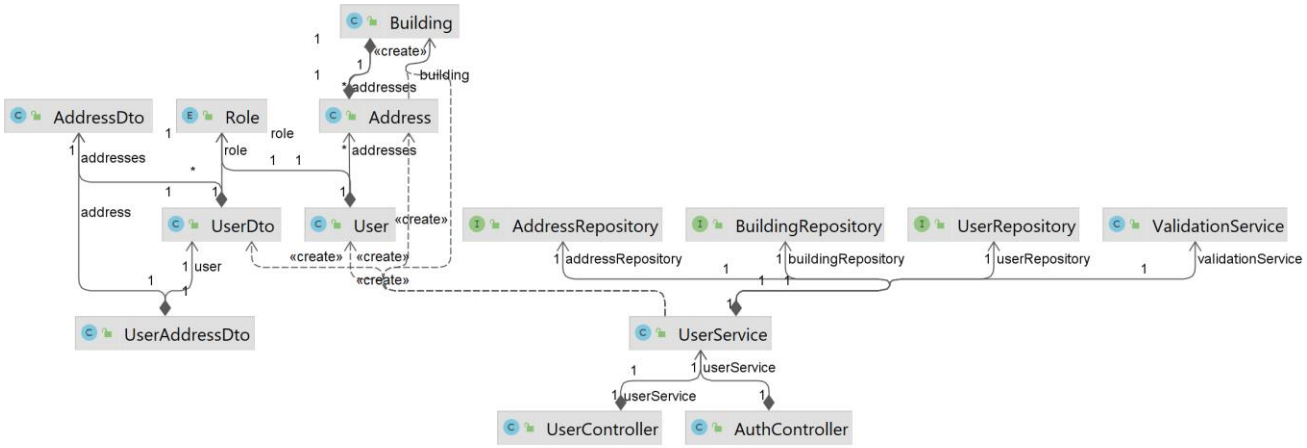


Рисунок 3.5 – UML діаграма компонент мікросервісу user-service

В рівень представлення входять класи UserController, AuthController, які оперують класами DTO (англ. Data Transfer Object) AddressDto, UserDto.

3.3.2 Мікросервіс billing-service

Даний мікросервіс реалізує білінг, тобто оплату телекомунікаційних послуг. API цього мікросервісу викликає мікросервіс order-service для оплати послуги при її підключенні. Також кожен день виконується метод, який проводить місячну оплату послуг. Детально розглянемо API, яке надає мікросервіс:

1) PUT */api/nrc-payment?userId&totalNRC* – виконує NRC платіж. Переказ здійснюється з рахунку користувача на рахунок провайдера на суму, що указана в параметрі запиту.
2) GET */api/check-nrc-payment?userId&totalNRC* – перевіряє чи можна провести NRC платіж.

Розглянемо структуру мікросервісу, яка представлена у вигляді UML діаграми компонентів системи на рисунку 3.6.

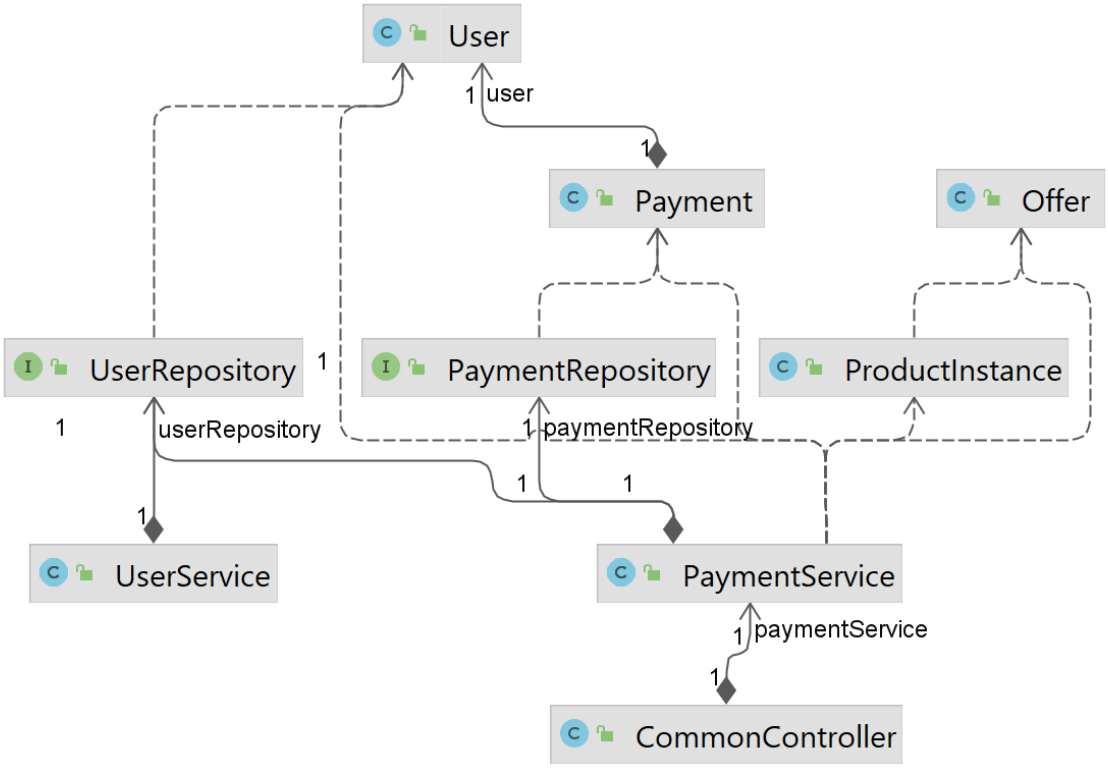


Рисунок 3.6 – UML діаграма компонент мікросервісу billing-service

Payment, User – сутності моделі даних та відповідають відповідним таблицям. Offer, ProductInstance – DTO, що використовуються для отримання інформації про продукти, які підключені у користувача. Рівень доступу до даних представлений класами PaymentRepository, UserRepository. Рівень бізнес логіки представлений класами UserService та PaymentService.

3.3.3 Мікросервіс order-service

Мікросервіс order-service є найбільшим в системі та має широку функціональність. Він відповідає за створення та виконання замовлень користувачів, активацію екземплярів продуктів та перевірки, які потрібні при цьому. Мікросервіс використовує RabbitMQ сервер для асинхронної взаємодії під час створення замовлення. Детально розглянемо API, що надає мікросервіс:

1) POST */api/orders* – створює замовлення та запускає потік його виконання. Тіло запиту містить об’єкт параметрів замовлення.
2) GET */api/orders/{orderId}* – шукає об’єкт замовлення зі всіма параметрами за переданим ID.
3) GET */api/orders?userId* – повертає список замовлень користувача з ID, що переданий в параметрах запиту.
4) GET */api/instances?userId* - повертає список екземплярів продуктів користувача з ID, що переданий в параметрах запиту.
5) GET */api/instances/mobile-product/{mobileProductId}* – повертає екземпляр мобільного продукту з всіма параметрами за вказаним ID.
6) GET */api/instances/internet-product/{internetProductId}* – повертає екземпляр Інтернет продукту з всіма параметрами за вказаним ID.
7) GET */api/instances/dtv-product/{dtvProductId}* – повертає екземпляр DTV продукту з всіма параметрами за вказаним ID.
8) GET */api/deliveries/available-times?deliveryDate&products* – повертає список годин, для яких працівники можуть підключити продукт в залежності від загруженості працівників.
9) GET */api/deliveries/{userId}* – повертає список доставок для працівника з ID, що переданий в URL запиту.
10) PUT */api/deliveries/finish* – завершує доставку та виконує задачі, які потрібні для остаточного підключення продукту. Тіло запиту містить об’єкт параметрів замовлення.

Всі кінцеві точки доступні тільки для авторизованого користувача.

На рисунку 3.7 представлена UML діаграма основних компонентів мікросервісу, так як даний мікросервіс оперує багатьма класами і повна діаграма буде занадто великою.

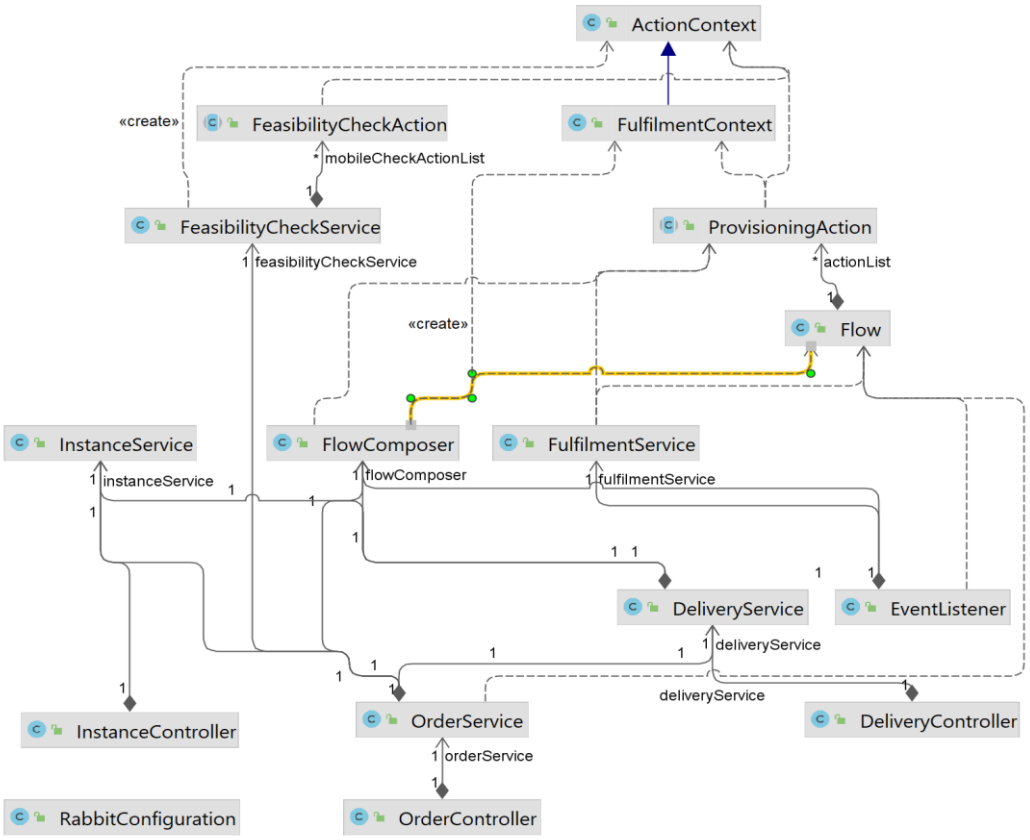


Рисунок. 3.7 – UML діаграма основних компонентів мікросервісу order-service

Детально розглянемо функціональність створення замовлення. На рисунку 3.8 зображений процес створення замовлення та запуску потоку його виконання.

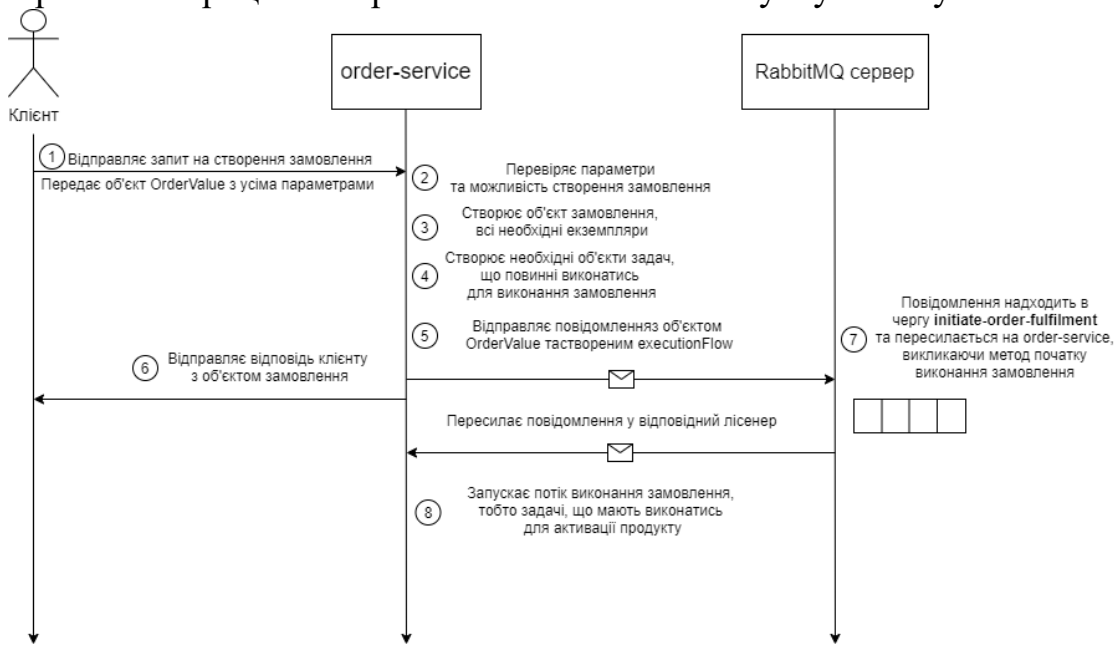


Рисунок 3.8 – Загальна схема процесу створення та виконання замовлення

Користувач заповнює всі необхідні параметри та відправляє об’єкт з параметрами замовлення в order-service. Спершу, перевіряється можливість

виконання замовлення з такими параметрами, наприклад, чи не зарезервував уже інший користувач обрано SIM карту, чи підключений будинок користувача до мережі провайдера тощо. Потім створюються всі необхідні екземпляри та зберігаються в базі даних. Після цього об'єкт параметрів замовлення відправляється в брокер повідомлень RabbitMQ в чергу initiate-order-fulfilment та відправляється відповідь для користувача. Водночас лісенер зчитує повідомлення з черги та запускає потік виконання замовлення.

Перевірка можливості виконання замовлення (англ. Feasibility Check) перед його виконанням реалізована за допомогою шаблону Ланцюжок Відповідальності (англ. Chain of Responsibility). На рисунку 3.9 зображена дана реалізація.

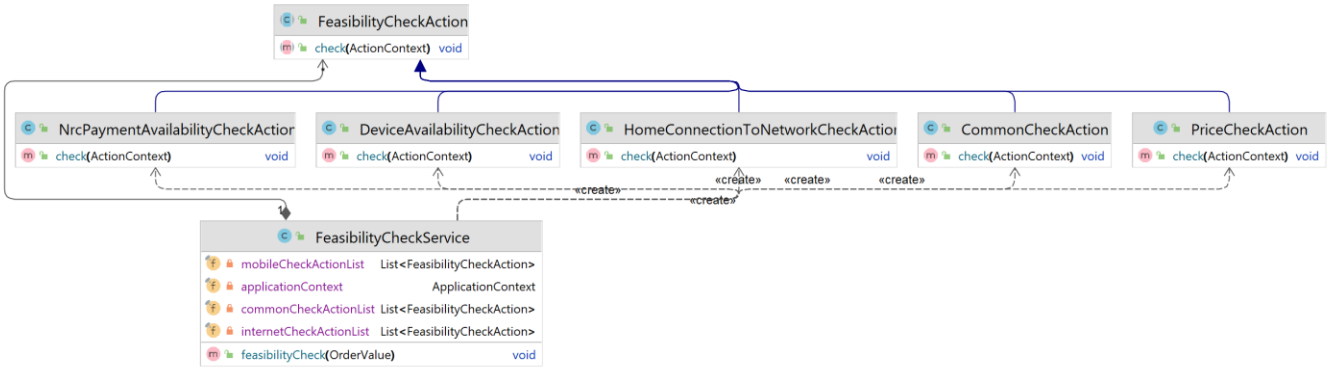


Рисунок 3.9 – UML діаграма класів функціональності перевірки можливості виконання замовлення

На діаграмі бачимо абстрактний клас FeasibilityCheckAction, що має один метод check(ActionContext), та який розширюють конкретні класи перевірок: NrcPaymentAvailabilityCheckAction, DeviceAvailabilityCheckAction, PriceCheckAction, CommonCheckAction, HomeConnectionToNetworkCheckAction та інші класи, що не попали в діаграму для збереження її читабельності. Клас FeasibilityCheckService створює екземпляри цих класів та зберігає їх в списках mobileCheckActionList, internetCheckActionList, commonCheckActionList. Метод feasibilityCheck(OrderValue) створює список всіх необхідних перевірок, базуючись на продуктах, які замовив користувач, та викликає метод check() з конкретними перевірками. Якщо певна перевірка не пройшла, то метод кидає виключення, яке

перехоплюється обробником виключень та повертається код 400 BAD REQUEST клієнта з повідомленням, що не так.

Окрім цього, шаблон Ланцюжок Відповідальності реалізований і для виконання потоку замовлення (англ. Order Fulfilment). На рисунку 3.10 зображена UML діаграма компонентів цієї функціональності.

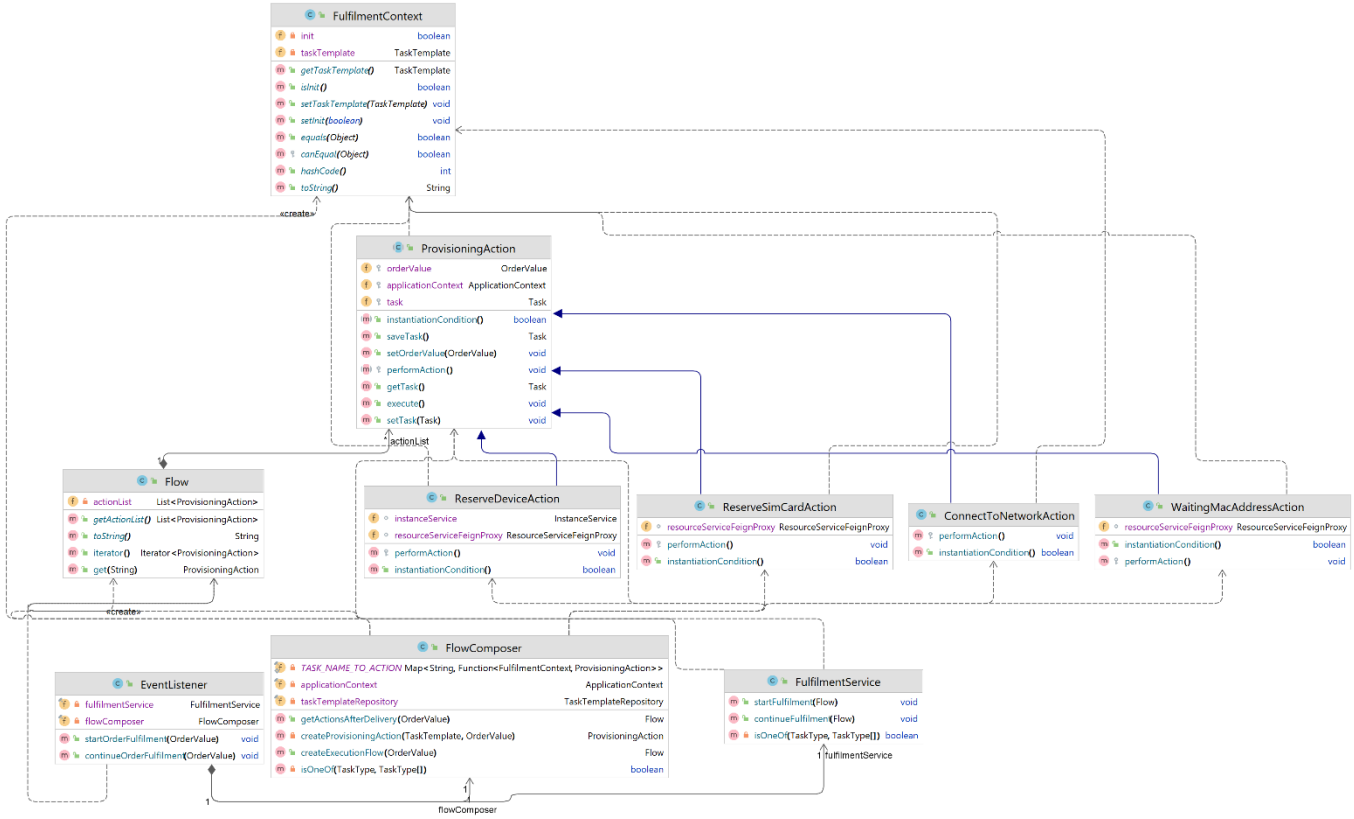


Рисунок 3.10 – UML діаграма класів функціональності створення та виконання замовлення

На діаграмі бачимо абстрактний клас ProvisioningAction, який розширюють класи, що виконують конкретні задачі, такі як підключення користувача до мережі, очікування на MAK-адресу від монтажника, резервування SIM карти тощо. Ці класи створюються на основі шаблону задач, що зберігається в таблиці task_templates, в класі FlowComposer. З об’єктів цих класів складається Flow, який реалізує інтерфейс Iterator (а також однойменний шаблон проєктування) для ітерації задач. Коли повідомлення з черги надходить до EventListener, він викликає метод createExecutionFlow(), який повертає об’єкт класу Flow. Після цього він

ітерує цей клас, виконуючи задачі згідно з їхньою черговістю. Таким чином реалізована функціональність виконання замовлення.

Була розглянута ключова функціональність мікросервісу order-service. Сервіс реалізує й іншу логіку, яка не була розглянута.

3.3.4 Мікросервіс offer-service

Мікросервіс offer-service відповідає за створення, відображення та зміну пропозицій продуктів, а також їхні ціни разом з дисконтами та знижками. Детально розглянемо API, який надає цей мікросервіс:

1) GET */api/offers/internet-offers* – повертає список пропозицій Інтернет продукту, включно зі знижками, якщо такі є.
2) GET */api/offers/internet-offers/{internetOfferId}* – повертає пропозицію Інтернет продукту за вказаним ID в URL, включно зі знижками, якщо такі є.
3) GET */api/offers/dtv-offers* – повертає список пропозицій DTV продукту, включно зі знижками, якщо такі є.
4) GET */api/offers/dtv-offers/{dtvOfferId}* – повертає пропозицію DTV продукту за вказаним ID в URL, включно зі знижками, якщо такі є.
5) GET */api/offers/tariffs* – повертає список тарифів мобільного продукту, включно зі знижками, якщо такі є.
6) GET */api/offers/tariffs/{tariffId}* – повертає тариф мобільного продукту за вказаним ID в URL, включно зі знижками, якщо такі є.
7) GET */api/offers/constant-prices* – повертає об’єкт з цінами на одноразові послуги, такі як підключення Інтернету, доставка, ціна за 1 метр кабелю, включно зі знижками, якщо такі є.
8) GET */api/active-discounts* – повертає список всіх активних дисконтів та знижок.

Всі кінцеві точки доступні і для неавторизованих користувачів, щоб вони могли переглянути пропозиції не реєструючись.

Розглянемо структуру мікросервісу, що зображена на рисунку 3.11 у вигляді UML діаграми компонентів.

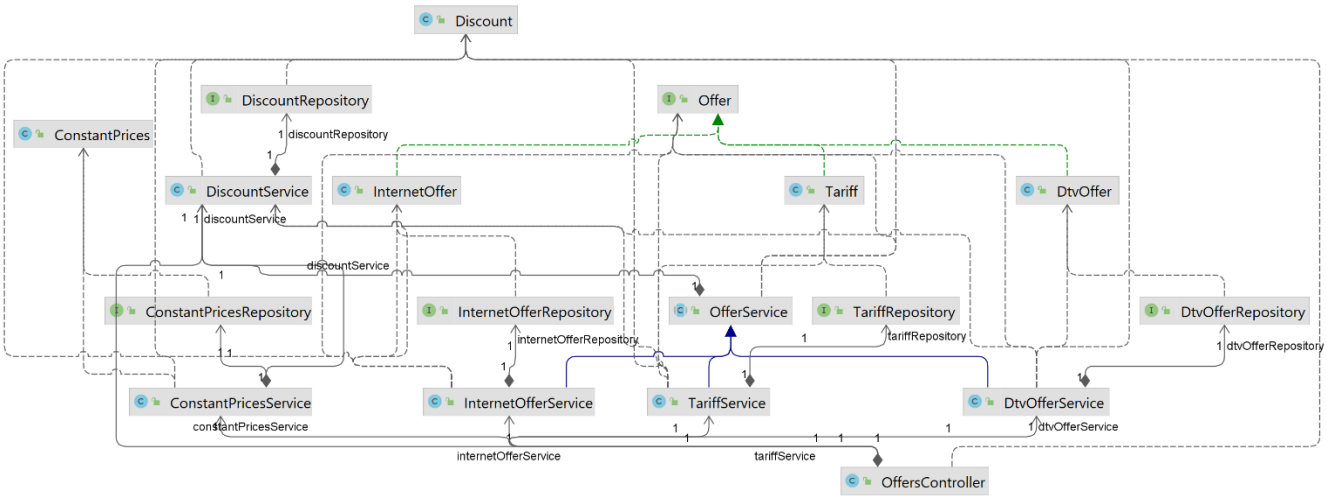


Рисунок 3.11 – UML діаграма компонент мікросервісу offer-service

Як бачимо на діаграмі, сервіс також реалізує багаторівневу архітектуру. Так як пропозиції мобільного, інтернет та DTV продуктів мають спільні поля та логіку, тому класи цих пропозицій реалізують інтерфейс Offer. Також, абстрактний клас OfferService має спільну логіку обробки пропозицій, а саме – застосування дисконтів, підрахунок кінцевих цін тощо. Класи наслідники тільки конкретизують, для яких пропозицій логіка буде виконана. Також в сервісі сутності бази даних конвертуються в DTO, які також мають ієрархічну структуру, але на діаграмі вони не були зображені для збереження читаємості.

3.3.5 Мікросервіс resource-service

Мікросервіс resource-service відповідає за роботу з ресурсами провайдера телекомунікаційних компаній, а саме з маршрутизаторами, комутаторами, платами, SIM картами, кабелями тощо. Ця функціональність є дуже важливою для компаній, її реалізує OSS в великих системах.

Розглянемо API, що надає цей мікросервіс:

1) GET */api/cables/twisted-pair-cable* – повертає скільки метрів кабелю витої пари залишилось в компанії та її ціну з врахуванням знижки.
2) PUT */api/cables?cableLength* – резервує кабель витої пари, довжина якого передана в параметрах запиту.
3) GET */api/connection/connected-buildings?buildingId* – перевіряє чи підключений до мережі будинок з ID, що переданий в запиті.
4) GET */api/connection/connected-address?addressId* – перевіряє чи підключена адреса до мережі з ID, що переданий в запиті.
5) PUT */api/connection/?buildingId&addressId&macAddress* – підключає користувача до мережі.
6) GET */api/devices* – повертає пристрої, які доступні для продажу.
7) GET */api/devices/{deviceId}* – повертає об’єкт пристрою з усіма параметрами.
8) PUT */api/devices?deviceId* – резервує пристрій для користувача.
9) GET */api/fixed-ip* – повертає першу знайдену вільну IP адресу з зарезервованих .
10) GET */api/phone-numbers* – повертає список мобільних номерів, які користувач може придбати.
11) GET */api/phone-numbers/{phone-number}* – повертає об’єкт мобільного номеру з ціною, з врахуванням знижок, якщо вони є.
12) PUT */api/phone-numbers?sim-card-number* – резервує SIM карту за переданим в параметрах запитом номеру SIM карти.

Розглянемо структуру мікросервісу, що зображена на рисунку 3.12 у вигляді UML діаграми компонентів.

Як бачимо на діаграмі, сервіс повністю реалізує багаторівневу архітектуру та є здебільшого простим CRUD сервісом (від англ. Create Read Update Delete), тобто реалізує операції створення, читання, зміни та видалення сутностей з бази даних без додаткової логіки обробки.

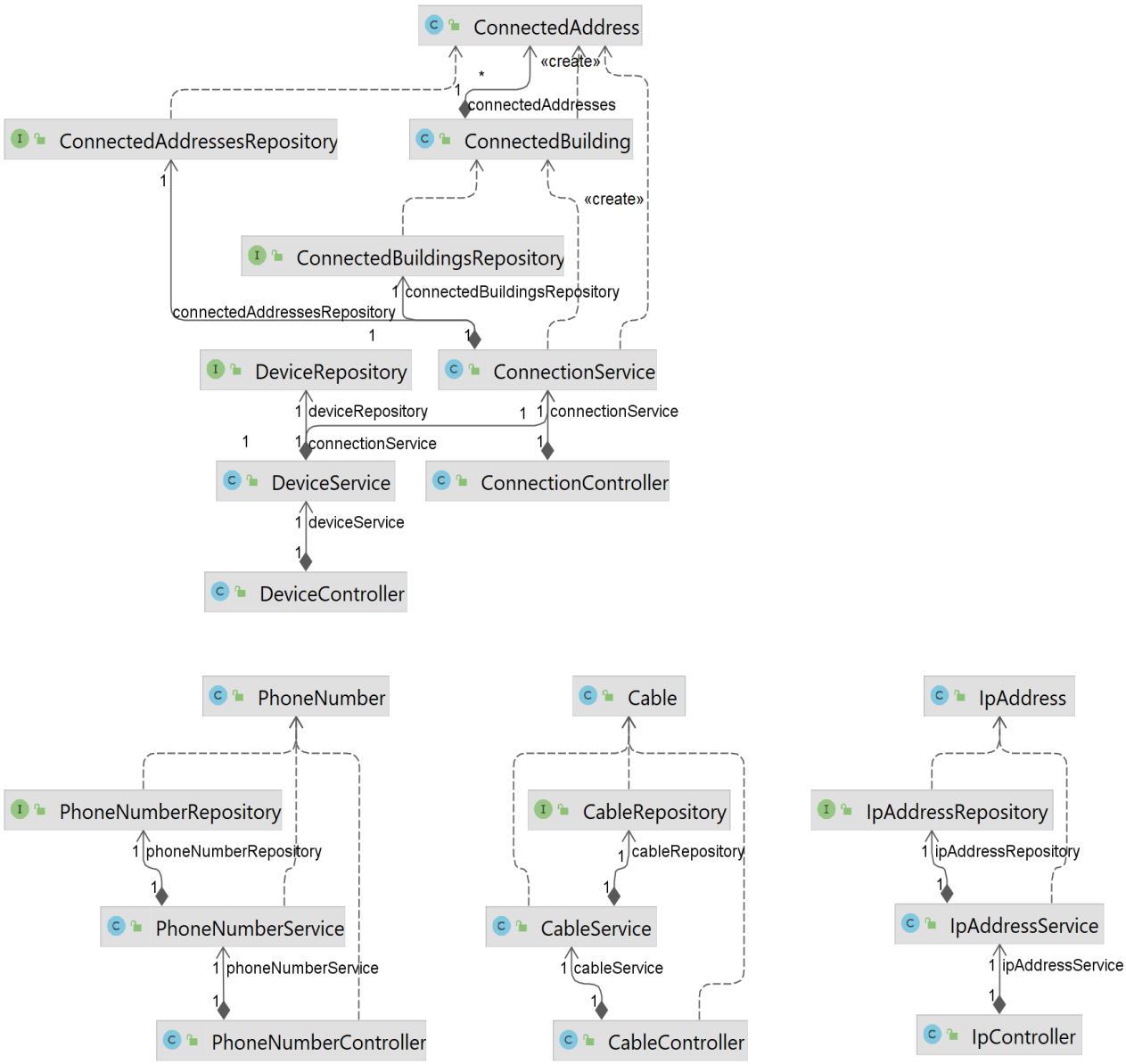


Рисунок 3.12 – UML діаграма компонент мікросервісу resource-service

Мікросервіс звертається до мікросервісу offer-service, використовуючи Spring Cloud Feign Proxy, для отримання знижок для цін на мобільні номера. Також цей мікросервіс використовує мікросервіс order-service під час виконання замовлення для резервування ресурсів та підключення користувача до мережі.

