НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2021 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою “Комп’ютерні системи та мережі”

спеціальності 123 “Комп’ютерна інженерія”

на тему: *Автоматизована система управління замовленнями та ресурсами для телекомунікаційних компаній*

Виконав: студент IV курсу, групи ІО-71

Комісаров Ілля Андрійович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник пос. Ружевський Микита Сергійович _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Консультант н. контроль доц. д.т.н. Сімоненко В.П. _____
(назва розділу) (вчені ступінь та звання, прізвище, ініціали) (підпис)

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

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

Київ - 2021 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“Комп’ютерні системи та мережі”

спеціальність 123 “Комп’ютерна інженерія”

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ Сергій СТІРЕНКО
(підпис)

“__” _____2021 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента

Комісарова Іллі Андрійовича

1. Тема проєкту: *Автоматизована система управління замовленнями та ресурсами для телекомунікаційних компаній*
   керівник проєкту пос. Ружевський Микита Сергійович ,
   (прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
   затверджені наказом по університету від “ 11 ” травня 2021 року № 1139-с
2. Термін здачі студентом закінченого проєкту: *29 травня 2021 р.*
3. Вихідні дані до проєкту: *технічна документація*
4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються): *опис завадання та аналіз існуючих рішень. Описання архітектури та використаних технологій серверної та клієнтської частин. Реалізація системи та інструкція щодо використання додатку.*
5. Перелік графічного матеріалу (з точним позначенням обов’язкових креслень) схема програми, схема роботи системи):
   ІАЛЦ.467100.003 Д1 Структурна схема: схема бази даних,
   ІАЛЦ.467100.004 Д2 Функціональна схема: діаграма класів,

ІАЛЦ.467100.005 Д3 Принципова схема: схема алгоритму створення замовлення.

6. Консультанта проєкт, з вказівкою розділів роботи, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
	Сімоненко В.П		

7. Дата видачі завдання: 01.09.2020

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	Затвердження теми роботи	01.09.2020-20.10.2020	
2.	Вивчення та аналіз завдання	20.10.2020-10.11.2020	
3.	Розробка архітектури та загальної структури системи	10.11.2020-12.12.2020	
4.	Розробка структур окремих підсистем	12.12.2020-01.01.2021	
5.	Програмна реалізація системи	01.01.2021-13.05.2021	
6.	Оформлення посянювальної записки	13.05.2021-24.05.2021	
7.	Захист програмного продукту	24.05.2021	
8.	Передзахист	25.05.2021	
9.	Захист	14.06.2021	

Студент-дипломник _____
(підпис)

Керівник роботи _____
(підпис)

Анотація

В даній бакалаврській дипломній роботі було розроблено систему для автоматизації деяких процесів для телекомунікаційних компаній. А саме - управління замовленнями та ресурсами.

Розроблена система дозволяє реєструватися та авторизуватися користувачам з різними правами доступу, створювати замовлення, переглядати стан їх виконання тощо.

Система складається з двох частин: клієнтська та серверна. Серверна частина розроблена на мові програмування Java та фреймворку Spring з використанням мікросервісної архітектури. Клієнтська частина розроблена на мові програмування TypeScript та фреймворку Angular.

Annotation

In this bachelor's thesis, a system was developed to automate some processes for telecommunications companies. Namely - order and resource management.

The developed system allows users with different access rights to register and log in, create orders, view the status of their provisioning etc.

The system consists of two parts: client and server. The server side is developed using Java programming language and Spring framework with a microservice architecture approach. The client part is developed using TypeScript programming language and Angular framework.

Опис альбому

	Формат	Позначення	Найменування	Кільк. аркушів	№ екземпляру	Примітка
1			Документація загальна			
			Знову розроблена			
2	А4	ІАЛЦ.467100.001 ТЗ	Технічне завдання	4		
3	А4	ІАЛЦ.467100.002 ПЗ	Пояснювальна записка	84		
4	А4	ІАЛЦ.467100.003 Д1	Структурна схема:	1		
			Схема бази даних			
5	А4	ІАЛЦ.467100.004 Д2	Функціональна схема:	1		
			Діаграма класів			
6	А4	ІАЛЦ.467100.005 Д3	Принципова схема:	1		
			Схема алгоритму			
			створення замовлення			
7	А4	ІАЛЦ.467100.006 Д4	Лістинг програми	20		

					ІАЛЦ.467100.001 ОА		
Змн	Арк.	№ докум.	Підпис	Дата			
Розроб.		Комісаров І.А.			Автоматизована система управління замовленнями та ресурсами для телекомунікаційних компаній Опис альбому	Літ.	Арк. 1 / Аркушів 1
Перевір.		Ружевський М.С					
Н. Контр.		Сімоненко В.П.				НТУУ ‘‘КПІ’’ ФІОТ ІО-71	
Затверд.		Стіренко С.Г.					

Технічне завдання

Технічне завдання до дипломної роботи

Зміст

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання розповсюджується на розробку системи автоматизації процесів для телекомунікаційних компаній, яка дозволяє створювати замовлення, відстежувати стан їх виконання користувачам та управляти ресурсами компанії користувачам з відповідними правами.

Область застосування: зменшення часу виконання замовлення, спрощення управління ресурсами для телекомунікаційних компаній.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для виконання дипломної роботи та розробки проекту є завдання на створення системи автоматизації процесів для телекомунікаційних компаній для отримання кваліфікаційно-освітнього рівня «бакалавр», затверджене НТУУ «Київський політехнічний інститут імені Ігоря Сікорського».

3. МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою даного проекта є розробка системи автоматизації процесів для телекомунікаційних компаній з використанням мікросервісної архітектури, коли система функціонально розбита на незалежні модулі для забезпечення швидкості та гнучкості розробки складних систем.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки є досвід роботи в компанії Netcracker, яка спеціалізується на розробці таких систем. Це і надихнуло на створення подібної системи з урізаною функціональністю.

5. ТЕХНІЧНІ ВИМОГИ

5.1.Вимоги до розроблюваного продукту

- Клієнтська частина з інтуїтивним та зрозумілим інтерфейсом з можливістю реєстрації та авторизації для роботи в системі.
- Серверна частина з реалізацією бізнес логіки, реєстрації та авторизації з використанням мікросервісної архітектури.
- Зберігання даних користувачів, замовлень та ресурсів компанії в реляційних та нереляційних базах даних.
- Використання сучасних принципів та технологій розробки.

5.2.Вимоги до програмного забезпечення

- Операційна система Windows 7/8/10, Linux або MacOS. Для мобільних пристроїв – Android, iOS.
- Веб-браузер.

5.3.Вимоги до апаратної частини

- Процесор Intel/AMD 2/Qualcom/MediaTech 1GHz або більше.
- Оперативна пам'ять 1 Гб або більше.
- Вільний простір на диску 500 Мб або більше.
- Підключення до інтернету.

6. ЕТАПИ РОЗРОБКИ

Вивчення літератури	20.10.2020
Складання та узгодження технічного завдання	10.11.2020
Аналіз структури розроблюваної системи	05.01.2021
Створення модулів розроблюваної системи	15.03.2021
Тестування	01.05.2021
Налагодження та виправлення помилок	15.05.2021
Оформлення документації дипломної роботи	19.05.2021

Пояснювальна записка

Зміст

СПИСОК СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

BSS	Business Support System
OSS	Operations Support System
GUI	Graphical User Interface
API	Application Programming Interface
REST	REpresentational State Transfer
SOAP	Simple Object Access Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
JSON	JavaScript Object Notation
XML	eXtensible Markup Language
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
E2E	End to End
JVM	Java Virtual Machine
AOP	Aspect-Oriented Programming
JAR	Java ARchive
WAR	Web ARchive
JPA	Java Persistence API
MVC	Model-View-Controller
JWT	JSON Web Token
SQL	Structured Query Language
AMQP	Advanced Message Queuing Protocol
CSS	Cascading Style Sheets
SPA	Single Page Application
DTV	Digital TeleVision

ВСТУП

З розвитком технологій програмування все більше процесів в побуті та в бізнесі почали автоматизуватися. Підприємства почали інвестувати кошти в розробку систем для автоматизації власних бізнес процесів, так як вони з роками окупаються сторицею. З кожним роком тенденція до автоматизації тільки збільшується і сфера телекомунікацій не стала виключенням. Ще в минулому столітті у телекомунікаційних компаній виникла потреба зберігати дані клієнтів, вести облік ресурсів, налагодити процес замовлення продуктів, заздалегідь проектувати локальній мережі, моніторити їх стан та швидко приймати необхідні рішення у випадку неполадок. Для цих потреб були створені відповідні системи, які зараз називаються BSS (англ. Business Support System) та OSS (англ. Operations Support System).

Рис.1 – Структура BSS та OSS

BSS – система підтримки бізнеса, яка відповідає за управління замовленнями, виставлення рахунків, продажі та маркетинг, управління даними користувачів, підтримка користувачів.

OSS – система підтримки операцій, яка відповідає за облік та управління ресурсів (кабелі, комутатори, маршрутизатори, SIM карти і т.д.), установку та налаштування необхідного забезпечення в рамках виконання замовлення певного продукту, програмне моделювання мережі та моніторинг її стану, спроби автоматично усунути неполадки у випадку їх виявлення і багато іншого.

Загальна структура та взаємодія цих систем зображена на рисунку 1. Потрібно зважати, що вказана функціональність не завжди реалізована у вказаних систем, як правило, кожна з них - це відносно незалежний модуль, за реалізацію якого платить замовник, тобто телекомунікаційна компанія. Тому часто можна зустріти системи з одним або декількома модулями, в залежності від потреб замовника. Але, загалом, дані системи володіють приблизно такою функціональністю.

Розроблена система в цій роботі буде мати урізану функціональність в порівнянні з комерційними аналогами, так як компанії, що займаються створенням таких систем, витрачають роки на розробку, для досягнення необхідного функціоналу та високої надійності, витрачаючи на неї (розробку) велику кількість ресурсів. Незважаючи на це, розроблена система буде мати мінімальний необхідний функціонал, для працездатності такої системи. Більший акцент буде наданий саме BSS, так як BSS оперує програмними засобами, а OSS – апаратними, що реалізувати набагато важче, не маючи їх у використанні.

РОЗДІЛ 1. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

1.1 Опис завдання

Завдання даної роботи полягає в створенні системи з наступними вимогами:

1) Можливість авторизації та реєстрації користувачів з різними правами, в залежності від ролі користувача. В системі повинні бути присутні такі ролі:
    - Користувач (User);
    - Монтажник (Fitter);
    - Менеджер з продажу (Sales manager);
    - Менеджер по персоналу (HR manager);
    - Менеджер ресурсів (Resource manager);
    - Адміністратор (Admin).
2) У ролі Користувач (User) повинні бути права на:
    - Перегляд пропозицій продуктів та цін;
    - Вибір продукту та задання відповідних параметрів замовлення для покупки;
    - Перегляд та зміну власних даних на персональній сторінці;
    - Перегляд стану замовлення;
    - Перегляд параметрів активованих продуктів.
3) У ролі Монтажник (Fitter) повинні бути права на:
    - Перегляд очікуваних доставок;
    - Підтвердження виконання доставки/установки обладнання/ підключення послуги;
    - Перегляд та зміну власних даних на персональній сторінці.
4) У ролі Адміністратор (Admin) повинні бути права на:
    - Всі вищеперелічені дії для інших ролей.

5) Система повинна мати такі сторінки:

- Сторінка пропозицій товарів: підключення Інтернету, підключення цифрового телебачення, купівля сім карти з тарифом. Сторінка доступна неавторизованому користувачу.
- Сторінки реєстрації та авторизації. Сторінки доступні неавторизованому користувачу.
- Сторінка профіля користувача з можливістю редагування даних. Сторінка доступна тільки авторизованому користувачу.
- Сторінка створення замовлення. Сторінка доступна тільки авторизованому користувачу.
- Сторінка всіх замовлень та підключених продуктів користувача. Сторінка доступна тільки авторизованому користувачу з роллю Користувач (User).
- Сторінка певного замовлення зі всіма параметрами, посиланнями на сторінки продуктів, які були підключені в рамках цього замовлення, та діями, які були виконані/виконуються в рамках цього замовлення, ці дії визначають потік виконання замовлення (англ. - execution flow). Сторінка доступна тільки авторизованому користувачу з роллю Користувач (User).
- Сторінка певного продукта зі всіма параметрами цього продукту. Сторінка доступна тільки авторизованому користувачу з роллю Користувач (User).
- Сторінка очікуваних доставок для монтажника/кур’єра зі всіма параметрами цих доставок та додатковою інформацією, необхідною для доставки. Сторінка має забезпечувати введення даних, необхідних для продовження виконання замовлення (MAC адреса тощо) та можливість підтвердження виконання доставки/підключення.

Сторінка доступна тільки авторизованому користувачу з роллю Монтажник (Fitter).

6) Система повинна забезпечувати всі етапи виконання замовлення: від ініціалізації замовлення до установки обладнання та підключення абонента.
7) Система повинна автоматично знімати місячну плату з абонентів, і якщо коштів не вистачає призупиняти послугу.

1.2 Порівняння аналогів

На даний момент у світі існує доволі багато рішень в цій сфері. Але, так як BSS/OSS – це закриті комерційні проекти, то інформації у відкритому доступі про них дуже мало. Тому в порівнянні не буде точних даних і оцінки функціональності, так як такої інформації немає у відкритому доступі. Існує декілька незалежних сервісів, які займаються порівнянням даних систем. Одним із таких є gartner.com[1]. Розглянувши статистику порівняння за 2015 рік, бачимо, що найбільш ефективними є системи таких компаній: Netcracker Technology, Amdocs, Oracle, Huawei.

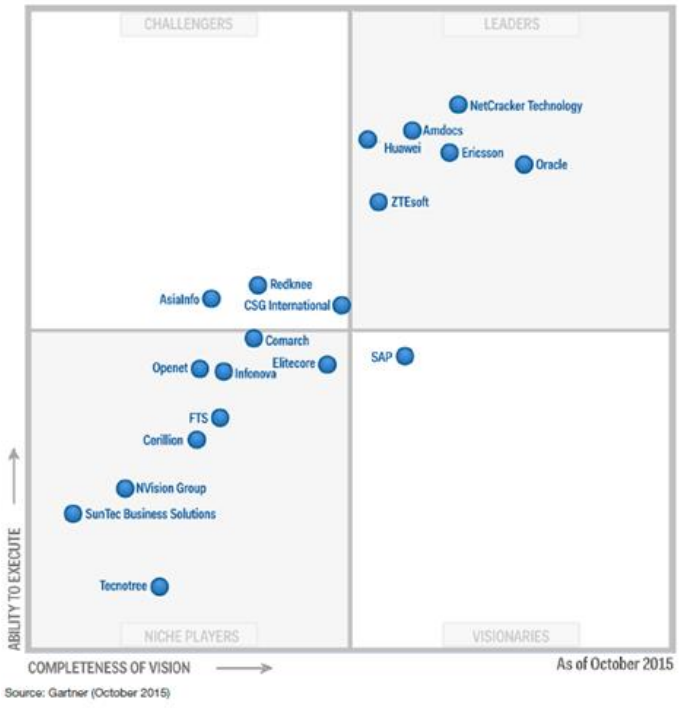


Рис. 1.1 - Рейтинг BSS систем різних компаній за 2015р. [1]

Рис. 1.2 - Рейтинг OSS систем різних компаній за 2015р. [1]

На рисунках 1.1 та 1.2 зображено 4 квадрати – в лівому нижньому знаходяться нішові компанії, які тільки «заходять на ринок», в правому нижньому – компанії, що давно на ринку, але почали втрачати позиції, в лівому верхньому – компанії, які недавно на ринку, але мають багатий функціонал, в правому верхньому – лідери сфери, які довго на ринку і мають багатофункціональні системи.

Зважаючи на результати вказаних графіків, розглянемо детальніше (наскільки це можливо) продукти таких компаній: Netcracker Technology, Amdocs, Oracle, Huawei.

1.2.1 Netcracker BSS/OSS [2] [3]

Компанія Netcracker є розробником BSS/OSS вже 25 років та вважається лідером у цій сфері для постачальників послуг зв'язку та підприємств у всьому світі.

Розглянемо спочатку BSS рішення. Система включає управління маркетингом, управління каналами, управління подорожами абонентів,

управління даними абонентів, управління доходами, управління продажами, управління партнерами та управління продуктами.

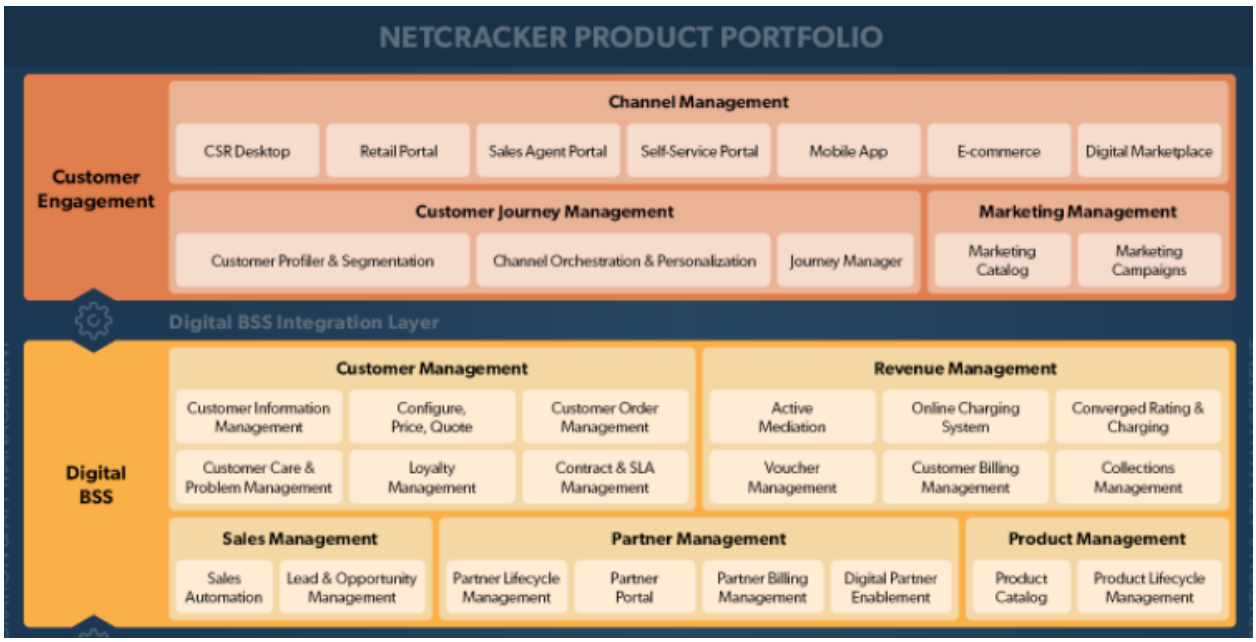


Рис. 1.3 – BSS портфоліо Netcracker Technology [2]

Розглянемо OSS рішення [3]. Netcracker OSS використовує власну архітектуру на базі мікросервісів, щоб забезпечити гнучкість та стійкість рішення. Дана система забезпечує управління такими сервісами: SDN / NFV, Cloud, IoT та 5G та автоматизує операції в складних гібридних середовищах.