3.3.6 Мікросервіс analytics-service

Мікросервіс analytics-service реалізує функціональність аналітики даних. Реалізований на мові Python з використанням фреймворку Flask. Звертається до бази даних baoss_db для отримання даних з таблиць users, orders, instances, mobile_products, internet_products, dtv_products, payments. Детально розглянемо API, який надає даний мікросервіс. Варто зазначити, що всі кінцеві точки приймають дати початку та кінця періоду для аналітики та крок групування дат, наприклад день, тиждень, місяць, квартал, рік.

1. GET */api/common-statistic* – розраховує загальну статистику за певний проміжок часу. Повертає об’єкт, який складається з таких елементів:
    - Кількість користувачів, що зареєструвалися за проміжок часу;
    - Залежність кількості клієнтів від продукту, тобто скільки клієнтів замовило того чи іншого продукту;
    - Залежності кількості нових клієнтів від дати, тобто скільки клієнтів зареєструвалося за день, тиждень, місяць тощо за весь період;
    - Залежності продуктів від дати, тобто скільки разів користувачі замовили певного продукту за день, тиждень, місяць тощо за весь період;
2. GET */api/business-metrics* – розраховує бізнес метрики за певний проміжок часу, а саме:
    - Загальний GMV, тобто прибуток за весь період.
    - Залежність GMV від дати, тобто прибуток за день, тиждень, місяць тощо за весь період;
    - Загальний AOV, або середню корзину користувача за весь період. Було розділено на 2 типи – для усереднення NRC та MRC, так як це різні значення;

- Залежність AOV від дати, тобто зміна середньої корзини користувача за весь період з групуванням по дню, тижню, місяцю тощо;
- Загальний ARPU, тобто середній дохід з одного користувача за весь період;
- Залежність ARPU від дати, тобто зміна середнього догоду з користувача за весь період з групуванням по дню, тижню, місяцю тощо;
- Залежність GMV від продукту, тобто який продукт приносить найбільше прибутку компанії за весь період;
- Залежність GMV від продукту та дати, тобто зміна прибутку за певний продукт за весь період з групуванням по дню, тижню, місяцю тощо;
- Загальний CLV, тобто життєву цінність користувача за весь період;
- Залежність CLV від дати, тобто зміна життєвої цінності користувача за весь період з групуванням по дню, тижню, місяцю тощо;

3. GET */api/profit-forecast* – прогнозує прибуток компанії на певний період часу з заданим кроком (ден, тиждень, місяць тощо). Використовує для цього алгоритми Lasso регресії, Gradient Boosting, Decision Tree, Prophet для більш об’єктивної оцінки. Також, алгоритми можна порівняти наглядно, так як весь датасет поділяється на дані для тренування моделі та тестування, таким чином можна порівняти алгоритми по точності прогнозування, порівнюючи з реальними даними. Варто зазначити, що для перевірки точності також використовується метрика MAPE (англ. Mean Absolute Percentage Error), яка розраховується для результатів кожного з алгоритмів. Вона є доволі популярною для оцінки результатів прогнозування.

Всі кінцеві точки доступні тільки авторизованому користувачу. Мікросервіс тільки зчитує дані з бази даних та обробляє їх, ніяким чином не змінюючи дані.

3.3.7 Застосування Docker

Розглянемо яким чином використовується засіб автоматизації та уніфікації збирання та розгортування, а також контейнеризації Docker. На рисунку 3.13 зображений вміст Dockerfile для мікросервісу resource-service.

```
FROM maven AS 🔨 MAVEN_BUILD
COPY src /home/resource-service/src
COPY pom.xml /home/resource-service
RUN mvn -f /home/resource-service/pom.xml clean package -DprofileActive=dev-docker -U

FROM openjdk
COPY --from=MAVEN_BUILD /home/resource-service/target/resource-service-0.0.1.jar /usr/local/lib/resource-service.jar
EXPOSE 8084
ENTRYPOINT ["java", "-jar", "/usr/local/lib/resource-service.jar", "--spring.profiles.active=dev-docker"]
```

Рисунок 3.13 – Вміст Dockerfile для мікросервісу resource-service

В першій частині вказується, що образ буде базуватися з існуючого образу maven, після цього копіюється вихідний код з робочої директорії до відповідної директорії в контейнері і наприкінці виконуємо команду по створенню JAR файлу. В другій частині вказується, що образ буде базуватися з існуючого образу openjdk, потім з контейнеру, в якому зберігається створений JAR файлу копіюється цей файл та запускається з профілем dev-docker. Таким чином маємо контейнер з запущеним сервісом, який прослуховує порт 8084. Ідентичний вміст Dockerfile має кожен мікросервіс окрім analytics-service, так як він розроблений на Python. Його вміст наданий в додатку Г.

Також розглянемо оркестрацію контейнерів за допомогою docker-compose. А саме приклад створення контейнера resource-service. Для цього в docker-compose.yml файл додана конфігурація, яка описує яким чином resource-service контейнер має бути запущений. На рисунку 3.14 зображена конфігурація resource-service в docker-compose.yml. В ній вказані такі параметри: назва контейнеру, директорію контексту виконання, шлях до Dockerfile, мапінг портів, які потрібно «прокидувати» за межі контейнера, так як контейнер не доступний зовні, параметри середовища, мережа, до якої належить сервіс та сервіс, який повинен бути запущений перед запуском resource-service.

```
resource-service:
  container_name: resource-service
  build:
    context: ./resource-service
    dockerfile: ./Dockerfile-dev
  ports:
    - 8084:8084
  expose:
    - 8084
  environment:
    - IMPORT_DEFAULT_RESOURCES=true
    - NUMBER_OF_PHONE_NUMBER=0
  restart: always
  networks:
    - common-network
  depends_on:
    - mongo-db
```

Рисунок 3.14 – Конфігурація resource-service в docker-compose.yml.

Схожа конфігурація описана для всіх мікросервісів в docker-compose.yml. Завдяки цьому інструменту можна легко запускати всі сервіси командою docker-compose up --build. Повний файл docker-compose.yml наданий в додатку Г.

3.4 Реалізація клієнтської частини

Клієнтська частина реалізована за допомогою веб-фреймворку Angular. Структура проєкту розробленого з використанням цього фреймворку складається з компонентів – структурних елементів, які реалізують певну функціональність та відображають результати роботи на сторінці в браузері. Розглянемо основні компоненти та інформацію, яку вони відображають користувачу на сторінці:

1) authorization-component – компонента, яка відповідає за авторизацію користувача. Сторінка має поля логіна та пароля, які збираються в об’єкт та відправляються на сервер для перевірки та авторизації користувача у разі успіху.
2) analytics-component – компонента, яка відповідає за відображення аналітики даних телекомунікаційних компаній. Має поля дат для задання періоду часу,

за який потрібно провести аналітику, зібрати статистику та спрогнозувати прибуток, та кроку для задання кроку для групування даних для графіків.

3) deliveries-component – компонента, яка відображає всі доставки для робітника, тобто інформацію про продукти, які він має підключити за день, в який час та за якою адресою.
4) device-info-component – компонента, яка відображає діалогове вікно для перегляду детальної інформації про пристрій. Відкривається з сторінки Order Entry.
5) edit-profile-component – компонента, яка відповідає за редагування даних користувача.
6) footer-component – компонента, яка відображає інформацію про компанію внизу сторінки, а саме контакти та копірайт.
7) header-component - компонента, яка відображає назву компанії вверху сторінки разом з кнопками авторизації та реєстрації для неавторизованих користувачів.
8) headerauth-component – компонента, яка відображає назву компанії вверху сторінки разом з іменем користувача та кнопкою виходу з системи для авторизованих користувачів.
9) home-component – композитна компонента, яка містить в собі меню, заголовок, футер та робочу область сторінки для неавторизованого користувача.
10) homeauth-component – композитна компонента, яка містить в собі меню, заголовок, футер та робочу область сторінки для авторизованого користувача.
11) instance-component – компонента, яка відображає детальну інформацію про екземпляр продукту.
12) menu-component – компонента, яка відображає меню з посиланнями на різні сторінки. Меню залежить від ролі авторизованого користувача. Наприклад, клієнти бачать тільки посилання на сторінки Order Entry,

пропозицій продуктів, своїх замовлень та підключених послуг, а користувач з роллю Sales Manager бачить посилання на сторінку аналітики даних, замість посилань на сторінки замовлень та продуктів.

13) offer-component – компонента, яка відображає пропозиції для всіх продуктів, які надає провайдер. Має кнопку-посилання для переходу на сторінку order entry.
14) order-component – компонента, яка відображає детальну інформацію про замовлення користувача.
15) order-entry-component – головна компонента, яка відповідає за створення замовлення. На сторінці цієї компоненти користувач обирає продукти, які бажає підключити, обирає тариф, додаткові послуги, заповнює поля адреси та часу доставки та підтверджує замовлення. Всі параметри валідуються при підтвердженні, щоб сервер отримав валідні дані.
16) orders-and-instances-component – компонента, яка відображає коротку інформацію про замовлення та екземпляри продуктів користувача.
17) profile-component – компонента, що відображає інформацію користувача. Має кнопку-посилання на сторінку редагування даних.
18) registration-component – компонента, яка надає можливість реєстрації для нових клієнтів. Сторінка містить всі необхідні поля, які валідуються перед відправкою на сервер.

Також використовуються класи сервіси, які виконують технічні задачі, такі як відправка запитів з необхідними даними на відповідний сервіс через ТВ-API, перехоплення запитів для додавання токену тощо.

Висновки до розділу 3

У даному розділі була розглянута реалізація всієї системи, а саме – її загальна структура, моделі бази даних, функціональність всіх мікросервісів на серверній частині та функціональність сервісу на клієнтській частині. Детально були описані всі таблиці та колекції, що зберігаються в 3 базах даних. Детально описані API всіх мікросервісів з поясненням їхньої роботи. Надані UML діаграми компонентів для всіх мікросервісів для розуміння їхньої структури. Детально розглянута функціональність створення та виконання замовлення в мікросервісі order-service разом з перевіркою можливості виконання замовлення. Також було описано використання Docker на прикладі resource-service, а саме – вміст відповідного Dockerfile для збирання образу та конфігурація даного сервісу в docker-compose.yml для запуску контейнера та його оркестрації.

4 ІНСТРУКЦІЯ КОРИСТУВАЧА

4.1 Огляд можливостей системи

Покроково розглянемо всю функціональність, що реалізована в системі. Титульна сторінка відображає всі пропозиції та тарифи, що надає провайдер.

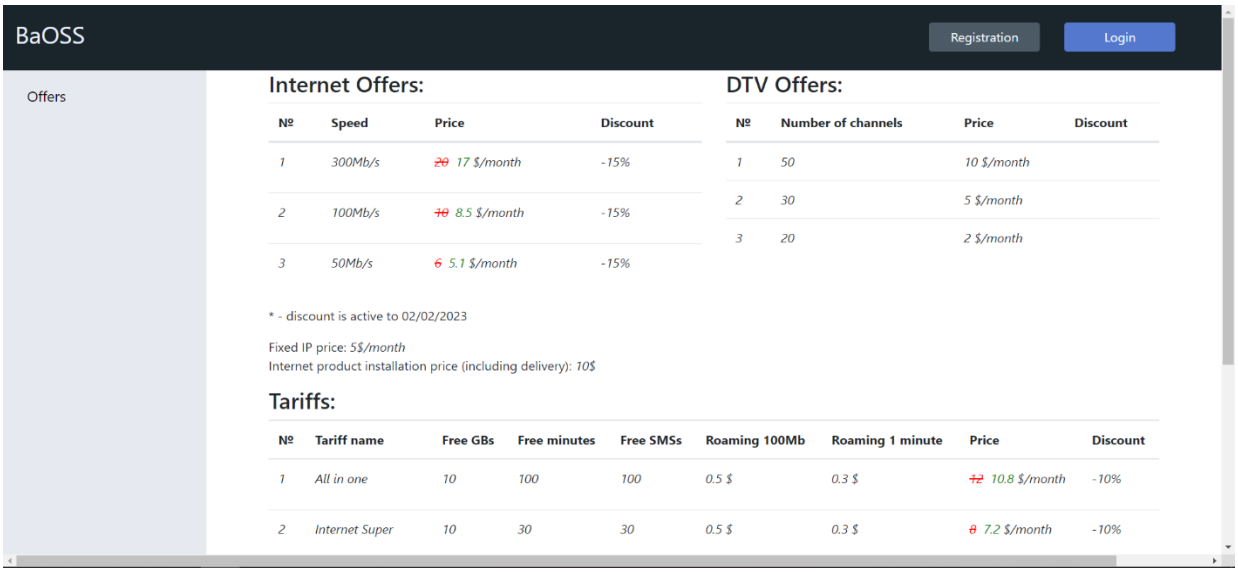


Рисунок 4.1 – Сторінка всіх пропозицій

Зліва знаходиться меню з одним посиланням на сторінку пропозицій, так як користувач неавторизований. Створимо новий обліковий запис та авторизуємось.

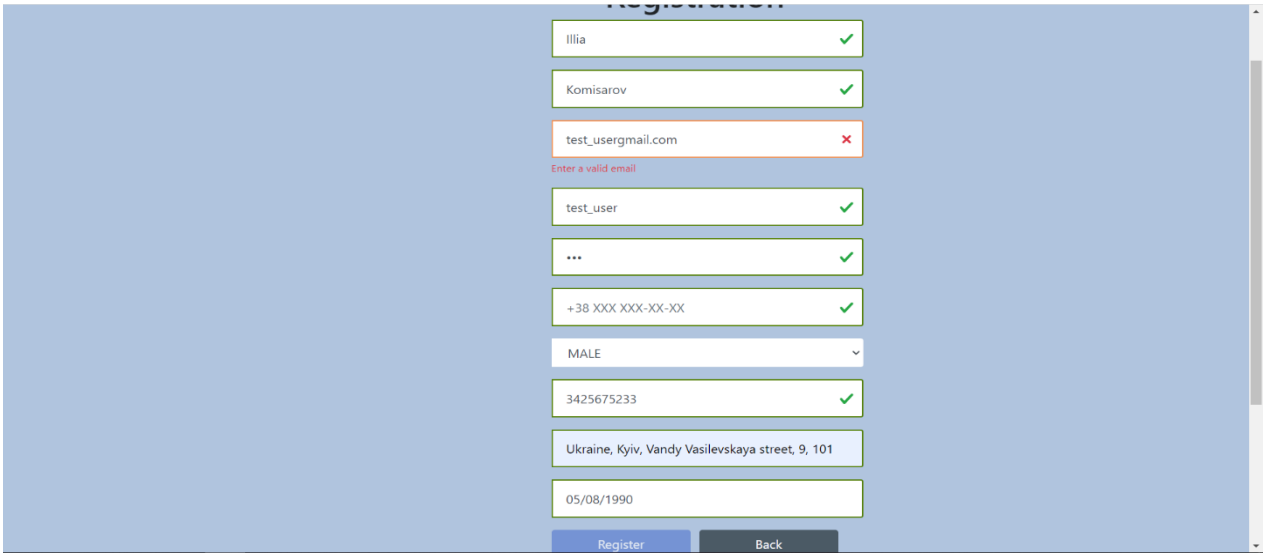


Рисунок 4.2 – Сторінка реєстрації користувача

Як бачимо, валідація працює коректно та не дозволяє користувачу зареєструватись, поки пошта не буде правильною.

Перейдемо до сторінки авторизації.

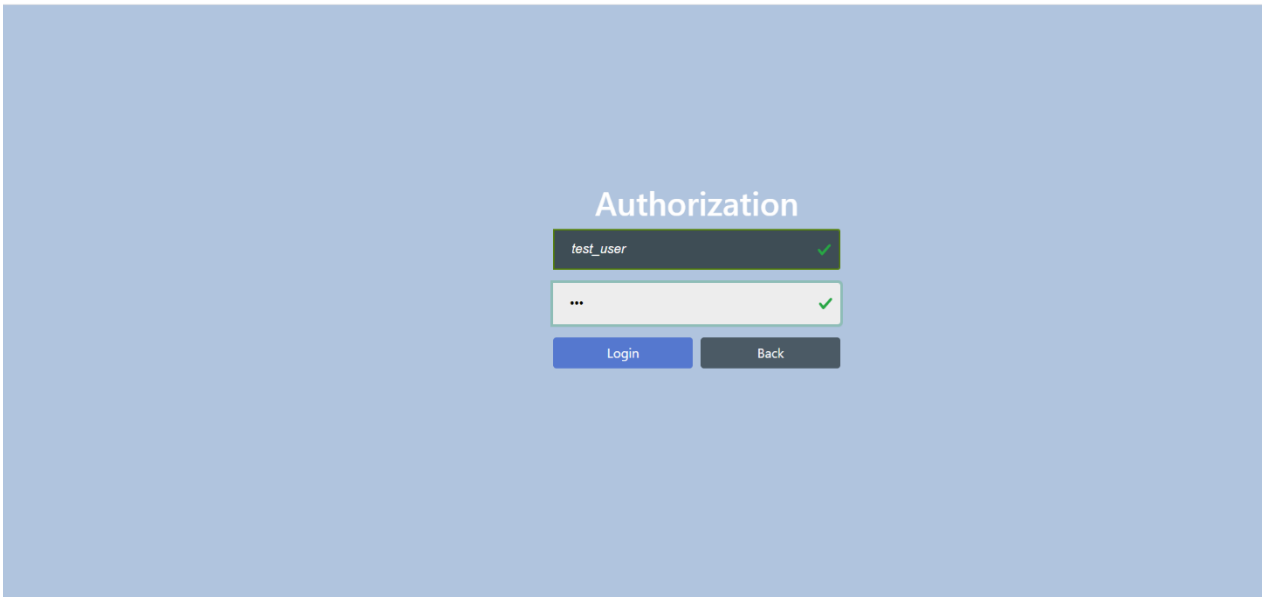


Рисунок 4.3 – Сторінка авторизації користувача

Після введення логіну та паролю відкривається сторінка профілю користувача. Також бачимо, що меню розширилось.

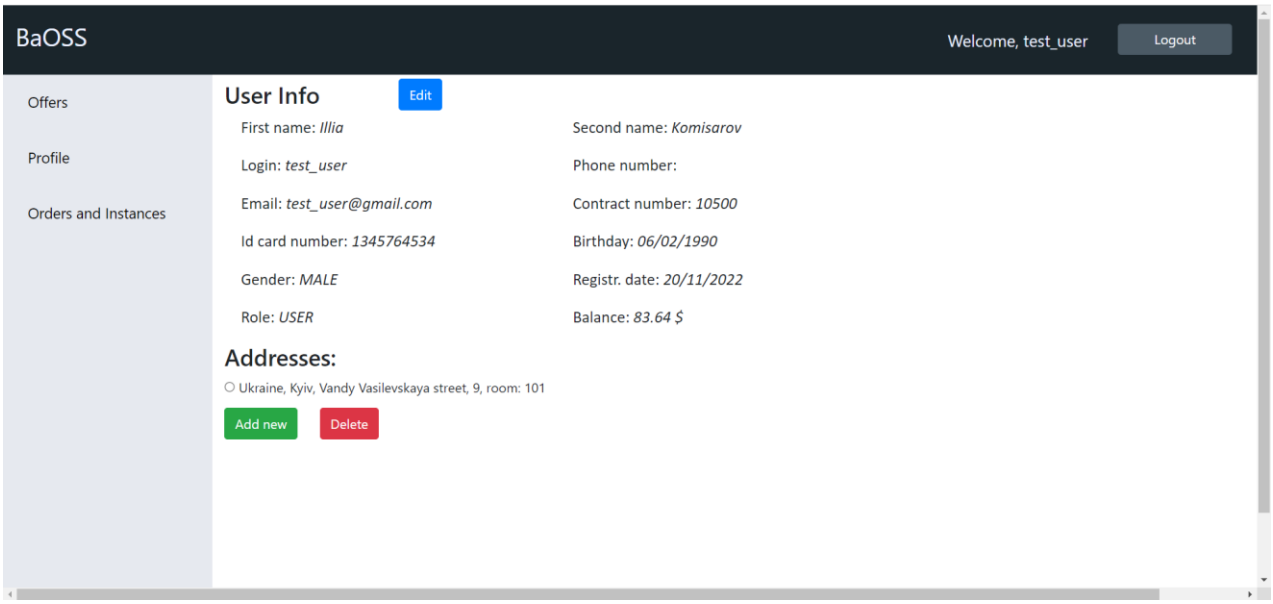


Рисунок 4.4 – Сторінка профілю користувача

На сторінці бачимо повну інформацію користувача разом з адресами. Також є можливість додати нову адресу або видалити існуючу. Разом з цим є кнопка-посилання на сторінку зміни даних. Вона зображена на рисунку 4.5. Всі поля також мають валідацію, як і на сторінці реєстрації.

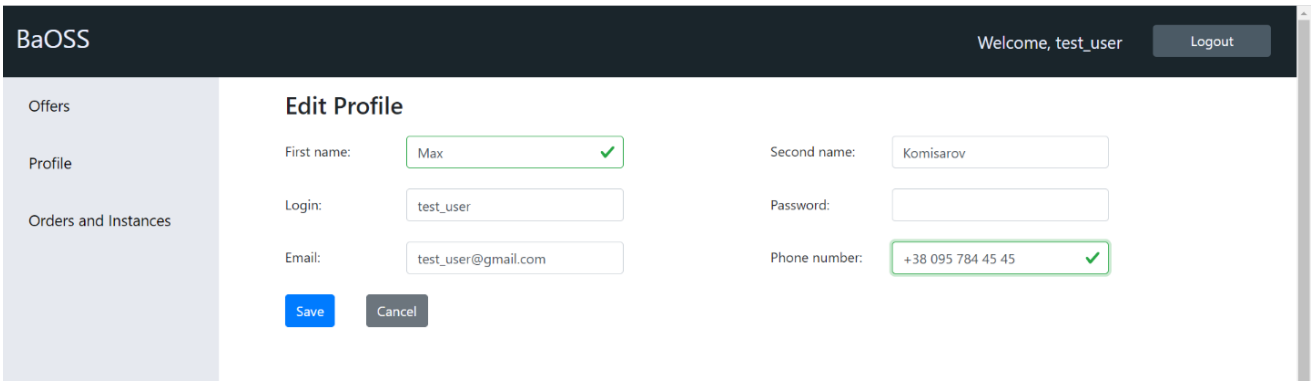


Рисунок 4.5 – Сторінка зміни даних користувача

Як бачимо, ім’я та номер телефону були успішно змінені.

Рисунок 4.6 – Перевірка зміни даних користувача

Тепер розглянемо ключову функціональність системи, а саме створення замовлення. Для цього перейдемо до сторінки Order Entry, посилання на яку доступно на сторінці пропозицій. Як бачимо з рисунку 4.7, сторінка розділена на 3 області: продукти та інформація про них, інформація про доставку та загальна інформація про обрані послуги та їх ціну NRC та MRC.

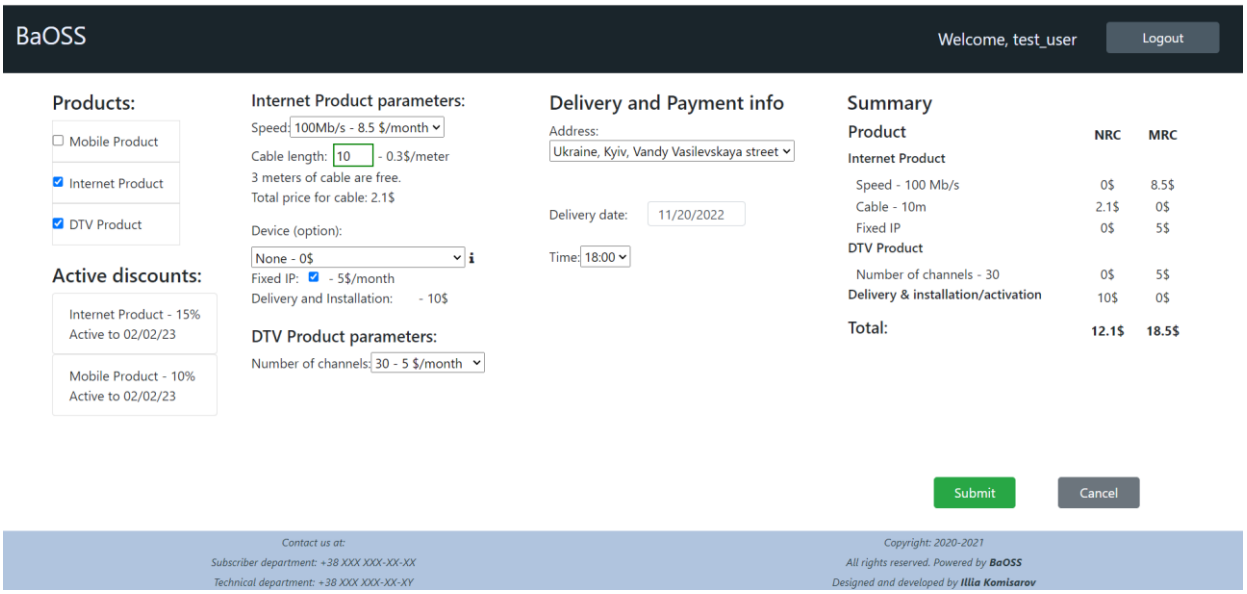


Рисунок 4.7 – Сторінка Order Entry

Оберемо Інтернет та DTV продукти та заповнимо всі необідні параметри, оберемо дату та час та підтвердимо замовлення.

Після підтвердження замовлення користувач переходить на сторінку своїх замовлень та екземплярів продуктів, яка зображена на рисунку 4.8. Як бачимо, екземпляри ще не активні, а замовлення в статусі виконання.

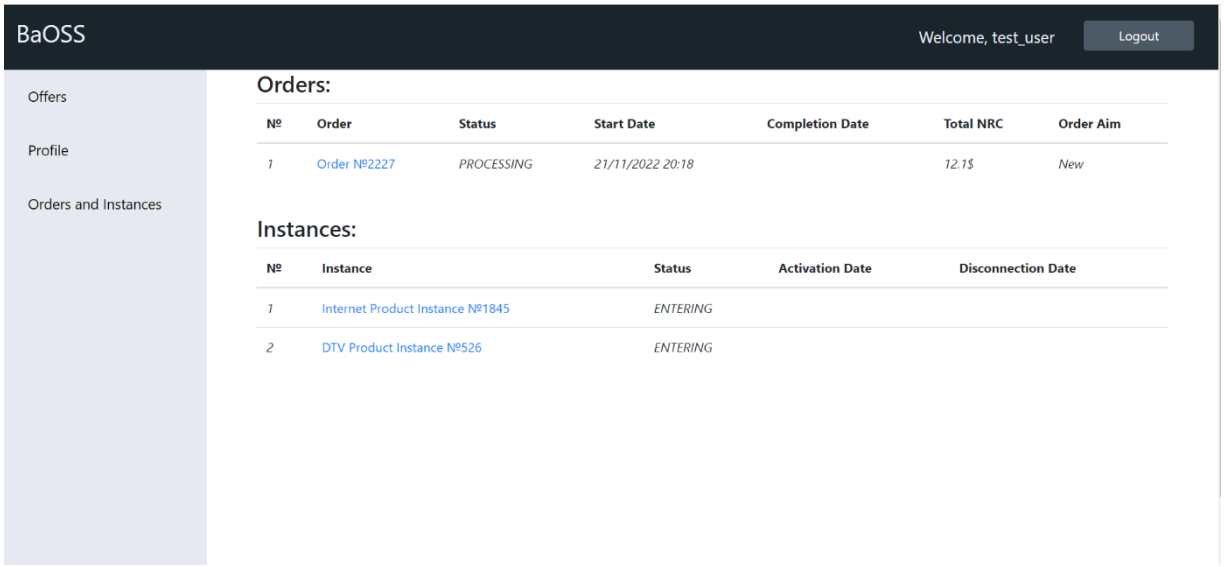


Рисунок 4.8 – Сторінка з замовленнями та екземплярами продуктів

Користувач може перейти на сторінку замовлення, щоб переглянути його параметри на статус виконання. Ця сторінка зображена на рисунку 4.9. Окрім параметрів відображаються екземпляри, які активуються цим замовленням.

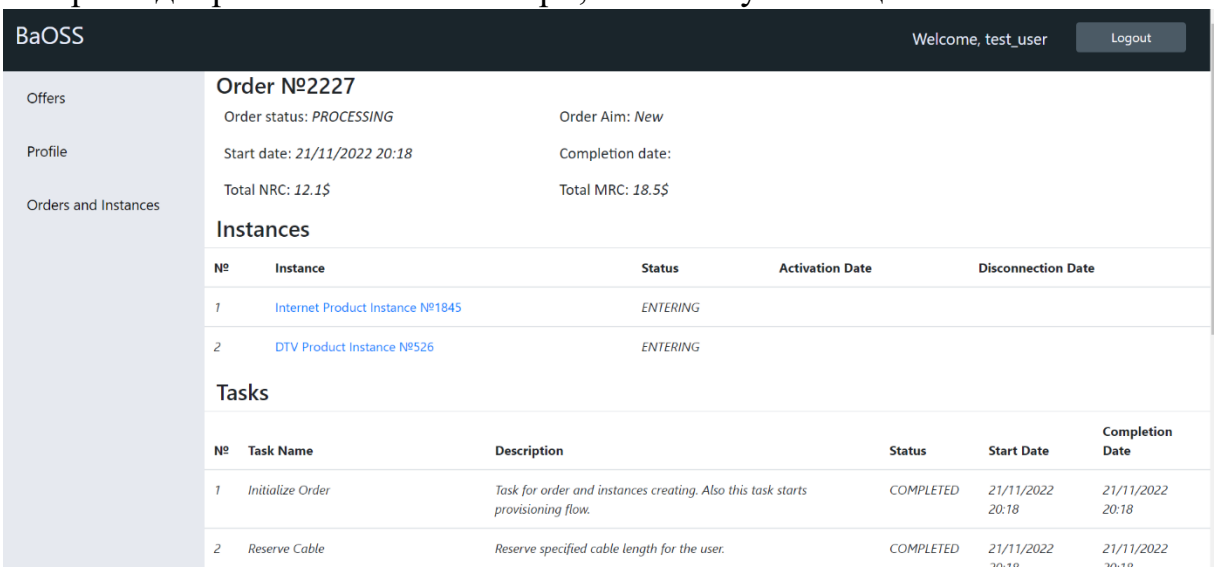


Рисунок 4.9 – Сторінка параметрів замовлення

Після списку екземплярів продуктів йде список задач, які потрібно виконати для виконання замовлення. Повний список задач зображений на рисунку 4.10.

№	Task Name	Description	Status	Start Date	Completion Date
1	Initialize Order	Task for order and instances creating. Also this task starts provisioning flow.	COMPLETED	21/11/2022 20:18	21/11/2022 20:18
2	Reserve Cable	Reserve specified cable length for the user.	COMPLETED	21/11/2022 20:18	21/11/2022 20:18
3	Notify the fitter about installation date	Send notification to the fitter about the order installation date.	COMPLETED	21/11/2022 20:18	21/11/2022 20:18
4	Waiting MAC Address for Internet connection	Waiting when fitter enter MAC-address of user.	WAITING		
5	Connect User to Network	Finalize user connection to network.	NOT_STARTED		
6	Activate DTV Channels	Finalize activation of DTV.	NOT_STARTED		
7	Provide Fixed IP	Providing fixed IP to user.	NOT_STARTED		
8	NRC Payment	Get NRC payment from the card.	NOT_STARTED		
9	Complete Order	Final processes for order fulfilment.	NOT_STARTED		

Рисунок 4.10 – Сторінка параметрів замовлення

Як бачимо, задача «Waiting MAC Address for Internet connection» знаходиться в статусі “Waiting”, тобто вона очікує підтвердження установки від монтажника. Для цього потрібно авторизуватися під роллю “Fitter” та зайти на сторінку доставок, що зображена на рисунку 4.11. На ній ми бачимо, сьогоднішні доставки для певного робітника. Вводимо MAK-адресу пристрою користувача та підтверджуємо доставку.

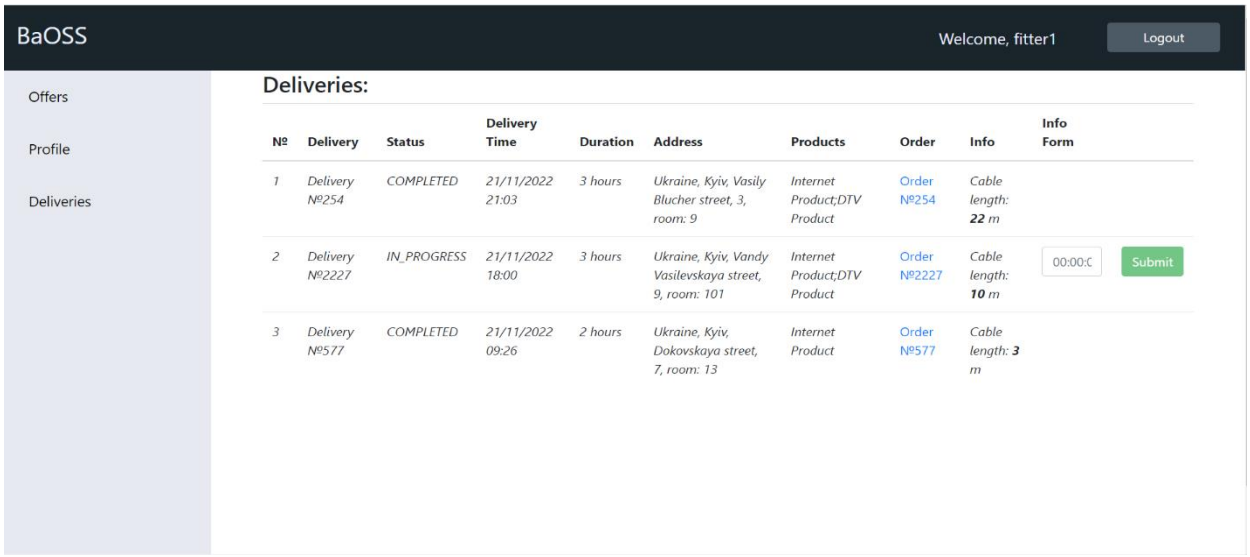


Рисунок 4.11 – Сторінка доставок робітника компанії

Після цього зайдемо на сторінку замовлення користувача та перевіримо статус замовлення та задач. Як бачимо з рисунку 4.12, замовлення в статусі «Completed» і всі задачі також.

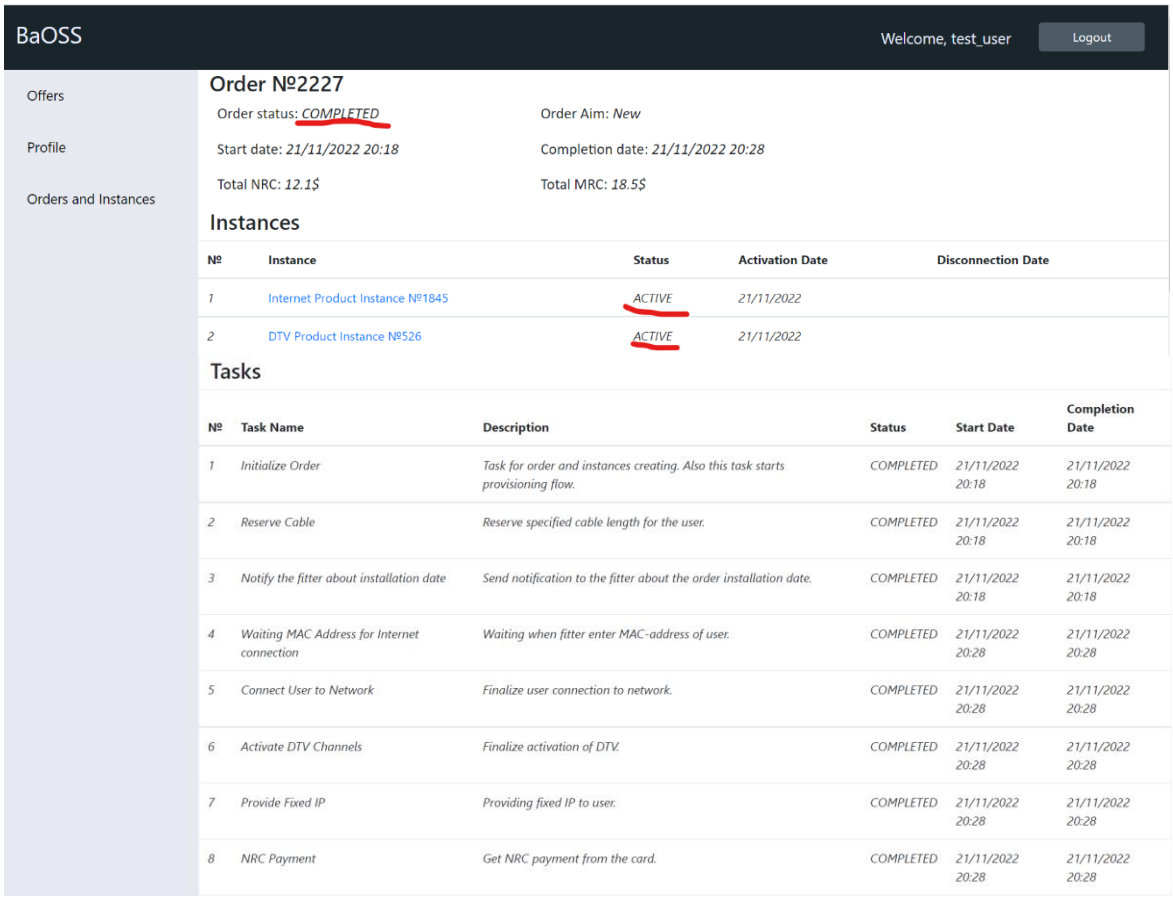


Рисунок 4.12 – Сторінка виконаного замовлення

Також переглянемо сторінки активованих продуктів. На рисунку 4.13 ми бачимо параметри активного Інтернет продукту.

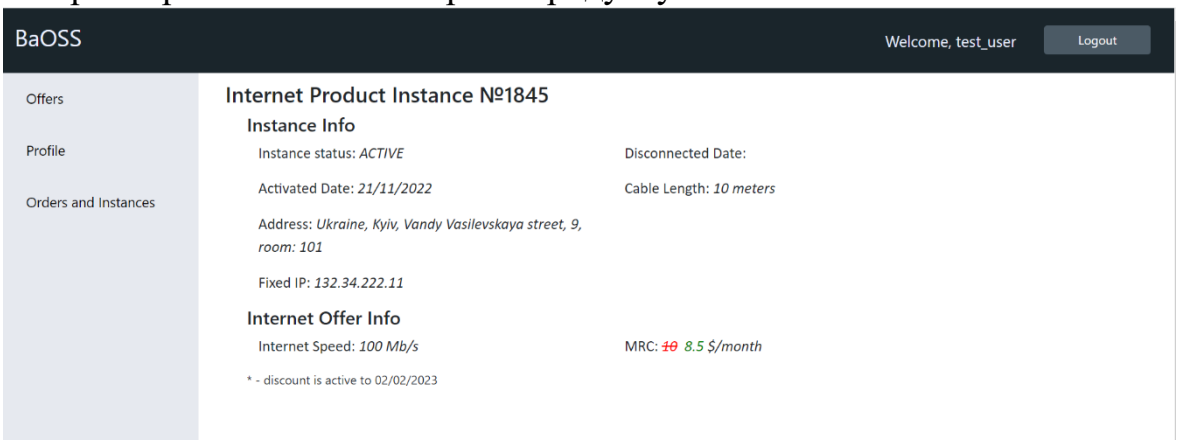


Рисунок 4.13 – Сторінка активного Інтернет продукту

А на рисунку 4.14 активного DTV продукту. Таким чином, можна підтвердити, що активація пройшла успішно.

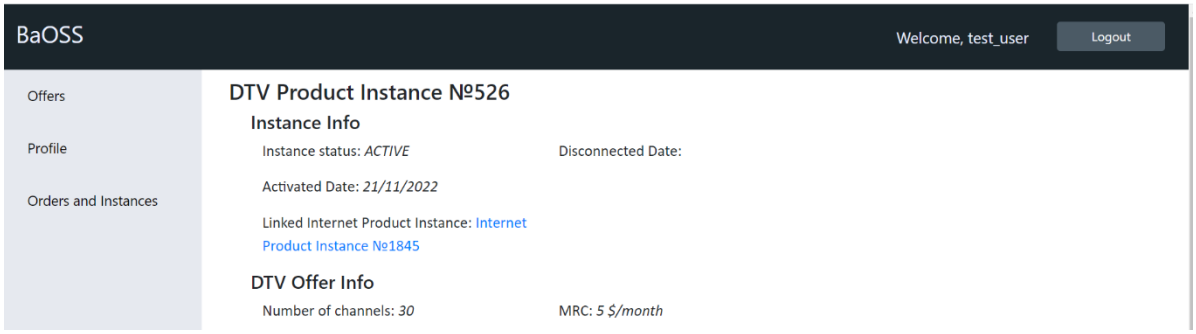


Рисунок 4.14 – Сторінка активного DTV продукту

Тепер перейдемо до розгляду аналітики даних. Для цього потрібно авторизуватись під роллю “Sales Manager”. В меню переходимо на сторінку аналітики та вводимо дати початку і кінця періоду для аналітики та крок групування по датам. Бізнес аналітик може вибрати загальну статистику, бізнес метрики та прогнозування прибутку.

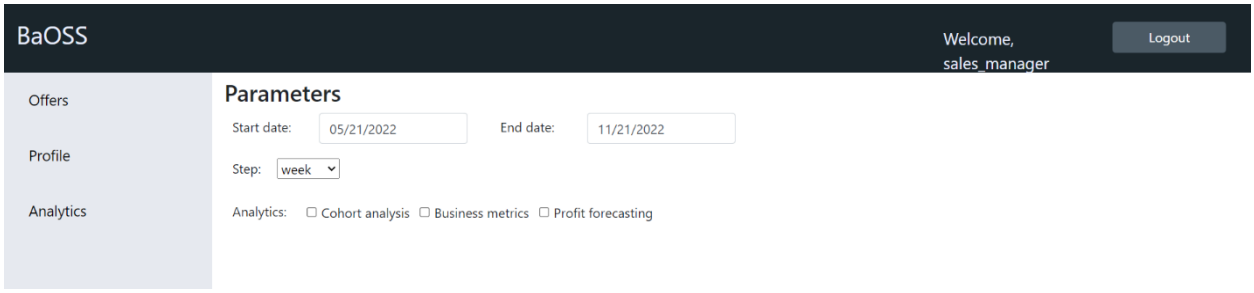


Рисунок 4.15 – Сторінка аналітики даних

Спершу розглянемо загальну статистику, перша частина результатів якої зображена на рисунку 4.16.

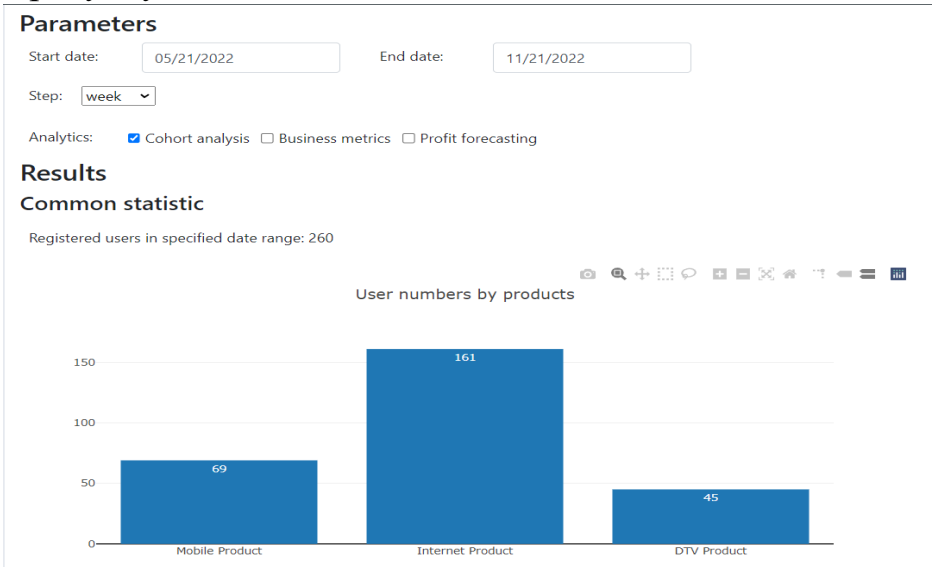


Рисунок 4.16 – Статистика кількості користувачів за пів року

Друга частина забражена на рисунку 4.17.

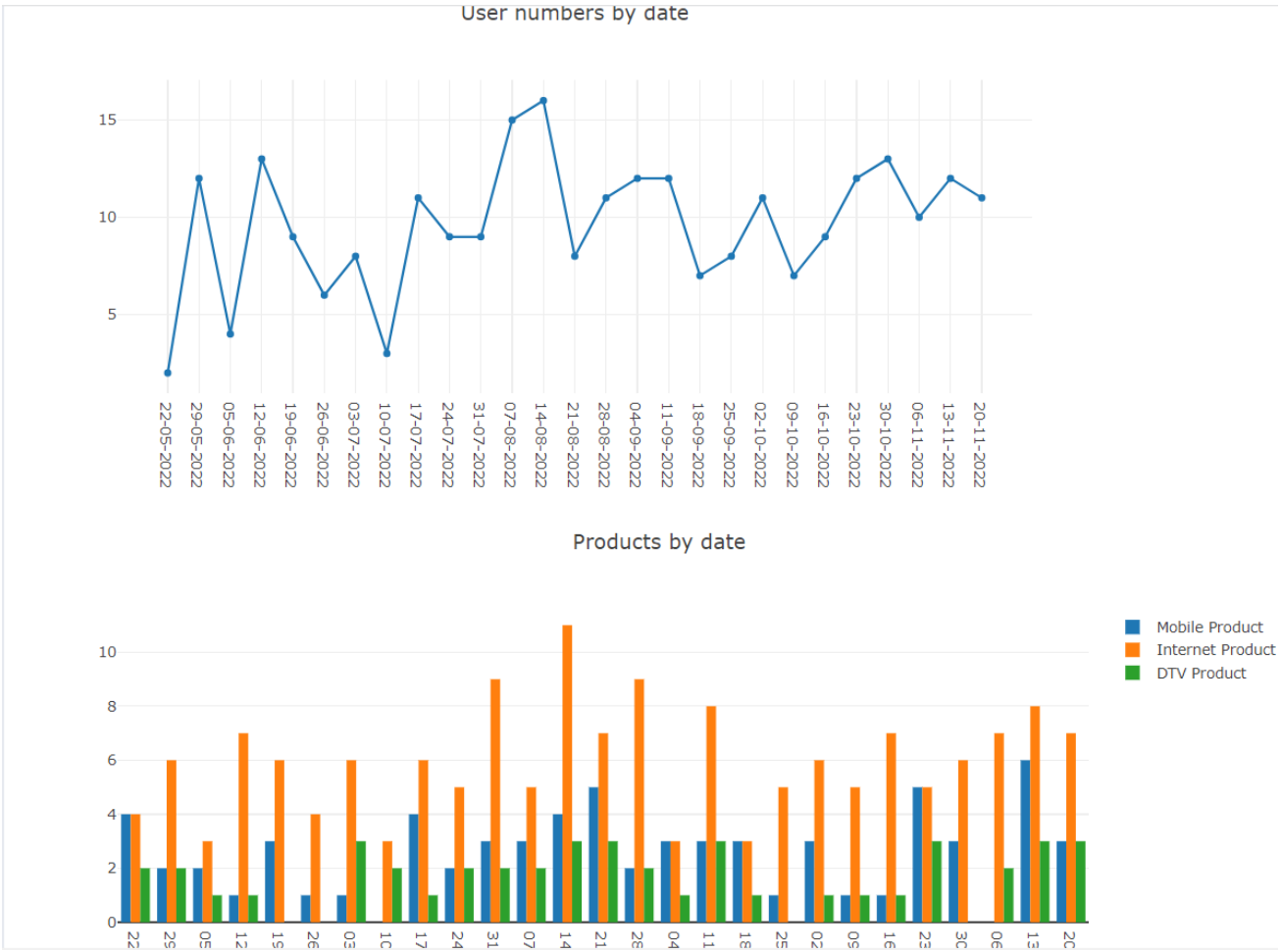


Рисунок 4.17 – Залежність кількіості користувачів та продуктів від дати

На рисунку 4.16 бачимо кількість користувачів, які зареєструвалися з 21 травня по 21 листопада – 260 штук. Після цього йде графік залежності кількості користувачів від продукту. Як бачимо, більшість користувачів купують Інтернет продукт, причому з великим відривом від мобільного чи DTV. На рисунку 4.17 зображені графіки залежності користувачів та продуктів від дати з групуванням по тижню. Як бачимо, кількість реєстрованих користувачів за тиждень близько 10, хоча бувають просадки до 1-2. Залежність продуктів від дати також підтверджує факт, що більше всього користувачі купляють Інтернет продукт, потім мобільний і наприкінці DTV.

Тепер розглянемо бізнес метрики, які розраховує система, вони разом з графіками зображені на рисунках 4.18-4.22.

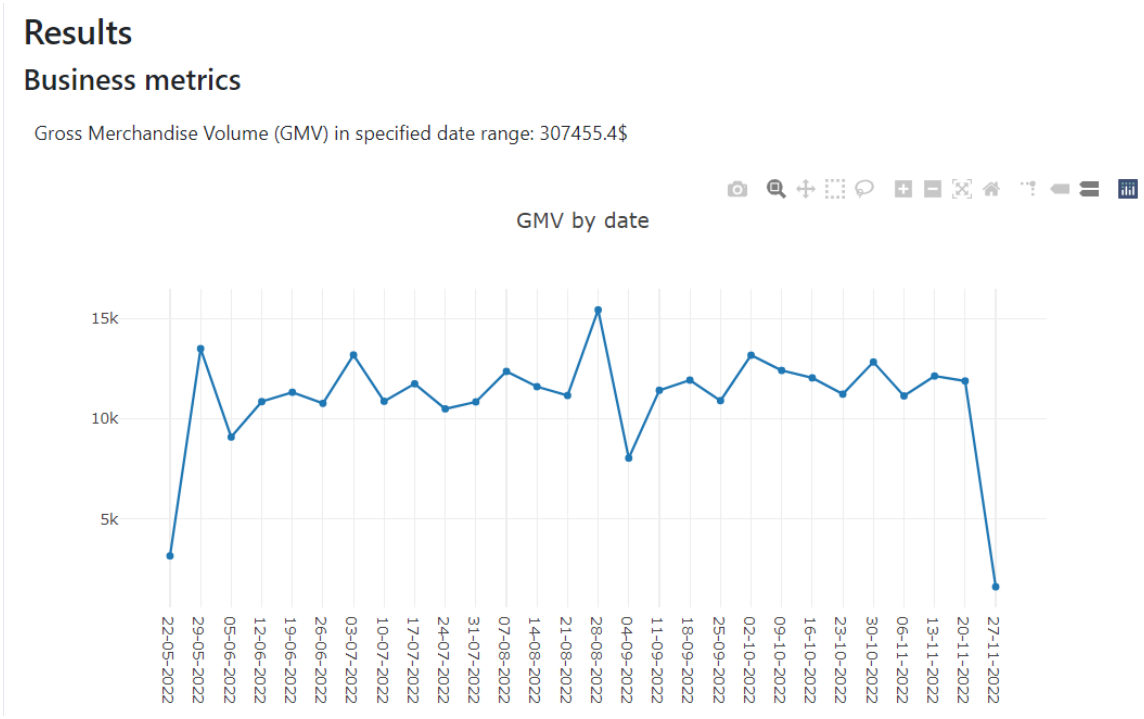


Рисунок 4.18 – Значення GMV

На рисунку 4.18 зображені значення загального прибутку (GMV) за пів року та графік зміни GMV за цей період з групуванням по тижню. Як бачимо, в тиждень, в середньому, компанія заробляє близько 10000$, хоча бувають просадки.

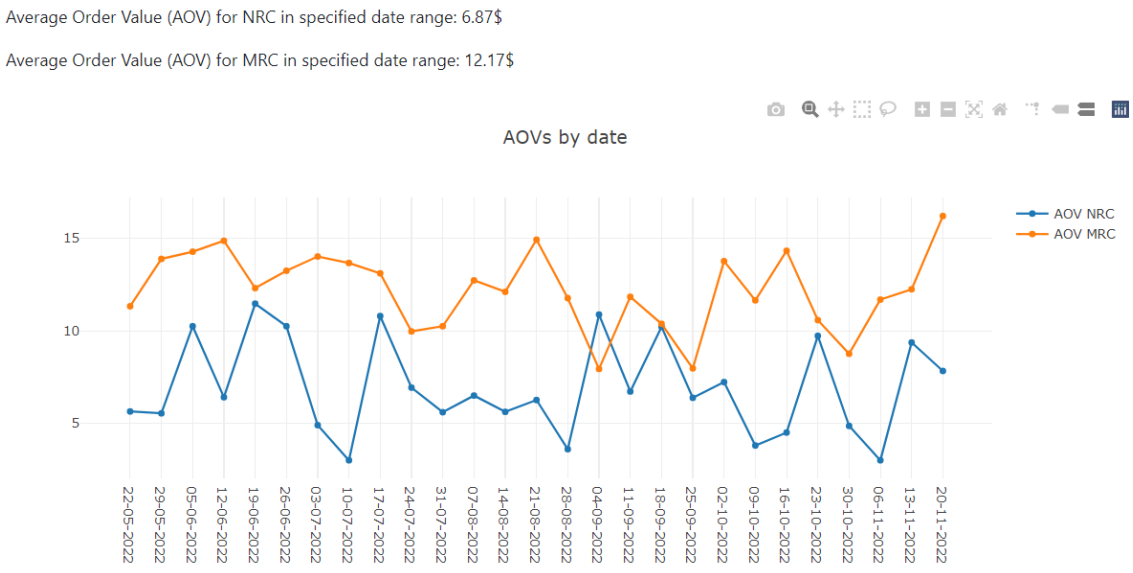


Рисунок 4.19 – Значення AOV

На рисунку 4.19 зображені значення середньої корзини замовлення MRC та NRC та графік зміни цих значень за період з групуванням по тижню. Як бачимо,

більше прибутку компанія отримує з MRC платежів, отже компанії є сенс зосередитись на заохоченню користувачів замовляти більш дорогі тарифи/додаткові місячні послуги – це принесе більше прибутку.

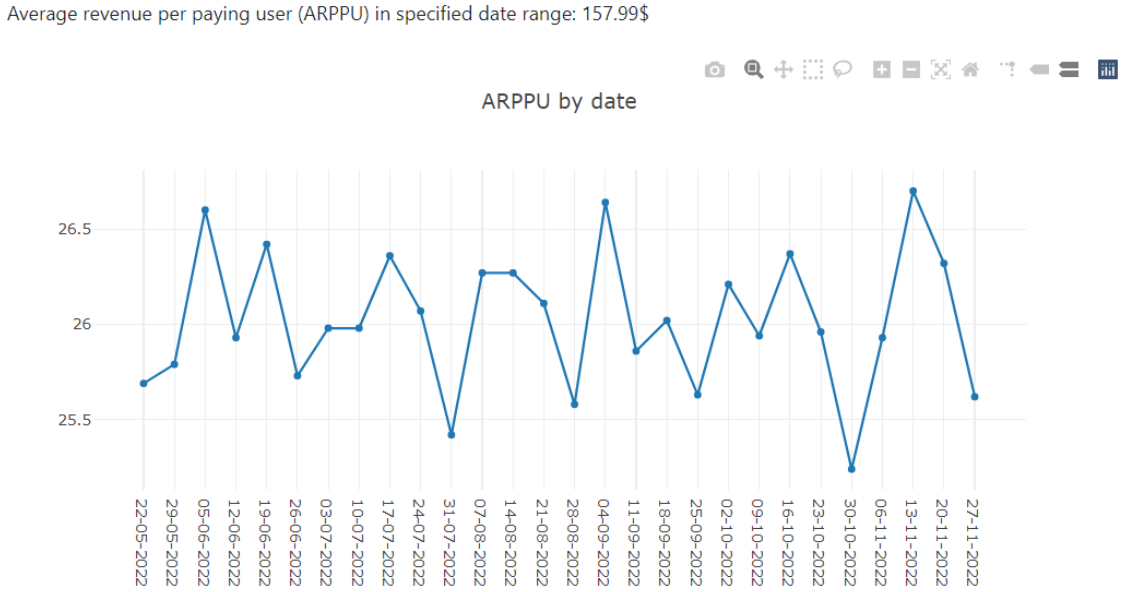


Рисунок 4.20 – Значення ARPU

На рисунку 4.20 зображені значення середнього прибутку з користувача та та графік зміни ARPU за період з групуванням по тижню. Як бачимо, за тиждень в середньому це значення дорівнює 26$.

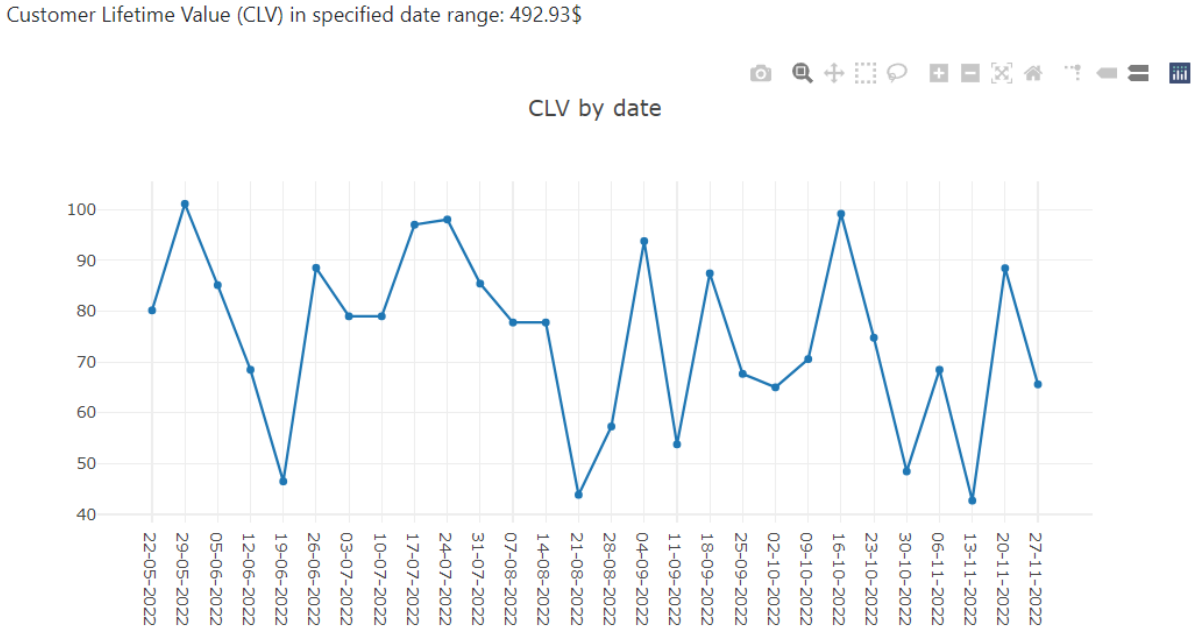


Рисунок 4.21 – Значення CLV

На рисунку 4.21 зображені значення цінності користувача та графік зміни CLV за період з групуванням по тижню. Це значення має корелювати з ARPU, але

як бачимо, є різниця, так як на CLV впливає значення відтоку користувачів. Як бачимо, є просадки по цій метриці, що каже про великий відток клієнтів в ці тижні. Компанії варто зосередитись на утриманні клієнтів, або можливо це пов'язано з якістю послуг.