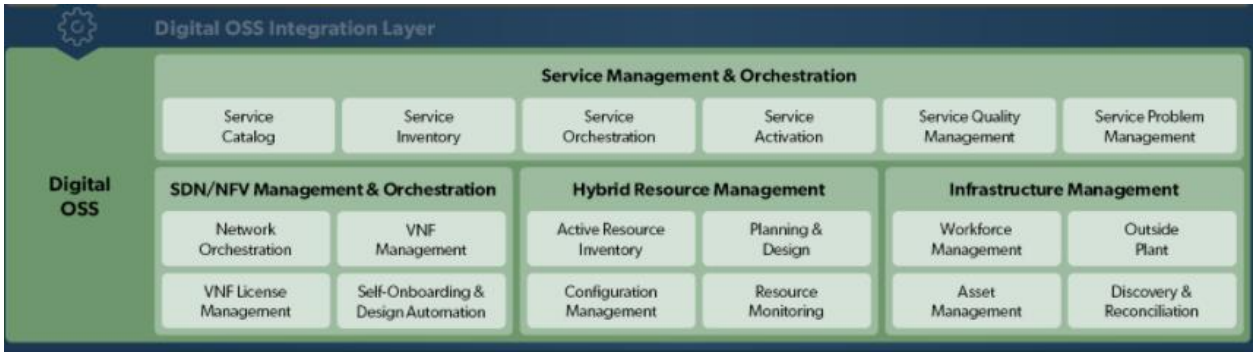


Рис. 1.4 – OSS портфоліо Netcracker Technology [3]

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

1.2.2 Amdocs BSS/OSS [4]

Компанія Amdocs розробляє системи для білінгу та системи управління взаємовідносинами з клієнтами (BSS). Також системи операційної підтримки (OSS) в основному націлені на державний сектор.

Основні складові портфоліо Amdocs:

1) Amdocs Billing – білінг система, що забезпечує облік транзакцій найбільших світових сервіс-провайдерів.
2) Amdocs CRM – дана система оптимізує бізнес-процеси, в центрі яких знаходиться клієнт; система спрямована на ліквідацію основних проблем, які впливають на сталість клієнтури і загальну ефективність діяльності;
3) Amdocs Order Management – масштабована система управління замовленнями, що задовольняє найскладніші вимоги сервіс-провайдерів щодо цієї діяльності;
4) Amdocs Content Revenue Management – система управління потоком виконання поставок контенту від партнера до кінцевого споживача;
5) Amdocs Mediation – система, що задовольняє різні вимоги і реалізує логіку широкомасштабного обслуговування клієнтів. Вона спрямована на вирішення найбільш задач щодо скорочення експлуатаційних витрат і підвищення ефективності бізнесу.
6) Amdocs Ensemble – білінгова система, що розширює Amdocs Billing. В дану систему входять такі модулі: підтримка продаж, підтримка клієнтських операцій і, власне, білінг.

На рисунку 1.5 графічно зображене детальне портфоліо продуктів компанії Amdocs.

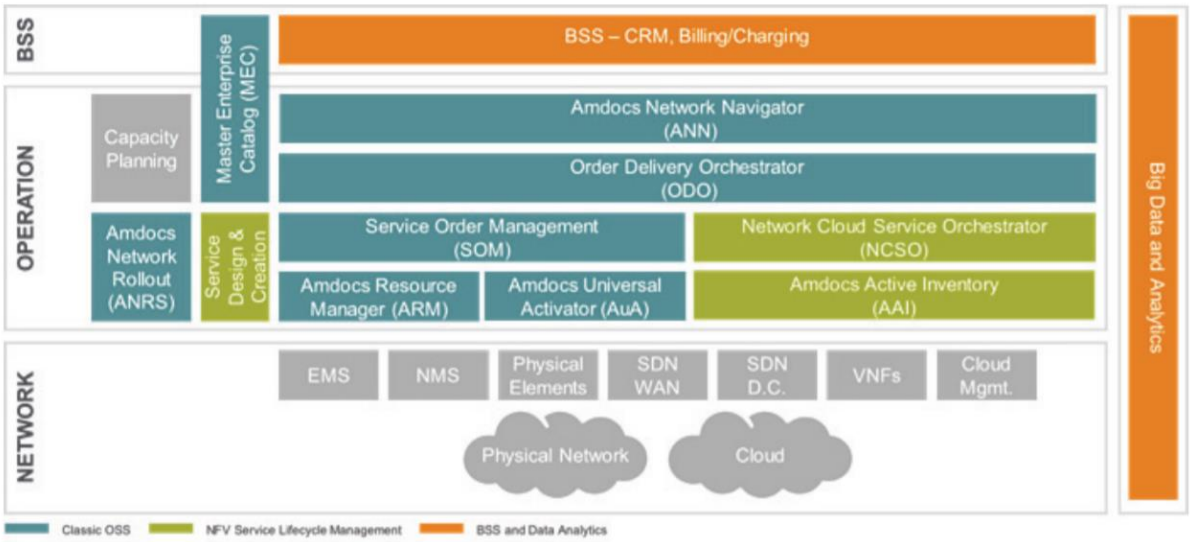


Рис. 1.5 – BSS/OSS портфоліо Amdocs [5]

1.2.3 Oracle OSS [6]

OSS система від Oracle називається Oracle Communications OSS і вона дозволяє постачальникам послуг зв'язку (сервіс-провайдерам) планувати, створювати, оптимізувати мережі, включаючи такі можливості: обслуговування мобільних мереж і оптоволоконних широкосмугових мереж, управління багатоканальними мережами і підтримку консолідації та міграції мереж.

Система Oracle Communications OSS покращує здатність сервіс-провайдерів (CSP) швидко і ефективно проектувати, запускати і розгортати нові телекомунікаційні сервіси, такі як високошвидкісні LTE-мережі, інтернет-телебачення (IPTV), технологію передачі голосового трафіку через IP-мережі (VoIP), широкосмуговий доступ та різноманітні інформаційні служби.

Система OSS від Oracle складається з таких модулів:

- Oracle Communications Design Studio;

- Oracle Communications Order and Service Management;
- Oracle Communications Unified Inventory Management;
- Oracle Communications Network Integrity;
- Oracle Communications Network Intelligence;
- Oracle Communications ASAP;
- Oracle Communications IPSA.

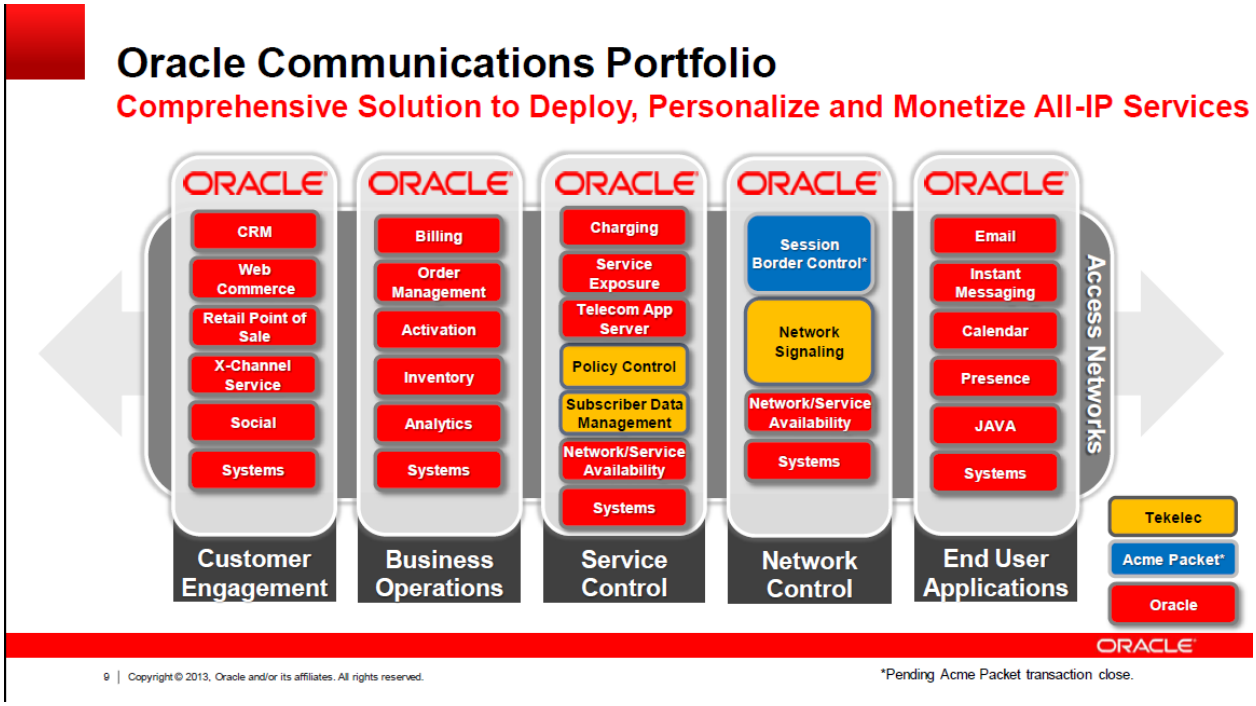


Рис. 1.6 – BSS/OSS портфоліо Oracle [7]

Таблиця 1.1. Оцінка можливостей Oracle OSS сервісом gartner.com [8]:

№	Параметр	Оцінка
1	Повне виконання замовлення: забезпечення та активація	4.6
2	Забезпечення обслуговування	4.4
3	Планування та інженерія	4.5
4	Управління співробітниками	4.8
5	Операційна аналітика / Мережева аналітика	4.8
6	Каталог товарів / послуг та управління життєвим циклом продукції	4.8
7	Управління політикою	4.3

Як бачимо, результати непогані, що ставить продукти Oracle на лідируючі позиції.

1.2.4 Huawei OSS [9]

OSS система у Huawei називаєтсья OSS-as-a-Service. OSS-as-a-Service – це модель бізнесу, яка пропонує цілісні можливості OSS через портнерство із сервісними службами протягом всього періоду контракту. OSSaaS Huawei складається з трьох рівнів: рівень бізнес-моделі, що описує потенційні бізнес-цінності, які будуть сформовані, та визначає, як вони будуть проявлятися з точки зору ефективності та задоволеності споживачів. Шари Use Case розробляють бізнес-додатки, які вирішуватимуть проблемні ситуації, щоб створити цінності, що вказані в рівні бізнес-моделі. Кожен випадок використання надє засоби вирішення однієї або декількох проблем і опирається на ряд компонентнів, які будуть доставлені під час процесу доставки. Шар Cloud OSS забезпечує можливості прогрмного забезпечення OSS через приватну, загальнодоступну або гібридну «хмару» (Cloud).

Незважаючи на те, що OSSaaS все ще розробляється, ринок телекомунікацій в цілому та сервіс-провайдери зараз активно досліджують його потенційні вигоди для бізнесу, переваги та економію коштів.

Як підтвердили провідні сервіс-провайдери, через OSSaaS вже реалізуються такі значення:

- Підвищена гнучкість для задоволення конкретних вимог до послуг;
- Уникнення дублікатів. Централізована модель управління OSSaaS дозволяє локальним мережам великих глобальних або регіональних сервіс-провайдерів спільно використовувати послуги OSS та операційні процеси;

- Передбачувані інвестиції. Останні п’ять років усі розробники BSS/OSS задіяні у проектах зменшення витрат на рівні всієї організації, щоб максимізувати ефективність та зменшити капіталовкладення, пов’язані з модернізацією та розширенням мереж;
- OSSaaS використовує модель підписки, розподіляючи витрати на весь час роботи мережі, відповідно до індивідуальних потреб. Початкові дослідження показують, що використовуючи OSSaaS можна здійснити зниження загальних витрат на 15-20%;
- Значно зменшено кількість циклів оновлення. Відтепер оновлення програмного забезпечення здійснюється безпосередньо на серверах сервіс-провайдера, і кінцеві користувачі переходять на найновіші версії протягом декількох годин, а не тижнів/місяців, як це відбувається з поточною застарілою моделлю;
- Швидке та еластичне масштабування. OSSaaS дозволяє сервіс-провайдерам розширювати свою архітектуру OSS разом з мережею, і робити це швидко та ефективно;
- Адаптація до нових технологі. Сервіс-провайдери, які експериментують з новими цифровими послугами, повинні швидко створити нову функціональності для пітримки служб, не роблячи значних вкладень коштів. OSSaaS дозволяє сервіс-провайдерам швидко вивести на ринок нову цифрову послугу для тестування PoC і, якщо потрібно, ефективно розширити операцію з мінімальними витратами;
- Програмне забезпечення, яке підтримується постачальником залишається в руках експертів, які його створили, що дозволяє сервіс-провайдеру отримати доступ до безцінного досвіду архітекторів та розробників систем постачальника.

Таблиця 1.2. Оцінка можливостей Huawei OSS сервісом gartner.com [10]:

№	Параметр	Оцінка
1	Повне виконання замовлення: забезпечення та активація	4.0
2	Забезпечення обслуговування	5.0
3	Планування та інженерія	4.0
4	Управління співробітниками	4.0
5	Операційна аналітика / Мережева аналітика	3.0
6	Каталог товарів / послуг та управління життєвим циклом продукції	4.0
7	Управління політикою	4.0

На жаль на дану систему був наданий тільки один відгук, тому результати не можна вважати об’єктивними.

ВИСНОВКИ ДО РОЗДІЛУ 1

У першому розділі були розглянуті системи BSS/OSS від різних компаній. Описання цих систем вийшло доволі абстрактним, більше схожим на рекламу, так як інформація була взята з офіційних сайтів компаній. Більше детальної інформації у відкритому доступі, на жаль, немає, тому порівняння не може бути повноцінним та об’єктивним. Але можна сказати непевно, що всі ці системи доволі схожі і мають практично однакову функціональність. Клієнти звертають увагу більше на конкретні модулі, які їм потрібні, і на їх ціну. З цієї причини не можна сказати, що якась система найкраща, це залежить від багатьох факторів.

У даній роботі розроблена система, яка має базову функціональність. Дана система може бути цікава сервіс-провайдерам, які представляють малий бізнес, так як вони не використовують технології, які використовують великі компанії; їх цікавить автоматизація базових процесів, таких як – повне виконання (fulfilment) замовлення, управління ресурсами та співробітниками. Ця функціональність і буде реалізована в розроблюваній системі.

РОЗДІЛ 2. ОПИС АРХІТЕКТУРИ ТА ВИКОРИСТАНИХ ТЕХНОЛОГІЙ

2.1 Використані архітектури

В рамках даної роботи було використані такі архітектури: клієнт-серверна архітектура, архітектурний шаблон REST, багатошарова архітектура та микросервісна архітектура. Всі ці архітектури були розроблені для різних задач, тому можуть комбінуватися між собою, а також з іншими архітектурами, які не перечать принципам цих архітектур. Розглянемо кожну з цих архітектур більш детально.

2.1.1 Клієнт-серверна архітектура

Клієнт-серверна архітектура представляє собою модель, в якій присутній сервер (тобто програмне забезпечення, яке розміщене на фізичному сервері та підключене до глобальної мережі Інтернет), в ролі постачальника послуг та клієнт, що представляє собою споживача, як правило клієнтом є браузер або мобільна прикладна програма, які звертаються до сервера, але клієнтом може бути довільний сервіс, який також може звертатися до сервера.

Рис. 2.1 – Схема клієнт-серверної архітектури

На рисунку 2.1 зображена схема клієнт-серверної архітектури. Як бачимо, вона, окрім сервера та клієнта, має також блок бази даних. База даних необов’язкова, якщо нам не потрібно зберігати дані після завершення клієнтської сесії. Але в 95% випадків, все таки, потрібно зберігати дані клієнта, тому база даних стала де-факто стандартом в клієнт-серверній архітектурі.

Розглянемо кожен з цих блоків більш детально:

1) Клієнт. Як зазначалось вище, клієнтом, як правило, є кінцевий користувач, який за допомогою браузера, в якому відкриваються HTML сторінки, переглядає/вводить дані або виконує інші дії та за допомогою мови JavaScript (або іншим мовам, які були створені на його основі) відправляє запити на сервер, використовуючи протокол передачі даних HTTP (або інший протокол передачі даних, але в 99% випадків використовується саме HTTP). Також клієнтом може бути мобільна програма на Android або IOS, яка за допомогою відповідних інструментів також може відображати/приймати дані від користувача та відправляти запити на сервер. Окрім забезпечення, з яким взаємодіє користувач, клієнтом може бути певний сервіс, який представляє собою, програмне забезпечення, яке, в свою чергу, розміщене на сервері та відправляє запити на сервер. В такому разі один сервер є клієнтом, а інший, власне, сервером, в залежності від того, який з них відправляє запит.
2) Сервер. Сервером називається як програмне забезпечення так і комп'ютер, на якому розміщене програмне забезпечення. Сервер приймає HTTP запити та повертає клієнту HTTP відповіді. Програма, яка встановлена на сервері, може бути написана на таких високорівневих мовах, як Java, C#, Python, PHP, Go тощо. Також сервер має доступ до бази даних. На запит клієнта сервер «дістає» дані з бази даних, обробляє їх та повертає клієнту в JSON або в XML форматі (як правило,

використовується JSON, так як цей формат легко конвертується в JavaScript об’єкти, але якщо використовується певні протоколи/технології, наприклад, SOAP, які працюють з XML форматом, то для передачі даних використовується саме він).

3) База даних. База даних є сховищем структурованих даних, здебільшого, в текстовій формі. Бази даних функціонують під управлінням спеціальних програмних засобів, які називаються Системами Управління Базами Даних (СУБД). СУБД дозволяє створювати бази даних, структури для зберігання даних, вставляти в ці структури самі дані, змінювати їх читати і багато іншого. Також вони забезпечують цілісність надійність та відмовостійкість бази даних. Найбільш популярними СУБД є Oracle, PostgreSQL, MySQL, Microsoft SQL Server, MongoDB.

В даному підрозділі було розглянуто базові принципи та поняття клієнт-серверної архітектури. Існує багато варіації цієї архітектури, в залежності від потреб програм.