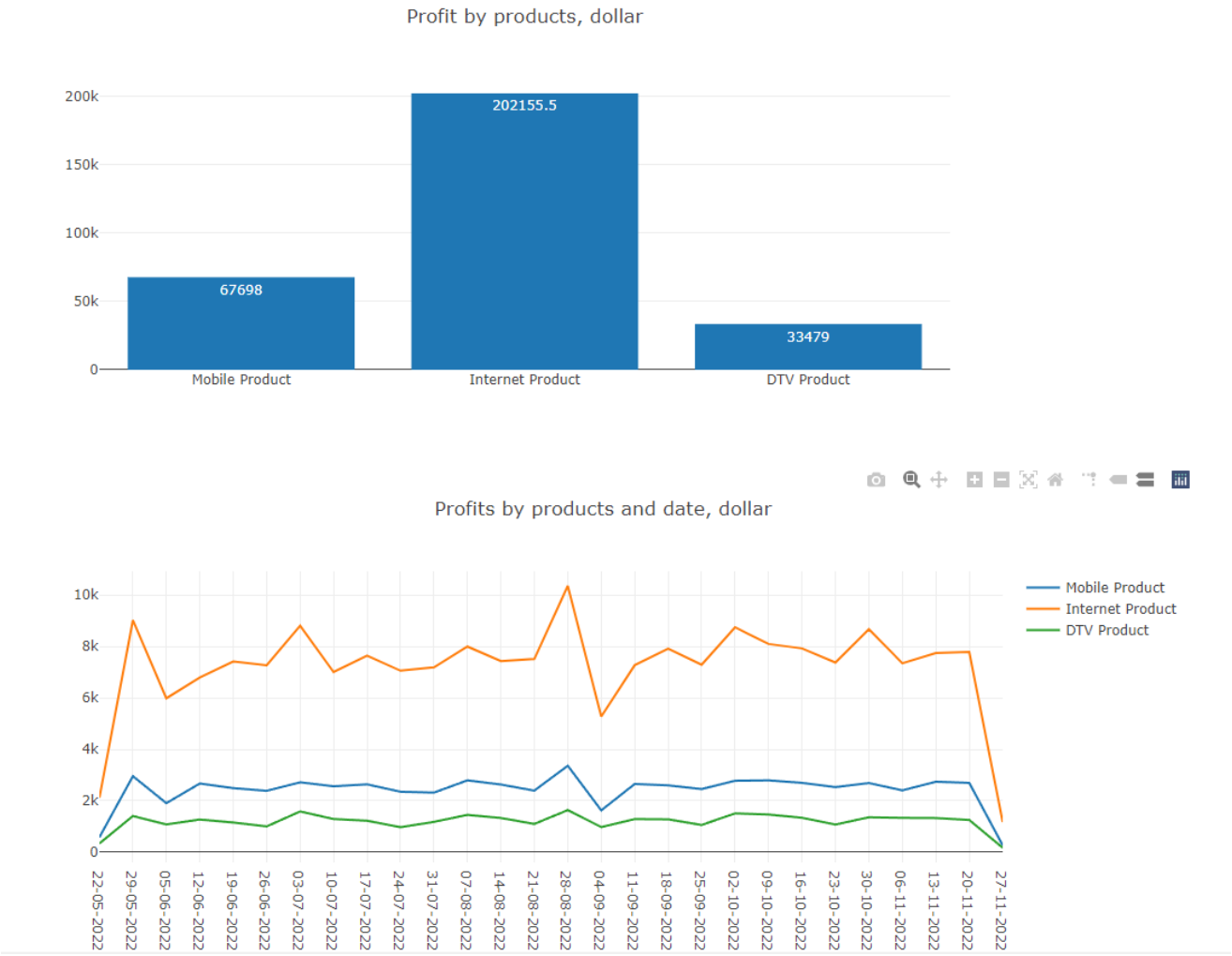


Рисунок 4.22 – Графіки залежності GMV від продукту

На рисунку 4.22 зображені графіки залежності прибутку (GMV) від продукту та зміни прибутку за період з групуванням по тижню для кожного продукта. Ще раз можемо впевнитись, що Інтернет продукт є найбільш прибутковим, таким чином компанії варто зосередитись на цьому продукті.

Під кінець проаналізуємо результати прогнозування прибутку на 2 місяці вперед алгоритмами Prophet, Lasso, Gradient Boosting, Decision Tree. Ця залежність зображена на рисунку 4.23.

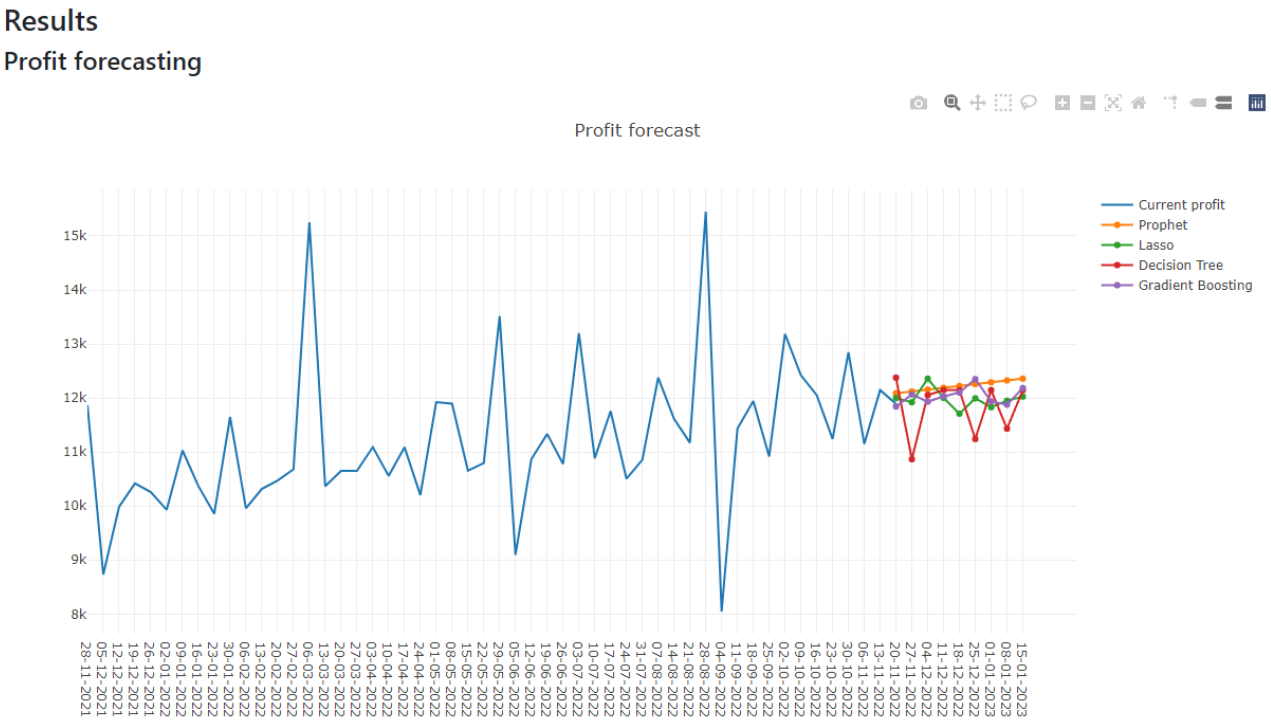


Рисунок 4.23 – Графік прогнозування прибутку 4-ма алгоритмами

З графіків бачимо, що, очевидно, алгоритм Prophet зовсім некоректно прогнозує прибуток, він пропонує лінійне зростання, що неможливо, судячи з графіку прибутку за останній рік. Регресія Lasso показує більш реалістичні результати, але піки зростання та падіння прибутку доволі, малі, що каже нам про можливі неточності. Результати роботи Gradient Boosting схожі з Lasso. А Decision Tree враховує перепади прибутку, хоча вони доволі рівномірні, чого скоріше за все не буде в реальності.

Звісно, щоб більш об’єктивно порівнювати алгоритми потрібно порівняти їхні результати з реальними даними, розбивши датасет на дані для тренування та для прогнозу. Результати цієї роботи продемонстровані на рисунку 4.24. На графіках бачимо, що найкраще себе показав алгоритм Gradient Boosting, піки прибутку майже збігаються з реальними даними. Також непогано себе показала Lasso регресія, але результати за тиждень 23 жовтня та 30 жовтня мають дуже великий розбіг. Decision Tree видає занадто великі піки, які не відповідають дійсності, але навідміну від Prophet він хоча б намагається. Prophet взагалі не намагається враховувати періодичність прибутку і показує нікудишні результати.

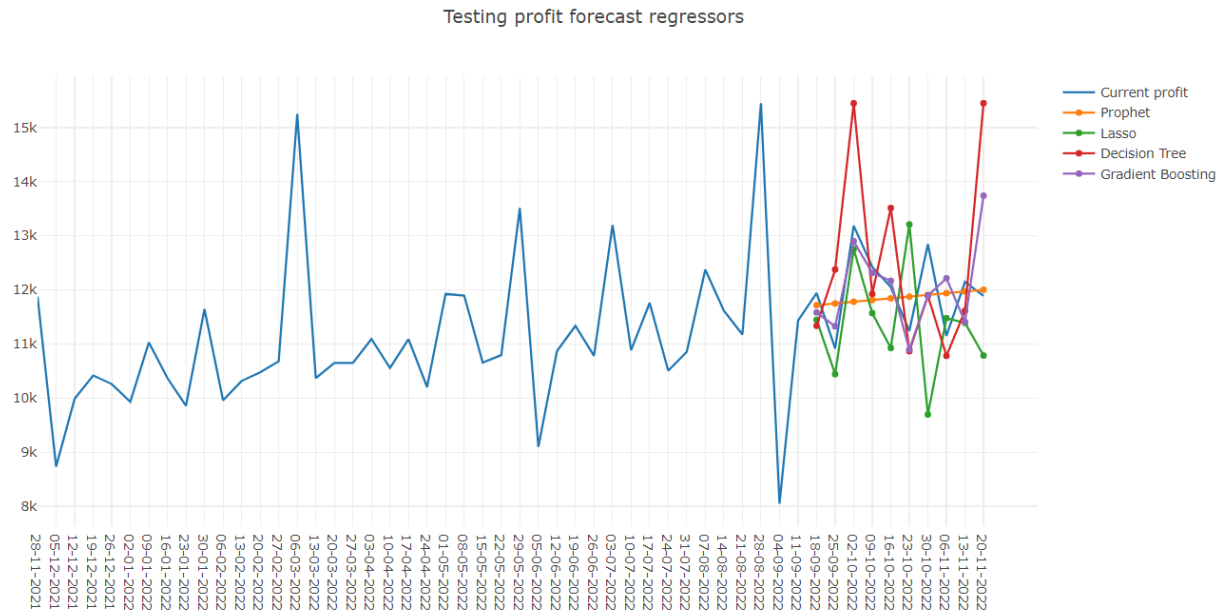


Рисунок 4.24 – Графік прогнозування прибутку на проміжку з існуючим прибутком

Також надамо результати розрахування метрики MAPE на рисунку 4.25, які схожі з вищеописаними висновками.

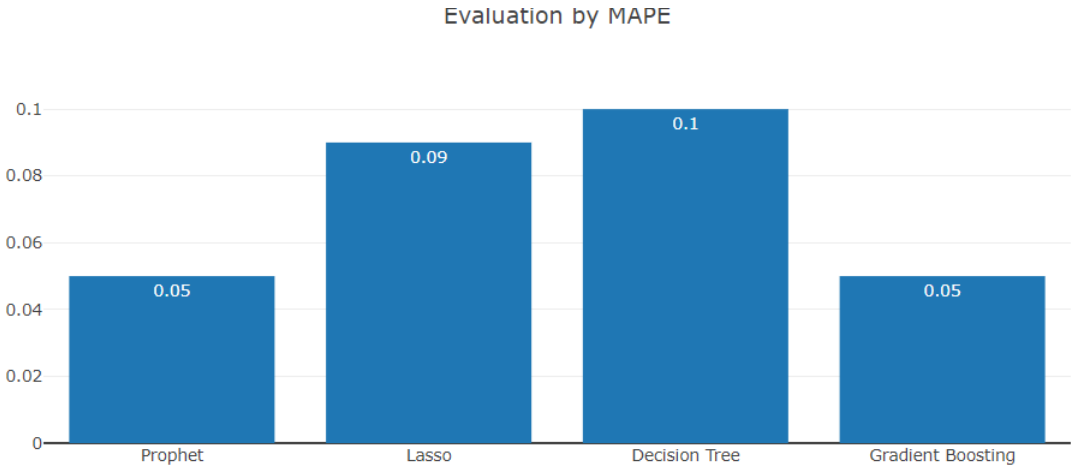


Рисунок 4.25 – Значення метрики оцінки MAPE

Таким чином була розглянута функціональність розробленої системи та надано детальну інструкцію користувачу.

Висновки до розділу 4

В даному розділі була розглянута та продемонстрована вся функціональність розробленої системи та надана детальна інструкція користувача. Система повністю вирішує поставлені задачі та може використовуватись в реальних бізнес задачах. Система має функціональність реєстрації нових користувачів, авторизації, перегляду параметрів користувача, перегляду пропозицій та тарифів продуктів, створення замовлення, виконання замовлення, резервація ресурсів, перегляду параметрів замовлення та екземплярів продуктів, розрахування загальної статистики, бізнес метрик та прогнозування прибутку різними алгоритмами.

5 РОЗРОБКА СТАРТАПУ ПРОЄКТА

В даному розділі буде проведений маркетинговий аналіз стартап проєкту, розглянуті можливості ринкового впровадження системи, проблеми, які можуть виникнути при виході на «ринок».

5.1 Описання ідеї проєкту

Розглянемо зміст ідеї проєкту, що пропонується, можливі напрямки застосування та вигоди для користувачів в таблиці 5.1.

Таблиця 5.1 – Описання ідеї проєкту

Зміст ідеї	Напрямки застосування	Вигоди для користувачів
Надання провайдерам телекомунікаційних послуг системи, що автоматизує процеси виконання замовлень, веде облік ресурсів та надає аналітику даних	Автоматизація виконання замовлень	Економія часу та людських ресурсів для виконання замовлення по підключенню клієнтів
	Облік ресурсів	Максимізація прибутку компанії
	Зберігання та управління даними клієнтів	Зменшення відтоку клієнтів
	Управління пропозиціями компанії	Можливість коригування стратегії розвитку компанії
	Використання аналітики даних	

Також для визначення конкурентноспроможності проєкту потрібно проаналізувати техніко-економічні переваги ідеї. Для цього потрібно порівняти розроблену систему з конкурентами, визначити сильні та слабкі сторони системи та спробувати мінімізувати вплив слабких сторін та відігрувати за рахунок сильних сторін. Це порівняння надано в таблиці 5.2.

Таблиця 5.2 – Визначення сильних, слабких та нейтральних характеристик

№	Характеристика	Розроблена система	Конкуренти	Результат
1	Налагодженість процесів	Так як система нова, тому потрібно підключати систему та налагоджувати процеси тощо	Конкуренти вже мають налагоджені системи, які легко можуть бути підключені	Слабка
2	Створення замовлення через інтерфейс	Надає повноцінну можливість створити замовлення через інтерфейс	Конкуренти підтримують тільки замовлення по телефону	Сильна
3	Облік ресурсів	Веде облік ресурсів при виконанні замовлення	Скоріше за все мають повноцінний облік ресурсів	Слабка
4	Зміна даних клієнта	Підтримує повне редагування даних	Зазвичай не мають такої можливості	Сильна
5	Повна інформація про замовлення та підключені послуги	Відображає детальну інформацію	Як правило коротку інформацію про послуги	Сильна
6	Надання аналітики	Надає необхідну аналітику	Зазвичай, не надають ніякої аналітики	Сильна
7	Загальна функціональність	Має необхідну функціональність	Кожна система реалізує конкретні потреби	Нейтральна

Дана таблиця показує чи є виграш у розробленої системи в порівнянні з конкурентами по ключовим характеристикам таких систем. Якщо значення в колонці результатів «Сильна», то система має перевагу у конкурентів по цій характеристиці, якщо «Слабка» - немає, якщо «Нейтральна», то системи приблизно рівні. Як бачимо, у системи є сильні та слабкі сторони, хоча сильних більше, тому є сенс намагатися впровадити стартап проєкт.

5.2 Технологічний аудит ідеї проєкту

Потрібно визначити наявність та доступність технологій для реалізації проєкту та зробити висновки щодо можливостей реалізації стартап проєкту в залежності від технологій, які він використовує. Для цього наведена таблиця 5.3.

Таблиця 5.3 – Технологічна здійсненність стартап проєкту

№	Складова проєкту	Технології реалізації	Наявність технологій	Доступність технологій
1	Можливість збереження пропозицій продуктів, інформаціх про ресурси та замовлення та стану системи	СУБД PostgreSQL, СУБД MongoDB	Наявні	Доступні. СУБД умовно безкоштовні
2	Автоматизація процесу виконання замовлення	Мова програмування Java, Фреймворк Spring, брокер повідомлень RabbitMQ	Наявні	Доступні, Java безкоштовна тільки з OpenJDK, Spring безкоштовний

Продовження таблиці 5.3

3	Оплата послуг (білінг)	Мова програмування Java, Фреймворк Spring, система оплати	Наявні	Доступні, система оплати платна та потребує інтеграції
4	Аналітика даних	Мова програму-вання Python, фреймворк Flask, бібліотеки numpy, pandas, scikit-learn, sktime	Наявні	Доступні та безкоштовні
5	Графічний інтерфейс та візуалізація даних	Мова програму-вання TypeScript, фреймворк Angular, бібліотека Plotly.js	Наявні	Доступні та безкоштовні
6	Розгортування всіх сервісів	Засіб розгортування та контейнеризації Docker, сервери для розгортування сервісів	Наявні	Docker умовно безкоштовний, сервери платні

Як бачимо з таблиці, всі технології, які потрібні для реалізації стартап проєкту наявні, більшість з них доступні і безкоштовні, але деякі платні. Можна зробити висновок, що реалізація стартап проєкту можлива.

5.3 Аналіз ринкових можливостей запуску стартап проєкту

Обов’язковою умовою перед впровадженням стартап проєкту є аналіз ринкових можливостей, які будуть сприяти впровадженню проєкту та загроз, які можуть нашкодити. Також потрібно урахувати стан ринку, потреби потенційних клієнтів і пропозиції та можливості конкурентів. Для початку розглянемо стан ринку та охарактеризуємо його в таблиці 5.4.

Таблиця 5.4 – Характеристика ринку стартап проєкту

№	Показник стану ринку	Характеристика
1	Кількість конкурентів	100+
2	Загальний обсяг продажів	мільйони доларів
3	Динаміка ринку	Постійна, без змін
4	Наявність обмежень для входу	Немає
5	Вимоги до специфікації	В деяких випадках рекомендовано використовувати специфікацію TMF
6	Середня норма рентабельності, %	35%

За попередньою оцінкою ринку та його характеристикою ми бачимо, що є вагомі проблеми, які не сприяють легкому впровадженню стартапу. Перш за все це багато конкурентів та той факт, що нові провайдери телекомунікаційних послуг не з’являються на ринку, тобто динаміка ринку не зростає. Також рентабельність хоча й вище банківської ставки, але вона не велика. Визначимо потенційних клієнтів, їхні характеристики та їхні вимоги.

Таблиця 5.5 – Характеристика потенційних клієнтів

№	Потреби ринку	Цільова аудиторія	Вимоги клієнтів
1	Автоматизація процесу виконання замовлення по підключенню телекомунікаційних послуг	Невеликі провайдери телекомунікаційних послуг	Легкість підключення до системи, можливості розширення параметрами замовлення
2	Облік телекомунікаційних ресурсів	Невеликі провайдери телекомунікаційних послуг	Можливість обліку широкого спектру ресурсів та можливості оперування ними
3	Аналітика даних	Будь-який бізнес	Загальна статистика, підрахунок бізнес метрик, прогнозування

Були розглянуті потенційні клієнти, їхні потреби та очікування, щоб розуміти наскільки реалізована система їх задовольняє, на на чому потрібно сконцетруватись та покращити. Також необхідно розглянути фактори загроз та можливостей, що наведено в таблицях 5.6 та 5.7.

Таблиця 5.6 – Фактори загроз стартап проєкту

№	Фактор	Зміст загрози	Реакції клієнта
1	Відсутність нових провайдерів телекомунікаційних послуг	Якщо немає нових провайдерів, які заходять на ринок, не буде кому продавати систему	-
2	Несумісність системи з інфраструктурою провайдера	Можливо таке, що у провайдера інфраструктура повністю відрізняється від середньої і доведеться переписувати кодову базу для адаптації	Клієнт може зачекати закінчення адоптації системи, а може придбати систему у конкурентів
3	Вісутність необхідної функціональності	Також можливо, що провайдер буде потребувати додаткової функціональності, якої немає в системі	Клієнт може зачекати закінчення реалізації необхідної функціо-нальності, а може придбати систему у конкурентів

Таблиця 5.7 – Фактори можливостей стартап проєкту

№	Фактор	Зміст можливості	Реакція клієнта
1	Зростання кількості провайдерів на ринку	Чим більше провайдерів на ринку тим більша вірогідність впровадження реалізованої системи	Покупка реалізованої системи
2	Зростання потреби в аналітиці даних	Так як конкуренти не надають можливості аналітики даних, а реалізована система надає, тому це стумулов до покупки систем	Покупка реалізованої системи

Кінцевим етапом аналізу ринкових можливостей є складання SWOT таблиці, тобто матриці сильних та слабких сторін (Strengths, Weaknesses) і можливостей та загроз (Opportunities, Threats). Ця матриця наведена в таблиці 5.8.

Таблиця 5.8 – Матриця SWOT аналізу

Сильні стороні	Слабкі стороні
Можливість створення користувачем замовлення через інтерфейс системи Детальная аналітика даних Легкість та швидкість розгортування Повний спектр валідацій під час створення замовлення	Недостатня функціональність для деяких провайдерів Складність з підключенням до інфраструктури провайдера
Можливості	Загрози
Збільшення кількості провайдерів телекомунікаційних послуг Зростання потреби в аналітиці бізнесу	Відсутність нових провайдерів телекомунікаційних послуг Зменшення рентабельності подібних систем Висока конкуренція з існуючими розробниками таких систем

Висновки до розділу 5

В даному розділі була описана ідея проєкту, проведений технологічний аудит проєкту та детальний аналіз ринкових можливостей. Базуючись на вищеописаних даних можна зазначити, що існують серйозні проблеми та загрози на шляху до впровадження стартапу, а саме – стогнуючий ринок, на якому не з’являються нові провайдери, достатня конкуренція та невисока рентабельність. Ці фактори ставлять під сумнів можливість успіху стартапу, незважаючи на вищевказані можливості.

ВИСНОВКИ

Була виконана магістерська дисертація, в рамках якої була реалізована система, що автоматизує процес виконання замовлення, облік ресурсів та надає аналітику даних для потреб бізнесу. Система має наступну функціональність: реєстрація та авторизація користувачів, перегляд пропозицій та тарифів продуктів, перегляд та зміна персональних даних, створення замовлення через інтерфейс, перевірка можливості виконання замовлення, резервація та облік ресурсів під час виконання замовлення та надання аналітики даних.

В роботі були використані наступні архітектури: клієнт-серверна архітектура, яка розділяє систему на 2 складові – клієнтську та серверну, багатошарова, яка розділяє зони відповідальності на абстрактні шари та мікросервісна архітектура, яка розділяє систему на незалежні компоненти, які реалізують конкретну функціональність. Такий набір архітектур робить систему надійною, відмовостійкою, її легко підтримувати та розробляти, а також розгортати на серверах.

Також в роботі були використані наступні технології: Java як основна мова програмування для серверної частини, Pyhon як мова програмування для мікросервісу аналітики даних, TypeScript як мова програмування для клієнтської частини, стек фреймворків Spring як основний фреймворк для серверної частини, Angular як основний фреймворк для клієнтської частини, RabbitMQ як брокер повідомлень та Docker як засіб розгортування та контейнеризації та бібліотеки numpy, pandas, scikit-learn, sktime для аналітики даних.

Розроблена система повністю виконує поставлену задачу, відповідає всім вимогам та може використовуватися провайдерами телекомунікаційни послуг. Окрім цього, був проведений маркетинговий та ринковий аналіз для оцінки можливості впровадження реалізованої сисетми в якості стартап проєкту. Можна сказати, що це можливо, але з суттєвими складнощами та проблемами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Invest Implications: “Magic Quadrant for Integrated Revenue and Customer Management for CSP” [Електронний ресурс]. – Режим доступу: https://www.gartner.com/en/documents/3156622
2. Netcracker BSS Portfolio [Електронний ресурс]. – Режим доступу: https://www.netcracker.com/portfolio/products/digital-bss/
3. Netcracker OSS Portfolio [Електронний ресурс]. – Режим доступу: https://www.netcracker.com/portfolio/products/digital-oss/
4. Amdocs OSS Portfolio Presentation [Електронний ресурс]. – Режим доступу: http://solutions.amdocs.com/rs/647-OJR-802/images/Beyond-Next-generation-OSS-whitepaper.pdf
5. Офіційна сторінка Bilink [Електронний ресурс]. – Режим доступу: http://bilink.ua/ukr/
6. Офіційна сторінка Softlink [Електронний ресурс]. – Режим доступу: https://www.softlink.kiev.ua/ua
7. Berson Alex Client/server architecture, 1992. 245p.
8. Tait James The Architecture Concept Book, 2018. 280p.
9. Masse Mark REST API Design Rulebook / Publisher by O'Reilly Media 1st edition, 2011. 112p.
10. Irakli Nadareishvili, Ronnie Mitra, Matt McLarty, Mike Amundsen Microservice Architecture Aligning Principles, Practices, and Culture / Published by O’Reilly Media, June 2016. 697p.
11. Herbert Schildt. Java: The Complete Reference, 10-th edition / McGraw-Hill Education, 2017. 1312 p.
12. Mark Lutz Programming Python, 4-th Edition / Published by O’Reilly Media, 2010. 378p.
13. Craig Walls. Spring in Action, 4-th Edition / Craig Walls – Wiley India Pvt. Limited, 2015. 424 p.

14. Miguel Grinberg Flask Web Development 2-nd edition / Published by O'Reilly Media, 2018. 178p.
15. Wes McKinney Python for Data Analysis / Published by O'Reilly Media, 2012. 312p.
16. Michael J. Hernandez Database Design for Mere Mortals / Published by O'Reilly Media, 2008. 462p.
17. Pramod Sadalage, Martin Fowler NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence 1st Edition / Addison-Wesley Professional, 2012. 192p.
18. Emrah Ayanoglu, Yusuf Aytas, Dotan Nahum, Mastering RabbitMQ / Packt Publishing, 2016. 123p.
19. Karl Matthias, Sean P. Kane Docker: Up & Running: Shipping Reliable Containers in Production 1st Edition / Published by O'Reilly Media, 2015. – 230p.
20. Lasse Koskela Effective Unit Testing 1st Edition / Manning Publications Company, 2013. 248p.
21. Boris Cherny Programming TypeScript: Making Your JavaScript Applications Scale 1st Edition / Published by O'Reilly Media, 2019. 324p.
22. Jeremy Wilken Angular in Action 1st Edition / Manning, 2018. 320p.
23. Kieran Healy Data Visualization: A Practical Introduction 1st Edition / Princeton University Press, 2018. 296p.
24. Norval D Glenn Cohort Analysis / Sage Pubns, 1977. 245p.
25. Mark Graban, Donald J. Wheeler Measures of Success: React Less, Lead Better, Improve More / Constancy, Inc., 2018. 319p.
26. Andrew Gelman, Jennifer Hill Data Analysis Using Regression and Multilevel/Hierarchical Models / Cambridge University Press, 2006. 648p.