2.1.2 Багаторівнева архітектура

Багаторівнева архітектура є різновидом клієнт-серверної архітектури. Дана архітектура призначена для розділення зон відповідальності функцій системи на декілька рівнів. Зазвичай виділяють 3 рівні – рівень представлення, рівень бізнес логіки та рівень доступу до даних. Ці рівні також називають рівнями абстракції, так як на рівні бізнес логіки ми не повинні перейматись правильним збереженням даних до бази даних. Абстракції допомагають вдало організувати роботи команди, так як легко можна розділити зони відповідальності. Таким чином, команда, яка займається розробкою бізнес логіки ніяким чином не впливає на діяльність команди, яка відповідальна за збереження даних, так як кожен рівень надає інтерфейс для використання його

функціональності іншому рівню. Такий підхід сприяє прискоренню розробки програми та полегшенню її підтримки в довготривалій перспективі.

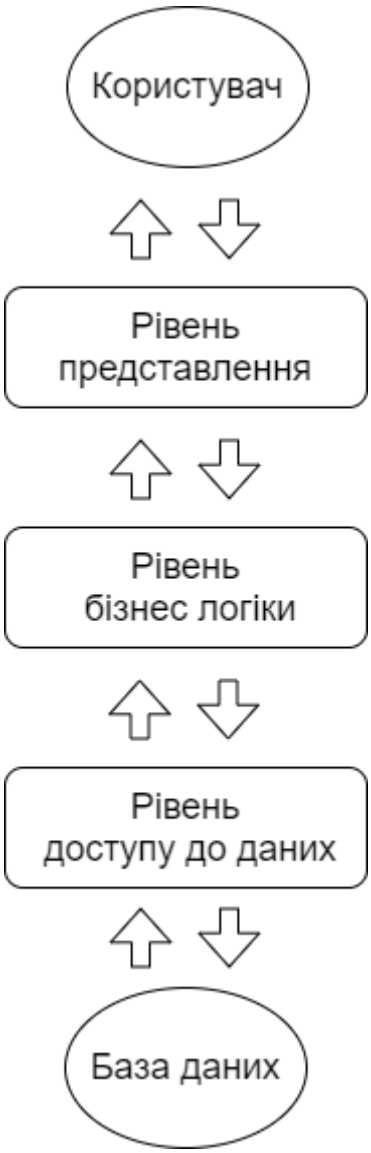


Рис. 2.2 – Схема багатошарової архітектури з 3-ма рівнями

На рисунку 2.2 схематично зображено ці рівні. Розглянемо їх більш детально:

1) Рівень представлення – це графічний інтерфейс, з яким взаємодіє користувач. Цей рівень переважно знаходиться на стороні клієнта, але для деяких програм він може знаходитися і на сервері. Також, він може дублюватися на сервері та на клієнті, так як API, що надається клієнту частково, також, є рівнем представлення.

2) Рівень бізнес логіки представляє собою основну програмну частину, яка знаходиться на сервері. На цьому рівні виконуються всі операції над даними перед їх збереженням за допомогою рівня доступу до даних, або їх відправленням на сторону клієнта. Також рівень бізнес логіки надає інтерфейс рівню представлення.
3) Рівень доступу до даних відповідає за читання та запис даних до бази даних або в файл. Може містити валідацію даних перед збереженням або початкову обробку для надання їх рівню бізнес логіки. Рівень доступу до даних, як правило, розміщується на сервері разом з рівнем бізнес логіки. Ніяким чином не повинен взаємодіяти з рівнем представлення.

Зобразимо яким чином ці рівні розміщені на клієнті та на сервері в даній роботі:

Рис. 2.3 – Схема розміщення рівнів абстракції на клієнті та сервері

Як бачимо рівень представлення дублюється на клієнті та на сервері. На клієнті рівень представлення являє собою графічний інтерфейс (англ. Graphical User Interface, GUI), а на сервері – прикладним програмним інтерфейсом (англ. Application Programming Interface, API).

Існують і інші варіації розміщення цих рівнів відносно платформ. Також їх може бути більше або менше, залежно від проекту.

2.1.3 Архітектурний принцип REST

REST (англ. REpresentational State Transfer) – архітектурний стиль/принцип/шаблон, що використовується для розробки багатьох видів систем. REST представляє собою низку вимог, дотримуючись яких, можна домогтись стабільності, надійності та високої швидкодії системи. Системи, які реалізують всі вимоги REST називаються RESTful. Цей архітектурний стиль був запропонований австрійським розробником Роєм Філдингом в його науковій дисертації. Розглянемо всі принципи, які він висовує в своїй роботі:

1) Модель клієнт-сервер. Ця вимога була розглянута в розділі 2.1.1. Вона потрібна для кращої масштабованості системи.
2) Відсутність стану. Сервер не повинен зберігати стан, тобто в різні моменти часу на один і той самий запит клієнт має отримувати одну і ту саму відповідь (якщо дані в базі даних не змінювались). Ця вимога потрібна для запобігання конфліктів на стороні клієнта та допомагає зберігати систему в консистентному (цілісному) стані.
3) Кешування. Кешування потрібно для значного збільшення швидкодії системи, адже клієнту не потрібно зайвий раз звертатися до сервера, так як він тримає дані «під рукою». Але кешування повинно бути явним, та з можливістю оновлення даних, що в ньому зберігаються в будь-який момент. Кешувати бажано дані, які якомога рідше змінюються. Таким чином можна досягти гарної швидкодії та стабільності роботи.

4) Єдиний інтерфейс. У клієнта та сервера має бути один інтерфейс, через який вони будуть «спілкуватися». Ця вимога має 4 принципи:
    - Ідентифікатор ресурсу. У REST, на відміну від SOAP (англ. Simple Object Access Protocol), фігурують не операції, а ресурси. Ресурсом є будь-яка сутність з іменем, наприклад користувач, замовлення, продукт, доставка і так далі. У кожного ресурсу має бути свій ідентифікатор доступу, який називається URI (англ. Uniform Resource Identifier). Приклад URI: */api/users/1* – користувач з id, що дорівнює 1.
    - Маніпуляції над ресурсами через представлення. Тобто ресурси можна змінювати через, так зване представлення. Представлення – це будь-яка форма, яка відображає стан певної сутності. Це може бути JSON, XML або HTML формат. Змінюючи представлення ми можемо змінити ресурс.
    - Повідомлення, що себе описує. Це означає, що на певний запит клієнта сервер повинен повертати повний об’єкт ресурсу, з яким можна маніпулювати. Тобто не можна зберігати якісь дані в проміжках між запитами. Клієнт і сервер повинні обмінюватись повідомленнями с повним корисним навантаженням.
    - HATEOAS (англ. hypermedia as the engine of application state). Це означає, що всі параметри повинні передаватись в посиланні разом з ідентифікатором ресурсу, а стан ресурсу повинен передаватися в тілі запиту.
5) Шари системи. Ця вимога аналогічна до багаторівневої архітектури. Вона означає, що система може бути розділена на декілька шарів (рівнів), які є абстракціями на іншими рівнями. Детально цей принцип був описаний в розділі 2.1.2.
6) Код на вимогу. Ця вимога не є обов’язковою, нею можна нехтувати за потребою. Вона означає, що клієнт може запитувати у сервера якийсь

виконуваний код для виконання його на стороні клієнта. Це може бути, наприклад, Java-аплети або JavaScript скрипти.

Для комунікації клієнта та сервера в REST, найчастіше, використовується протокол HTTP. Цей протокол використовує такі методи для вказання цілі запиту: GET, POST, PUT, DELETE, HEAD, PATCH, TRACE, CONNECT. Як правило, використовуються перші 4 методи, тому розглянемо їх більш детально:

1) GET. Цей тип запиту потрібен, щоб отримати від серверу певні дані. Дані запиту передаються в URL (англ. Uniform Resource Locator). Приклад: *GET /api/users?role=fitter* – отримати всіх користувачів з роллю fitter.
2) POST. Цей тип запиту потрібен, щоб передавати серверу об'єкти ресурсів для їх створення. Дані запиту передаються в тілі запиту, а не в URL. Приклад:

   POST /api/user

   {

   "id": 1,

   "name": "John",

   "role": "admin"

   }

   Цей запит передає на сервер користувача з id=1, іменем=John та роллю=fitter.
3) PUT. Використовується для зміни ресурсу. Методом POST також можна змінювати ресурси, але прийнято використовувати метод PUT для цих цілей. Приклад:

   PUT /api/user/1

   { "name": "John Gold",

   "role": "admin" }

   Цей запит змінить ім'я користувача з id = 1 з "John" на "John Gold".

4) DELETE. Використовується для видалення ресурсу. Приклад: DELETE */api/user/1* – видалить користувача з id = 1.

2.1.4 Мікросервісна архітектура

Мікросервісна архітектура знайшла популярність в розробці веб-додатків в 2010 році і з тих пір тільки нарощувала популярність. Компанії, які запровадили такий підхід в розробці їх систем в ті часи вже отримали досвід і, загалом, від виявився позитивним. Тому все більше і більше компаній починають використовувати цей стиль.

Мікросервісна архітектура – це стиль, в якому система розбивається на незалежні модулі (мікросервіси), які взаємодіють між собою для передачі даних. Ці сервіси існують незалежно один від одного і можуть бути написані на різних мова та технологіях, які більш підходять для того чи іншого сервісу, та використовувати різні бази даних. Детально розглянемо властивості мікросервісної архітектури:

1) Логіка всього додатку розбита на невеликі незалежні частини з чіткими рамками відповідальності;
2) Мікросервіси комунікують між собою, як правило, використовуючи протокол HTTP для передачі даних.
3) Мікросервіси можуть використовувати різні технології, залежно від своїх потреб. Один мікросервіс може бути написаний на мові Java, так як йому потрібно використовувати бібліотеки, які написані для Java, другий мікросервіс займається аналітикою даних, тому його вигідно написати на Python або Scala, третьому потрібно взаємодіяти з операційної системою або апаратним забезпеченням, тому його можна написати на Rust або C тощо.

4) Так як доменна область системи розділена на мікросервіси, тому для кожного мікросервіса можна використовувати різні бази даних, що гарно позначається на швидкодії та масштабованості.
5) Так як мікросервіси незалежні один від одного, значить різні мікросервіси можуть розроблювати різні команди з чіткими рамками відповідальності. Але потрібно узгодити між командами інтеграційну складову розробки.

Мікросервісна архітектура була створена в противагу класичній монолітній, коли вся система запущена в одному процесі, має одну базу даних та написана на одній мові програмування. Схематично зобразимо ці архітектури для кращого розуміння:

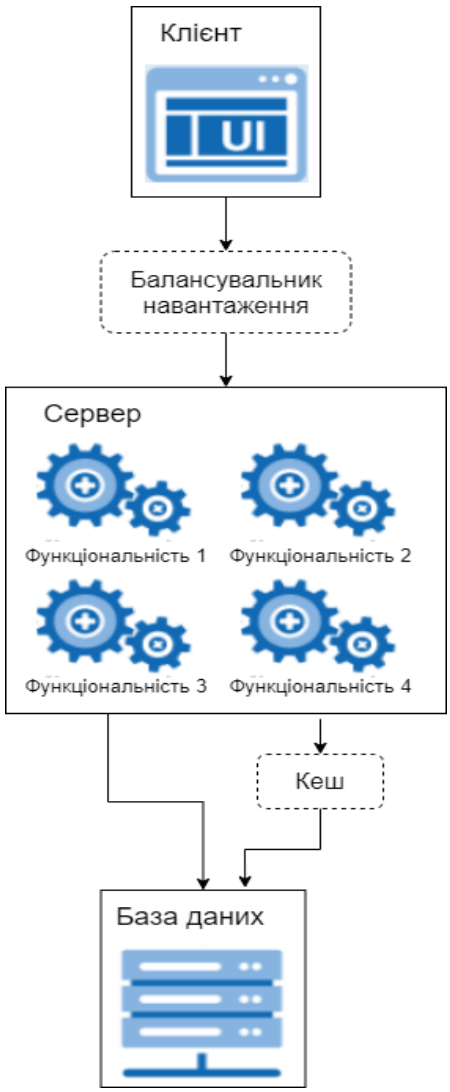


Рис. 2.4 – Модель монолітної архітектури

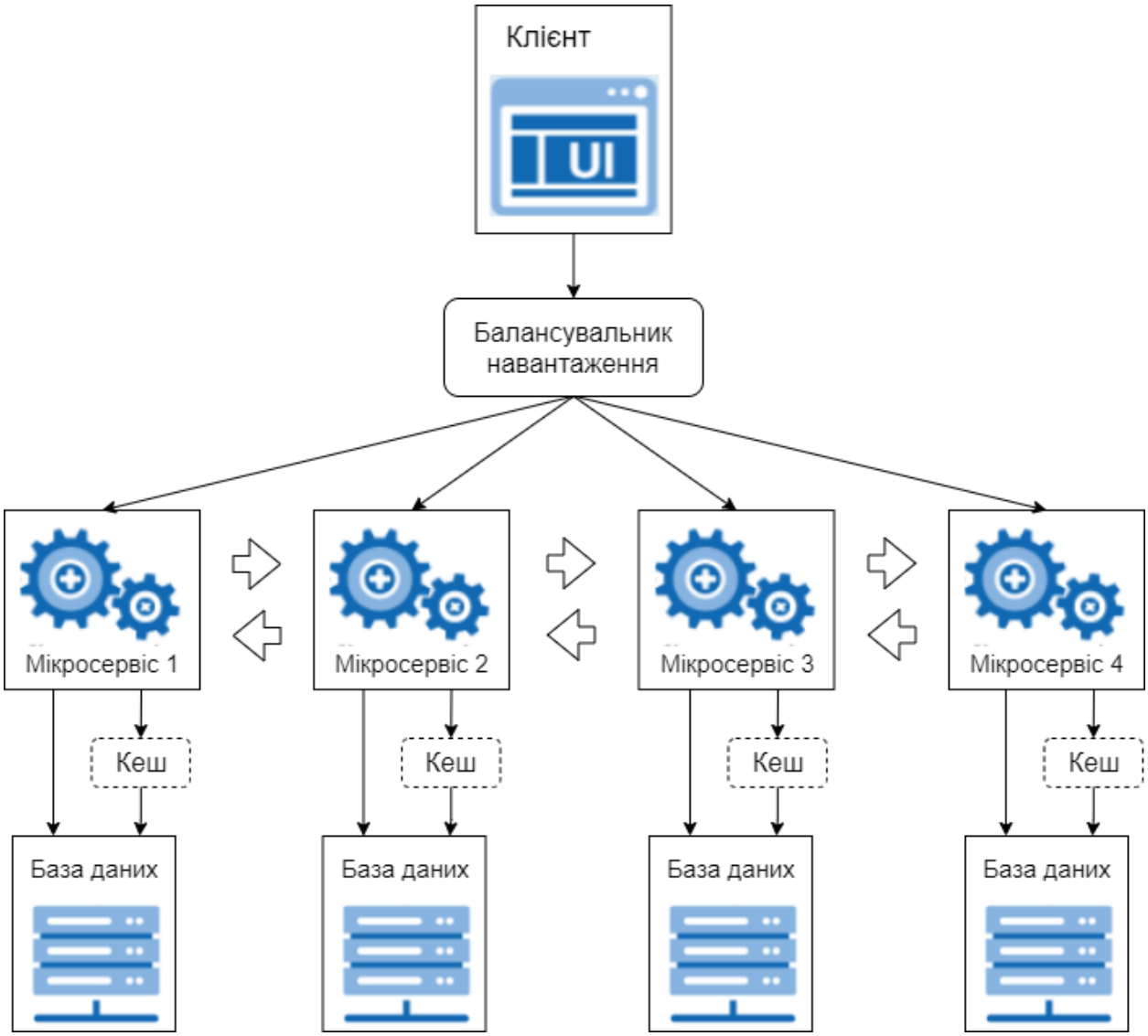


Рис. 2.5 – Модель мікросервісної архітектури

Довгий час монолітний підхід використовувався для багатьох систем і всіх він влаштовував. Але у нього є ряд недоліків, які частково або повністю вирішує мікросервісна архітектура. Розглянемо основні недоліки монолітної архітектури:

1) При розробці проекту більше 2 років, якщо з самого початку архітектура була вибрана не зовсім правильно або були допущені незначні помилки в її реалізації, то з часом вони виростуть в значні проблеми при розробці нового фунціоналу та вдосконаленні старого, так як проблеми будуть накоплюватись в прогресії подібно лавині. В такому випадку

довготривала підтримка проекту супроводжується постійним переписуванням старого коду та тестів, що значно зменшує швидкість розробки продукту. В сучасному світі такі затримки коштують дуже дорого.

2) Обмежена масштабованість. Монолітний додаток легко можна масштабувати, додаючи фізичні сервери, таким чином можна досягти більшої швидкодії всієї системи. Але якщо нам потрібно збільшити швидкодії конкретного функціоналу? Наприклад, пошуком користуються 2 млн. користувачів в день і 2 фізичних сервера не справляються з таким навантаженням, але функціоналом створення замовлень користуються тільки 50 тисяч користувачів в день і з таким навантаженням 2 фізичних сервери легко справляються. В такому випадку додавання нових серверів є неефективним, так як ми масштабуємо всю систему, а нам потрібно збільшити швидкодію тільки для пошуку. Виникає проблема неефективного використання ресурсів.
3) Відсутність ізоляції та чітких зон відповідальності в системі. Так як весь фунціонал знаходиться в одному процесі, зміна якоїсь частини може суттєво вплинути на інші, які зав’язані на змінювану частину. Це, в свою чергу, створює проблему, вказану в 1 пункті.
4) Використання одного стеку технологій для всієї системи. На початку створенням системи архітектори та розробники повинні узгодити стек технологій (мова програмування та фреймворк для серверної частини, мова програмування та фреймфорк/бібліотека для клієнтської частини база даних, суміжні бібліотеки для конкретних задач), який буде використовуватись протягом всієї розробки. Таким чином вибір стеку є компромісом, так як для певних задач може підійти інший/а фреймворк/мова програмування/СУБД, а для інших інший/а. Але монолітна архітектура змушує для всіх задач використовувати одні й ті самі технології.

5) При зміні невеликої частини в коді проекту потрібно перезбирати весь проект та заново завантажувати його на сервер, що на великих проектах може зайняти декілька годин.

Ці недоліки і вирішує мікросервісна архітектура. Але нічого ідеального не буває, тому і у мікросервісної архітектури достатньо недоліків:

1) Розробка мікросервісів на почтакових стадіях важча за моноліт. Перед розробкою потрібно досконало продумати архітектуру кожного мікросервісу та його комунікацію з іншими мікросервісами. Для такої роботи потрібні якісні спеціалісти.
2) Проблема з написанням тестів для всієї системи. Так як нам потрібно враховувати взаємодії мікросервісів між собою, тому деякі види тестування ускладнюється, наприклад, E2E (англ. End to End).
3) Зі збільшенням кількості мікросервісів збільшується складність їх підтримки та комунікації між ними.
4) Виникає необхідність у використанні балансувальника навантаження, що в деяких випадках може негативно позначитись на швидкодії.