ДОДАТОК А

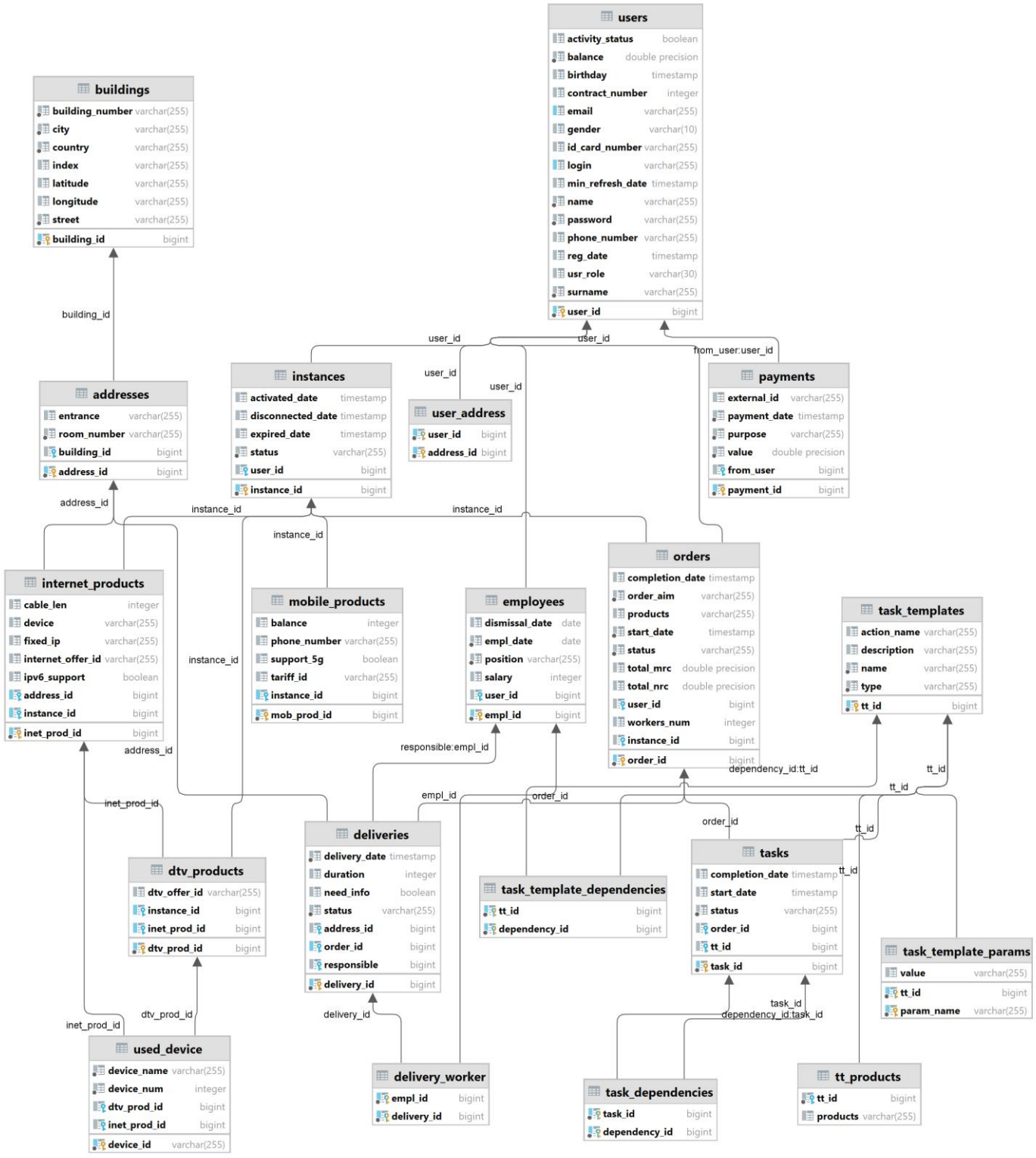
ER діаграма бази даних baoss_db

Інформаційна система управління та аналізу замовлень телекомунікаційних компаній з використанням мікросервісної архітектури

УКР.НТУУ«КПІ»_НН ІАТЕ_ІПЗЕ_ТВ-12мп

Аркушів 1

2022


users
activity_status boolean
balance double precision
birthday timestamp
contract_number integer
email varchar(255)
gender varchar(10)
id_card_number varchar(255)
login varchar(255)
min_refresh_date timestamp
name varchar(255)
password varchar(255)
phone_number varchar(255)
reg_date timestamp
usr_role varchar(30)
surname varchar(255)
user_id bigint
buildings
building_number varchar(255)
city varchar(255)
country varchar(255)
index varchar(255)
latitude varchar(255)
longitude varchar(255)
street varchar(255)
building_id bigint
addresses
entrance varchar(255)
room_number varchar(255)
building_id bigint
address_id bigint
instances
activated_date timestamp
disconnected_date timestamp
expired_date timestamp
status varchar(255)
user_id bigint
instance_id bigint
user_address
user_id bigint
address_id bigint
payments
external_id varchar(255)
payment_date timestamp
purpose varchar(255)
value double precision
from_user bigint
payment_id bigint
orders
completion_date timestamp
order_aim varchar(255)
products varchar(255)
start_date timestamp
status varchar(255)
total_mrc double precision
total_nrc double precision
user_id bigint
workers_num integer
instance_id bigint
order_id bigint
internet_products
cable_len integer
device varchar(255)
fixed_ip varchar(255)
internet_offer_id varchar(255)
ipv6_support boolean
address_id bigint
instance_id bigint
inet_prod_id bigint
mobile_products
balance integer
phone_number varchar(255)
support_5g boolean
tariff_id varchar(255)
instance_id bigint
mob_prod_id bigint
employees
dismissal_date date
empl_date date
position varchar(255)
salary integer
user_id bigint
empl_id bigint
task_templates
action_name varchar(255)
description varchar(255)
name varchar(255)
type varchar(255)
tt_id bigint
deliveries
delivery_date timestamp
duration integer
need_info boolean
status varchar(255)
address_id bigint
order_id bigint
responsible bigint
delivery_id bigint
dtv_products
dtv_offer_id varchar(255)
instance_id bigint
inet_prod_id bigint
dtv_prod_id bigint
task_template_dependencies
tt_id bigint
dependency_id bigint
tasks
completion_date timestamp
start_date timestamp
status varchar(255)
order_id bigint
tt_id bigint
task_id bigint
task_template_params
value varchar(255)
tt_id bigint
param_name varchar(255)
used_device
device_name varchar(255)
device_num integer
dtv_prod_id bigint
inet_prod_id bigint
device_id varchar(255)
delivery_worker
empl_id bigint
delivery_id bigint
task_dependencies
task_id bigint
dependency_id bigint
tt_products
tt_id bigint
products varchar(255)
building_id
user_id
user_id
user_id
from_user:user_id
user_id
user_id
address_id
instance_id
instance_id
instance_id
instance_id
address_id
responsible:empl_id
dependency_id:tt_id
tt_id
tt_id
empl_id
order_id
order_id
tt_id
tt_id
inet_prod_id
inet_prod_id
dtv_prod_id
delivery_id
task_id
dependency_id:task_id

ДОДАТОК Б

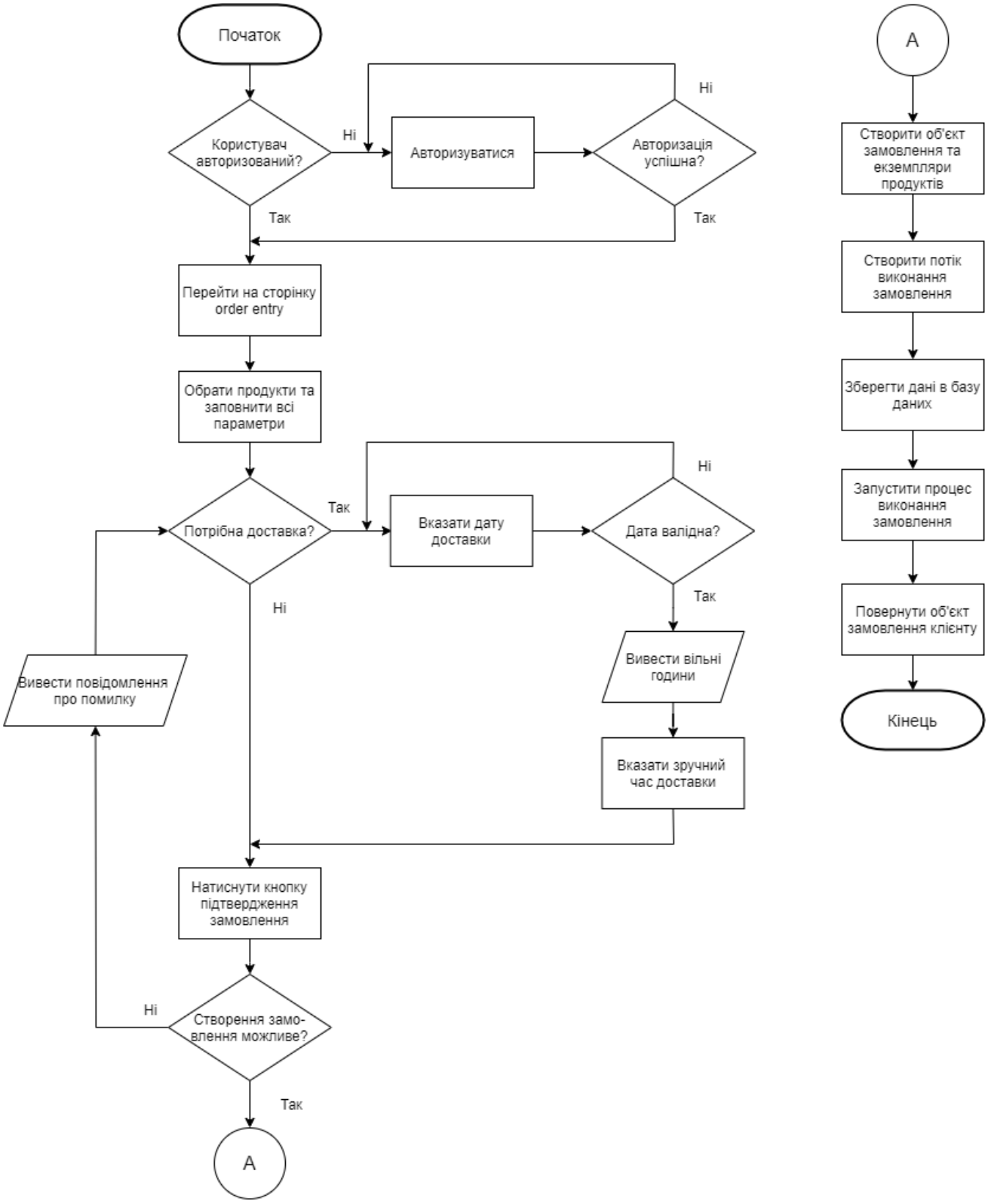
Блок-схема алгоритму створення замовлення

Інформаційна система управління та аналізу замовлень телекомунікаційних компаній з використанням мікросервісної архітектури

УКР.НТУУ«КПІ»_НН ІАТЕ_ІПЗЕ_ТВ-12мп

Аркушів 1

2022


Початок
Користувач авторизований?
Ні
Авторизуватися
Авторизація успішна?
Ні
Так
Так
Перейти на сторінку order entry
Обрати продукти та заповнити всі параметри
Потрібна доставка?
Так
Вказати дату доставки
Дата валідна?
Ні
Так
Ні
Вивести повідомлення про помилку
Вивести вільні години
Вказати зручний час доставки
Натиснути кнопку підтвердження замовлення
Створення замовлення можливе?
Ні
Так
A
A
Створити об'єкт замовлення та екземпляри продуктів
Створити потік виконання замовлення
Зберегти дані в базу даних
Запустити процес виконання замовлення
Повернути об'єкт замовлення клієнту
Кінець

ДОДАТОК В

Публікації

Інформаційна система управління та аналізу замовлень телекомунікаційних компаній з використанням мікросервісної архітектури

УКР.НТУУ«КПІ»_НН ІАТЕ_ІПЗЕ_ТВ-12мп

Аркушів 11

2022

Міністерство освіти і науки України
Мелітопольський державний педагогічний університет
імені Богдана Хмельницького
Українська асоціація з прикладної геометрії
Мелітопольська школа прикладної геометрії

СУЧАСНІ ПРОБЛЕМИ МОДЕЛЮВАННЯ

ЗБІРНИК НАУКОВИХ ПРАЦЬ

Випуск 25

м. Мелітополь

УДК 004.75/515.14

Залевська О.В., к.т.н.,
o.zalevska@kpi.ua, 0000-0002-3163-1695
Гагарін О.О., к.т.н.,
gagarin.info@gmail.com, 0000-0001-5130-7043
Комісаров І.А,
emidot32@gmail.com, 0000-0001-8177-9092

ЗАЛЕВСЬКА О.В., К.Т.Н.
О.О. ГАГАРІН, І.А. КОМІСАРОВ
Національний технічний університет України “Київський політехнічний інститут імені Ігоря Сікорського”

ОСОБЛИВОСТІ СТРУКТУРИ СИСТЕМ ОБРОБКИ ЗАМОВЛЕНЬ АБОНЕНТІВ ТЕЛЕКОМУНІКАЦІЙНИХ ПОСЛУГ

З розвитком технологій програмування все більше процесів в побуті та в бізнесі почали автоматизуватися і сфера телекомунікацій не стала виключенням. Ще в минулому столітті у телекомунікаційних компаній виникла потреба зберігати дані клієнтів, вести облік ресурсів, налагоджувати процеис підключення нових клієнтів до мережі, заздалегідь проектувати локальні мережі, моніторити їх стан та швидко приймати необхідні рішення в не спиятливих ситуаціях.

У даній статті буде розглянуто структуру, функціональність ініціалізації замовлень клієнтів телекомукаційних послуг та способи їх обробки. Розглянута проблема організації системи обробки замовлень для підключення абонентів до мобільної або інтернет мережі провайдера. Розроблено та проаналізовано моделі систем обробки замовлень, які базуються на системах масового обслуговування та використовують черги для ініціалізації їх обробки. Параметрами порівняння моделей слугують такі показники, яч темп надходження заявок, середній час перебування заявки в черзі, середня довжина черги тощо.

В матеріалах статті наведено типи та необхідна функціональність для системи обробки замовлень клієнтів, моделі системи масового обслуговування,

організація черг із замовлень клієнтів, приведено сєхеми таких моделей та проаналізовано їх недоліки та переваги при використанні їх на практиці.

*В роботі наведено вхідні параметри для моделювання кожного з отриманих варіантів системи, що залежать від типу замовлення. В результатами процесу моделювання стають відомими такі параметри системи, як дані каналу обслуговування (NOH, MOH,SOH,ROH,DOH,IFOH, TON, EON), прийняті замовлення, коєфіцієнт використання та очікуючі замовлення. Побудова варіантів систем реалізовувалась в середовищі GPSS World*ю.

Ключові слова: черга, система масового обслуговування, замовлення, телекомунікації, GPSS World.

Постановка проблеми. Процес ініціалізації замовлення починається з веб-порталу, на якому користувач вибирає необхідні продукти (Інтернет, цифрове телебачення, мобільний тариф тощо). Процес подання заяки потребує перевірки параметрів виконання замовлення. Такими параметрами можуть бути підключеня будиноку користувача до мережі провайдера, наявність знижки на товар і так далі. Обробка замовлення займає досить великий проміжок часу, оскільки має враховувати розбиття замовлення на об’єкти більше конкретних підзамовлень, розрахунок технічних параметрів, організація та підготовка до білінгу, додання параметрів до бази даних. Для реалізації обробки таких замовлень використовуються асинхронні запити та черги, які потребують їх орієнтації. Проблема організації черг для досягнення максимальної ефективності системи та їх моделювпання розглянуто в роботі.

Аналіз досліджень та публікацій Існує досить багато розробок по системах масового обслуговування(СМО), в яких розглядається як математична модель так і результати імітаційного моделювання системи. Недоліком більшісті з них є неврахування способів організації черг для конкретних сфер бізнесу та їх впливу на нього. Наприклад, в статті [1] пропонується оптимізація для торгового підприємства "Атол" на основі моделювання СМО, що враховує не експоненціальній закон розподілу заявок (клієнтів), модель стимуляції персоналу, витрати підприємства. Але в виду специфіки бізнесу, організація черг заявок не розглядається. Специфіка роботи інтернет-магазину, яка розглядається в статті [2] доволі схожа на роботу провайдерів телекомунікаційних послуг без процеса моделювання. В даній роботі для моделювання використовується система GPSS

World були детально вивчені публікації [3][4]. Вони надають чіткі інструкції і приклади, та охоплюють широкий спектр можливостей для моделювання в цьому середовищі.

Цілі статті. Побудувати та дослідити математичну мадель процесу обробки замовлень з урахуванням огранізації їх черг. Врахувати розбиття замовлення на об'єкти більш конкретних підзамовлень, розрахунок технічних параметрів, організація та підготовка до білінгу, додання параметрів до бази даних.

Головна частина. Потреби і навантаження в таких компаніях можуть значно відрізнятися в залежності від їх особливостей. Розглянемо можливості та потреби великих телекомунікаційних провайдерів у яких існує декілька типів замовлень, які мають різну логіку обробки. До них належать [5]:

- замовлення на підключення нового абонента до мережі, надання певного обладнання тощо;
- модифікація вже виконаного замовлення;
- тимчасове призупинення надання послуги;
- поновлення призупиненої послуги;
- відключення послуги;
- підключення послуги в іншій локації і відключення на поточній,;
- випуск замовлення, яке змінює попереднє, поки попереднє замовлення ще не було виконане.

В процесі виконання замовлення можуть виникнути непередбачувані обставини або помилки. В такому разі замовлення має бути оброблене іншим чином, що необхідно врахувати.

Розглянемо декілька варіантів організації черг:

1) Одна черга для всіх типів замовлень, черга для оплати, черга для помилок.
2) Інший варіант, який може спасти на думку, зважаючи на недоліки першого є створення черг для кожного типу замовлення.

Перший варіант є прийнятним і, у випадках, коли кількість замовлень невелика, або замовлення обробляються швидко, така модель може себе гарно показувати. При ускладненні системи, при збільшенні часу обробки замовлення та при збільшенні кількості поступаючих замовлень, такі технічні показники, як час заявки в черзі, довжина черги, час знаходження в системі обслуговування, та такі

економічні показники, як витрати очікування в черзі/системі обслуговування, погіршаться.

На рисунку 1 зображена схема моделі СМО. На ній бачимо всі можливі варіанти розвитку подій та сервіси, які відповідальні за ту чи іншу функціональність. Як і зазначалось вище, всі замовлення попадають в загальну чергу, після цього розподільник замовлень забирає замовлення та визначає відповідного до типу та продукту замовлення обробника. Якщо замовлення було оброблене успішно, то воно зберігається в базу даних та використовується в подальшій роботі, поки послуга замовлення не буде підключена. Якщо ж виникли проблеми, то замовлення відправляється в чергу замовлень з помилками і звідти до обробника таких замовлень. Також, коли послуга буде підключена запускається billing сервіс, який отримує в бази даних об’єкт відповідного замовлення та відправляє відповідні дані в чергу на оплату, звідки система оплати їх забирає, виконує оплату та відправляє підтвердження про успішну оплату до відповідної черги.

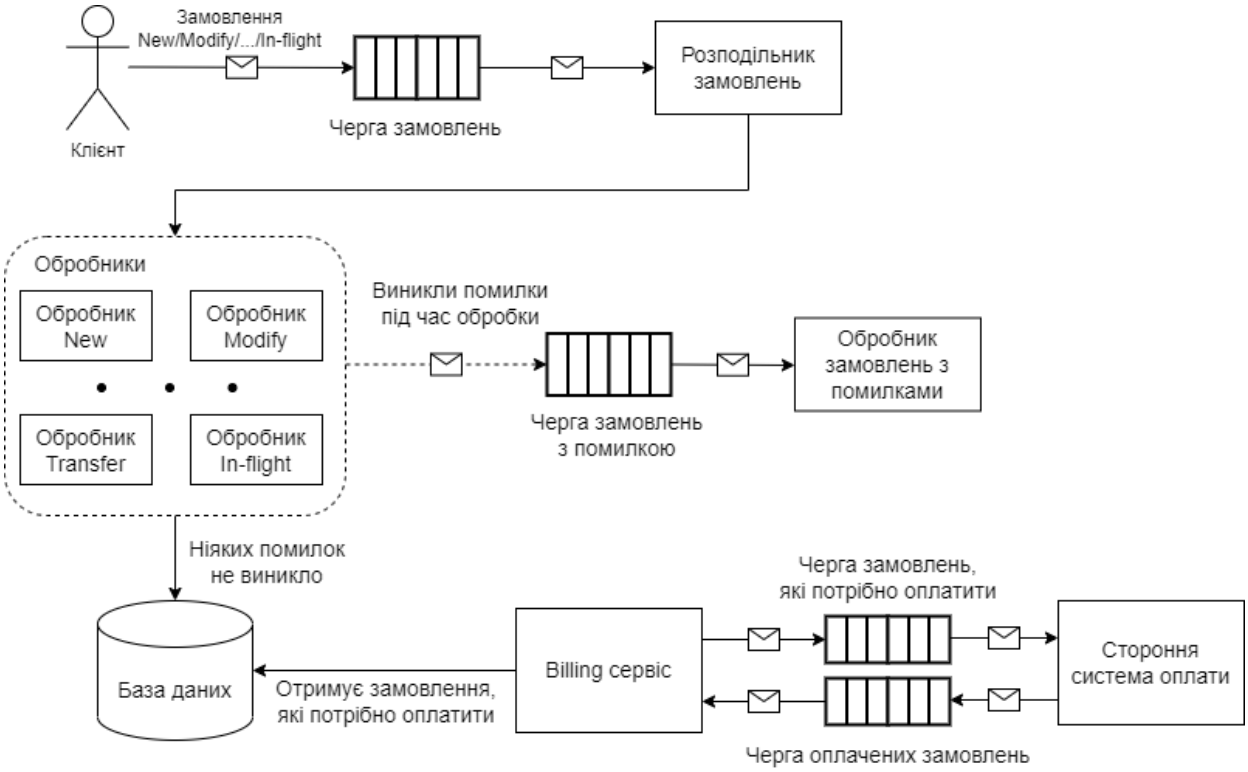


Рис.1 – Схема моделі СМО першого варіанту

Наведемо переваги та недоліки такого варіанту:

Переваги:

- Простота реалізації;
- Простота підтримки в майбутньому;

- Економія обчислювальних ресурсів.

Недоліки:

- Необхідність в реалізації додаткової фільтрації замовлень по типу для вибору відповідного обробника;
- При збільшенні кількості замовлень погіршуються показники СМО;\

На рисунку 2 зображена схема другого варіанту. Черги названі скорочено англійскою відповідно до типу замовлення: замовлення New – NOQ (New Order Queue), замовлення Transfer – TOQ (Transfer Order Queue), замовлення In-flight – IFOQ (In-Flight Order Queue) і т.д. Як бачимо, замовлення після черги відразу попадають до відповідного обробника, у випадку помилки замовлення будь-якого типу попадає в спільну чергу, так як замовлень з помилками не дуже багато в будь-якому випадку.

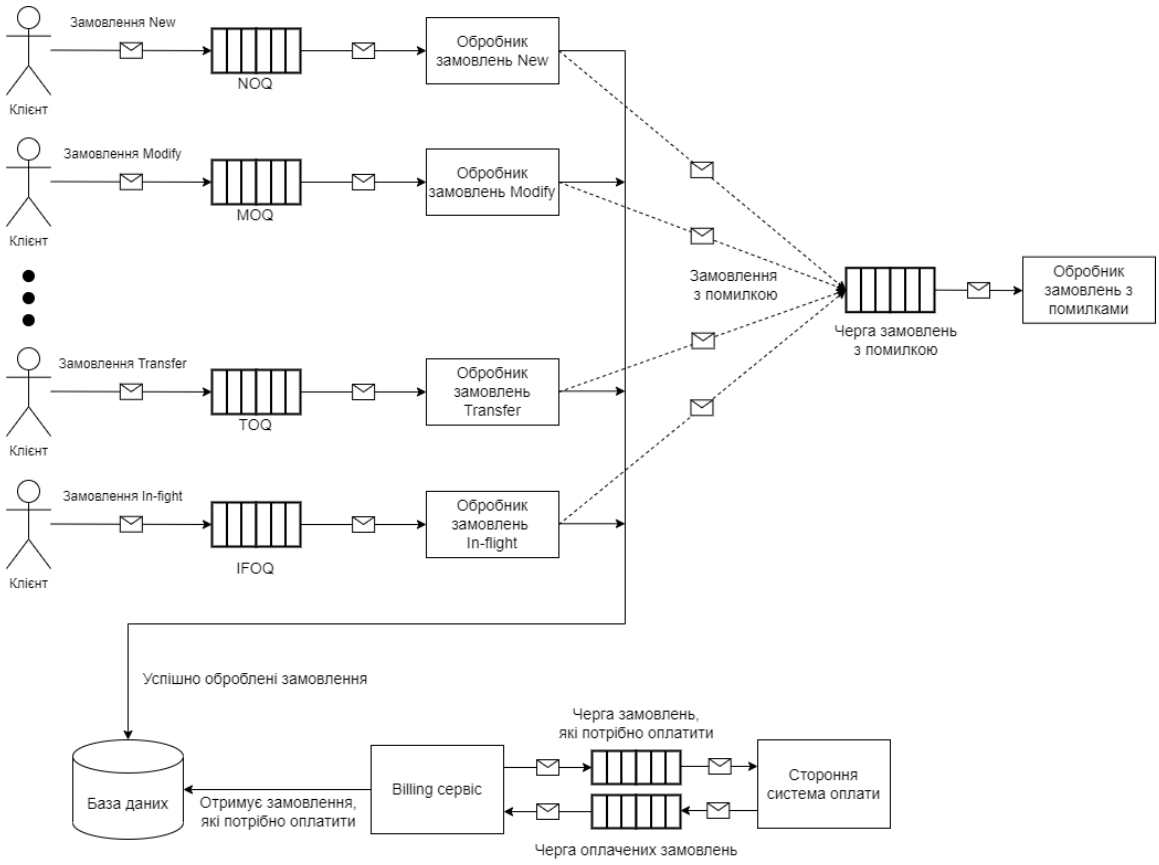


Рис.2 – Схема моделі СМО другого варіанту

Наведемо переваги та недоліки такого варіанту:

Переваги:

- Повна масштабованість;

- Стійка до великих навантажень.

Недоліки:

- Дорога в реалізації, якщо на кожну чергу створювати власний сервер;
- Неефективна, так як деякі типи замовлень рідко надходять до системи, що зумовлює простоювання цих черг.

З урахуванням недоліків першого та другого варіантів є сенс скомбінувати деякі черги, типи замовлень яких не дуже часто надходять до системи. Комбінувати можна по-різному. Наприклад, такі замовлення як New та Modify, які складають більшість замовлень, що надходять в систему будуть мати свої незалежні черги, типи замовлень Suspend, Resume, Disconnect – свою спільну чергу, і такі рідкі замовлення як Transfer та In-flight також будуть відправлятися в спільну чергу. Якщо у провайдера великий відсоток замовлень Resume, Suspend, Disconnect, то вони також можуть бути розділені на власні черги. Розподільник замовлень буде реалізований тільки для спільних черг.

Наведемо переваги та недоліки такого варіанту:

Переваги:

- Стійка до великих навантажень;
- Ефективна – при вдалому групуванні черги будуть заповнюватись рівномірно, так само як і навантаження на систему.

Недоліки:

- Відносно складна;
- Потрібна деяка статистика по кількості замовлень кожного типу до всіх замовлень, що надходять в систему в проміжок часу, для коректного групування по типу замовлень в одну чергу;
- При додаванні нового типу замовлення виникає питання до якої черги його додавати чи створювати нову.

Для моделювання використаємо середовище та мову моделювання GPSS World. Цей інструмент дозволяє моделювати СМО різної складності, збирати статистичні дані після моделювання та відображати різні графіки.

Кожен варіант промодельований при потенційно максимальному навантаженні та при середньому. Параметри, які можна змінювати в GPSS World для регулювання навантаження – час між надходженням замовлень (вимог, транзактів), час обробки одного замовлення відповідного типу. Очевидно, що в залежності від типу замовлення ці 2 параметра будуть відрізнятись. Наведемо таблицю з вхідними параметрами для моделювання для всіх типів замовлення.

Таблиця №1. Вхідні параметри для моделювання для кожного типу замовлення.

Параметр / Тип	Час надход-ження макс. наванта ж., с	Інтенси-вність макс, замов./хв	Час надход-ження сер. наванта ж., с	Інтенси-вність сер, замов./хв	Час об-робки з розпод., с	Час об-робки без розпод., с
New	0,1	600	6	10	2	1.7
Modify	0,3	200	10	6	1.5	1.2
Suspend	0,6	100	20	3	1.3	1
Resume	0,6	100	20	3	1.3	1
Disconnect	0,5	120	15	4	1.5	1.2
In-flight	2	30	60	1	3	2.7
Transfer/Ch.Own.	10	6	120	0.5	4.5	4.3
Error	120	0.5	600	0.1	-	5

Очевидно, що найчастіше до системи надходять New замовлення та трохи рідще Modify, тому їх буде найбільше, також приймемо, що Suspend та Resume замовлень надходить однаково для балансу. In-flight та Transfer замовлення найбільш дуже рідко зустрічаються, а помилкових замовлень повинно бути по мінімуму в налагодженій системі. В деяких системах при максимальному навантаженні кількість замовлень може бути десятки тисяч в хвилину, в нашому випадку, приймемо, що сумарна їх кількість дорівнює ~1200 в хвилину.

З часом обробки важче визначитись, так як в залежності від системи воно може відрізнятися на порядок. В середньому, якщо прийняти до уваги складну структуру замовлення, яка може розбиватися на свої підзамовлення, взяти до уваги величезну кількість інтеграцій, яким потрібно відправити доволі об’ємні дані,

збереження всіх записів до бази даних, та запустити потік виконання замовлення, то вийде не менше 1 секунди, для New, Disconnect, Transfer, In-flight цей час буде близько 2 секунд. Час на розподілення замовлень з черги до конкретного обробника – 0.3 секунди.

Для моделювання максимального навантаження запустимо модель на 600 секунд для потенційних 5000 замовлень (при можливості більше). Для моделювання середнього навантаження будемо моделювати 5000 секунд при 10000 замовлень.

При таких параметрах в першому варіанті при максимальному навантаженні максимальна довжина черги найбільша, як і середня довжина черги. Це свідчить про багато замовлень, які не встигають оброблюватись і навіть замовлення, які могли б бути швидко опрацьовані системою повинні чекати, поки до них дійде черга, адже найбільше складнощів виникає саме з New та Modify замовленнями, яких найбільше, інші замовлення вимушені чекати на їхню обробку. Але при середньому навантаженні всі замовлення встигають оброблюватися та коефіцієнт використання каналі найбільший серед всіх варіантів, що каже про ефективність такої системи.

У другому варіанті ми бачимо, що, навіть, якщо повністю розділити замовлення по типах, то це не вирішить проблему, коли замовлення не встигають оброблюватись, але це вирішує проблему, коли замовлення, які рідко надходять до системи не чекають, поки обробляться замовлення, які часто надходять. В цьому варіанті найменша середня кількість замовлень в чергах, але при цьому деякі черги простоюють навість при максимальному навантаженні, низька ефективність підтверджується і найменшим коефіцієнтом використання.

У наступносму було згенеровано найбільше замовлень в порівнянні з іншими, тому деякі характеристики завищені. В цілому, в цьому варіанті черги заповнюються максимально рівномірно, якщо не рахувати чергу error_orders, за виключенням черги IFTOQ, так як замовлення In-flight, Transfer доволі рідко надходять. Також коефіцієнт використання другий після варіанту 1, що гарним показником ефективності.

Кінцевий варіант є середнім між варіантом 2 та 3.1, він менш ефективний ніж попередній, але черги не так сильно простоюють. Цей варіант має місце на існування, якщо замовлень Suspend/Resume/Disconnect доволі багато в системі.

Висновки та перспективи. Запропоновані варіанти організації черг СМОЗ показують, що для компаній, в яких не передбачається великих навантажень, достатньо однієї черги для всіх замовлень, такий варіант максимально ефективний і з точки зору використання ресурсів компанії і з точки зору систем масового обслуговування. Для компаній в яких в деякі моменти часу навантаження можуть доходити до десятків-сотні тисяч замовлень в секунду цю проблему можна вирішити шляхом розділення однієї черги на декілька та групуванням по типу замовлення, та використовуючи горизонтальне масштабування вже в розгортанні реалізованих систем. Найбільш ефективним буде для замовлень New, Modify, Disconnect створити власні черги, а замовлення Suspend, Resume, In-flight, Transfer об'єднати в одну з подальшим розподіленням.

Література

1. Дуплякин В. М., Скогарева Ю. В. Моделирование системы массового обслуживания торгового предприятия // Вестник ОГУ. 2009. №1.
2. Скіцько Володимир Іванович, Ігнатова Юлія Володимирівна МОДЕЛЮВАННЯ ЛОГіСТИЧНИХ ПРОЦЕСіВ ВИКОНАННЯ ЗАМОВЛЕНЬ ІНТЕРНЕТ-МАГАЗИНОМ ЯК МЕРЕЖі МАСОВОГО ОБСЛУГОВУВАННЯ // БИ. 2015. №8 (451).
3. В. Б. Толубко, А.Д. Кожухівський, В.В. Вишнівський, Г.І. Гайдур, О.А. Кожухівська. // Імітаційне моделювання систем масового обслуговування – Київ: 175 с
4. Аверьянов В. Т., Полынько С. В. Имитационное моделирование системы массового обслуживания на языке GPSS World // Научно-аналитический журнал «Вестник Санкт-Петербургского университета Государственной противопожарной службы МЧС России». 2010. №3
5. Афанасьев М.Ю., Суворов Б.П., Исследование операций в экономике: модели, задачи, решения. // Модели систем массового обслуживания. – 2004. – Р. 201-245

O.O. Haharin, I.A. Komisarov

National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute"

MODELING OF QUEUING SYSTEMS AND ORGANIZATION OF ORDER QUEUES FOR CONNECTING SUBSCRIBERS TO THE NETWORK

In this article the problem of organization of the order processing system for connection of subscribers to the network (Internet or mobile) of the telecommunication service provider was considered. As a solution to this problem, models of order processing systems have been developed, which are based on queuing systems and use queues to initialize the processing of these orders.

During the development of models, the following needs were considered: stable operation under heavy loads, processing of orders, processed with errors, implementation of the functionality of payment for the order, organization of queues depending on the type of order. In total, the following types of orders were considered: New, Modify, Suspend, Resume, Disconnect, Transfer/Change Ownership, In-flight Modify, In-flight Cancel, In-flight Resume, In-flight Suspend.

The following models have been proposed: model with one common queue for all types of orders, a model where each order type has its own queue and models with different combination options by order types. The models are compared by the following parameters: the rate of receipt of applications, the average time that the application is in the queue, the average length of the queue, and so on. The results were analyzed and appropriate conclusions were made.

Keywords: queue, queuing system, order, telecommunications, GPSS World.

References

1. Duplyakin V. M., Skogareva Yu. V. Modeling of the queuing system of a commercial enterprise // Bulletin of the OSU. 2009. No. 1.

2. Skitsko Volodymyr Ivanovich, Ignatova Yuliya Volodymyrivna, Modeling of logistics processes of order fulfillment

Online store as a mass service network// BI. 2015. No. 8 (451).

3. V. B. Tolubko, A.D. Kozhukhivsky, V.V. Vishnivsky, G.I. Gaidur, O.A. Kozhukhivska. // Simulation modeling of mass service systems - Kiev: 175 s

4. Averyanov V. T., Polynko S. V. Simulation modeling of a queuing system in the GPSS World language // Scientific and analytical journal “Bulletin of the St. Petersburg University of the State Fire Service of the Ministry of Emergency Situations of Russia”. 2010. №3

5. Afanasiev M.Yu., Suvorov B.P., Research of operations in the economy: models, tasks, solutions. // Models of queuing systems. - 2004. - R. 201-245

ДОДАТОК Г

Лістинг програми

Інформаційна система управління та аналізу замовлень телекомунікаційних компаній з використанням мікросервісної архітектури

УКР.НТУУ«КПІ»_НН ІАТЕ_ІПЗЕ_ТВ-12мп

Аркушів 34

2022

docker-compose.yml

```
version: '3'
services:
  postgres-db:
    image: postgres:14.4
    container_name: postgres-db
    restart: always
    ports:
      - 5432:5432
    environment:
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=Aposuv74
      - POSTGRES_DB=baoss_db
      - POSTGRES_HOST_AUTH_METHOD=md5
    volumes:
      - postgres-volume:/var/lib/postgresql/data
      - ./logs:/logs
    networks:
      - common-network

  mongo-db:
    image: mongo
    container_name: mongo-db
    ports:
      - 27017:27017
    volumes:
      - mongo-volume:/var/lib/mongo/data
      - ./logs:/logs
    #command: mongod
    networks:
      - common-network

  rabbitmq:
    image: rabbitmq:3.9.8-management-alpine
    container_name: 'rabbitmq'
    hostname: rabbitmq
    ports:
      - 5672:5672
      - 15672:15672
    volumes:
      - rabbit-volume:/var/lib/rabbitmq/
      - ./logs:/logs
    networks:
      - common-network

  eureka-server:
    container_name: eureka-server
    build:
      context: ./eureka-server
      dockerfile: ./Dockerfile-dev
    ports:
      - 8080:8080
    expose:
      - 8080
    restart: always
    networks:
      - common-network

  tb-api:
    container_name: tb-api
    build:
      context: ./tb-api
      dockerfile: ./Dockerfile-dev
    ports:
      - 5555:5555
```

```
    expose:
      - 5555
    restart: always
    networks:
      - common-network

  user-service:
    container_name: user-service
    build:
      context: ./user-service
      dockerfile: ./Dockerfile-dev
    ports:
      - 8082:8082
    expose:
      - 8082
    restart: always
    networks:
      - common-network
    depends_on:
      - postgres-db

  order-service:
    container_name: order-service
    build:
      context: ./order-service
      dockerfile: ./Dockerfile-dev
    ports:
      - 8081:8081
    expose:
      - 8081
    environment:
      - SPRING_RABBITMQ_HOST=rabbitmq
    restart: always
    networks:
      - common-network
    depends_on:
      - postgres-db
      - offer-service
      - rabbitmq

  offer-service:
    container_name: offer-service
    build:
      context: ./offer-service
      dockerfile: ./Dockerfile-dev
    ports:
      - 8083:8083
    expose:
      - 8083
    environment:
      - IMPORT_DEFAULT_OFFERS=true
    restart: always
    networks:
      - common-network
    depends_on:
      - mongo-db

  resource-service:
    container_name: resource-service
    build:
      context: ./resource-service
      dockerfile: ./Dockerfile-dev
    ports:
      - 8084:8084
    expose:
      - 8084
```

```
    restart: always
    networks:
      - common-network
    depends_on:
      - mongo-db

  billing-service:
    container_name: billing-service
    build:
      context: ./billing-service
      dockerfile: ./Dockerfile-dev
    ports:
      - 8085:8085
    expose:
      - 8085
    restart: always
    networks:
      - common-network
    depends_on:
      - postgres-db

  analytics-service:
    container_name: analytics-service
    build:
      context: ./analytics-service
    ports:
      - 8086:8086
    expose:
      - 8086
    restart: always
    networks:
      - common-network
    depends_on:
      - postgres-db

networks:
  common-network:
    driver: bridge

volumes:
  postgres-volume:
  mongo-volume:
  rabbit-volume:
```

order-service

```
package edu.baoss.orderservice.services;

import edu.baoss.orderservice.Constants;
import edu.baoss.orderservice.model.Flow;
import edu.baoss.orderservice.model.dtos.DeliveryAdditionalInfo;
import edu.baoss.orderservice.model.dtos.DeliveryDto;
import edu.baoss.orderservice.model.dtos.OrderValue;
import edu.baoss.orderservice.model.dtos.UserDevice;
import edu.baoss.orderservice.exceptions.NoEmployeeFoundException;
import edu.baoss.orderservice.model.entities.*;
import edu.baoss.orderservice.model.enums.DeliveryStatus;
import edu.baoss.orderservice.model.enums.OrderStatus;
import edu.baoss.orderservice.model.enums.Position;
import edu.baoss.orderservice.repositories.DeliveryRepository;
import edu.baoss.orderservice.repositories.EmployeeRepository;
import lombok.AllArgsConstructor;
import lombok.NoArgsConstructor;
import lombok.RequiredArgsConstructor;
import lombok.Setter;
```

```
    import org.apache.commons.lang.time.DateUtils;
    import org.apache.commons.lang3.tuple.ImmutablePair;
    import org.apache.commons.lang3.tuple.Pair;
    import org.springframework.amqp.rabbit.core.RabbitTemplate;
    import org.springframework.beans.factory.annotation.Autowired;
    import org.springframework.stereotype.Service;

    import java.text.ParseException;
    import java.text.SimpleDateFormat;
    import java.util.*;
    import java.util.stream.Collectors;
    import java.util.stream.IntStream;

    import static edu.baoss.orderservice.Constants.ONLY_DATE_FORMAT;

    @Service
    @AllArgsConstructor
    @NoArgsConstructor
    public class DeliveryService {
        @Autowired
        private DeliveryRepository deliveryRepository;
        @Setter
        @Autowired
        private  EmployeeRepository employeeRepository;
        @Autowired
        private InstanceService instanceService;
        @Autowired
        private RabbitTemplate template;
        @Autowired
        private FulfilmentService fulfilmentService;
        @Autowired
        private FlowComposer flowComposer;

        private static final int[] HOURS =  IntStream.rangeClosed(10, 18).toArray();

        public Delivery createDelivery(OrderValue orderValue, Order order) throws ParseException {
            Pair<Integer, Integer> durationAndNumberOfEmployees =
getDurationAndNumberOfEmployees(orderValue.getSelectedProducts());
            int hour = Integer.parseInt(orderValue.getDeliveryTime().substring(0, 2));
            Date deliveryDate = Constants.DATE_HOUR_FORMAT.parse(orderValue.getDeliveryDateStr() + "
" + hour);
            List<Employee> allEmployees =
employeeRepository.findAllByPosition(Position.FITTER.getName());
            Set<Employee> workers = getFreeEmployees(deliveryDate, allEmployees,
durationAndNumberOfEmployees.getLeft(), hour)
                    .stream()
                    .limit(durationAndNumberOfEmployees.getRight())
                    .collect(Collectors.toSet());
            Employee responsible =
workers.stream().findFirst().orElseThrow(NoEmployeeFoundException::new);
            return deliveryRepository.save(Delivery.builder()
                    .deliveryDate(deliveryDate)
                    .address(new Address(orderValue.getSelectedAddress()))
                    .duration(durationAndNumberOfEmployees.getLeft())
                    .status(DeliveryStatus.NOT_STARTED)
                    .order(order)
                    .workers(workers)
                    .responsible(responsible)
                    .needInfo(orderValue.getSelectedSpeed() != null &&
orderValue.getSelectedDevice() == null)
                    .build());
        }

        public List<DeliveryDto> getTodayDeliveriesForEmployeeAndUpdateIfStarted(long userId) {
            Employee employee =
employeeRepository.getEmployeeByUserId(userId).orElseThrow(NoEmployeeFoundException::new);
```

```
            Date now = new Date();
            List<DeliveryDto> deliveriesForEmployee = new ArrayList<>();
            List<Delivery> employeeDeliveries = employee.getDeliveries().stream()
                    .sorted(Comparator.comparing(Delivery::getDeliveryDate).reversed()).toList();
            for (Delivery delivery : employeeDeliveries) {
                if (sameDay(delivery.getDeliveryDate(), now)) {
                    boolean responsible = employee.equals(delivery.getResponsible());
                    boolean deliveryStarted = isDeliveryStarted(delivery);
                    if (deliveryStarted) {
                        updateDelivery(delivery, DeliveryStatus.IN_PROGRESS,
OrderStatus.IN_DELIVERY);
                    }
                    deliveriesForEmployee.add(
                            new DeliveryDto(delivery, getAdditionalInfo(delivery), responsible,
deliveryStarted));
                }
            }
            return deliveriesForEmployee;
        }

        public List<String> getAvailableHours(Date deliveryDate, String[] products) {
            List<Employee> allEmployees =
employeeRepository.findAllByPosition(Position.FITTER.getName());
            Pair<Integer, Integer> durationAndNumberOfEmployees =
getDurationAndNumberOfEmployees(products);
            List<String> availableHours = new ArrayList<>();
            for (int hour : HOURS) {
                List<Employee> freeEmployees = getFreeEmployees(deliveryDate, allEmployees,
                        durationAndNumberOfEmployees.getLeft(), hour);
                if (freeEmployees.size() >= durationAndNumberOfEmployees.getRight())
                    availableHours.add(String.format("%d:00", hour));
            }
            return availableHours;
        }

        public DeliveryDto finishDelivery(OrderValue orderValue) {
            Delivery delivery =
deliveryRepository.findById(orderValue.getDeliveryId()).orElseThrow(RuntimeException::new);
            orderValue.setOrder(delivery.getOrder());
            updateDelivery(delivery, DeliveryStatus.COMPLETED, OrderStatus.PROCESSING);
            template.convertAndSend(Constants.ORDER_FULFILMENT_EXCHANGE, Constants.CONTINUE,
orderValue);
            return new DeliveryDto(delivery, getAdditionalInfo(delivery), false, false);
        }

        private List<Employee> getFreeEmployees(Date deliveryDate, List<Employee> allEmployees, int
duration, int hour) {
            return allEmployees.stream()
                    .filter(employee ->
                            isEmployeeFreeForHour(employee, hour, deliveryDate, duration))
                    .collect(Collectors.toList());
        }

        private boolean isEmployeeFreeForHour(Employee employee, int hour, Date deliveryDate, int
duration) {
            return employee.getDeliveries()
                    .stream()
                    .filter(delivery -> sameDay(deliveryDate, delivery.getDeliveryDate())
                            && isDeliveriesConflicted(hour, delivery.getDeliveryDate(), duration)
                            && isDeliveriesConflicted(hour, delivery))
                    .findAny()
                    .isEmpty();
        }

        private boolean isDeliveriesConflicted(int possibleHour, Date plannedDate, int duration) {
            int endOfPossibleDelivery = possibleHour + duration;
```

```
            return endOfPossibleDelivery >
Integer.parseInt(Constants.ONLY_HOUR_FORMAT.format(plannedDate));
        }

        private boolean isDeliveriesConflicted(int possibleHour, Delivery delivery) {
            int endOfPlannedDelivery =
Integer.parseInt(Constants.ONLY_HOUR_FORMAT.format(delivery.getDeliveryDate())) +
delivery.getDuration();
            return endOfPlannedDelivery > possibleHour;
        }

        public Pair<Integer, Integer> getDurationAndNumberOfEmployees(String[] products) {
            if (hasProduct(Constants.DTV_PRODUCT_STR, products))
                return new ImmutablePair<>(3, 2);
            else if (hasProduct(Constants.INTERNET_PRODUCT_STR, products))
                return new ImmutablePair<>(2, 2);
            else
                return new ImmutablePair<>(1, 1);
        }

        private DeliveryAdditionalInfo getAdditionalInfo(Delivery delivery) {
            Instance instance = delivery.getOrder().getInstance();
            UserDevice userDevice = instanceService.getDeviceIfPresentInternetProduct(instance);
            return DeliveryAdditionalInfo.builder()
                    .simCardNumber(instanceService.getSimCardNumberIfPresentMobileProduct(instance))
                    .cableLength(instanceService.getCableLengthIfPresentInternetProduct(instance))
                    .deviceSerialNumber(userDevice == null ? null : userDevice.getSerialNumber())
                    .macAddress(userDevice == null ? null : userDevice.getMacAddress())
                    .build();
        }

        private boolean hasProduct(String product, String[] products) {
            return Arrays.asList(products).contains(product);
        }

        private boolean isDeliveryStarted(Delivery delivery) {
            Date now = new Date();
            return (delivery.getStatus().equals(DeliveryStatus.NOT_STARTED) ||
delivery.getStatus().equals(DeliveryStatus.IN_PROGRESS))
                    && now.after(delivery.getDeliveryDate()) &&
now.before(DateUtils.addHours(delivery.getDeliveryDate(), delivery.getDuration()));
        }

        private void updateDelivery(Delivery delivery, DeliveryStatus deliveryStatus, OrderStatus
orderStatus) {
            delivery.setStatus(deliveryStatus);
            delivery.getOrder().setStatus(orderStatus);
            deliveryRepository.save(delivery);
        }

        private boolean sameDay(Date date1, Date date2) {
            return ONLY_DATE_FORMAT.format(date1).equals(ONLY_DATE_FORMAT.format(date2));
        }
    }

    package edu.baoss.orderservice.services;

    import edu.baoss.orderservice.model.Flow;
    import edu.baoss.orderservice.model.dtos.OrderDto;
    import edu.baoss.orderservice.model.dtos.OrderValue;
    import edu.baoss.orderservice.model.dtos.OrderWithInstances;
    import edu.baoss.orderservice.model.dtos.TaskDto;
    import edu.baoss.orderservice.model.entities.Instance;
    import edu.baoss.orderservice.model.entities.Order;
    import edu.baoss.orderservice.model.entities.Task;
    import edu.baoss.orderservice.model.enums.OrderStatus;
```

```
    import edu.baoss.orderservice.repositories.OrderRepository;
    import lombok.RequiredArgsConstructor;
    import org.springframework.amqp.rabbit.core.RabbitTemplate;
    import org.springframework.stereotype.Service;
    import org.springframework.transaction.annotation.Transactional;

    import java.io.Serializable;
    import java.text.ParseException;
    import java.util.Comparator;
    import java.util.Date;
    import java.util.List;
    import java.util.stream.Collectors;

    @Service
    @RequiredArgsConstructor
    public class OrderService implements Serializable {
        private final FeasibilityCheckService feasibilityCheckService;
        private final RabbitTemplate template;
        private final OrderRepository orderRepository;
        private final DeliveryService deliveryService;
        private final InstanceService instanceService;
        private final FulfilmentService fulfilmentService;
        private final FlowComposer flowComposer;

        @Transactional
        public Order orderEntryPoint(OrderValue orderValue) throws ParseException {
            feasibilityCheckService.feasibilityCheck(orderValue);
            Order order = createOrderAndInstance(orderValue);
            orderValue.setOrder(order);
            //throw new RuntimeException();
            //template.convertAndSend(Constants.ORDER_FULFILMENT_EXCHANGE, Constants.START,
orderValue);
            Flow executionFlow = flowComposer.createExecutionFlow(orderValue);
            fulfilmentService.startFulfilment(executionFlow);
            return order;
        }

        @Transactional
        public Order createOrderAndInstance(OrderValue orderValue) throws ParseException {
            String productsString = String.join(";", orderValue.getSelectedProducts());
            Order order = orderRepository.save(Order.builder()
                    .startDate(new Date())
                    .status(OrderStatus.PROCESSING)
                    .products(productsString)
                    .orderAim(orderValue.getOrderAim())
                    .totalMRC(orderValue.getTotalMRC())
                    .totalNRC(orderValue.getTotalNRC())
                    .userId(orderValue.getUserId())
                    .workersNum(0)
                    .build());
            boolean neededDelivery = orderValue.getDeliveryTime() != null;
            if (neededDelivery) {
                deliveryService.createDelivery(orderValue, order);
            }
            Instance instance = instanceService.createInstances(orderValue);
            order.setInstance(instance);
            return order;
        }

        public List<OrderDto> getAllOrders(Long userId) {
            return (userId == null
                ? orderRepository.findAll()
                : orderRepository.getOrdersByUserId(userId))
                    .stream()
                    .sorted(Comparator.comparingLong(Order::getId))
                    .map(OrderDto::new)
```

```
                .collect(Collectors.toList());
        }

        public OrderWithInstances getFullOrder(long orderId) {
            OrderWithInstances orderWithInstances = new OrderWithInstances();
            orderRepository.findById(orderId)
                    .ifPresent(order -> {
                        orderWithInstances.setOrder(new OrderDto(order));
                        orderWithInstances.setTasks(order.getTasks().stream()
                                .sorted(Comparator.comparingLong(Task::getId))
                                .map(TaskDto::new)
                                .collect(Collectors.toList()));
                        orderWithInstances.setProductInstances(
                                instanceService.getProductInstancesForInstance(order.getInstance(),
true));
                    });
            return orderWithInstances;
        }

        public Order saveOrder(Order order) {
            return orderRepository.save(order);
        }

    }

    package edu.baoss.orderservice.services;

    import edu.baoss.orderservice.actions.provisioning.ProvisioningAction;
    import edu.baoss.orderservice.model.Flow;
    import edu.baoss.orderservice.model.entities.Task;
    import edu.baoss.orderservice.model.enums.TaskStatus;
    import edu.baoss.orderservice.model.enums.TaskType;
    import org.springframework.stereotype.Service;

    import java.util.List;

    import static edu.baoss.orderservice.model.enums.TaskType.EVENT_LISTENER_TASK;

    @Service
    public class FulfilmentService {
        public void startFulfilment(Flow executionFlow) {
            boolean breakPointPassed = false;
            for (ProvisioningAction action: executionFlow.getActionList()) {
                Task task = action.getTask();
                if (EVENT_LISTENER_TASK.equals(task.getTaskTemplate().getType())) {
                    task.setStatus(TaskStatus.WAITING);
                    breakPointPassed = true;
                }
                if (breakPointPassed) {
                    action.saveTask();
                } else {
                    action.execute();
                }
            }
        }

        public void continueFulfilment(Flow executionFlowAfterDelivery) {
            for (ProvisioningAction action: executionFlowAfterDelivery.getActionList()) {
                action.execute();
            }
        }

    }

    package edu.baoss.orderservice.services;
```

```
    import edu.baoss.orderservice.actions.checking.FeasibilityCheckAction;
    import edu.baoss.orderservice.actions.checking.CommonCheckAction;
    import edu.baoss.orderservice.actions.checking.NrcPaymentAvailabilityCheckAction;
    import edu.baoss.orderservice.actions.checking.PriceCheckAction;
    import edu.baoss.orderservice.actions.checking.dtv.CommonDtvCheckAction;
    import edu.baoss.orderservice.actions.checking.internet.AddressConnectionCheckAction;
    import edu.baoss.orderservice.actions.checking.internet.CommonInternetCheckAction;
    import edu.baoss.orderservice.actions.checking.internet.DeviceAvailabilityCheckAction;
    import edu.baoss.orderservice.actions.checking.internet.HomeConnectionToNetworkCheckAction;
    import edu.baoss.orderservice.actions.checking.mobile.CommonMobileCheckAction;
    import edu.baoss.orderservice.actions.checking.mobile.SimCardAvailabilityCheckAction;
    import edu.baoss.orderservice.model.ActionContext;
    import edu.baoss.orderservice.model.dtos.OrderValue;
    import lombok.RequiredArgsConstructor;
    import org.springframework.context.ApplicationContext;
    import org.springframework.context.annotation.Lazy;
    import org.springframework.stereotype.Service;

    import java.util.ArrayList;
    import java.util.List;

    import static edu.baoss.orderservice.Constants.*;

    @Service
    @RequiredArgsConstructor
    public class FeasibilityCheckService {
        @Lazy
        private final ApplicationContext applicationContext;

        private final List<FeasibilityCheckAction> mobileCheckActionList = List.of(
                new CommonMobileCheckAction(), new SimCardAvailabilityCheckAction());
        private final List<FeasibilityCheckAction> internetCheckActionList = List.of(
                new CommonInternetCheckAction(), new DeviceAvailabilityCheckAction(),
                new HomeConnectionToNetworkCheckAction(), new AddressConnectionCheckAction());
        private final List<FeasibilityCheckAction> commonCheckActionList =  List.of(
                new CommonCheckAction(), new PriceCheckAction(), new
NrcPaymentAvailabilityCheckAction());

        public void feasibilityCheck(OrderValue orderValue) {
            List<FeasibilityCheckAction> checkActions = new ArrayList<>(commonCheckActionList);
            for (String product: orderValue.getSelectedProducts()) {
                if (MOBILE_PRODUCT_STR.equals(product)) {
                    checkActions.addAll(mobileCheckActionList);
                }
                if (INTERNET_PRODUCT_STR.equals(product)) {
                    checkActions.addAll(internetCheckActionList);
                }
                if (DTV_PRODUCT_STR.equals(product)) {
                    checkActions.add(new CommonDtvCheckAction());
                }
            }
            System.out.println(checkActions);
            checkActions.forEach(checkAction -> checkAction.check(new ActionContext(orderValue,
applicationContext)));
        }

    }

    package edu.baoss.orderservice.services;

    import edu.baoss.orderservice.actions.provisioning.ProvisioningAction;
    import edu.baoss.orderservice.actions.provisioning.common.*;
    import edu.baoss.orderservice.actions.provisioning.dtv.ActivateDtvChannelsAction;
    import edu.baoss.orderservice.actions.provisioning.internet.*;
    import edu.baoss.orderservice.actions.provisioning.mobile.Activate5GAction;
    import edu.baoss.orderservice.actions.provisioning.mobile.NotifyUserAction;
```

```
    import edu.baoss.orderservice.actions.provisioning.mobile.ReserveSimCardAction;
    import edu.baoss.orderservice.actions.provisioning.mobile.WaitingDeliverymanConfirmationAction;
    import edu.baoss.orderservice.model.Flow;
    import edu.baoss.orderservice.model.FulfilmentContext;
    import edu.baoss.orderservice.model.dtos.OrderValue;
    import edu.baoss.orderservice.model.entities.Task;
    import edu.baoss.orderservice.model.entities.TaskTemplate;
    import edu.baoss.orderservice.model.enums.TaskType;
    import edu.baoss.orderservice.repositories.TaskTemplateRepository;
    import lombok.RequiredArgsConstructor;
    import org.apache.commons.lang3.BooleanUtils;
    import org.springframework.context.ApplicationContext;
    import org.springframework.context.annotation.Lazy;
    import org.springframework.stereotype.Service;

    import java.lang.reflect.InvocationTargetException;
    import java.util.*;
    import java.util.function.Function;

    import static edu.baoss.orderservice.Constants.*;

    @Service
    @RequiredArgsConstructor
    public class FlowComposer {
        @Lazy
        private final ApplicationContext applicationContext;

        private final TaskTemplateRepository taskTemplateRepository;

        private static final Map<String, Function<FulfilmentContext, ProvisioningAction>>
TASK_NAME_TO_ACTION = new HashMap<>() {{
                put(INIT_ORDER_TASK_NAME, InitializeOrderAction::new);
                put(RESERVE_SIM_CARD_TASK_NAME, ReserveSimCardAction::new);
                put(RESERVE_CABLE_TASK_NAME, ReserveCableAction::new);
                put(RESERVE_DEVICE_TASK_NAME, ReserveDeviceAction::new);
                put(NOTIFY_USER_TASK_NAME, NotifyUserAction::new);
                put(NOTIFY_FITTER_TASK_NAME, NotifyFitterOrDeliverymanAction::new);
                put(ACTIVATE_5G_TASK_NAME, Activate5GAction::new);
                put(WAITING_FITTER_CONFIRMATION_TASK_NAME, WaitingFitterConfirmationAction::new);
                put(WAITING_DELIVERYMAN_CONFIRMATION_TASK_NAME,
WaitingDeliverymanConfirmationAction::new);
                put(WAITING_MAC_ADDRESS_TASK_NAME, WaitingMacAddressAction::new);
                put(CONNECT_TO_NETWORK_TASK_NAME, ConnectToNetworkAction::new);
                put(ACTIVATE_DTV_CHANNELS_TASK_NAME, ActivateDtvChannelsAction::new);
                put(PROVIDE_FIXED_IP_TASK_NAME, ProvideFixedIpAction::new);
                put(NRC_PAYMENT_TASK_NAME, GetPaymentAction::new);
                put(COMPLETE_ORDER_TASK_NAME, CompleteOrderAction::new);
            }};

        public Flow createExecutionFlow(OrderValue orderValue) {
            Map<String, Task> nameToTask = new HashMap<>();
            List<TaskTemplate> all = taskTemplateRepository.findAll();
            List<ProvisioningAction> actions = all.stream()
                    .filter(tt -> Arrays.stream(orderValue.getSelectedProducts())
                            .anyMatch(product -> tt.getProducts().contains(product)))
                    .map(tt -> TASK_NAME_TO_ACTION.get(tt.getName()).apply(new
FulfilmentContext(orderValue, applicationContext, tt, true)))
                    .filter(ProvisioningAction::instantiationCondition)
                    .peek(action -> nameToTask.put(action.getTask().getTaskTemplate().getName(),
action.getTask()))
                    .toList();
            actions.forEach(action -> action.getTask().getTaskTemplate().getDependencies()
                    .forEach(dep -> {
                        if (nameToTask.get(dep.getName()) != null) {
                            action.getTask().addDependency(nameToTask.get(dep.getName()));
                        }
```

```
                    })
            );
            return new Flow(actions);
        }

        public Flow getActionsAfterDelivery(OrderValue orderValue) {
            List<ProvisioningAction> executionFlowAfterDelivery = new ArrayList<>();
            for (Task task: orderValue.getOrder().getTasks()) {
                if (BooleanUtils.toBoolean(task.getTaskTemplate().getParams().get(AFTER_DELIVERY)))
{
                    FulfilmentContext context = new FulfilmentContext(orderValue,
applicationContext, task.getTaskTemplate(), false);
                    ProvisioningAction action =
TASK_NAME_TO_ACTION.get(task.getTaskTemplate().getName()).apply(context);
                    action.setTask(task);
                    executionFlowAfterDelivery.add(action);
                }
            }
            return new Flow(executionFlowAfterDelivery);
        }

        private ProvisioningAction createProvisioningAction(TaskTemplate taskTemplate, OrderValue
orderValue) {
            try {
                FulfilmentContext fulfilmentContext = new FulfilmentContext(orderValue,
applicationContext, taskTemplate, true);
                return (ProvisioningAction)
Class.forName(taskTemplate.getActionName()).getConstructor(FulfilmentContext.class).newInstance(fulf
ilmentContext);
            } catch (NoSuchMethodException | ClassNotFoundException | InvocationTargetException
                     | InstantiationException | IllegalAccessException e) {
                throw new RuntimeException("Impossible to create " + taskTemplate.getActionName() +
" object due to: " + e);
            }
        }

    }


    package edu.baoss.orderservice.actions.provisioning;

    import edu.baoss.orderservice.model.FulfilmentContext;
    import edu.baoss.orderservice.model.dtos.OrderValue;
    import edu.baoss.orderservice.model.entities.Task;
    import edu.baoss.orderservice.model.enums.TaskStatus;
    import edu.baoss.orderservice.repositories.TaskRepository;
    import org.springframework.context.ApplicationContext;

    import java.util.Date;

    public abstract class ProvisioningAction {
        protected Task task;
        protected OrderValue orderValue;
        protected ApplicationContext applicationContext;

        public ProvisioningAction(FulfilmentContext fulfilmentContext) {
            this.orderValue = fulfilmentContext.getOrderValue();
            this.applicationContext = fulfilmentContext.getApplicationContext();
            if (fulfilmentContext.isInit()) {
                this.task = Task.builder()
                        .taskTemplate(fulfilmentContext.getTaskTemplate())
                        .status(TaskStatus.NOT_STARTED)
                        .order(orderValue.getOrder())
                        .build();
            }
        }
```

```
        public void execute() {
            task.setStatus(TaskStatus.IN_PROGRESS);
            task.setStartDate(new Date());
            saveTask();
            performAction();
            task.setCompletionDate(new Date());
            task.setStatus(TaskStatus.COMPLETED);
            saveTask();
        }

        public abstract boolean instantiationCondition();

        protected abstract void performAction();

        public Task getTask() { return task; }

        public Task saveTask() {
            return applicationContext.getBean(TaskRepository.class).save(task);
        }

        public void setTask(Task task) {
            this.task = task;
        }

        public void setOrderValue(OrderValue orderValue) {
            this.orderValue = orderValue;
        }
    }
```