Складемо таблицю для явного порівняння двох підходів:

Таблиця 2.1. Порівняння монолітної та мікросервісної архітектур

Архітектура / Характеристика	Монолітна архітектура	Мікросервісна архітектура
Складність розробки	Простіше на початку	Простіше після 1-2 років розробки
Можливість використання різних технологій для задач	-	+
Перезбирання та завантаження системи при змінах в проекті	Потребує більше часу	Потребує менше часу
Масштабованість	Тільки всієї системи	Окремих компонентів
Рівень підготовки розробників	Менш кваліфіковані	Більш кваліфіковані

Тестування	Простіше для E2E, складніше для модульного	Простіше для модульного, складніше для E2E
Зони відповідальності функціональності	Розмиті/неявні	Чіткі/явні

В висновку, можна сказати, що мікросервісна архітектура підійде для таких випадків: розроблювана система є велика та складна, мається на увазі, що розробка буде тривати довго (2+ років), навантаження на різні функціональності системи неоднорідне, розробка проходить динамічно, часто потрібно змінювати функціональність. Для інших випадків можна обійтись монолітної архітектурою.

2.2 Технології серверної частини

Детально розглянемо технології (мову програмування, фреймворки, СУБД тощо), що використовуються на серверній частині проекту.

2.2.1 Мова програмування Java

Java – високорівнева об’єктно-орієнтована мова програмування зі строгою статичною типізацією, розроблена компанією Sun Microsystems в 1995 році. Наразі права на ліцензію належать компанії Oracle. Основна ідея мови заключається в виконанні одного і того скомпільованого коду на будь-якій платформі. Така можливість називається кросплатформністю і реалізована вона за допомогою JVM (англ. Java Virtual Machine). JVM – віртуальна машина, яка виконує байт-код, який є результатом компіляції вихідного коду компілятором javac.

Java доволі швидко стало популярною та зберігає її і сьогодні. За версіями IEEE Spectrum та TIOBE Java займає 2 місце (після JavaScript) в рейтингу популярності серед мов програмування. Приваблює розробників вона надійністю, передбачуваністю та оберненою сумісністю (код, який був

написаний 20 років назад, може компілюватися та виконуватись на нових версіях мови). Останньою властивістю більшість сучасних мов не можуть похвалитися, але в неї є суттєвий недолік – мова наразі є застарілою та «багатослівною» в порівнянні з сучасними популярними мовами. Але завдяки цій властивості багато компаній вибирають Java як основну мову програмування. Як правило, на Java пишуть складні enterprise системи, тобто системи підтримки певного бізнесу, Android додатки, веб-додатки, IoT системи, ігри (Maincraft), також використовується для обробки великих об’ємів даних (Big Data), так як для цього для Java існують потужні інструменти – Apache Sparks та Hadoop.

Java використовує статичну типізацію, в якій кожна змінна має певний тип, який задається перед компіляцією. Також Java має сильну типізацію, яка не дозволяє змішувати різні типи даних, наприклад, конкатенувати число і строку. Java є об’єктно-орієнтованою мовою, тобто ключовим поняттям є об’єкт, який описується по певному шаблону – класу. Клас описує, які атрибути та властивості може мати об’єкт цього класу. Об’єктно-орієнтоване програмування має 3 принципи:

1) Інкапсуляція – приховування деталей реалізації та надання простого зрозумілого інтерфейсу.
2) Наслідування – можливість певної сутності запозичувати атрибути та властивості іншої сутності та розширювати їх.
3) Поліморфізм – можливість одного інтерфейсу мати різні реалізації або можливість функції приймати дані різних типів.

Ця мова була вибрана для даного проекту, так як розроблювану систему можна назвати enterprise системою, хоча набагато меншою та простішою за комерційні.

2.2.2 Фреймворк Spring

Spring Framework – найбільш популярний та універсальний фрейморк для Java, який створений для полегшення розробки складних систем на мові Java. Фреймворк був створений 2002 року Родом Джонсоном як заміна існуючої технології Enterprise Java Beans. Надає рішення для широкого спектру задач та є де-факто стандартом для сучасних enterprise систем, написаних на Java. Також приваблює розробників зрозуміла й детальна документація та відносна простота використання.

Spring Framework, фактично, є набором фреймворків, які вирішують різні задачі. Ці фреймворки є незалежними, тому розробники можуть використовувати лише ту функціональність, що їм потрібна. Детально розглянемо найбільш популярні компоненти та ті, що використовувались в даній роботі:

1) Inversion of Control контейнер. Ця компонента є ключовою функціональністю Spring фреймворку. Вона відповідає за автоматичну ініціалізацію залежностей модулів системи. Розглянемо детально, як працює інверсія контролю. Залежність – це модуль/клас, який використовує для своєї роботи інший модуль/клас, тобто залежить від першого. Але для функціонування програми потрібно ініціалізувати всі залежності. Це можна роботи в самому класі, таким чином сам клас контролює створення об’єктів своїх залежностей, але це поганий підхід, так як клас займається сторонньою роботою, окрім основної роботи, для якої він створювався. Краще передавати вже створені об’єкти залежностей через конструктор класу або через методи при створенні самого об’єкту класу. Таким чином ініціалізацією займається інший клас, тобто він контролює цей процес замість початкового. Це і є інверсією контролю. Але це все одно не дуже зручно, ми просто змінили місце створення об’єкту залежності, але ми все одно створюємо її

вручну. Для вирішення цієї задачі використовується механізм ін’єкції залежностей (англ. Dependency Injection). Це і є основна робота цієї компоненти. Цей механізм реалізований за допомогою ApplicationContext-у, в якому при запуску додатку створюються об’єкти всіх залежностей та автоматично «розсилаються» по класам, де вони використовуються. Створити об’єкт ApplicationContext-у можна різними способами: XML-конфігурація, Java анотації тощо.

2) Spring AOP або аспектно-орієнтоване програмування в Spring. Аспектно-орієнтоване програмування – парадигма, суть якої полягає в додаванні «наскрізної» функціональності. Гарним прикладом АОП є проксі. При створенні біна (залежності) Spring може створити об’єкт не самого класу, а проксі-клас, який має таку саму функціональність та розширення до неї. Це може знадобитися для розширення логіки створення/видалення об‘єкта або відкриття/закривання сесії роботи з базою даних.
3) Spring Boot. Ця компонента спрощує початок роботи з Spring. Так як Spring надає широкий функціонал, то в ньому легко заплутатись початківцям, тому розробники Spring створили компоненту, яка має всю необхідну функціональність для роботи з Spring. Spring Boot надає вбудований сервер (Tomcat або Jetty) для запуску додатку без необхідності розгортання JAR/WAR на сервері вручну, спрощує конфігурацію збірки додатку та не потребує XML конфігурації для роботи.
4) Spring Data. Spring Data надає інструменти з простим інтерфейсом для взаємодії з різними базами даних. Ця компонента дає змогу створювати класи-мапінгу для сутностей моделі бази даних, має реалізовані репозиторії для доступу до даних, використовуючи створені класи-мапінгу, має потужний механізм генерації запитів до бази даних, базуючись на назві методу та не потребує складної конфігурації для

підключення до бази даних. Для цього потрібно тільки прописати URL бази даних, логін та пароль і створити класи-мапінгу сутностей, після цього можна працювати з базою даних. Spring Data складається з багатьох підкомпонент, наприклад: Spring Data JPA, Spring Data MongoDB, Spring Data JDBC, Spring Data LDAP тощо. Детальніше розглянемо перші дві, так як вони використовувалися в роботі. Spring Data JPA є однією з реалізацій специфікації JPA (англ. Java Persistence API). JPA – специфікація, яка надає інтерфейс та стандарти для об'єктно-реляційного представлення сутностей бази даних в Java класи та роботи з ними. Spring Data JPA не повністю реалізує дану специфікації, а тільки доповнює вже існуючу, яка називається Hibernate. Spring Data MongoDB має схожий функціонал з Spring Data JPA, але призначений для доступу до нереляційної бази даних MongoDB, яка буде розглянута в розділі 2.2.3.2.

5) Spring MVC. MVC (англ. Model-View-Controller) – це архітектурний шаблон для розробки, переважно, веб-додатків, в якому присутні такі сутності – модель, вид, контролер. Тобто система має модель з даними, з якою працює контролер та змінює вид, використовуючи оброблені дані. Spring MVC є фреймворком для створення веб-додатків. Для Spring MVC моделлю є власне дані, видом – HTML, JSON, XML, а контролером – кастомні класи контролерів, які є розширенням Java сервлетів. Також в цій моделі присутній ключовий модуль – DispatcherServlet, який перехоплює всі запити від клієнта та оброблює їх для подальшого використання класами контролерами.

Рис. 2.6 – Схема роботи DispatcherServlet

На рисунку 2.6 зображено схему роботи DispatcherServlet. Після обробки запиту класом контролером відповідь відсилається спочатку DispatcherServlet-у, а потім клієнту.

6) Spring Security. Spring Security є фреймворком для реалізації аутентифікації та авторизації для Spring додатків. Spring Security підтримує багато видів аутентифікації та авторизації, найбільш популярним на даний момент є використання JWT (англ. JSON Web Token). JWT – специфікація для створення токенів в форматі JSON. Токен створюється сервером, заповнюється корисними даними, підписується та відправляється клієнту для подальшого використання. Схематично розглянемо, як працює аутентифікація з використанням JWT:

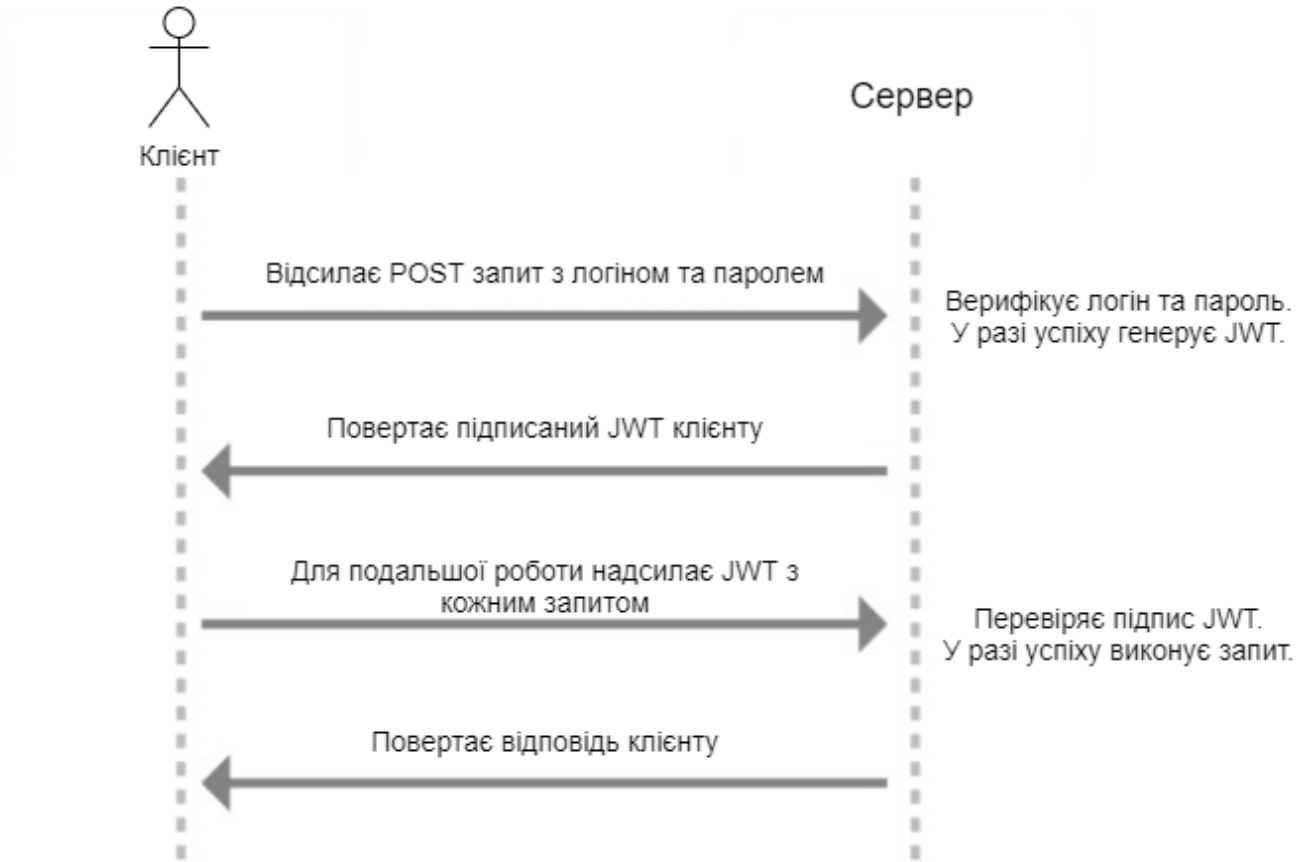


Рис. 2.7 – Схема аутентифікації з JWT

7) Spring Cloud. Spring Cloud включає в себе багато компонент, для розробки мікросервісних систем, містить в собі компоненти для інтеграції з Netflix OSS, яка також має інструменти для роботи з мікросервісами. Розглянемо детально компоненти з Spring Cloud та Netflix OSS, які були використані в даній роботі:

- Spring Cloud Netflix Eureka. Netflix Eureka або Eureka сервер реєструє кожний сервіс в системі та зберігає дані про нього. Також Eureka Server називають Discovery сервером. Опишемо алгоритм, за яким працює Eureka сервер:
  1) Кожен мікросервіс має ім’я, IP адресу та порт, на якому запущений. Під час запуску системи кожен мікросервіс відправляє запит на Eureka сервер з вищевказаною інформацією;
  2) Eureka сервер реєструє всі мікросервіси;
  3) Коли певному мікросервісу потрібно звернутися до іншого, він має тільки ім’я мікросервісу, тому він відправляє спочатку запит на Eureka сервер;
  4) Eureka сервер перевіряє в реєстрі, яку IP адресу має мікросервіс, з таким іменем і повертає знайдені IP адресу та порт;
  5) Мікросервіс, знаючи IP адресу та порт потрібного мікросервісу, може надіслати йому запит.

  Eureka сервер дуже легко конфігурується, тому його часто використовують при розробці мікросервісів.
- Spring Cloud Netflix Zuul або Zuul сервер. Zuul сервер є шлюзом для клієнта і мікросервісів, тобто виконує функцію маршрутизації, також може виконувати функції балансувальника навантаження. Клієнт замість того, щоб надсилати запити напряму мікросервісам, що не дуже зручно, адже потрібно знати IP адрес кожного мікросервісу, відсилає запит, в якому прописано ім’я мікросервісу,

на Zuul сервер, а він в свою чергу запитує у Eureka серверу, яку IP адресу має мікросервіс з таким іменем і надсилає запит вже мікросервісу. Таким чином клієнту достатньо знати IP адресу тільки Zuul серверу. Схематично ілюструємо процес маршрутизації:

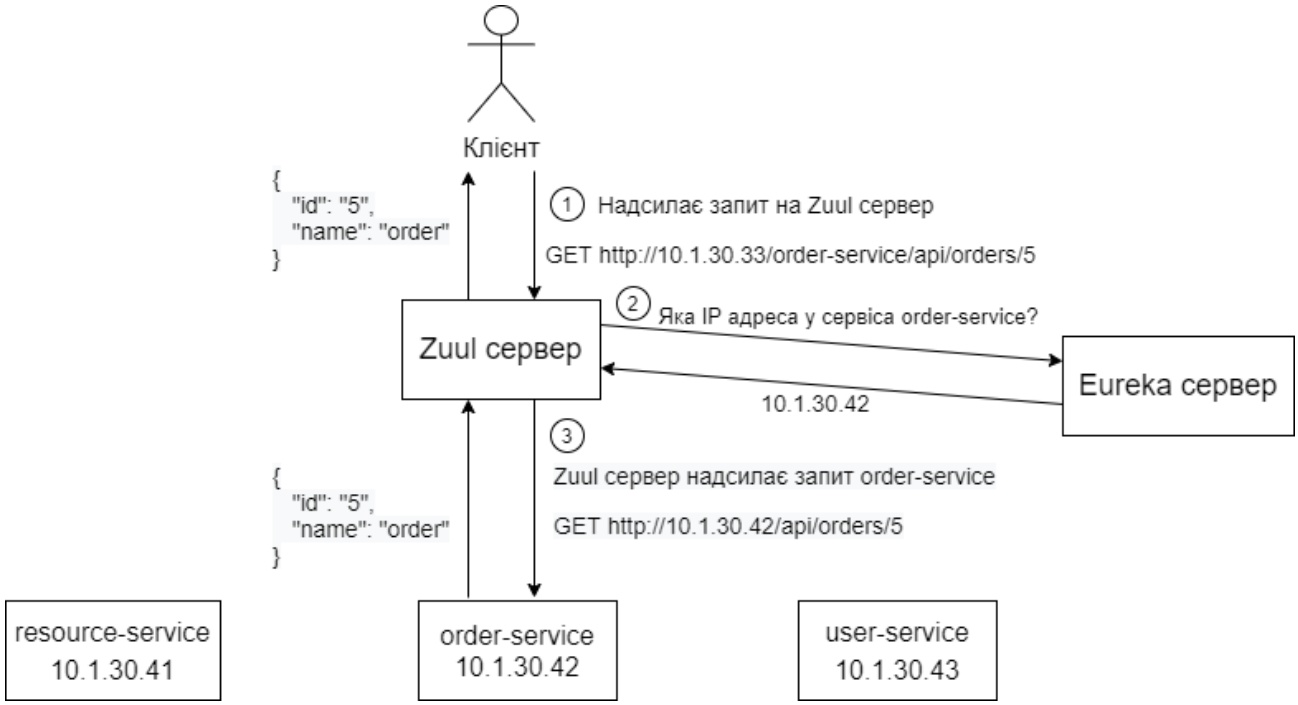


Рис.2.8 – Схема процесу маршрутизації Zuul сервера

Zuul сервер може обробляти запити, які надходять з клієнта і відповіді з мікросервісів, якщо вказати логіку обробки. Це реалізовано за допомогою ланцюгу фільтрів, що є реалізацією патерну Ланцюжок відповідальності, який буде детально розглянутий в розділі 2.2.6. Також Zuul сервер може виконувати функції балансувальника навантаження, адже через нього проходить весь трафік і в нього є доступ до всіх мікросервісів. В Spring даний функціонал міститься в компоненті Netflix Ribbon, що реалізує алгоритм планування Round Ribbon для рівномірного розподілення запитів на екземпляри мікросервісів.

- Spring Cloud Config. Так як мікросервіси – це окремі проекти, то і конфігурація для них буде зберігатися в різних файлах, що

відповідають конкретним мікросервісам. Але що робити, якщо деяка конфігурація спільна для декількох мікросервісів? Для вирішення цієї проблеми існує компонента Spring Cloud Config, яка зберігає файли конфігурації всіх мікросервісів на одному сервері, до якого звертаються мікросервіси для отримання своєї конфігурації. Таким чином ми централізуємо зберігання конфігурації, що спрощує підтримку системи.