offer-service

```
    package edu.baoss.offerservice.services;

    import edu.baoss.offerservice.dto.OfferDto;
    import edu.baoss.offerservice.model.Discount;
    import edu.baoss.offerservice.model.Offer;
    import org.springframework.beans.factory.annotation.Autowired;
    import org.springframework.stereotype.Service;

    import java.util.*;
    import java.util.stream.Collectors;

    @Service
    public abstract class OfferService {
        @Autowired
        DiscountService discountService;

        public List<OfferDto> getOffers(List<? extends Offer> offers) {
            Discount totalDiscount = discountService.getTotalDiscount(getProductName());
            return offers.stream()
                    .sorted(Comparator.comparingDouble(Offer::getRent).reversed())
                    .map(offer -> getOfferDto(offer, totalDiscount))
                    .collect(Collectors.toList());
        }

        public OfferDto getOfferDto(Offer offer, Discount totalDiscount) {
            double startPrice = offer.getRent();
            double discountedPrice = discountService.calcDiscountedPrice(startPrice, totalDiscount);
            String priceEnding = Converter.getConvertedString(offer.getCurrency(),
offer.getRentTime());
            OfferDto offerDto = createConcreteDto(offer);
            offerDto.setId(offer.getId());
            offerDto.setStartingPrice(offer.getRent());
```

```
            offerDto.setDiscountedPrice(discountedPrice);
            offerDto.setPriceEnding(priceEnding);
            offerDto.setDiscount(totalDiscount.getValue());

offerDto.setDiscountEndDate(Constants.onlyDateFormat.format(totalDiscount.getEndDate()));
            offerDto.setCurrency(offer.getCurrency());
            offerDto.setRentTime(offer.getRentTime());
            return offerDto;
        }

        public abstract String getProductName();

        public abstract OfferDto createConcreteDto(Offer offer);
    }
```

```
    package edu.baoss.offerservice.services;

    import edu.baoss.offerservice.dto.InternetOfferDto;
    import edu.baoss.offerservice.dto.OfferDto;
    import edu.baoss.offerservice.exceptions.NoOfferFoundException;
    import edu.baoss.offerservice.model.Discount;
    import edu.baoss.offerservice.model.Offer;
    import edu.baoss.offerservice.model.InternetOffer;
    import edu.baoss.offerservice.repositories.InternetOfferRepository;
    import edu.baoss.offerservice.util.StreamUtil;
    import org.springframework.beans.factory.annotation.Autowired;
    import org.springframework.stereotype.Service;

    import java.util.List;
    import java.util.stream.Collectors;

    @Service("internetService")
    public class InternetOfferService extends OfferService {
        @Autowired
        InternetOfferRepository internetOfferRepository;

        public List<InternetOfferDto> getInternetOffers() {
            List<? extends Offer> offers = internetOfferRepository.findAll();
            return StreamUtil.toStream(getOffers(offers))
                    .map(InternetOfferDto.class::cast)
                    .collect(Collectors.toList());
        }

        public InternetOfferDto getInternetOffer(String internetOfferId) {
            InternetOffer internetOffer =
internetOfferRepository.getById(internetOfferId).orElseThrow(NoOfferFoundException::new);
            Discount totalDiscount = discountService.getTotalDiscount(getProductName());
            return (InternetOfferDto) getOfferDto(internetOffer, totalDiscount);
        }

        @Override
        public String getProductName() {
            return Constants.INTERNET_PRODUCT_STR;
        }

        @Override
        public OfferDto createConcreteDto(Offer offer) {
            return InternetOfferDto.builder()
                    .speed(((InternetOffer) offer).getSpeed())
                    .build();
        }
    }
```

resource-service

```
    package edu.baoss.resourceservice.services;

    import edu.baoss.resourceservice.exceptions.NoConnectedDeviceFoundException;
    import edu.baoss.resourceservice.model.ConnectedAddress;
    import edu.baoss.resourceservice.model.ConnectedBuilding;
    import edu.baoss.resourceservice.repositories.ConnectedAddressesRepository;
    import edu.baoss.resourceservice.repositories.ConnectedBuildingsRepository;
    import lombok.RequiredArgsConstructor;
    import org.apache.commons.collections4.CollectionUtils;
    import org.apache.commons.lang3.RandomStringUtils;
    import org.springframework.beans.factory.annotation.Autowired;
    import org.springframework.stereotype.Service;

    import java.util.List;
    import java.util.Locale;
    import java.util.Optional;
    import java.util.Random;
    import java.util.stream.Collectors;
    import java.util.stream.IntStream;

    @Service
    @RequiredArgsConstructor
    public class ConnectionService {
        private final ConnectedBuildingsRepository connectedBuildingsRepository;
        private final ConnectedAddressesRepository connectedAddressesRepository;

        public boolean isBuildingConnectedToNetwork(long buildingId) {
           return
connectedBuildingsRepository.findConnectedBuildingByBuildingId(buildingId).isPresent();
        }

        public boolean isAddressConnected(long addressId) {
            return connectedAddressesRepository.findAll().stream()
                    .map(ConnectedAddress::getAddressId)
                    .anyMatch(address -> address == addressId);
        }

        public String getMacAddressByAddressId(long addressId) {
            return connectedAddressesRepository.findAll().stream()
                    .filter(connectedAddress -> connectedAddress.getAddressId() == addressId)
                    .map(ConnectedAddress::getMacAddress)
                    .findFirst()
                    .orElseThrow(NoConnectedDeviceFoundException::new);
        }

        public void connectUser(long buildingId, long addressId, String macAddress) {
            Optional<ConnectedBuilding> connectedBuilding =
connectedBuildingsRepository.findConnectedBuildingByBuildingId(buildingId);
            connectedBuilding.orElseThrow(() -> new RuntimeException("The building is not connected
to the network"));
            connectedBuilding.ifPresent(cb -> {
                boolean alreadyConnected = cb.getConnectedAddresses().stream()
                        .anyMatch(ca -> ca.getAddressId() == addressId &&
ca.getMacAddress().equals(macAddress));
                if (!alreadyConnected) {
                    ConnectedAddress connectedAddress = connectedAddressesRepository
                            .save(new ConnectedAddress(addressId, macAddress));
                    cb.getConnectedAddresses().add(connectedAddress);
                    connectedBuildingsRepository.save(cb);
                }
            });
        }

        public void connectBuildings(List<Long> buildingIds) {
            Random rand = new Random();
            List<Long> connectedBuildingIds = connectedBuildingsRepository.findAll().stream()
```

```
                    .peek(System.out::println)
                    .map(ConnectedBuilding::getBuildingId)
                    .toList();
            buildingIds.stream()
                    .filter(buildingId -> !connectedBuildingIds.contains(buildingId))
                    .map(buildingId -> ConnectedBuilding.builder()
                            .buildingId(buildingId)
                            .switchMacAddress(randomMACAddress())
                            .switchSerialNumber(String.format("2%04d", rand.nextInt(10000)))
                            .build())
                    .forEach(connectedBuildingsRepository::save);
        }

    }

    package edu.baoss.resourceservice.services;

    import edu.baoss.resourceservice.dtos.UserDevice;
    import edu.baoss.resourceservice.exceptions.NoDeviceFoundException;
    import edu.baoss.resourceservice.feignproxies.OfferFeignProxy;
    import edu.baoss.resourceservice.model.Device;
    import edu.baoss.resourceservice.repositories.DeviceRepository;
    import org.springframework.beans.factory.annotation.Autowired;
    import org.springframework.data.domain.Sort;
    import org.springframework.stereotype.Service;

    import java.util.Arrays;
    import java.util.List;
    import java.util.stream.Collectors;

    @Service
    public class DeviceService {
        @Autowired
        DeviceRepository deviceRepository;
        @Autowired
        OfferFeignProxy offerFeignProxy;
        @Autowired
        ConnectionService connectionService;

        public List<Device> getDevicesForSale() {
            return deviceRepository.findAll(Sort.by(Sort.Direction.DESC, "price"))
                    .stream()
                    .filter(device -> device.isForSale() && device.getType().equals("Router") &&
hasInstance(device))
                    .peek(device -> device.setPrice(offerFeignProxy.getDiscountedPrice("Devices",
device.getPrice())))
                    .collect(Collectors.toList());
        }

        public UserDevice getUserDeviceByDeviceId(String deviceId, long addressId) {
            System.out.println(deviceId);
            Device device =
deviceRepository.findById(deviceId).orElseThrow(NoDeviceFoundException::new);
            System.out.println(device.getName());
            String macAddress = connectionService.getMacAddressByAddressId(addressId);
            int index = -1;
            String[] macAddresses = device.getMacAddresses();
            for (int i = 0; i < macAddresses.length; i++) {
                if (macAddresses[i].equals(macAddress))
                    index = i;
            }
            UserDevice userDevice = new UserDevice(device);
            userDevice.setMacAddress(macAddress);
            userDevice.setSerialNumber(device.getSerialNumbers()[index]);
            return userDevice;
        }
```

```
        public UserDevice getReservedDevice(String deviceId) {
            Device device =
deviceRepository.findById(deviceId).orElseThrow(NoDeviceFoundException::new);
            System.out.println(device.getName());
            int index = getFirstFreeInstance(device) - 1;
            UserDevice userDevice = new UserDevice(device);
            userDevice.setMacAddress(device.getMacAddresses()[index]);
            userDevice.setSerialNumber(device.getSerialNumbers()[index]);
            return userDevice;
        }

        public UserDevice reserveDevice(String deviceId) {
            Device device =
deviceRepository.findById(deviceId).orElseThrow(NoDeviceFoundException::new);
            int index = getFirstFreeInstance(device);
            System.out.println(index);
            System.out.println(Arrays.toString(device.getUsed()));
            device.getUsed()[index] = true;
            System.out.println(Arrays.toString(device.getUsed()));
            deviceRepository.save(device);
            UserDevice userDevice = new UserDevice(device);
            userDevice.setSerialNumber(device.getSerialNumbers()[index]);
            userDevice.setMacAddress(device.getMacAddresses()[index]);
            return userDevice;
        }

        boolean hasInstance(Device device) {
            return getFirstFreeInstance(device) != -1;
        }

        int getFirstFreeInstance(Device device) {
            for (int i = 0; i < device.getUsed().length; i++) {
                if (!device.getUsed()[i])
                    return i;
            }
            return -1;
        }

    }

    package edu.baoss.resourceservice.controllers;

    import edu.baoss.resourceservice.services.ConnectionService;
    import org.springframework.beans.factory.annotation.Autowired;
    import org.springframework.web.bind.annotation.*;

    import java.util.List;

    @RestController
    @CrossOrigin(origins = "http://localhost:4200")
    @RequestMapping("/connection")
    public class ConnectionController {
        @Autowired
        ConnectionService connectionService;

        @GetMapping("/connected-buildings")
        public boolean isBuildingConnectedToNetwork(@RequestParam long buildingId) {
            System.out.println(connectionService.isBuildingConnectedToNetwork(buildingId));
            return connectionService.isBuildingConnectedToNetwork(buildingId);
        }

        @GetMapping("/connected-address")
        public boolean isAddressConnected(@RequestParam long addressId) {
            return connectionService.isAddressConnected(addressId);
        }
```

```
        @PutMapping("/connect-user")
        void connectUser(@RequestParam("buildingId") long buildingId,
                         @RequestParam("addressId") long addressId,
                         @RequestParam("macAddress") String macAddress) {
            connectionService.connectUser(buildingId, addressId, macAddress);
        }

        @PostMapping("/connect-buildings")
        void connectBuildings(@RequestBody List<Long> buildingIds) {
            connectionService.connectBuildings(buildingIds);
        }
    }
```

analytics-service

```
    from datetime import datetime
    from datetime import timedelta
    from dateutil.relativedelta import relativedelta
    import numpy as np
    import pandas as pd
    from flask import current_app as app, Response
    from flask import request, jsonify, json
    from sqlalchemy import func
    from sklearn.linear_model import Lasso
    from sklearn.linear_model import LinearRegression
    from sklearn.tree import DecisionTreeRegressor
    from sklearn.ensemble import GradientBoostingRegressor
    from sktime.forecasting.base import ForecastingHorizon
    from sktime.forecasting.fbprophet import Prophet
    from sktime.forecasting.naive import NaiveForecaster
    from sktime.forecasting.compose import make_reduction
    from sktime.forecasting.model_selection import temporal_train_test_split
    from sktime.performance_metrics.forecasting import mean_absolute_percentage_error
    from models import *

    date_pattern = "%d-%m-%Y"
    internet_product = "Internet Product"
    mobile_product = "Mobile Product"
    dtv_product = "DTV Product"
    day = "day"
    week = "week"
    month = "month"
    quarter = "quarter"
    year = "year"
    x = 'x'
    y = 'y'
    step_to_pattern = {day: "%d-%m-%Y", week: "%d-%m-%Y", month: "%m-%Y", quarter: "%m-%Y", year:
"%Y"}
    step_to_period = {day: 1, week: 7, month: 30, quarter: 90, year: 365}
    end_date_condition_to_result = {
        "010": lambda end_date_param: datetime.now() + relativedelta(months=+2),
        "000": lambda end_date_param: datetime.now(),
        "100": lambda end_date_param: end_date_param,
        "101": lambda end_date_param: datetime.now(),
        "110": lambda end_date_param: datetime.now() + relativedelta(months=+2),
        "111": lambda end_date_param: end_date_param
    }


    @app.route('/common-statistic', methods=['GET'])
    def cohort_analysis():
        return jsonify({
            "userNumber": get_user_number(),
            "userNumberByProducts": get_user_number_by_product(),
```

```
            "usersByDate": get_users_by_date(),
            "productsByDate": get_products_by_date(),
        })


    @app.route('/business-metrics', methods=['GET'])
    def business_metrics():
        return jsonify({
            "profitByDate": get_profit_by_date(),
            "profit": get_profit(),
            "aovsByDate": get_aovs_by_date(),
            "aovs": get_aovs(),
            "arppuByDate": get_arppu_by_date(),
            "arppu": get_arppu(),
            "profitByProduct": get_profits_by_product(),
            "profitByProductAndDate": get_profits_by_product_and_date(),
            "clv": get_clv(),
            "clvByDate": get_clv_by_date()
        })


    @app.route('/profit-forecast', methods=['GET'])
    def profit_forecast():
        try:
            forecast = get_profit_forecast()
        except NotEnoughDataForPrediction:
            response = Response(status=404)
            response.data = json.dumps({"timestamp": datetime.now(),
                            "status": 404,
                            "error": "Not Found",
                            "message": "Not enough data for prediction"})
            return response
        return jsonify({
            "profitForecast": forecast[0],
            "forecastWithTest": forecast[1],
            "mapeEvaluation": forecast[2],
            "regressors": ["Prophet", "Lasso", "Decision Tree", "Gradient Boosting"]
        })


    def get_user_number():
        start_date, end_date = get_date_range(lambda:
db.session.query(func.min(User.reg_date)).first()[0])
        return len(User.query.filter(User.reg_date >= start_date).filter(User.reg_date <=
end_date).filter(
            User.usr_role == 'USER').all())


    def get_user_number_by_product():
        start_date, end_date = get_date_range(lambda:
db.session.query(func.min(Order.completion_date)).first()[0])
        return get_number_of_users_by_products(start_date, end_date)


    def get_users_by_date():
        start_date, end_date = get_date_range(lambda:
db.session.query(func.min(User.reg_date)).first()[0])
        step = request.args.get('step') if request.args.get('step') is not None else day
        users = User.query.filter(User.reg_date >= start_date) \
            .filter(User.reg_date <= end_date) \
            .filter(User.usr_role == 'USER').all()
        if len(users) == 0:
            return {x: [], y: []}
        return get_nums_grouped_by_date(users, step, 'reg_date', 'user_id', lambda df: df.count())
```

```
    def get_products_by_date():
        start_date, end_date = get_date_range(lambda:
db.session.query(func.min(Order.completion_date)).first()[0])
        step = request.args.get('step') if request.args.get('step') is not None else day
        date_and_product_nums = [get_product_numbers_by_date(start_date, end_date, step,
mobile_product),
                                 get_product_numbers_by_date(start_date, end_date, step,
internet_product),
                                 get_product_numbers_by_date(start_date, end_date, step,
dtv_product)]
        return date_and_product_nums


    def get_profit_by_date():
        start_date, end_date = get_date_range(lambda:
db.session.query(func.min(Payment.payment_date)).first()[0])
        step = request.args.get('step') if request.args.get('step') is not None else day
        payments = Payment.query.filter(Payment.payment_date >= start_date) \
            .filter(Payment.payment_date <= end_date).all()
        if len(payments) == 0:
            return {x: [], y: []}
        return get_nums_grouped_by_date(payments, step, 'payment_date', 'value', lambda df:
df.sum())


    def get_profit():
        start_date, end_date = get_date_range(lambda:
db.session.query(func.min(Payment.payment_date)).first()[0])
        payment_values = [res[0] for res in Payment.query.filter(Payment.payment_date >= start_date)
        .filter(Payment.payment_date <= end_date).with_entities(Payment.value).all()]
        return round(sum(payment_values), 2)


    def get_aovs_by_date():
        start_date, end_date = get_date_range(lambda:
db.session.query(func.min(Order.completion_date)).first()[0])
        step = request.args.get('step') if request.args.get('step') is not None else day
        orders = Order.query \
            .filter(Order.completion_date is not None) \
            .filter(Order.completion_date >= start_date) \
            .filter(Order.completion_date <= end_date).all()
        if len(orders) == 0:
            return [{x: [], y: []}, {x: [], y: []}]
        aovs_by_date = [get_nums_grouped_by_date(orders, step, 'completion_date', 'total_nrc',
lambda df: df.mean()),
                        get_nums_grouped_by_date(orders, step, 'completion_date', 'total_mrc',
lambda df: df.mean())]
        return aovs_by_date
    def get_aovs():
        start_date, end_date = get_date_range(lambda:
db.session.query(func.min(Order.completion_date)).first()[0])
        orders = Order.query \
            .filter(Order.completion_date is not None) \
            .filter(Order.completion_date >= start_date) \
            .filter(Order.completion_date <= end_date).all()
        if len(orders) == 0:
            return 0, 0
        mean_nrc = np.array([order.total_nrc for order in orders]).mean()
        mean_mrc = np.array([order.total_mrc for order in orders]).mean()
        return round(mean_nrc, 2), round(mean_mrc, 2)


    def get_arppu_by_date():
        start_date, end_date = get_date_range(lambda:
db.session.query(func.min(Payment.payment_date)).first()[0])
        step = request.args.get('step') if request.args.get('step') is not None else day
```

```
        payments = Payment.query.filter(Payment.payment_date >= start_date) \
            .filter(Payment.payment_date <= end_date).all()
        if len(payments) == 0:
            return {x: [], y: []}
        return get_arppu_grouped_by_date(payments, step)


    def get_arppu():
        start_date, end_date = get_date_range(lambda:
db.session.query(func.min(User.reg_date)).first()[0])
        payments = Payment.query.filter(Payment.payment_date >=
start_date).filter(Payment.payment_date <= end_date).all()
        payment_values = [payment.value for payment in payments]
        user_number = len(set([payment.from_user for payment in payments]))
        return round(sum(payment_values) / user_number, 2)


    def get_profits_by_product():
        start_date, end_date = get_date_range(lambda:
db.session.query(func.min(Payment.payment_date)).first()[0])
        payments = Payment.query.filter(Payment.payment_date >=
start_date).filter(Payment.payment_date <= end_date).all()
        if len(payments) == 0:
            return {x: [], y: []}
        product_to_profit = dict()
        product_to_profit[mobile_product] = round(
            sum([payment.value for payment in payments if mobile_product in payment.purpose]), 2)
        product_to_profit[internet_product] = round(
            sum([payment.value for payment in payments if internet_product in payment.purpose]), 2)
        product_to_profit[dtv_product] = round(
            sum([payment.value for payment in payments if dtv_product in payment.purpose]), 2)
        return {x: list(product_to_profit.keys()), y: list(product_to_profit.values())}


    def get_profits_by_product_and_date():
        start_date, end_date = get_date_range(lambda:
db.session.query(func.min(Payment.payment_date)).first()[0])
        step = request.args.get('step') if request.args.get('step') is not None else day
        all_payments = Payment.query.filter(Payment.payment_date >=
start_date).filter(Payment.payment_date <= end_date)
        mobile_payments = all_payments.filter(Payment.purpose.contains(mobile_product)).all()
        internet_payments = all_payments.filter(Payment.purpose.contains(internet_product)).all()
        dtv_payments = all_payments.filter(Payment.purpose.contains(dtv_product)).all()
        return [get_nums_grouped_by_date(mobile_payments, step, 'payment_date', 'value', lambda df:
df.sum()) if len(
            mobile_payments) > 0 else {x: [], y: []},
                get_nums_grouped_by_date(internet_payments, step, 'payment_date', 'value', lambda
df: df.sum()) if len(
                    internet_payments) > 0 else {x: [], y: []},
                get_nums_grouped_by_date(dtv_payments, step, 'payment_date', 'value', lambda df:
df.sum()) if len(
                    dtv_payments) > 0 else {x: [], y: []}]


    def get_clv():
        arppu = get_arppu()
        churn = (arppu*10) % 3 + 2
        return round(arppu * 0.8 * churn, 2)


    def get_clv_by_date():
        arppu_by_date = get_arppu_by_date()
        arppu_by_date[y] = [round((arppu * 0.8 * ((arppu*10) % 3 + 2)), 2) for arppu in
arppu_by_date[y]]
        return arppu_by_date
```

```
    def get_profit_forecast():
        start_date, forecast_end_date = get_date_range(lambda:
db.session.query(func.min(Payment.payment_date)).first()[0], True)
        step = request.args.get('step') if request.args.get('step') is not None else day
        start_date = get_start_date_by_step(step)
        payment_end_date = get_payments_end_date(step)
        payments = Payment.query.filter(Payment.payment_date >= start_date) \
            .filter(Payment.payment_date < payment_end_date).order_by(Payment.payment_date).all()
        if len(payments) == 0:
            return {x: [], y: []}
        df = pd.DataFrame.from_records([v.__dict__ for v in payments])[['payment_date', 'value']]
        date_to_profit = df.groupby(pd.Grouper(key='payment_date', axis=0,
freq=step[0].upper())).sum()
        forecast_with_test, mape_results = get_sktime_forecast(date_to_profit, payment_end_date,
forecast_end_date, step, True)
        return get_sktime_forecast(date_to_profit, payment_end_date, forecast_end_date, step,
False)[0], forecast_with_test, mape_results


    def get_date_range(func_for_start_date, for_forecast=False):
        start_date_str = request.args.get('start_date')
        end_date_str = request.args.get('end_date')
        start_date = datetime.strptime(start_date_str,
                                       date_pattern) if start_date_str is not None and
start_date_str != '' else None
        end_date = datetime.strptime(end_date_str,
                                     date_pattern) if end_date_str is not None and end_date_str !=
'' else None
        start_date = start_date if start_date is not None else func_for_start_date()
        condition = str(int(end_date is not None)) + str(int(for_forecast)) + str(int(end_date >
datetime.now() if end_date is not None else False))
        return start_date if start_date is not None else func_for_start_date(),
end_date_condition_to_result[condition](end_date)


    def get_number_of_users_by_products(start_date, end_date):
        product_to_number = dict()
        product_to_number[mobile_product] =
len(set(Order.query.filter(Order.products.contains(mobile_product))
                                                    .filter(Order.completion_date is not None)
                                                    .filter(Order.completion_date >= start_date)
                                                    .filter(Order.completion_date <= end_date)
                                                    .with_entities(Order.user_id).all()))
        product_to_number[internet_product] =
len(set(Order.query.filter(Order.products.contains(internet_product))
                                                      .filter(Order.completion_date is not None)
                                                      .filter(Order.completion_date >= start_date)
                                                      .filter(Order.completion_date <= end_date)
                                                      .with_entities(Order.user_id).all()))
        product_to_number[dtv_product] =
len(set(Order.query.filter(Order.products.contains(dtv_product))
                                                .filter(Order.completion_date is not None)
                                                .filter(Order.completion_date >= start_date)
                                                .filter(Order.completion_date <= end_date)
                                                .with_entities(Order.user_id).all()))
        return {x: list(product_to_number.keys()), y: list(product_to_number.values())}


    def get_product_numbers_by_date(start_date, end_date, step, product):
        products = Order.query.filter(Order.products.contains(product)) \
            .filter(Order.completion_date is not None) \
            .filter(Order.completion_date >= start_date) \
            .filter(Order.completion_date <= end_date).all()
        if len(products) == 0:
            return {x: [], y: []}
```

```
        return get_nums_grouped_by_date(products, step, 'completion_date', 'order_id', lambda df:
df.count())


    def get_nums_grouped_by_date(values, step, date_field, value_field, apply_to_group):
        df = pd.DataFrame.from_records([v.__dict__ for v in values])[[date_field, value_field]]
        date_to_num = apply_to_group(df.groupby(pd.Grouper(key=date_field, axis=0,
freq=step[0].upper())))
        dates = [order_date.strftime(step_to_pattern[step])
                 for order_date in date_to_num.index.get_level_values(date_field).tolist()]
        values = np.array(date_to_num[value_field].values.tolist())
        values[np.isnan(values)] = 0
        return {x: dates, y: [round(float(v), 2) for v in values]}


    def get_arppu_grouped_by_date(values, step):
        df = pd.DataFrame.from_records([v.__dict__ for v in values])[['payment_date', 'value',
'from_user']]
        date_to_sum = df[['payment_date', 'value']].groupby(
            pd.Grouper(key='payment_date', axis=0, freq=step[0].upper())).sum()
        date_to_users = df[['payment_date', 'from_user']].groupby(
            pd.Grouper(key='payment_date', axis=0, freq=step[0].upper())).agg(set)
        dates = [order_date.strftime(step_to_pattern[step]) for order_date in
                 date_to_sum.index.get_level_values('payment_date').tolist()]
        sums = list(date_to_sum['value'].values)
        list_of_grouped_users = list(date_to_users['from_user'].values)
        values = [calculate_arppu_by_date(list_of_grouped_users, sums, i) for i in range(len(sums))]
        return {x: dates, y: values}


    def calculate_arppu_by_date(list_of_grouped_users, sums, i):
        unique_users = len(list_of_grouped_users[i])
        if unique_users == 0:
            return 0
        sum = 0 if np.isnan(sums[i]) else sums[i]
        return round(sum / unique_users, 2)


    def get_sktime_forecast(date_to_profit, payment_end_date, forecast_end_date, step, with_test):
        if len(date_to_profit) < 15:
            raise NotEnoughDataForPrediction()
        y_train, y_test = temporal_train_test_split(date_to_profit,
test_size=int(0.2*len(date_to_profit)) if with_test else 1)
        fh = ForecastingHorizon(pd.date_range(start=payment_end_date, end=forecast_end_date,
freq=step[0].upper()), is_relative=False) \
            if not with_test else ForecastingHorizon(y_test.index, is_relative=False)
        result_data = []
        dates = [grouped_date.strftime(step_to_pattern[step])
                 for grouped_date in date_to_profit.index.get_level_values('payment_date').tolist()]
        grouped_values = np.array(date_to_profit['value'].values.tolist())
        grouped_values[np.isnan(grouped_values)] = 0
        result_data.append({x: dates, y: [round(float(v), 2) for v in grouped_values]})

        mape_results = []
        # NaiveForecaster(sp=12, strategy='last'),
        regressors = [Prophet(), make_reduction(Lasso()), make_reduction(DecisionTreeRegressor()),
make_reduction(GradientBoostingRegressor())]
        for regressor in regressors:
            if not with_test:
                result_data.append(fit_predict(regressor, y_train, fh, step)[0])
            else:
                prediction, mape_res = fit_predict(regressor, y_train, fh, step, y_test)
                result_data.append(prediction)
                mape_results.append(round(mape_res, 2))
        return result_data, mape_results
```

```
    def fit_predict(regressor, train_data, fh, step, test_data=None):
        regressor.fit(train_data)
        profit_forecast_result = regressor.predict(fh)
        profit_forecast_result = profit_forecast_result.fillna(0)
        dates = [grouped_date.strftime(step_to_pattern[step])
                 for grouped_date in profit_forecast_result.index.tolist()]
        values = np.array(profit_forecast_result['value'].values.tolist())
        values[np.isnan(values)] = 0
        results = {x: dates, y: [round(float(v), 2) for v in values]}
        return results, mean_absolute_percentage_error(test_data, profit_forecast_result,
symmetric=False) if test_data is not None else []


    def get_start_date_by_step(step):
        if step == day:
            return datetime.now() + relativedelta(months=-2)
        elif step == week:
            return datetime.now() + relativedelta(months=-12)
        elif step == month:
            return datetime.now() + relativedelta(years=-3)
        else:
            return db.session.query(func.min(Payment.payment_date)).first()[0]


    def get_payments_end_date(step):
        now = datetime.now()
        if step == day:
            return now.replace(hour=23, minute=59, second=59) - timedelta(days=1)
        elif step == week:
            return (now - timedelta(days=now.weekday())) - timedelta(days=1)
        elif step == month:
            return now.replace(day=1, hour=23, minute=59, second=59) - timedelta(days=1)
        elif step == quarter:
            return datetime(now.year, 3 * ((now.month - 1) // 3) + 1, 1) - timedelta(days=1)
        else:
            return now.replace(month=1, day=1, hour=23, minute=59, second=59) - timedelta(days=1)
```