- Spring Cloud OpenFeign. Ця компонента спрощує взаємодію між мікросервісами. Spring Cloud OpenFeign представляє собою простий REST Client, який для звернення до іншого мікросервіса потребує тільки оголошення інтерфейсу з методами.

Spring Cloud міть дуже багато корисних компонент і є потужним інструментом для розробки мікросервісів.

2.2.3 Бази даних

Практично будь-яка комерційна система має базу даних для збереження даних користувачів. На даний момент існує доволі багато СУБД. Загалом, бази даних поділяються на реляційні та нереляційні.

Реляційна база даних – база даних, в якій дані, зберігаються в структурованих таблицях, в яких колонки – поля/атрибути сутності, яка зберігається в таблиці, а рядки – конкретні записи; таблиці можуть бути зв’язані між собою реляційними зв’язками, які можуть бути представленими реляційною моделлю. Реляційні бази даних мають структуровану мову запитів SQL (англ. Structured Query Language), за допомогою якої можна створювати, модифікувати, видаляти та читати дані. Прикладами реляційних СУБД є Oracle, PostgreSQL, MySQL, Microsoft SQL Server.

Нереляційна база даних – база даних, яка зберігає дані в неструктурованих документах в певному форматі, наприклад, JSON або в форматі <ключ>: <значення> тощо. Нереляційні бази даних також називають NoSQL (англ. Not

only SQL). Прикладами нереляційних СУБД є MongoDB, Redis, Apache Cassandra, Neo4j.

В даній роботі використовуються як реляційна СУБД, так і нереляційна. Для реляційної СУБД було вибрано PostgreSQL, так як це популярна СУБД з графічним інтерфейсом, високою швидкість та відмовостійкістю. Для нереляційної СУБД було обрано MongoDB, так як це популярна СУБД з графічним інтерфейсом та документованим форматом збереження даних, що підходить під потреби проекту.

2.2.4 Брокер повідомлень RabbitMQ

Брокер повідомлень – програмний модуль, що пересилає повідомлення між компонентами системи по певному протоколу. Часто використовується для відкладення ресурсоємних задач, наприклад, в ситуаціях, коли потрібно повернути відповідь клієнту і запустити певну часозатратну задачу, тому, щоб не чекати, коли задача закінчить виконання, ми відправляємо повідомлення про початок виконання задачі і повертаємо відповідь клієнту. Для брокерів повідомлень існують наступні ключові терміни:

- Відправник (англ. Producer) – модуль, що відправляє повідомлення в брокер.
- Черга (англ. Queue) – «почтовий ящик», структура, в якій зберігаються повідомлення до відправки отримувачу.
- Отримувач (англ. Consumer) – модуль, що приймає повідомлення від брокеру.

Брокер повідомлень RabbitMQ для пересилки повідомлень використовує протокол AMQP (англ. Advanced Message Queuing Protocol). Зобразимо схематично роботу брокера повідомлень:

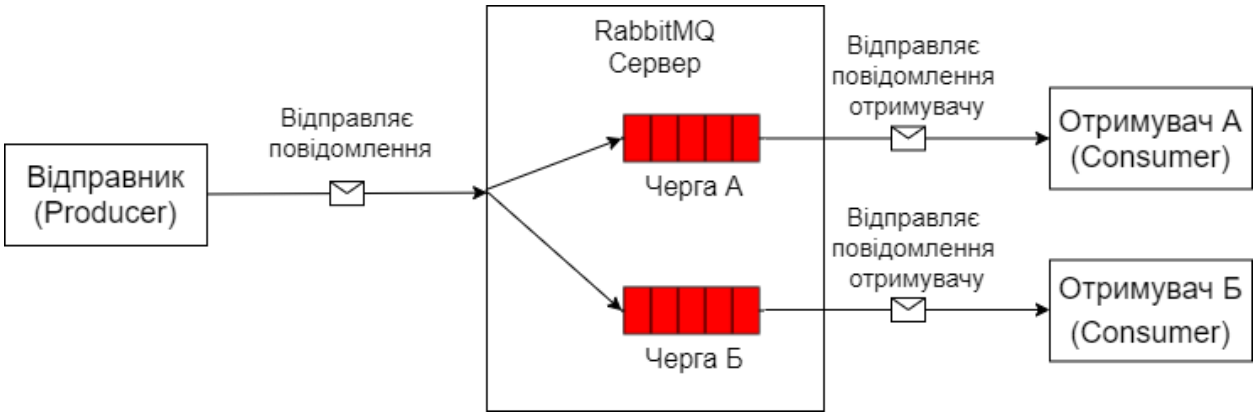


Рис. 2.9 – Схема роботи брокера повідомлень RabbitMQ

Також існує брокер повідомлень Apache Kafka, який є доволі потужним інструментом і використовується для високонавантажених систем, так як може забезпечити більшу відмовостійкість та витримати більше навантаження ніж RabbitMQ. Але разом з цим, Apache Kafka важче конфігурувати для коректної роботи, і так як RabbitMQ є більш простим інструментом, який покриває всі задачі для даної роботи, то був вибраний саме він.

2.2.5 Засіб автоматизації збірки проекту Apache Maven

Apache Maven – це засіб автоматизації збірки проекту та пакетний менеджер, що базується на XML конфігурації та використовується для систем, написаних на мовах JVM платформи. Maven був створений як заміни існуючого інструменту Apache Ant, який був доволі громіздким та незручним в використанні. Для всієї конфігурації в Maven використовується файл pom.xml, в якому прописуються модулі проекту, залежності, які використовуються в проекті, конфігурація окремих команд збірки проекту, використані плагіни тощо.

Розглянемо основні концепції Maven:

- Архетип – базова структура проекту. Майже всі проекти з використанням Maven мають структуру, зображену на рисунку 2.10.

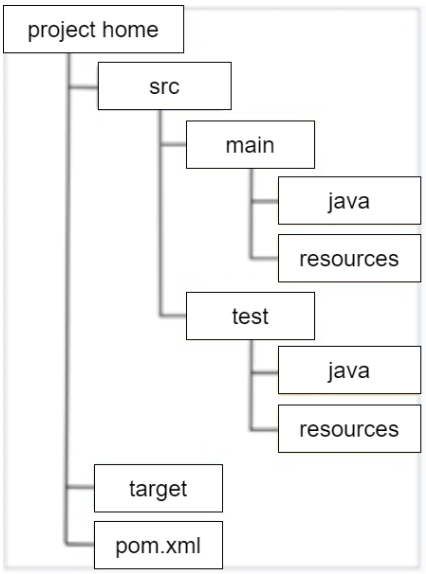


Рис. 2.10 – Базова структура проекту Maven

- Залежності – будь-яка бібліотека або фреймворк, що використовується в проекті. Залежності прописуються в файлі pom.xml під тегом <dependencies>. Залежності містяться в репозиторіях. В Maven існуть 3 види репозиторіїв:
    1. Центральний репозиторій Maven – «офіційний» репозиторій Maven з величезної кількістю публічних бібліотек та фреймворків.
    2. Локальний репозиторій – директорія, в якій зберігаються завантажені залежності проекту.
    3. Віддалений репозиторій використовується, коли для проекту потрібна приватна залежність, наприклад, в середині однієї компанії є компонента, яка розроблялась для багатьох проектів, тому вона поміщається в певний закритий репозиторій компанії та вказується в pom.xml.
- Життєвий цикл проекту – набір фаз, які містять команди, що виконуються під час збірки проекту. Розглянемо основні фази та команди:
    1. clean потрібен для очищення target директорії проекту, в якій зберігаються скомпільовані класи та виконувані JAR.

2. default основний життєвий цикл, що має такі задачі:
   validate – перевірка структури проекта.
   compile – компіляція вихідного коду.
   test compile та test – компіляція вихідного коду тестів та їх запуск.
   package – упаковка скомпільованих файлів в JAR або WAR.
   install – установка виконуваного архіву в локальний репозиторій.
   deploy – копіювання виконуваного архіву в віддалений або центральний репозиторій.
3. Site – генерація проектної документації.

Apache Maven був вибраний як засіб автоматизації збирання проекту, так як є найбільш популярним та являється де-факто стандартом для enterprise систем.

2.3 Технології клієнтської частини

Детально розглянемо технології (мови та фреймворк), що використовуються на клієнтській частині.

2.3.1 Мова програмування TypeScript

TypeScript – мова програмування з явною строгою статичною типізацією, розроблена Microsoft 2012 року як альтернатива JavaScript для розробки веб-додатків. TypeScript не є повноцінною мовою програмування, в тому сенсі, що вона є розширенням мови JavaScript і компілюється саме в JavaScript код. Основною проблемою JavaScript є неявна слабка динамічна типізація, яка не подобається багатьом розробникам, так як легко можна передати в функцію параметр з непідтримуваним типом і помилка з’явиться тільки під час роботи програми, а не під час компіляції. Також код написаний на мові з динамічною

типізацією важче підтримувати. Саме це і стало причиною створення TypeScript, яка вирішує ці проблеми. І так як TypeScript – це всього надбудова над JavaScript, то ми маємо всі переваги JavaScript, а саме – велика кількість бібліотек, що написані для JavaScript можна використовувати і в TypeScript. Також в TypeScript реалізовано ООП, яке в JavaScript реалізовано частково.

Наведемо порівняльну таблицю для наглядності.

Таблиця 2.2. Порівняння TypeScript та JavaScript

Мова / Характеристика	TypeScript	JavaScript
Типізація	Явна, строга, статична	Неявна, слабка, динамічна
Підтримка ООП	Реалізовані всі 3 принципи	Частково реалізоване
Складність вивчення	Вимагає більше часу	Легкий для вивчення
Модульність	Підтримується	Не підтримується
Необхідність компіляції	Є	Немає
Спільнота розробників	Невелика, мова тільки розвивається	Величезна, мова дуже популярна

У висновку можна сказати, що мови мають трохи різне призначення. JavaScript більше підходить для написання простеньких скриптів для веб-сторінок, TypeScript краще підійде у випадках, коли потрібно використовувати ООП та розбивати програму на модулі. Була обрана мова TypeScript, так як це мова за замовчування для фреймворку Angular, також типізація і ООП суттєво спрощують розробку складних клієнтських додатків.

2.3.2 Мова розмітки гіпертексту HTML

HTML (англ. HyperText Markup Language) – мова розмітки гіпертексту для веб-сторінок, що підтримується Консорціумом Всесвітньої Павутини. Файли HTML відображаються в браузері на веб-сторінці у певному вигляді, який

залежить, які теги були використані для тексту. HTML тег – це структурний елемент, який певним чином перетворює зміст тегу. Також тег має атрибути, які можуть змінювати поведінку тегу. Приклад тегу - Курсив. Даний тег відображає в браузері текст у вигляді курсиву. Документи HTML, як правило, використовуються разом зі стилями CSS.

2.3.3 Мова стилю CSS

CSS (англ. Cascading Style Sheets) – мова описання зовнішнього виду веб-сторінки, що використовується як доповнення до HTML документів. CSS дозволяє змінювати колір тексту, стилі шрифтів, форму кнопок, розміщувати елементи на екрані, зсувати елементи і багато іншого. Для застосування стилю в HTML документі достатньо вказати шлях до файлу зі стилями в тезі <style>

2.3.4 Веб-фреймворк Angular

Angular – фреймворк для створення веб-додатків, розроблений компанією Google. Angular написаний на мові TypeScript та використовує TypeScript як основну мову, але, за бажанням, можна використовувати JavaScript.

На Angular, як правило, розроблюють односторінкові додатки, які називаються SPA (англ. Single Page Application). Основним будівельним блоком в Angular є компонент. Компонент представляє собою директорію з 4 файлами: HTML документ, CSS стиль, TypeScript файл, файл з тестами для даного компоненту. HTML документ та CSS стиль є представленням, яке змінює код в TypeScript файл. Один компонент може описувати цілу сторінку, або тільки певну функціональність на ній. Наприклад, сторінка може мати меню, заголовок та основну частину – і це все буде окремі компоненти. Таким чином програма чітко розбита на незалежні модулі, які легко розроблювати паралельно, компонувати між собою та довго підтримувати.

Також Angular представляє багато бібліотек для роботи. Наприклад, для маршрутизації по сторінкам, для комунікації з клієнтом, для підтримки сесії авторизації, для управління формами і багато інших корисних інструментів «з коробки». Це дуже добре, так як не потрібно придумувати свої велосипеди або використовувати сторонні розробки, які можуть бути несумісні з собою. Також, так як Angular надає певний шаблон/структуру для проекту. З цих причин на Angular легко можна розроблювати великі додатки для enterprise систем і довго їх підтримувати, так як розробка йде «по шаблону» і сторонні бібліотеки використовуються по мінімуму. Також це корисно початківцям, які можуть йти по «протоптаній топинці» і не спіткнутись, як це можна інших бібліотеках та фреймворках.

Для розробки веб-додатків також використовуються бібліотека React та фреймворк Vue. Детальніше розглянемо дані альтернативи:

React доволі популярна бібліотека, яка є доволі гнучкою для розробки веб-додатків, це є одночасно як сильною стороною, так і недоліком, так як початківцю легко можна заплутатись в безлічі плагінів та додатків, несумісність яких може зламати всю систему. Також на великих проектах, гнучкість може більше заважати ніж допомагати, так як легко можна заплутатись. Тому React потрібно використовувати бажано досвідченими розробниками на невеликих або середніх проектах.

Vue є наймолодшим фреймворком з даної трійки, тому є найменш популярним, але так як він доволі простий в освоєнні, є не таким об’ємним як Angular, то доволі швидко набирає популярність. Він є гарною альтернативою Angular та React, так як є чимось середнім і має гарну документацію, що пришвидшує вивчення фреймворку та написання додатків.

Для даної роботи був вибраний саме Angular, так як він надає багато інструментів «з коробки» і шаблон, за яким можна легко розроблювати чітко структуровану систему.

ВИСНОВКИ ДО РОЗДІЛУ 2

Існує дуже багато технологій та архітектур, які використовуються при розробці систем і важливо їх знати, щоб вдало підібрати стек технологій та архітектури для конкретної системи, щоб розробка йшла легко і потім можна було так само підтримувати систему. Саме знання існуючих технологій та вдалий їх вибір відрізняє професіонала від новачка. І так як технологій та інструментів з кожним днем стає все більше і більше, тому все важче бути в курсі всіх останніх змін. Таким чином роль архітектора стає все більш і більш важливою на проектах.

Був проведений досконалий аналіз технологій та архітектур і було вирішено використати вищеописані технології для даної роботи. Даний стек технологій часто використовується для систем автоматизації процесів для телекомунікаційних систем, тобто BSS/OSS, тому такий вибір полегшить розробку та підтримку проекту в майбутньому.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Загальна структура системи

Схематично зобразимо структуру проекту зі всіма ключовими елементами:

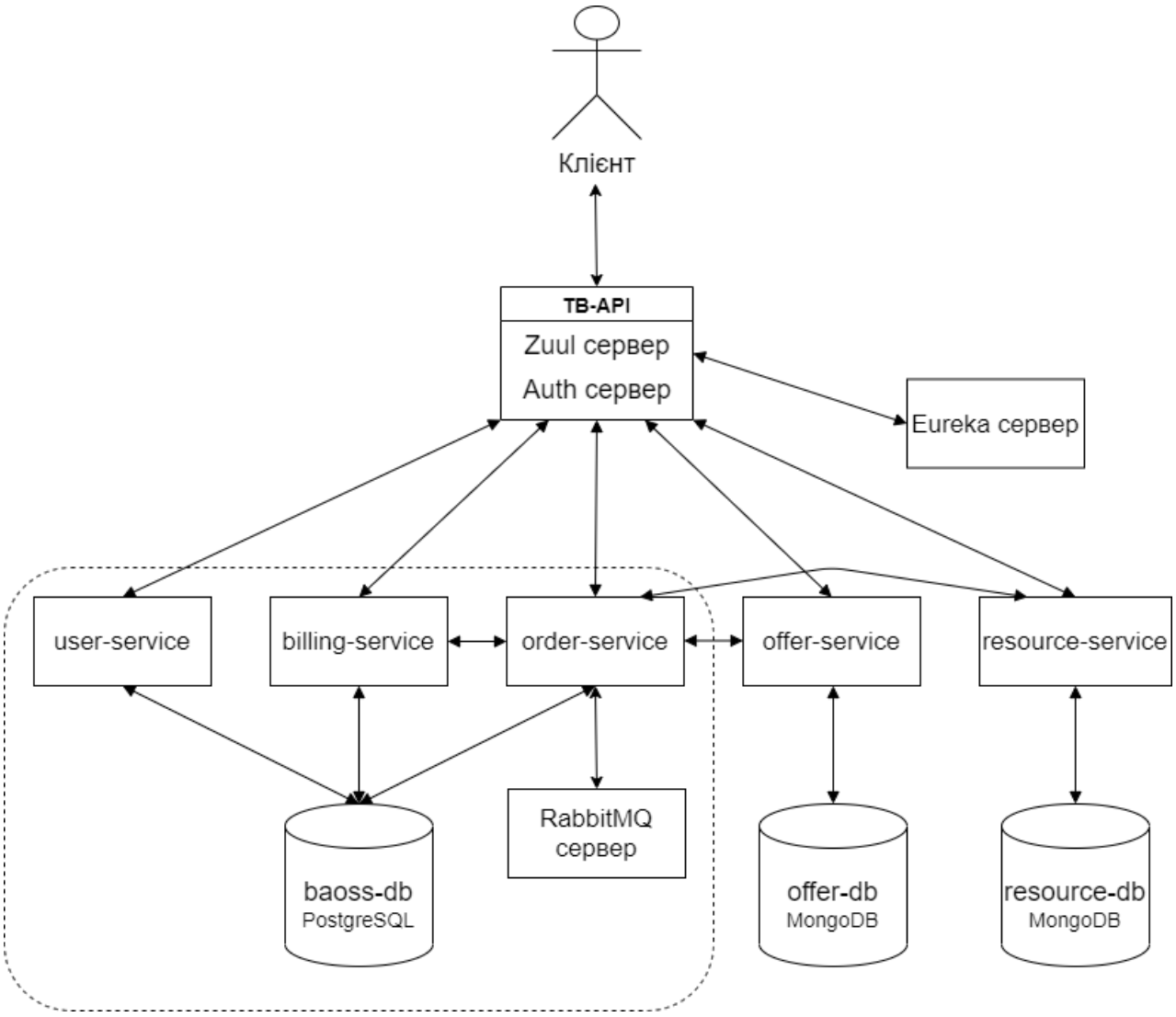


Рис. 3.1 – Схема структури проекту

На схемі бачимо клієнта, що відсилає запити на TB-API (Telecom Business API) сервер, що представляє собою шлюз до мікросервісів. TB-API сервер є реалізацією Zuul сервер, тобто займається маршрутизацією запитів. Також виконує роль сервера аутентифікації та авторизації, так як через нього проходять всі запити та в нього є доступ до всіх мікросервісів. Тому було

вирішено скомбінувати Zuul сервер та сервер авторизації. За IP адресами мікросервісів TB-API сервер звертається до Eureka сервера, в якому зареєстровані всі мікросервіси.