front-end-service

```
    import {Component, OnInit, Pipe, PipeTransform} from '@angular/core';
    import {
        Address,
        Cable,
        ConstantPrices,
        Device,
        Discount,
        DtvOffer,
        InternetOffer, OrderValue,
        PhoneNumber,
        Tariff
    } from '../_models/interface';
    import {DEFAULT_ERROR_MESSAGE, PRODUCTS} from '../_models/constants';
    import { DatePipe } from '@angular/common';
    import {OffersService} from '../_services/offers.service';
    import {ToastrService} from 'ngx-toastr';
    import {FormControl, FormGroup, ValidationErrors, Validators} from '@angular/forms';
    import {ResourceService} from '../_services/resource.service';
    import {error} from 'util';
    import {MatDialog, MatDialogConfig} from '@angular/material';
    import {DeviceInfoComponent} from '../device-info/device-info.component';
    import {AuthService} from '../_services/auth.service';
    import {UserService} from '../_services/user.service';
    import {Location} from '@angular/common';
    import {Router} from '@angular/router';
```

```
import {OrderService} from '../_services/order.service';

@Component({
    selector: 'app-order-entry',
    templateUrl: './order-entry.component.html',
    styleUrls: ['./order-entry.component.css']
})
export class OrderEntryComponent implements OnInit {
    products = [
        {id: 0, name: PRODUCTS.MOBILE_PRODUCT, active: false, check: false},
        {id: 1, name: PRODUCTS.INTERNET_PRODUCT, active: false, check: false},
        {id: 2, name: PRODUCTS.DTV_PRODUCT, active: false, check: false}
    ];
    internetOffers: InternetOffer[];
    dtvOffers: DtvOffer[];
    tariffs: Tariff[];
    constantPrices: ConstantPrices;
    activatedDiscounts: Discount[];
    phoneNumbers: PhoneNumber[];
    selectedPhoneNumber: PhoneNumber;
    selectedTariff: Tariff;
    support5g =  false;
    deliveryAndActivationMobile = true;
    selectedSpeed: InternetOffer;
    devices: Device[];
    selectedDevice: Device;
    selectedChannelNumber: DtvOffer;
    fixedIpSupport = false;
    ipv6Support = false;
    installation = false;
    cable: Cable;
    cableLengthForm = new FormGroup({
        cableLength: new FormControl('', [
            Validators.required,
            Validators.min(0),
            Validators.max(30)
        ])});
    validationMessages = [
        {type: 'required', message: 'Cable length is required'},
        {type: 'min', message: 'Cable length must be greater than 0'},
        {type: 'max', message: 'We don`t have so much cable :)'},
    ];
    cablePriceTotal = 0;
    userAddresses: Address[];
    selectedAddress: Address;
    homeDelivery = true;
    storeAddress = '';
    deliveryDate: Date;
    deliveryTime: string;
    availableTimes: string[];
    dateErrorMessage: string;
    totalNRC: number;
    totalMRC: number;
    deliveryPrice = 0;

    constructor(private offerService: OffersService,
                private toaster: ToastrService,
                private resourceService: ResourceService,
                public dialog: MatDialog,
                private authService: AuthService,
                private userService: UserService,
                private location: Location,
                private router: Router,
                private orderService: OrderService,
                private dateFormatPipe: DatePipe) {
    }
```

```
        ngOnInit() {
            this.offerService.getActiveDiscounts()
                .subscribe(data => this.activatedDiscounts = data);
            this.offerService.getInternetOffers()
                .subscribe(data => this.internetOffers = data,
                    error => this.toaster.error(error.error.message));
            this.offerService.getDtvOffers()
                .subscribe(data => this.dtvOffers = data,
                    error => this.toaster.error(error.error.message));
            this.offerService.getTariffs()
                .subscribe(data => this.tariffs = data,
                    error => this.toaster.error(error.error.message));
            this.offerService.getConstantPrices()
                .subscribe(data => this.constantPrices = data,
                    error => this.toaster.error(error.error.message));
            this.userService.getAddressesByLogin(this.authService.currentUserValue.login)
                .subscribe(data => {
                        this.userAddresses = data;
                        this.selectedAddress = data[0];
                    },
                    error => {
                        if (error.message != null && error.message != '') {
                            this.toaster.error(error.message);
                        } else if (error.error.message != null && error.error.message != '') {
                            this.toaster.error(error.error.message);
                        } else {
                            this.toaster.error(DEFAULT_ERROR_MESSAGE);
                        }
                    });
        }

        submit() {
            const selectedProducts = this.getSelectedProductList();
            const selectedDevice = this.selectedDevice != null && this.selectedDevice.name == 'None'
? null : this.selectedDevice;
            const error = this.validateAll(selectedProducts);
            if (error != '') {
                this.toaster.error(error);
            } else {
                const orderValue = {userId: this.authService.currentUserValue.id,
                                    selectedProducts: selectedProducts,
                                    selectedPhoneNumber: this.selectedPhoneNumber,
                                    selectedTariff: this.selectedTariff,
                                    support5g: this.support5g,
                                    deliveryAndActivationMobile: this.deliveryAndActivationMobile,
                                    selectedSpeed: this.selectedSpeed,
                                    selectedDevice: selectedDevice,
                                    selectedChannelNumber: this.selectedChannelNumber,
                                    fixedIpSupport: this.fixedIpSupport,
                                    installation: this.selectedSpeed != null ||
this.selectedChannelNumber != null,
                                    cableLength: +this.cableLengthForm.controls.cableLength.value,
                                    cablePriceTotal: this.cablePriceTotal,
                                    selectedAddress: this.selectedAddress,
                                    deliveryDateStr:
this.dateFormatPipe.transform(this.deliveryDate, 'dd/MM/yyyy'),
                                    deliveryTime: this.deliveryTime,
                                    totalNRC: this.totalNRC,
                                    totalMRC: this.totalMRC,
                                    deliveryPrice: this.deliveryPrice,
                                    constantPrices: this.constantPrices,
                                    orderAim: 'New'} as OrderValue;
                console.log(orderValue);
                this.orderService.createOrder(orderValue)
                    .subscribe(data => {
```

```
                    this.toaster.success('Order is created');
                    this.router.navigate(['/homeath/orders-and-instances']);
                },
                error => {
                    console.log(error.error.message);
                    if (error.error.message != null && error.error.message != '') {
                        this.toaster.error(error.error.message);
                    } else if (error.message != null && error.message != '') {
                        this.toaster.error(error.message);
                    } else {
                        this.toaster.error(DEFAULT_ERROR_MESSAGE);
                        this.router.navigate(['/homeath/orders-and-instances']);
                    }
                });
    }
}

selectProduct(productId: number) {
    this.products[productId].check = !this.products[productId].check;
    if (productId == 2 && !this.products[1].check) {
        this.products[1].check = true;
    }
    if (productId == 1) {
        if (this.products[2].check) {
            this.products[2].check = false;
        }
        const defaultDevice = {name: 'None', price: 0} as Device;
        if (this.devices == null) {
            this.resourceService.getDevicesForSale()
                .subscribe(data => {
                    this.devices = data;
                    this.devices.splice(0, 0, defaultDevice);
                });
        }
        this.selectedDevice = defaultDevice;
        this.resourceService.getTwistedPairCable()
            .subscribe(data => {
                    this.cable = data;
                    this.cableLengthForm.controls.cableLength.setValue( 3);
                    this.cableLengthForm.controls.cableLength.setValidators([
                        Validators.required,
                        Validators.min(0),
                        Validators.max(data.length / 2)
                    ]);
                },
                error => {
                    //this.toaster.error(error.error.message);
                    this.toaster.error('We can not provide you any cable');
                    this.cableLengthForm.controls.cableLength.setValue( 0);
                    this.cableLengthForm.controls.cableLength.disable();
                });
    }
    if (productId == 0) {
        this.resourceService.getPhoneNumbersForSale()
            .subscribe(data => this.phoneNumbers = data,
                error => {
                    //this.toaster.error(error.error.message);
                    this.toaster.error('Mobile product is not available at the moment');
                    this.products[productId].check = false;
                });
    }
}

validateAll(selectedProducts: string[]): string {
    for (const item of selectedProducts) {
        if (item == PRODUCTS.MOBILE_PRODUCT) {
```

```
                    if (this.selectedPhoneNumber == null) { return 'Please select phone number'; }
                    if (this.selectedTariff == null) { return 'Please select tariff'; }
                    if (this.deliveryAndActivationMobile && this.selectedAddress == null) { return
'Please select address for activation'; }
                }
                if (item == PRODUCTS.INTERNET_PRODUCT) {
                    if (this.selectedSpeed == null) { return 'Please select speed of Internet'; }
                    if (this.selectedAddress == null) { return 'Please select address for
installation'; }
                }
                if (item == PRODUCTS.DTV_PRODUCT) {
                    if (this.selectedChannelNumber == null) { return 'Please select number of
channels'; }
                }
                // if (this.selectedAccount == null) { return 'Please select billing account'; }
                if (this.deliveryDate == null) {
                    if ((this.deliveryAndActivationMobile || this.products[1].check) &&
this.deliveryTime == null) {
                        return 'Please select date and time';
                    } else {
                        return 'Please select date';
                    }
                }
            }
            return '';
        }


        validateAndGetAvailableTimes(value: Date) {
            const curDate = new Date();
            this.deliveryDate = value;
            if (curDate.getDay() === value.getDay()) {
                // this.dateErrorMessage = 'The date cannot be today';
                this.dateErrorMessage = '';
            } else if (value < curDate) {
                this.dateErrorMessage = 'The date cannot be earlier then today';
            } else if (value > this.addDays(curDate, 30)) {
                this.dateErrorMessage = 'The date cannot be more than 30 days from the current';
            } else {
                this.dateErrorMessage = '';
            }
            if (this.dateErrorMessage == '') {
                const selectedProducts = this.getSelectedProductList();
                this.orderService.getAvailableTimes(this.dateFormatPipe.transform(value,
'dd/MM/yyyy'), selectedProducts)
                    .subscribe(data => this.availableTimes = data,
                        error => this.toaster.error(error.error.message));
            }
        }

        addDays(date: Date, days: number) {
            date.setDate(date.getDate() + days);
            return date;
        }

        calculateTotalPrices() {
            if (this.products[1].check) {
                this.deliveryPrice = this.constantPrices.installationPricesForInternetProduct[1];
            } else if (this.deliveryAndActivationMobile) {
                this.deliveryPrice = this.constantPrices.deliveryPrices[1];
            } else {
                this.deliveryPrice = 0;
            }
            this.calculateTotalCablePrice();
            this.calculateTotalMRC();
            this.calculateTotalNRC();
```

```
        }

        calculateTotalMRC() {
            this.totalMRC = 0;
            if (this.products[0].check && this.selectedTariff) {
                this.totalMRC += this.selectedTariff.discountedPrice;
            }
            if (this.products[1].check && this.selectedSpeed) {
                this.totalMRC += this.selectedSpeed.discountedPrice;
            }
            if (this.products[1].check && this.fixedIpSupport) {
                this.totalMRC += this.constantPrices.fixedIpPrices[1];
            }
            if (this.products[2].check && this.selectedChannelNumber) {
                this.totalMRC += this.selectedChannelNumber.discountedPrice;
            }
        }

        calculateTotalNRC() {
            this.totalNRC = this.deliveryPrice;
            if (this.products[0].check && this.selectedPhoneNumber) {
                this.totalNRC += this.selectedPhoneNumber.price;
            }
            if (this.products[0].check && this.support5g) {
                this.totalNRC += this.constantPrices.supportOf5gPrices[1];
            }
            if (this.products[1].check) {
                this.totalNRC += this.cablePriceTotal;
                if (this.selectedDevice) {
                    if (this.selectedDevice.name !== 'None') {
                        this.totalNRC += this.selectedDevice.price;
                    }
                }
            }
        }

        calculateTotalCablePrice() {
            const length = +this.cableLengthForm.controls.cableLength.value;
            if (length > 3) {
                this.cablePriceTotal = +((length - 3) *
this.constantPrices.cableOneMeterPrice[1]).toFixed(2);
            } else {
                this.cablePriceTotal = 0;
            }
        }


        noBillingAccountAction(message: string) {
            this.toaster.error(message);
            this.router.navigate([`/homeath/profile`]);
        }

        openDeviceInfoWindow() {
            this.dialog.open(DeviceInfoComponent, {data: {device: this.selectedDevice}});
        }

        calculateClassForMobile() {
            const classes = {
                first: false,
                second: false,
                third: false,
                mobile_block: true
            };
            if (this.products[1].check) {
                if (this.products[2].check) {
                    classes.third = true;
```

```
                } else {
                    classes.second = true;
                }
            } else {
                classes.first = true;
            }
            return classes;
        }

        getSelectedProductList(): string[] {
            const selectedProducts = [];
            for (const item of this.products) {
                if (item.check) {
                    selectedProducts.push(item.name);
                }
            }
            return selectedProducts;
        }

        getValidationMessage() {
            const controlErrors: ValidationErrors = this.cableLengthForm.get('cableLength').errors;
            let someError = null;
            if (controlErrors != null) {
                for (const controlError in controlErrors) {
                    if (controlErrors[controlError]) {
                        someError = this.validationMessages.find((valMsg) => {
                            return valMsg.type === controlError;
                        });
                        break;
                    }
                }
            }
            return someError;
        }

        goBack() {
            this.location.back();
        }
    }

    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-
awesome.min.css">
    <link rel="stylesheet" href="https://unpkg.com/ngx-bootstrap/datepicker/bs-datepicker.css">
    <app-headerauth></app-headerauth>
    <div class="offers-instances">
        <h4>Products:</h4>
        <div *ngFor="let item of products">
            <label for="{{item.name}}" class="list-group-item">
                <input id="{{item.name}}" [value]="item.id" [checked]="item.check"
(click)="selectProduct(item.id)"
                       class="form-check-input"
                       type="checkbox"> {{ item.name }}
            </label>
        </div>
    </div>
    <div class="activated-discounts">
        <h4>Active discounts:</h4>
        <ul *ngFor="let discount of activatedDiscounts" class="list-group">
            <li class="list-group-item">{{discount.appliedFor}} - {{discount.value}}%<br/>
                Active to {{discount.endDate | date:"dd/MM/yy"}}
            </li>
        </ul>
    </div>
    <div *ngIf="products[0].check" [ngClass]="calculateClassForMobile()">
        <h5>Mobile Product parameters:</h5>
        <label for="phoneNumberList">Phone number: </label>
```

```
        <select id="phoneNumberList" [(ngModel)]="selectedPhoneNumber" name="phoneNumberList"
(ngModelChange)="calculateTotalPrices()">
            <option *ngFor="let phoneNumber of phoneNumbers" [ngValue]="phoneNumber">
                {{phoneNumber.phoneNumber}} - {{phoneNumber.price}}$
            </option>
        </select><br/>
        <label for="tariffList">Tariff: </label>
        <select id="tariffList" [(ngModel)]="selectedTariff" name="tariffList"
(ngModelChange)="calculateTotalPrices()">
            <option *ngFor="let tariff of tariffs" [ngValue]="tariff">
                {{tariff.tariffName}} - {{tariff.discountedPrice}}{{tariff.priceEnding}}</option>
        </select><br/>
        <span class="span-120w">5G support:</span>
        <input [(ngModel)]="support5g"
               class="form-check-input"
               type="checkbox" (ngModelChange)="calculateTotalPrices()">
        <span> - {{constantPrices.supportOf5gPrices[1]}}$</span>
        <br/>
        <div *ngIf="!products[1].check">
            <span class="delivery-span">Delivery and activation:</span>
            <input id="mobile_delivery" [(ngModel)]="deliveryAndActivationMobile"
                   class="form-check-input"
                   type="checkbox" (ngModelChange)="calculateTotalPrices()">
            <span> - {{constantPrices.deliveryPrices[1]}}$ </span><br/>
    <!--        <p>You can pick up your SIM card at the store for the address:
<i>{{storeAddress}}</i></p>-->
        </div>
    </div>

    <div *ngIf="products[1].check" class="first internet-block">
        <h5>Internet Product parameters:</h5>
        <label for="speedList">Speed: </label>
        <select id="speedList" [(ngModel)]="selectedSpeed" name="speedList"
(ngModelChange)="calculateTotalPrices()">
            <option *ngFor="let intOffer of internetOffers" [ngValue]="intOffer">
                {{intOffer.speed}}Mb/s -
{{intOffer.discountedPrice}}{{intOffer.priceEnding}}</option>
        </select><br/>
        <form [formGroup]="cableLengthForm" name="form" (ngSubmit)="submit()" novalidate>
            <div class="form-group">
                <span class="cable-length-span">Cable length: </span>
                <input type="number" class="cable-length" [ngClass]="{'is-invalid':
cableLengthForm.get('cableLength').invalid
                && cableLengthForm.get('cableLength').touched,
                       'is-valid': cableLengthForm.get('cableLength').valid}"
(ngModelChange)="calculateTotalPrices()"
                       name="cableLength" placeholder="3"
                       formControlName="cableLength" required/>
                <span class="cable-price-span"> -
{{constantPrices.cableOneMeterPrice[1]}}$/meter</span><br/>
                <div class="invalid-feedback"
                     *ngIf="cableLengthForm.get('cableLength').invalid &&
cableLengthForm.get('cableLength').touched">
                    {{getValidationMessage().message}}
                </div>
                <span>3 meters of cable are free.</span><br/>
                <span>Total price for cable: {{cablePriceTotal}}$</span>
            </div>
        </form>
        <div *ngIf="devices" class="device-block">
            <label for="deviceList">Device (option):</label>
            <select id="deviceList" [(ngModel)]="selectedDevice" name="deviceList"
(ngModelChange)="calculateTotalPrices()">
                <option *ngFor="let device of devices" [ngValue]="device">
                    {{device.type}} {{device.name}} - {{device.price}}$
                </option>
```

```
            </select>
            <i class="fa fa-info info-btn" aria-hidden="true" (click)="openDeviceInfoWindow()"></i>
        </div>
        <span class="fixed-ip-span">Fixed IP:</span>
        <input [(ngModel)]="fixedIpSupport" (ngModelChange)="calculateTotalPrices()"
               class="form-check-input"
               type="checkbox">
        <span> - {{constantPrices.fixedIpPrices[1]}}$/month</span>
        <br/>
        <!--    <span class="ipv6-support-span">IPv6 support:</span>-->
        <!--    <input [checked]="ipv6Support"-->
        <!--           class="form-check-input"-->
        <!--           type="checkbox">-->
        <!--    <span> - {{constantPrices.ipv6SupportPrices[1]}}$/month</span>-->
        <span class="int-installation-span">Delivery and Installation:</span>
    <!--    <input [checked]="installation"-->
    <!--           class="form-check-input"-->
    <!--           type="checkbox" disabled>-->
        <span> - {{constantPrices.installationPricesForInternetProduct[1]}}$ </span>
    </div>

    <div *ngIf="products[2].check" class="second dtv-block">
        <h5>DTV Product parameters:</h5>
        <label for="channelsList">Number of channels: </label>
        <select id="channelsList" [(ngModel)]="selectedChannelNumber" name="channelsList"
(ngModelChange)="calculateTotalPrices()">
            <option *ngFor="let dtvOffer of dtvOffers" [ngValue]="dtvOffer">
                {{dtvOffer.channelNumber}} -
{{dtvOffer.discountedPrice}}{{dtvOffer.priceEnding}}</option>
        </select><br/>
    </div>

    <div class="delivery-payment-info">
        <h4 class="delivery-payment-info-header">Delivery and Payment info</h4>
        <div class="address-selector">
            <span class="address-span">Address:</span>
            <div>
                <select class="address-select" *ngIf="userAddresses" [(ngModel)]="selectedAddress"
name="addressList">
                <option *ngFor="let address of userAddresses" [ngValue]="address">
                    {{address.country}}, {{address.city}}, {{address.street}},
{{address.buildingNum}}, {{address.roomNum}}  {{address.index}}</option>
                </select>
            </div>
        </div>
        <br/>
        <br/>
        <div class="date-picker">
            <span class="span-120w">Delivery date:</span>
            <input type="text"
                   placeholder="mm/dd/yyyy"
                   class="form-control"
                   (bsValueChange)="validateAndGetAvailableTimes($event)"
                   bsDatepicker>
            <p *ngIf="dateErrorMessage" class="date-error">{{dateErrorMessage}}</p>
        </div>
        <br/>
        <div *ngIf="(deliveryAndActivationMobile || products[1].check) && deliveryDate"
class="delivery-time-selector">
            <span class="span-80w">Time:</span>
            <select [(ngModel)]="deliveryTime" name="deliveryTime">
                <option *ngFor="let time of availableTimes" [ngValue]="time">{{time}}</option>
            </select><br/>
        </div>
    </div>
    <div class="summary-block">
```

```
        <h4>Summary</h4>
        <table style="width: 420px">
            <thead>
            <tr>
                <th scope="col" class="summary-product-name main-block"><h5>Product</h5></th>
                <th scope="col" class="price-cell">NRC</th>
                <th scope="col" class="price-cell">MRC</th>
            </tr>
            </thead>
            <tbody>
                <tr *ngIf="products[0].check">
                    <td class="main-block">
                        <h6>Mobile Product</h6>
                    </td>
                </tr>
                <tr *ngIf="selectedPhoneNumber">
                    <td class="summary-product-name">
                        <span class="child-value">Phone number -
{{selectedPhoneNumber.phoneNumber}}</span>
                    </td>
                    <td class="price-cell">
                        <span>{{selectedPhoneNumber.price}}$</span>
                    </td>
                    <td class="price-cell">
                        <span>0$</span>
                    </td>
                </tr>
                <tr *ngIf="selectedTariff">
                    <td class="summary-product-name">
                        <span class="child-value">Tariff - {{selectedTariff.tariffName}}</span>
                    </td>
                    <td class="price-cell">
                        <span>0$</span>
                    </td>
                    <td class="price-cell">
                        <span>{{selectedTariff.discountedPrice}}$</span>
                    </td>
                </tr>
                <tr *ngIf="support5g">
                    <td class="summary-product-name">
                        <span class="child-value">5G support</span>
                    </td>
                    <td class="price-cell">
                        <span>{{constantPrices.supportOf5gPrices[1]}}$</span>
                    </td>
                    <td class="price-cell">
                        <span>0$</span>
                    </td>
                </tr>

                <tr *ngIf="products[1].check">
                    <td class="main-block">
                        <h6>Internet Product</h6>
                    </td>
                </tr>
                <tr *ngIf="selectedSpeed">
                    <td class="summary-product-name">
                        <span class="child-value">Speed - {{selectedSpeed.speed}} Mb/s</span>
                    </td>
                    <td class="price-cell">
                        <span>0$</span>
                    </td>
                    <td class="price-cell">
                        <span>{{selectedSpeed.discountedPrice}}$</span>
                    </td>
                </tr>
```

```
                <tr *ngIf="products[1].check">
                    <td class="summary-product-name">
                        <span class="child-value">Cable -
{{cableLengthForm.controls.cableLength.value}}m</span>
                    </td>
                    <td class="price-cell">
                        <span>{{cablePriceTotal}}$</span>
                    </td>
                    <td class="price-cell">
                        <span>0$</span>
                    </td>
                </tr>
                <tr *ngIf="(selectedDevice && selectedDevice.name != 'None')">
                    <td class="summary-product-name">
                        <span class="child-value">Device - {{selectedDevice.type}} -
{{selectedDevice.name}}</span>
                    </td>
                    <td class="price-cell">
                        <span>{{selectedDevice.price}}$</span>
                    </td>
                    <td class="price-cell">
                        <span>0$</span>
                    </td>
                </tr>
                <tr *ngIf="fixedIpSupport">
                    <td class="summary-product-name">
                        <span class="child-value">Fixed IP</span>
                    </td>
                    <td class="price-cell">
                        <span>0$</span>
                    </td>
                    <td class="price-cell">
                        <span>{{constantPrices.fixedIpPrices[1]}}$</span>
                    </td>
                </tr>

                <tr *ngIf="products[2].check">
                    <td class="main-block">
                        <h6>DTV Product</h6>
                    </td>
                </tr>
                <tr *ngIf="selectedChannelNumber">
                    <td class="summary-product-name">
                        <span class="child-value">Number of channels -
{{selectedChannelNumber.channelNumber}}</span>
                    </td>
                    <td class="price-cell">
                        <span>0$</span>
                    </td>
                    <td class="price-cell">
                        <span>{{selectedChannelNumber.discountedPrice}}$</span>
                    </td>
                </tr>

                <tr *ngIf="(deliveryPrice!=0)">
                    <td class="summary-product-name main-block">
                        <h6>Delivery & installation/activation</h6>
                    </td>
                    <td class="price-cell">
                        <span>{{deliveryPrice}}$</span>
                    </td>
                    <td class="price-cell">
                        <span>0$</span>
                    </td>
                </tr>
```

```
                <tr *ngIf="(deliveryPrice!=0)">
                    <td class="summary-product-name total-cell">
                        <h5>Total:</h5>
                    </td>
                    <td class="price-cell">
                        <b>{{totalNRC}}$</b>
                    </td>
                    <td class="price-cell">
                        <b>{{totalMRC}}$</b>
                    </td>
                </tr>
            </tbody>
        </table>
    </div>
    <div class="submit-cancel-buttons">
        <button type="button" class="btn btn-success add-btn submit-button"
(click)="submit()">Submit</button>
        <button type="button" class="btn btn-secondary cancel-button"
(click)="goBack()">Cancel</button>
    </div>

    <app-footer class="footer"></app-footer>
```

ДОДАТОК Д

Презентація

Інформаційна система управління та аналізу замовлень телекомунікаційних компаній з використанням мікросервісної архітектури

УКР.НТУУ«КПІ»_НН ІАТЕ_ІПЗЕ_ТВ-12мп

Аркушів 6

2022

НАЦІОНАЛЬНИЙТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці

Інформаційна система управління та аналізу замовлень телекомунікаційних компаній з використанням мікросервісної архітектури

Виконав
студент групи ТВ-12мп
Комісаров Ілля

Науковий керівник
доц. Ходаковський О.В.

Актуальність роботи

- Незручність та великі затрати часу на підключення та надання телекомунікаційних послуг вручну
- Дорога вартість існуючих систем для невеликих провайдерів
- Існуючі системи потребують достатньо багато обчислювальних ресурсів, на обслуговування яких у невеликих провайдерів немає коштів
- Потреба аналітики замовлень та продажів для корекції пропозицій та збереження клієнтів

Постановка задачі

- Реєстрація та авторизація користувачів
- Управління даними клієнтів
- Повна автоматизація процесу виконання замовлень
- Управління ресурсами
- Аналітика даних

Об’єкт та предмет дослідження

- **Об’єктом дослідження** є процес виконання замовлень для підключення абонентів до мережі
- **Предметом дослідження** є система, що автоматизує процес виконання замовлення та надає аналітику даних для невеликих провайдерів телекомунікаційних послуг

Мета дослідження

Що досліджується? Система управління та аналізу замовлень телекомунікаційних компаній.

Для чого досліджується? Для розробки системи автоматизації та прискорення процесу надання послуг, мінімізації витрат телекомунікаційних компаній та покращення якості послуг.

Очікуваний результат? Система, що підходить для невеликих провайдерів телекомунікаційних послуг та задовольняє їхні потреби в автоматизації певних процесів, а також надає аналітику даних.

Яким шляхом планується отримати результат? Дослідження потреб та існуючих систем, розробка системи, тестування, впровадження.

Яким буде результат у соціально-економічному контексті? Автоматизації та прискорення процесу надання послуг, надання аналітики даних, що разом максимізує прибуток телекомунікаційних компаній та покращує якість послуг.

Дослідження існуючих рішень

Розглянуті системи:

- Netcracker BSS/OSS
- Amdocs BSS/OSS
- Bilink
- Softlink

Використані технології

СУБД:

- PostgreSQL;
- MongoDB.

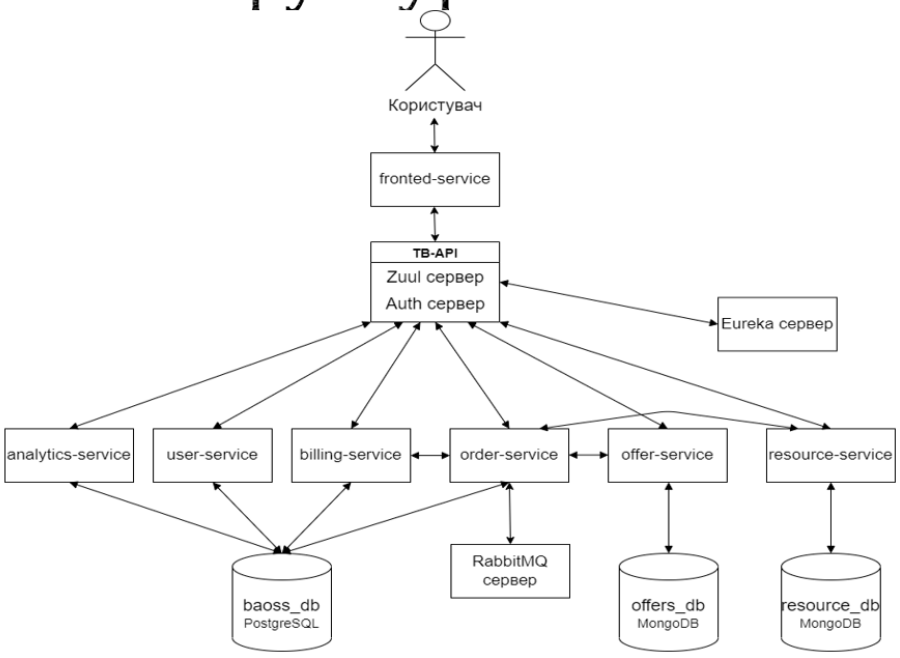
Серверна частина:

- мови програмування – Java, Python;
- фреймворки – Flask, Spring Boot, Spring Data, Spring Cloud, Spring Security;
- бібліотеки аналітики даних – numpy, pandas, scikit-learn, sktime;
- брокер повідомлень – RabbitMQ;
- засіб контейнеризації – Docker.

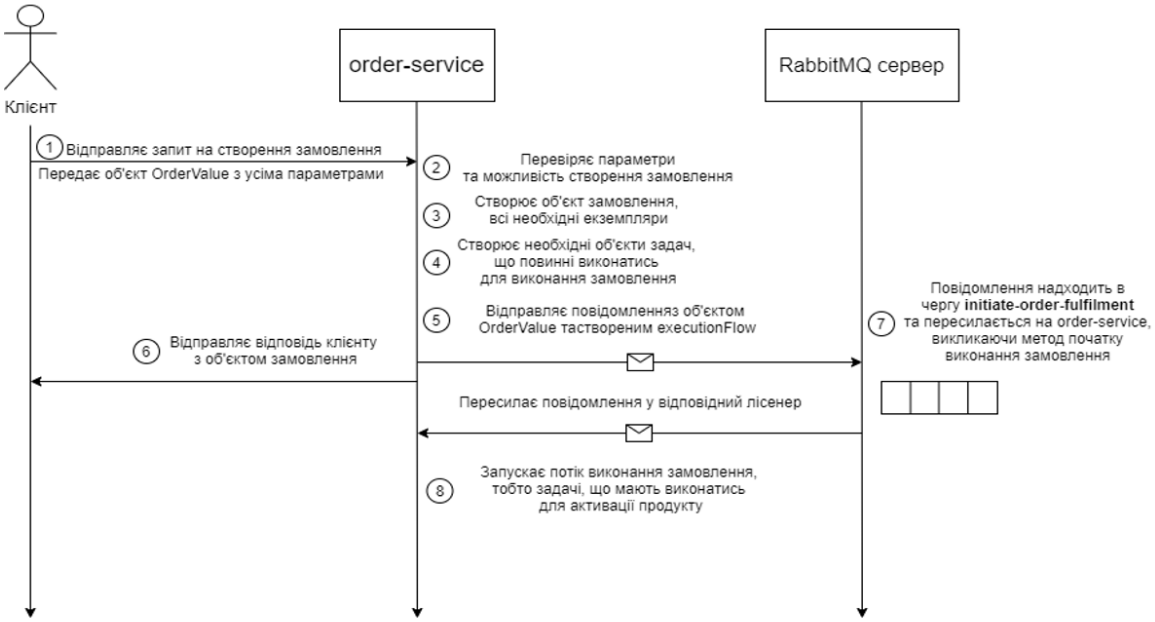
Клієнтська частина:

- мова програмування – TypeScript;
- веб-фреймворк – Angular;
- засоби верстки – HTML/CSS;
- засіб візуалізації даних – Plotly.js

Загальна структура системи



Процес створення замовлення



Практична цінність та новизна

Практичне значення полягає в можливості використання системи реальними провайдерами телекомунікаційних послуг, що зекономить час та кошти цільових компаній, а також дозволить більш коректно скласти стратегію по розвитку бізнесу та утриманню клієнтів.

Новизна полягає в наданні в рамках системи для невеликих провайдерів аналітики даних та інтерфейсу для клієнтів для створення замовлення, чого в подібних системах немає.

Висновки

Розроблена система підійде невеликим сервіс-провайдерам, які надають основні телекомунікаційні послуги (Інтернет, DTV) кінцевим клієнтам, так як система майже повністю покриває основні потреби таких компаній.

Переваги розробленої системи перед іншими аналогами:

1) простота використання/інтуїтивний інтерфейс;
2) швидкість та простота розгортування системи завдяки Docker;
3) можливість легкого масштабування/додавання нової функціональності;
4) швидкість/простота розробки/підтримки системи;
5) надання аналітики даних

Недоліки системи:

1) Урізана функціональність;

ДЯКУЮ ЗА УВАГУ!