Система складається з 5 мікросервісів, які будуть детально розглянуті в розділі 3.3. Мікросервіси user-service, order-service, billing-service мають спільну базу даних (яка підтримується СУБД PostgreSQL) так як мікросервіси user-service, billing-service використовують всього 2 таблиці, дані з яких використовує мікросервіс order-service, тому було вирішено об'єднати бази даних для цих мікросервісів. Мікросервіси offer-service, resource-service використовують відповідні роздільні NoSQL бази даних, які підтримуються СУБД MongoDB. Мікросервіс order-service звертається до сервера RabbitMQ для ініціалізації потоку виконання замовлення та його виконання.

3.2 Моделі баз даних

Система використовує 3 різні бази даних: baoss-db, resource-db, offer-db. База даних baoss-db підтримується СУБД PostgreSQL, інші – MongoDB. Спершу розглянемо ER (Entity-Relationship) діаграму для реляційної бази даних baoss-db.

3.2.1 База даних baoss-db

Діаграма сутність-зв'язок показує всі сутності, що фігурують в системі та їх взаємодію між собою (зв'язки). ER діаграма для baoss-db зображена на рисунку 3.2. Детально опишемо таблиці:

- usr – таблиця з даними користувача. Є ключовою сутністю системи, має такі поля: ім'я, прізвище, логін, пароль, пошта, роль та поле min_refresh_date, яке потрібне для того, щоб перевіряти та підтримувати авторизацію користувача в системі. Таблиця названа usr, а не user, так як user є ключовим словом PostgreSQL.

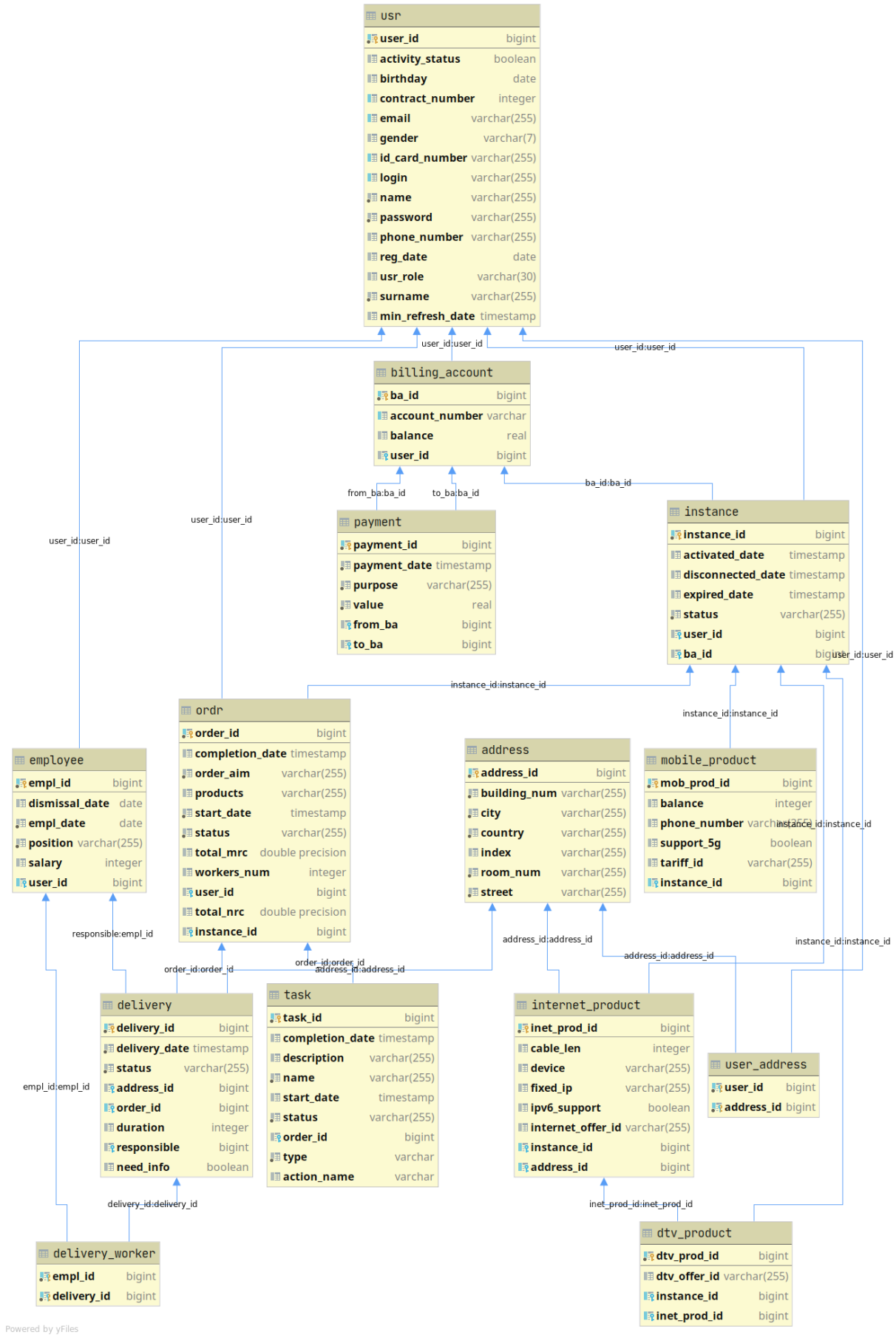


Рис. 3.2 – ER діаграма бази даних baoss-db

- billing_account – таблиця білінг акаунтів, тобто рахунків, з яких проходить оплата послуг. Має поля номеру рахунку, поточного балансу та зовнішнього ключа на таблицю usr, так як користувач може мати декілька білінг акаунтів, але білінг акаунт належить тільки одному користувачу, тобто реалізується зв'язок «багато до одного» (англ. Many to One).
- payment – таблиця з транзакціями між рахунками. Містить поля суми переказу, дати проведення транзакції та зовнішні ключі на таблицю billing_account, тобто білінг акаунт з якого і на який були переведені кошти.
- instance – таблиця екземплярів продуктів користувачів. Тобто, коли користувачу «проводиться інтернет» або користувач купляє SIM карту, то створюється екземпляр придбаного продукту, який є активним, поки користувач не відмовиться від послуг. Має поля дат активації/деактивації, статусу екземпляру (активний, неактивний тощо), зовнішній ключ на таблицю usr, тобто користувач може мати декілька екземплярів, а екземпляр може належати тільки одному користувачу та зовнішній ключ на таблицю billing_account, тобто білінг акаунт, з якого будуть зніматися щомісячно кошти за надання послуг.
- ordr – таблиця замовлень. Створене користувачем замовлення забезпечує потік активації екземпляру продукту. Містить такі поля: дата виконання, мета замовлення (нове замовлення, зміна активного продукту тощо), продукти, які активуються даним замовленням (Mobile, Internet, DTV), дата створення замовлення, статус замовлення, сума, яку користувач має сплатити відразу після активації продуктів, зовнішній ключ на таблицю usr, тобто користувач може мати декілька замовлень, а замовлення може належати тільки одному користувачу, зовнішній ключ на таблицю

instance, тобто на конкретний екземпляр, який активується даним замовленням, одне замовлення активує тільки один екземпляр, але екземпляр може змінюватись декількома замовленнями. Таблиця названа ordr, а не order, так як order є ключовим словом PostgreSQL.

- task – таблиця задач (етапів) замовлення. Кожне замовлення містить ряд обов'язкових та необов'язкових задач, які є етапами активації замовлення. Таблиця має такі поля: коротка назва задачі (Ініціалізація замовлення, Очікування підтвердження активації тощо), описання задачі, дата початку виконання, дата закінчення виконання, статус задачі, тип задачі (автоматична задача, «задача-очікувач» (listener) тощо), назва классу, що викликається для виконання задачі, зовнішній ключ на таблицю ordr для зв’язку Багато до Одного (Many to One).
- address – таблиця адресів користувача. Містить поля країни, міста, назви вулиці, номеру дому, номеру квартири та індексу.
- user_address – таблиця для підтримання зв’яку Many to Many для таблиць usr та address, так як один користувач може мати декілька будинків і в одному будинку може проживати декілька людей. Містить тільки зовнішні ключі на відповідні таблиці.
- mobile_product – таблиця, що зберігає дані для екземпляру мобільного продукту певного користувача. Має такі поля: номер телефону, флажок для підтримки 5G, баланс рахунку карти, ID тарифу, дані якого зберігаютья в offer-db, зовнішній ключ на таблицю instance.
- internet_product – таблиця, що зберігає дані для екземпляру інтернет продукту певного користувача. Має такі поля: довжина кабелю, яка була придбана користувачем, фіксована IP адреса або біла адреса, якщо користувач придбав таку послугу, ID роутера, якщо такий був придбаний (дані всіх пристроїв зберігаються в resource-db), ID

пропозиції швидкості інтернету, яка була придбана, що зберігається в offer-db, зовнішній ключ на таблицю instance, зовнішній ключ на таблицю address, тобто в якому саме будинку був проведений Інтернет.

- dtv_product – таблиця, що зберігає дані для екземпляру DTV продукту певного користувача. Має такі поля: ID пропозиції кількості каналів, яка була придбана, що зберігається в offer-db, зовнішній ключ на таблицю instance та зовнішній ключ на таблицю internet_product, так як DTV продукт не може існувати без активованого Інтернет продукту.
- employee – таблиця даних співробітників компанії. Має такі поля: дата трудовлаштування, дата звільнення, посада, заробітна плата співробітника та зовнішній ключ на таблицю usr, так як сутність Employee наслідує сутність User.
- delivery – таблиця доставок замовлень. Дана таблиця зберігає інформацію про доставки замовлень, тобто в який час та який продукт потрібно доставити та активувати за певною адресою. Має такі поля: дата закінчення доставки, статус, час виконання доставки, зовнішній ключ на таблицю ordr з унікальним обмеженням, так як у одного замовлення може бути одна доставка і навпаки, зовнішній ключ на адресу та зовнійшній ключ на таблицю employee, тобто відповідальний співробітник, який повинен підтвердити закінчення доставки/підключення послуги.
- delivery_worker – таблиця для підтримення зв’язку Many to Many між таблицями delivery та employee, так як підключенням послуг інтернету та DTV займаються декілька монтажників, а в одного співробітника може бути багато доставок.

Таблиці usr, user_address, address використовує переважно user-service, таблиці billing_account, payment використовує тільки billing-service. В даному розділі було представлено реалізацію реляційної бази даних baoss-db.

3.2.2 База даних offer-db

Розглянемо реалізацію нереляційної бази даних offer-db, що зберігає дані пропозицій продуктів, їх ціни та дисконти/акції на пропозиції. Ця база даних використовується тільки offer-service-ом. Хоча база даних є нереляційною і дані зберігаються не у вигляді таблиць, але можемо виділити спільні поля документів, що зберігаються в колекціях MongoDB.

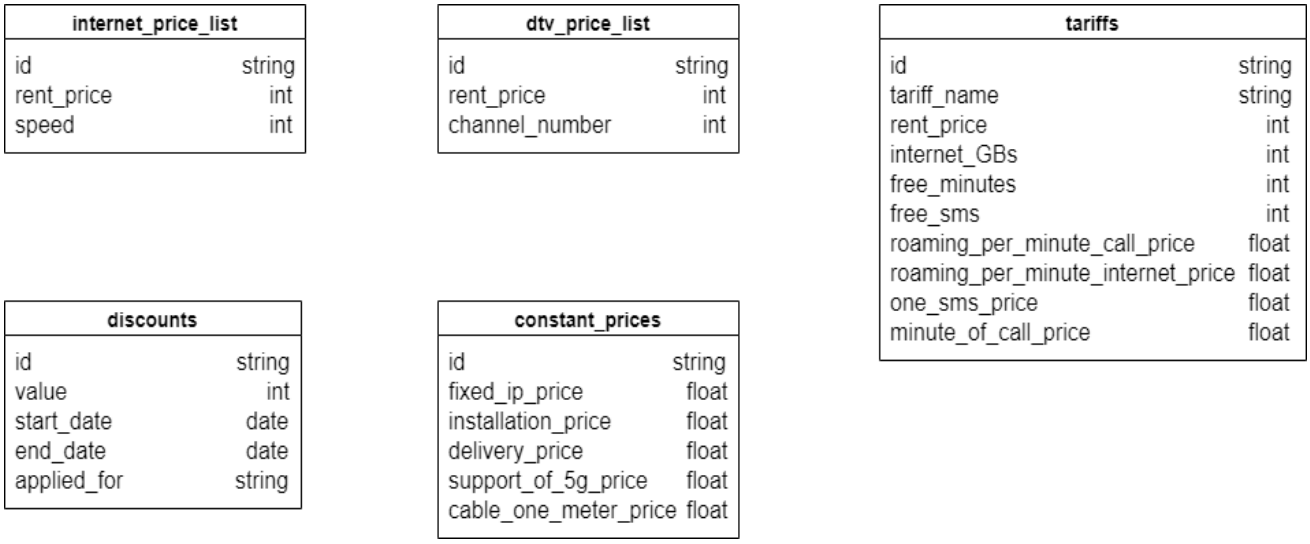


Рис. 3.3 – Схема колекцій бази даних offer-db

Детально опишемо кожну з колекцій:

- internet_price_list – колекція, що зберігає пропозиції інтернет продукту. Має такі поля: сума щомісячного платежу та швидкість інтернету.
- dtv_price_list – колекція, що зберігає пропозиції DTV продукту. Має такі поля: сума щомісячного платежу та кількість доступних каналів.
- tariffs – колекція, що зберігає тарифи. Має такі поля: назва тарифу, сума щомісячного платежу, кількість вільних Гігабайтів інтернету, кількість вільних хвилин тощо.

- discounts – колекція, що зберігає дисконти та акції. Має такі поля: знижка, тобто відсоток від суми продукту, дата початку дії знижки, дата кінця дії знижки та список (у вигляді строки) назв продуктів для яких діє знижка.
- constant_prices – колекція, що зберігає ціни на конкретні послуги, такі як доставка, підключення інтернету/DTV, ціна фіксованої IP адреси, ціна за 1 метр кабелю, ціна підтримки 5G.

3.2.3 База даних resource-db

Розглянемо реалізацію нереляційної бази даних resource-db, що зберігає дані всіх ресурсів компанії, тобто кабелі, пристрої (комутатори, маршрутизатори), SIM карти, список зарезервованих IP адрес, підключені до мережі будинки. Ця база даних використовується тільки resource-service-ом. Хоча база даних є нереляційною і дані зберігаються не у вигляді таблиць, але можемо виділити спільні поля документів, що зберігаються в колекціях MongoDB.

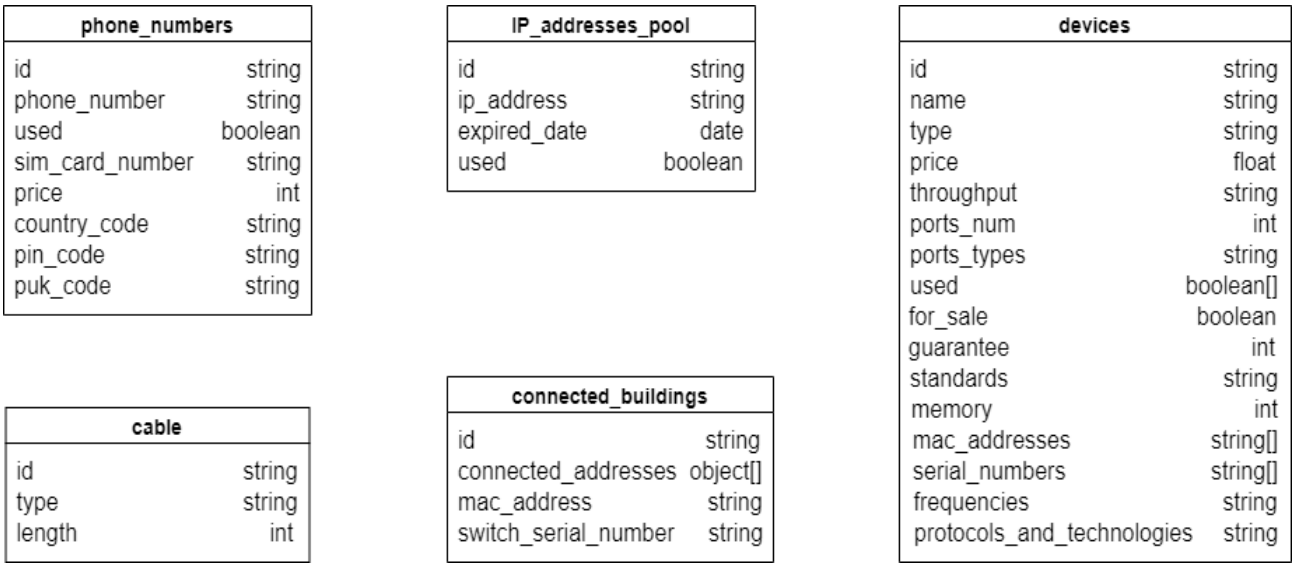


Рис. 3.4 – Схема колекцій бази даних resource-db

Детально опишемо кожну з колекцій:

- phone_numbers – колекція, що зберігає дані про SIM карти та телефонні номери. Містить такі поля: телефонний номер, який є унікальним в колекції, номер SIM карти, що теж є унікальним ідентифікатором, ціну телефонного номеру, наприклад, номер 0955557777 буде дорожче за простий набір цифр, код країни та флажок використання даної SIM карти.
- IP_addresses_pool – колекція, що зберігає зарезервовані провайдером IP адреси. Має такі поля: власне IP адреса, флажок використання певним користувачем, дата закінчення резерву IP адреси.
- devices – колекція, що містить повну інформацію про пристрої провайдера, тобто комутатори та марушрутизатори. Містить такі поля: назва пристрою, тип пристрою (маршрутизатор, комутатор), ціна пристрою, всі технічні характеристики, флажок, що вказує чи доступний даний пристрій для придбання користувачем, список флажків використання, MAC адрес та серійних номерів, що вказують конкретні екземпляри пристроїв - така реалізація дозволяє уникнути використання посилань на інші колекції, що не бажано для нереляційних баз даних, хоча більшість підтримує таку функцію.
- cable – колекція, що зберігає інформацію про кабелі, які використовує провайдер. Містить поля довжини кабелю та типу кабелю (вита пара, коаксіальний, оптоволоконний тощо).
- connected_buildings – колекція, що зберігає інформації про будинки, які підключені до мережі провайдера. Має такі поля: список підключених адрес, що предсавляє собою список об’єктів з полями MAC адреси маршрутизатора, який підключений до комутатора будинку та ID адреси користувача, який підкючений за цією адресою, MAC адресу комутатора, що встановлений в будинку та його серійний номер.

3.3 Реалізація серверної частини

Серверна частина розбита на 5 функціональних мікросервісів та 2 допоміжних сервери (TB-API, Eureka). Мікросервіси, що виконують бізнес логіку: user-service, billing-service, order-service, offer-service, resource-service. Розглянемо реалізацію даних сервісів більш детально.

3.3.1 Мікросервіс user-service

Даний мікросервіс відповідає за додавання нових користувачів та їх адрес в систему, редагування та відображення їх даних тощо. Мікросервіс пропонує наступний API для роботи:

1) GET */auth/user?login* – повертає користувача за переданим параметром “login”.
2) PUT */auth/min-refresh-date?login&date* – змінює дату останньої дії користувача в системі за логіном користувача.
3) POST */auth/user* – створює нового користувача в системі, приймає об’єкт користувача в тілі запиту.
4) GET */users* – повертає список всіх користувачів, зареєстрованих в системі.
5) POST */users/user-address* – додає адресу для користувача, приймає об’єкт, що містить ID користувача та об’єкт адреси в тілі запиту.
6) PUT */users/user* – приймає об’єкт існуючого користувача та змінює його дані.
7) GET */users/addresses?login* – повертає список адрес для користувача з переданим логіном у параметрі.

«Кінцеві точки» (англ. endpoints) 1, 3 доступні, як авторизованим, такі неавторизованим користувачам, а 2, 4, 5, 6, 7 – тільки авторизованим. Дані обмеження описані в конфігурації TB-API сервера.

Розглянемо загальну структуру мікросервісу, що представлена у вигляді UML діаграми компонентів системи. UML діаграма компонентів – структурна діаграми, що показує взаємодію між модулями, класами, пакетами, файлами, бібліотеками тощо.

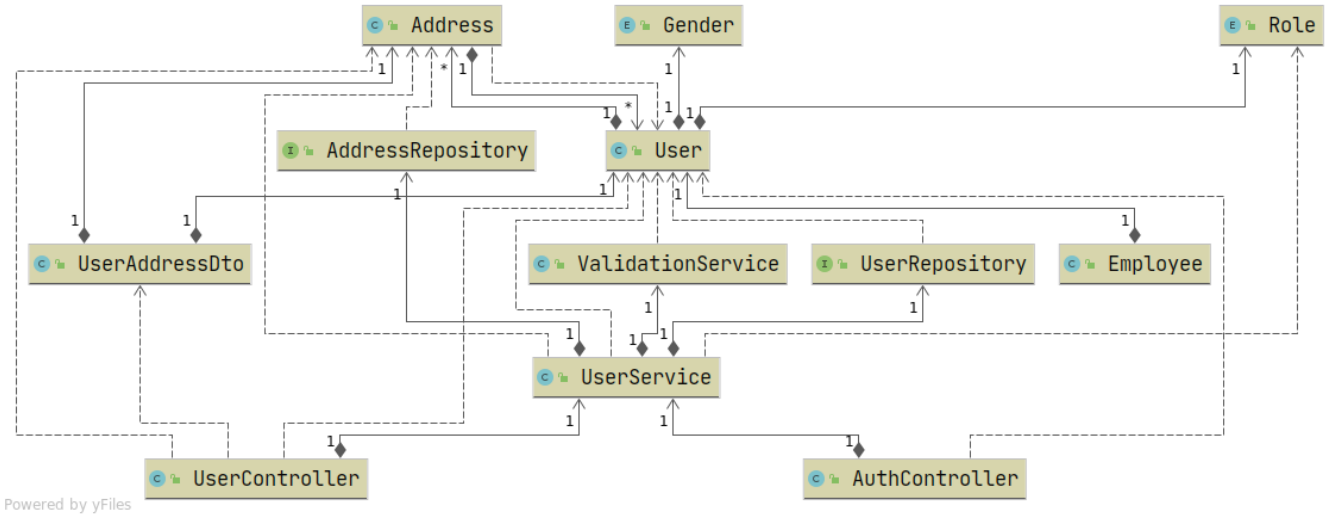


Рис. 3.5 – UML діаграма компонентів мікросервісу user-service

Діаграма була згенерована за допомогою середовища розробки IntellIJ IDEA. Діаграма відображає реалізацію багаторівневої архітектури в перевернутому вигляді, бачимо, що класи та єнами моделі бази даних (Address, Gender, Role, User) знаходяться зверху, а класи рівня представлення (UserController, AuthController) знаходяться внизу.

Рівень бізнес логіки представлений класами UserService, ValidationService. UserService відповідає за створення, зміну користувачів та операції з адресами користувача, ValidationService використовується для перевірок під час реєстрації нового користувача, наприклад, чи існує користувач з переданими логіном, електронною поштою, номером паспорту тощо.

Рівень доступу до даних представлений класами AddressRepository та UserRepository, що займаються таблицями адресу та користувача відповідно.

3.3.2 Мікросервіс billing-service

Даний мікросервіс відповідає за білінг, тобто оплата телекомунікаційних послуг. В реальних BSS/OSS системах білінг сервіс відповідає за генерацію чеків, інвойсів, мінімальне управління білінг акаунтами, а перекази між рахунками виконуються за допомогою сторонньої системи платежів, яка надає API для відповідних операцій. Білінг сервіс повинен тільки використовувати наданий API. Але так як дані системи платні і для даної роботи є надлишковою, тому була змодельована функціональність переказів між рахунками. Розглянемо API, що надає даний мікросервіс:

1) GET */billing-accounts?id* – повертає список білінг акаунтів користувача з переданим id.
2) POST */billing-account* – створює новий білінг акаунт для користувача. Приймає об‘єкт білінг акаунта в тілі запиту з посиланням на користувача, для якого буде створено білінг акаунт.
3) PUT */nrc-payment?baId&totalNRC* – проводить NRC платіж. Переказ здійснюється з білінг акаунту, ID якого передається в параметрі на білінг акаунт провайдера на суму, передану в параметрах запиту.

Розглянемо загальну структуру мікросервісу, що представлена у вигляді UML діаграми компонентів:

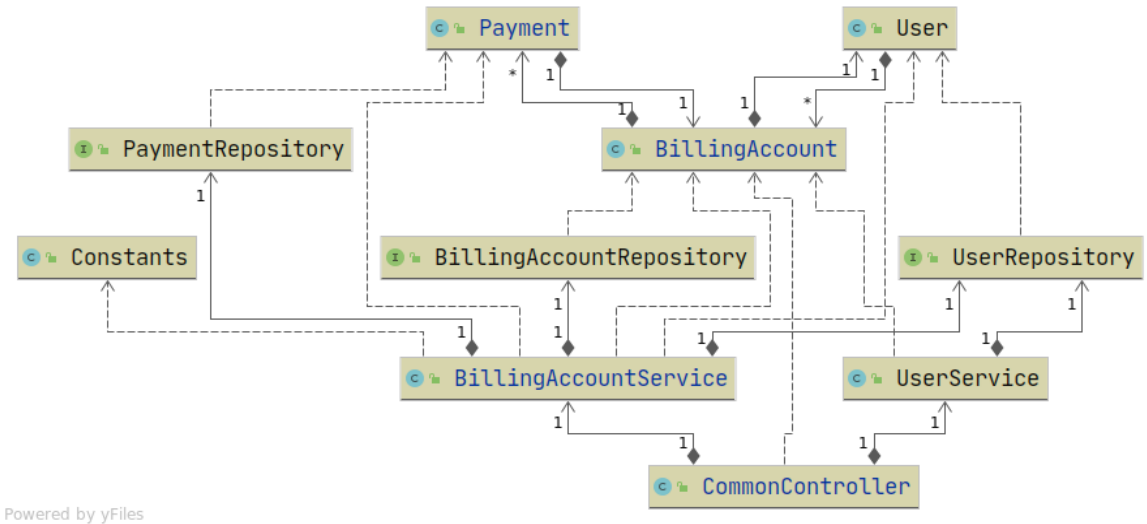


Рис. 3.6 – UML діаграма компонентів мікросервісу billing-service

Модель даних представлена дата-класами Payment, User, BillingAccount, що відповідають відповідним таблицям. Рівень доступу до даних представлений класами PaymentRepository, BillingAccountRepository, UserRepository, що займаються відповідними до їх назви таблицями. Рівень бізнес логіки представлений класами BillingService та UserService. Всі перекази виконуються як транзакції, тобто у випадку помилки система автоматично відмінить всі зміни, таким чином реалізується консистентність системи.

3.3.3 Мікросервіс order-service

Даний мікросервіс є найбільшим в системі та має доволі багатий функціонал. Він відповідає за повне виконання замовлень користувачів, створення, зміну екземпляру продуктів та відображення його параметрів. Також цей мікросервіс використовує сервер RabbitMQ для реалізації асинхронних запитів користувачів для створення замовлень. Розглянемо API, що надає даний мікросервіс:

1) POST */orders* – створює замовлення та запускає потік виконання замовлення. Приймає від клієнту об’єкт OrderValue зі всіма параметрами замовлення. Дана кінцева точка є точкою входу до ключової функціональності мікросервісу order-service, що буде розглянута детальніше далі.
2) GET */orders/{orderId}* – повертає об’єкт замовлення з ID, переданим в URI, зі всіма параметрами та посиланнями на активовані/змінені цим замовленням екземпляри продуктів.
3) GET */orders?userId* – повертає список замовлень для користувача з переданим ID в параметрах запиту.
4) GET */instances?userId* - повертає список екземплярів продуктів для користувача з переданим ID в параметрах запиту.

5) GET */instances/mobile-product/{mobileProductId}* – повертає об’єкт екземпляру мобільного продукту зі всіма параметрами за вказаним в URI ID.
6) GET */instances/internet-product/{internetProductId}* – повертає об’єкт екземпляру інтернет продукту зі всіма параметрами за вказаним в URI ID.
7) GET */instances/dtv-product/{dtvProductId}* – повертає об’єкт екземпляру DTV продукту зі всіма параметрами за вказаним в URI ID.
8) GET */deliveries/available-times?deliveryDate&products* – повертає список годин для певного дня, на які може бути запланована доставка в залежності від продукту та завантаженості працівників в цей день.
9) GET */deliveries/employee-deliveries?userId* – повертає список доставок в певний день для певного робітника за ID користувача.
10) PUT */deliveries/finish* – завершує доставку. Приймає об’єкт OrderValue в тілі запиту.

Для всіх кінцевих точок користувач має бути авторизованим.

Розглянемо загальну структуру мікросервісу, що представлена у скороченому вигляді UML діаграми компонентів, так як діаграма всіх компонентів занадто велика, тому вона представлена в додатку Б.

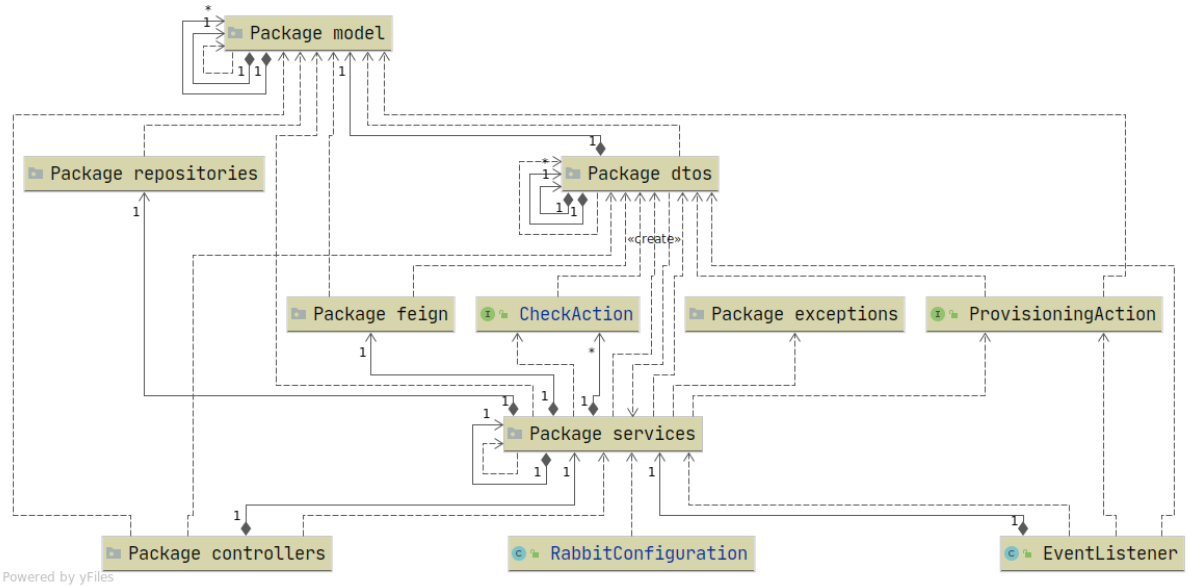


Рис. 3.7 – Скорочена UML діаграма компонентів мікросервісу order-service

Детально розглянемо ключову функціональність даного мікросервісу, а саме створення замовлення та його виконання. Для створення та записку потоку виконання замовлення використовується RabbitMQ сервер, для того щоб під час надсилання запиту на створення замовлення повернути користувачу певну відповідь і одночасно з цим запустити потік виконання замовлення. Схематично зобразимо цей процес:

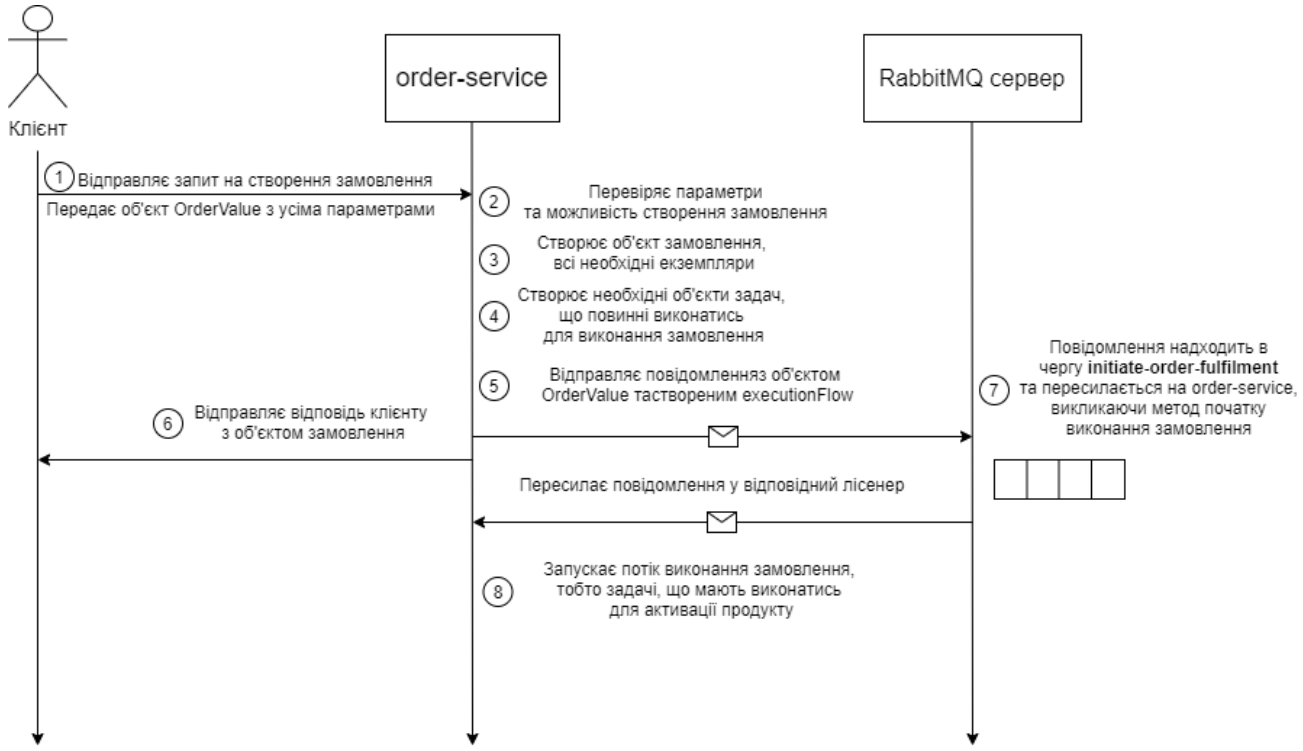


Рис. 3.8 – Загальна схема створення та початку виконання замовлення

Перед створенням замовлення потрібно перевірити чи правильні дані надіслав клієнт та чи можливе взагалі виконання замовлення. Наприклад, для підключення послуги Інтернет потрібно бути певним, що будинок користувача вже підключений до мережі провайдеру, якщо ні, то Інтернет неможливо підключити в короткий термін.

Для реалізації валідації параметрів замовлення був використаний шаблон проектування Ланцюжок Відповідальності (англ. Chain of Responsibility). Суть даного шаблону заключається в уникненні безкінечних блоків if {} else if {}. Замість цього створюється загальний інтерфейс з певним методом, наприклад, check(), та створюються класи, що реалізають даний інтерфейс з заданим

методом. Потім в основному класі валідації створюється список з об'єктами класів певного інтерфейсу і в циклі ітерації цього списку з кожного об'єкту викликається вказаний метод. Таким чином, ми можемо легко масштабувати систему, додавати нові класи реалізації та додавати їх в список.

Реалізація цього шаблону представлена на UML діаграмі класів рисунку 3.9.

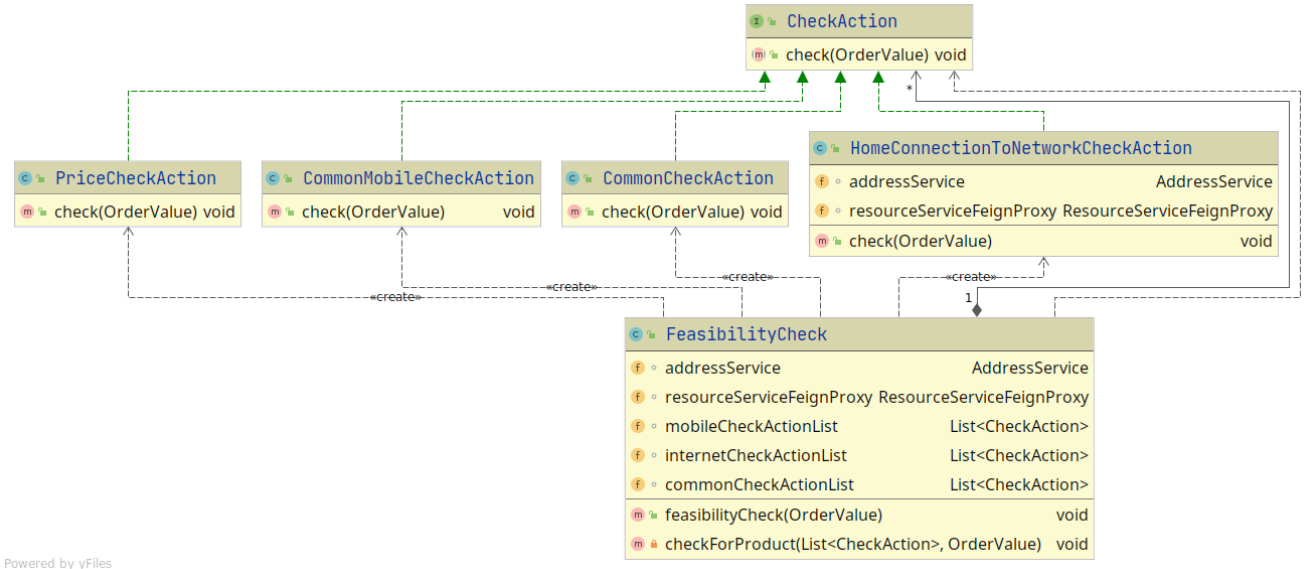


Рис. 3.9 – UML діаграма класів функціональності перевірки можливості виконання замовлення (англ. Feasibility Check)

Як бачимо з UML діаграми класів, у нас є інтерфейс CheckAction, який представляє інтерфейс для різних класів перевірок. Цей інтерфейс має метод check(OrderValue), що приймає об'єкт OrderValue з усіма параметрами замовлення. Цей інтерфейс реалізують класи CommonMobileCheckAction, HomeConnectionToNetworkCheckAction, CommonCheckAction, PriceCheckAction, CommonInternetCheckAction і інші, що не показані на діаграмі. Кожен з цих класів відповідає за певну перевірку, що відповідає його назві. В залежності від продукту, для якого відбувається валідація, об'єкти цих класів зберігаються у списках mobileCheckActions, commonCheckActions класу FeasibilityCheck, який в методі feasibilityCheck(OrderValue) в циклі ітерації по відповідних до продуктів списках валідаторів викликає метод check(OrderValue). Якщо перевірка пройшла успішно, то метод нічого не

повертає та викликається інша валідація, якщо перевірка не пройшла, то кидається виключення, що супроводжується відповіддю клієнту з статусом 400 (BAD REQUEST) та з конкретним повідомленням про помилку. Таким чином забезпечується безпечність подальшого виконання замовлення з мінімізацією ризику створення конфлікті чи інших проблем.

Також шаблон Ланцюжок Відповідальності реалізований для виконання потоку замовлення. UML діаграма класів цієї функціональності зображена на рисунку 3.10.

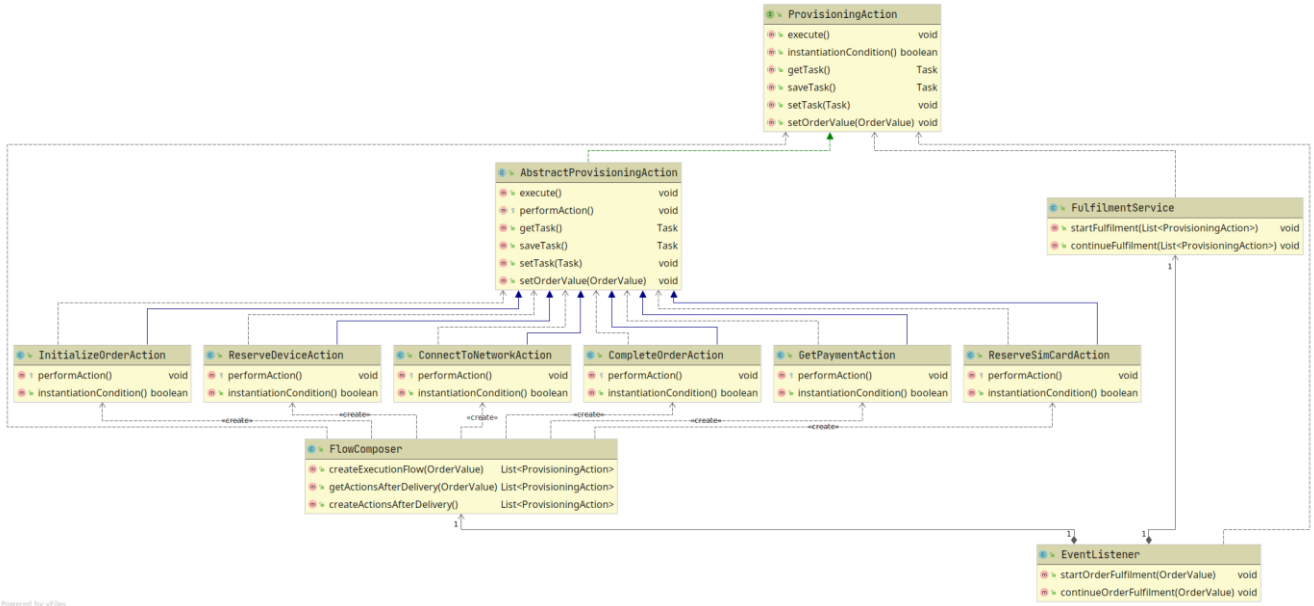


Рис. 3.10 – UML діаграма класів функціональності створення та виконання замовлення (англ. Order Fulfilment)

Як бачимо з рисунку 3.10, ми маємо інтерфейс ProvisioningAction, що має 6 методів, основними з яких є execute() та instantiationCondition(), та класи реалізації цього інтерфейсу InitializeOrderAction, ReserveDeviceAction, ConnectToNetworkAction, CompleteOrderAction, GetPaymentAction, ReserveSimCardAction, що представляють собою певну бізнес логіку задач з потоку виконання замовлення. Метод execute() відповідає за, власне, виконання певної корисної роботи, наприклад, резервування маршрутизатора, SIM карти, відправку повідомлень, оплату послуг тощо. Метод instantiationCondition() відповідає за умову створення тої чи іншої задачі,

наприклад, користувач може придбати «білу адресу», за підключення цієї послуги відповідає задача Get Fixed IP, але цю задачу не потрібно створювати, якщо користувач не придбав дану послугу. Тому під час створення потоку виконання зі всіма задачами в методі createExecutionFlow(OrderValue) класу FlowComposer, спочатку створюються об'єкти класів, що реалізують інтерфейс ProvisioningAction та додаються у список, потім в циклі для кожного об'єкту викликається метод instantiationCondition(), що перевіряє чи потрібно додавати цю задачу в потік виконання. Після цього створений потік виконання зберігається в базу даних, та передається класу FulfilmentService, що в циклі викликає метод execute() для всіх задач з потоку виконання.

Даний мікросервіс виконує ще багато корисної роботи, але в даному розділі вона не буде розглядатись. Була розглянута основна функціональність даного мікросервісу.

3.3.4 Мікросервіс offer-service

Даний мікросервіс відповідає за відображення, створення та зміну пропозицій різних продуктів та їх ціну, включаючи всі дисконти та знижки. Розглянемо API, що надає цей мікросервіс:

1) GET */offers/internet-offers* – повертає список всіх пропозицій для інтернет продукту, включаючи всі знижки, якщо такі є.
2) GET */offers/internet-offer?internetOfferId* – повертає пропозицію для інтернет продукту за вказаним в параметрах ID, включаючи всі знижки, якщо такі є.
3) GET */offers/dtv-offers* – повертає список всіх пропозицій для DTV продукту, включаючи всі знижки, якщо такі є.
4) GET */offers/dtv-offer?dtvOfferId* – повертає пропозицію для DTV продукту за вказаним ID, включаючи всі знижки, якщо такі є.

5) GET */offers/tariffs* – повертає список всіх тарифів мобільного продукту, включаючи всі знижки, якщо такі є.
6) GET */offers/tariff?tariffId* – повертає тариф для мобільного продукту за вказаним в параметрах ID, включаючи всі знижки, якщо такі є.
7) GET */offers/constant-prices* – повертає об’єкт зі всіма цінами на кінцеві послуги, включаючи всі знижки, якщо такі є.
8) GET */active-discounts* – повертає список всіх активних знижок.

Для всіх кінцевих точок користувач може бути неавторизованим.

Розглянемо загальну структуру мікросервісу, що представлена у вигляді UML діаграми компонентів:

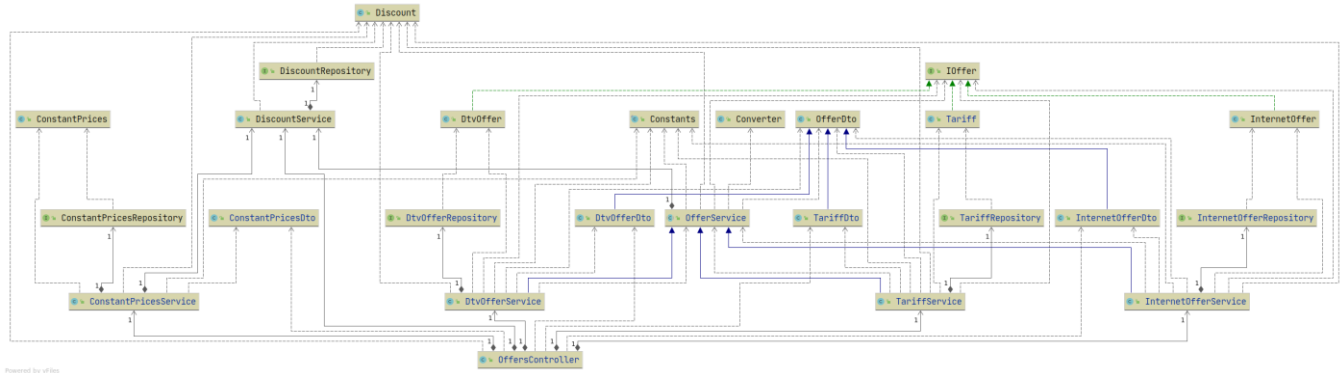


Рис. 3.11 – UML діаграма компонентів мікросервісу offer-service

Як бачимо з рисунку 3.11 ми маємо інтерфейс-маркер (інтерфейс без методів) IOffer, що є спільним інтерфейсом для класів Tariff, InternetOffer, DtvOffer. Ці класи є класами моделі, що представляють відповідні таблиці в базі даних. На відміну від них, класи TariffDto, DtvOfferDto, InternetOfferDto, що розширяють клас OfferDto використовуються для відправки даних пропозицій клієнту. Для цього використовується абстрактний клас-сервіс OfferService, що має спільну логіку конвертації об’єктів класів моделі в об’єкти dto-класів. Dto-класи містять дані з класів моделі, а також ціну пропозиції з урахуванням знижок, ця ціна і розраховується в класі OfferService.

В цьому розділі було розглянуто основну функціональність мікросервісу offer-service.

3.3.5 Мікросервіс resource-service

Даний мікросервіс використовується для створення, резервування, видалення, зміни ресурсів провайдера. Ресурсами вважаються маршрутизатори, комутатори, плати, IP адреси, SIM карти, телефонні номера та інші пристрої, що використовуються в телекомунікаціях. Управління ресурсами є ключовою функціональністю OSS систем, без якої вона не може існувати, так як це (а також управління замовленнями) є першим, що сервіс-провайдери бажають автоматизувати.

Розглянемо API, що надає цей мікросервіс:

1) GET */cables/twisted-pair-cable* – повертає довжину кабелю типу витої пари та ціну за 1 метр, враховуючи знижку.
2) PUT */cables?cableLength* – резервує довжину кабелю, що указана в параметрах запиту.
3) GET */connection?addressId&macAddress* – підключає до мережі адресу користувача.
4) GET */devices* – повертає всі пристрої, які може придбати користувач.
5) GET */devices/{deviceId}* – повертає об’єкт з ID, що вказане в URI, зі всіма параметрами.
6) PUT */devices?deviceId* – резервує пристрій для користувача з ID, що переданий в URI.
7) GET */fixed-ip* – повертає першу вільну IP адресу.
8) GET */phone-numbers* – повертає список телефонних номерів, які може придбати користувач.
9) GET */phone-numbers/{phone-number}* – повертає весь об’єкт телефонного номеру з ціною, враховуючи знижки, за телефонним номером.
10) PUT */phone-numbers/{sim-card-number}* – резервує SIM карту та номер телефону за переданим номером SIM карти в URI.

Розглянемо загальну структуру мікросервісу, що представлена у вигляді UML діаграми компонентів:

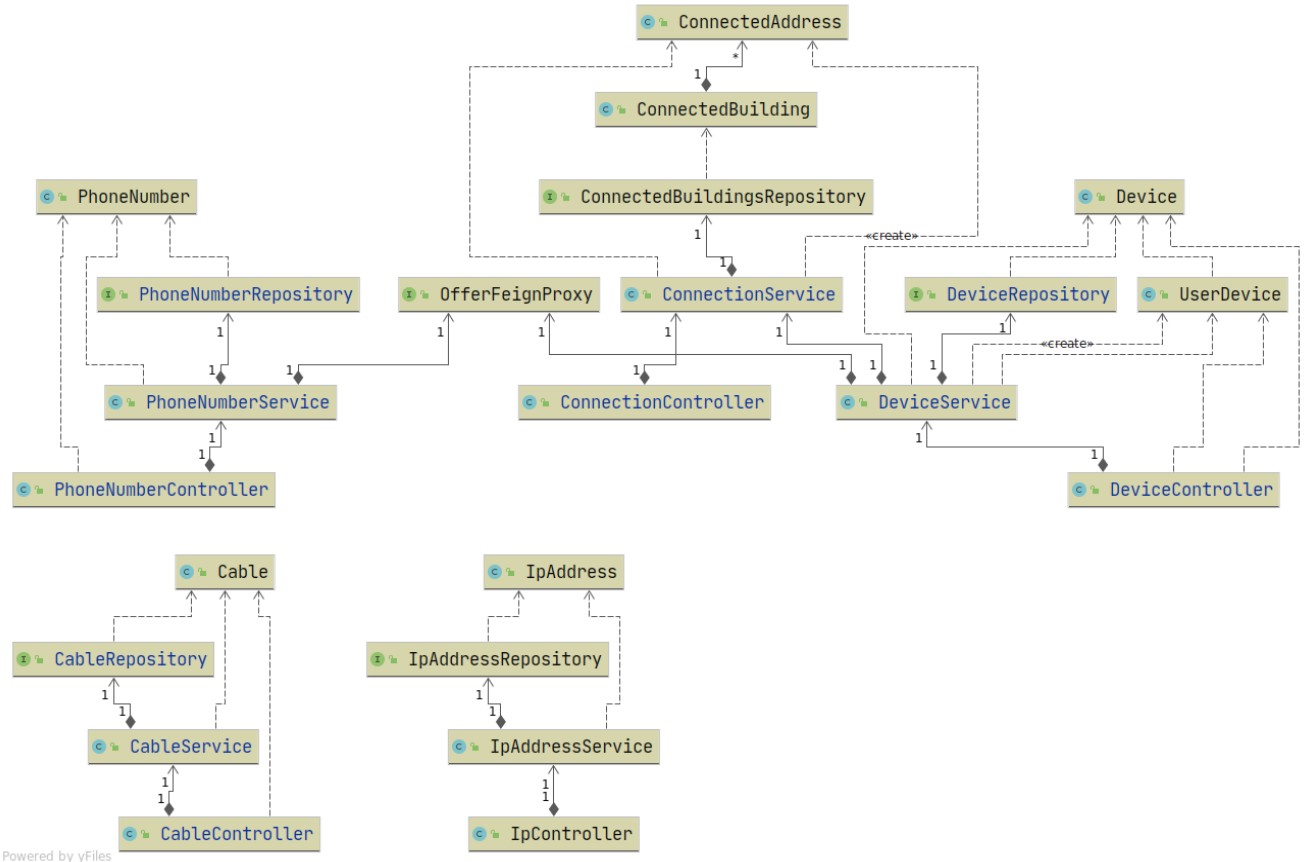


Рис. 3.12 – UML діаграма компонентів мікросервісу resource-service

Даний мікросервіс має доволі просту функціональність, він займається звичайними CRUD операціями (англ. Create, Read, Update, Delete) для відповідних класів моделей, тобто – пристроїв, IP адрес, телефонних номерів, підключених абонентів, кабелів тощо. Мікросервіс використовує offer-service для визначення ціни пристрої чи номери, враховуючи знижку, використовуючи OfferFeignProxy інтерфейс, який не потребує реалізації методів, так як цим займається компонента Spring Open Feign, яка була детально розглянута в розділі 2.2.2. Також цей мікросервіс використовується order-service-ом під час виконання замовлення для резервації ресурсів та підключення абонентів.

3.4 Реалізація клієнтської частини

Клієнтська частина, тобто та частина, яка виконується в браузері, і яку використовує користувач написана на фреймворку Angular, який використовує для побудови проекти незалежні структурні елементи, що називаються компоненти. Кожен компонент використовується для реалізації певної функціональності та відображенні її роботи на сторінці. Розглянемо всі компоненти, що були розроблені на клієнтській частині проекту та опишемо функціональність, яку вони надають:

1) authorization-component – компонент, що відповідає за авторизацію користувача. Має сторінку з полями логіна та пароля, які має ввести користувач. Введені дані валідуються та відправляються на сервер.
2) deliveries-component – компонент, що відображає доставки для співробітника. На сторінці також є поля для вводу даних, необхідних для підтвердження доставки.
3) device-info-component – компонент, що відображає діалогове вікно для перегляну інформації про пристрій. Відкривається з вікна Order Entry.
4) edit-profile-component – компонент, що надає користувачу можливість редагувати власні дані.
5) footer-component – компонент, що використовується для відображення футеру в кінці сторінки з інформацією про провайдера та копірайт.
6) header-component - компонент, що використовується для відображення заголовку на початку сторінки з кнопками авторизації та реєстрації.
7) headerauth-component – компонент, що використовується для відображення заголовку на початку сторінки з логіном авторизованого користувача та кнопкою виходу з системи.
8) home-component – компонент, що включає заголовок, футер, меню та робочу область сторінки.

9) homeauth-component – компонент, що зберігає заголовок автризованого користувача, футер, меню та робочу область сторінки.
10) instance-component – компонент, що відображає всю інформацію про екземпляр певного продукту.
11) menu-component – компонент, що зберігає меню з посиланнями на різні сторінки. Представлення меню залежить від ролі авторизованого користувача. Для неавторизованого користувача відображається посилання тільки на сторінку пропозицій.
12) offer-component – компонент, що відображає всі пропозиції для всіх продуктів. Має кнопку переходу на сторінку order entry.
13) order-component – компонент, що відображає всю інформацію про замовлення.
14) order-entry-component – головний компонент системи, є «магазином» або «вхідною точкою» для створення замовлення. Надає можливість користувачу обрати продукти відредагувати параметри. Після цього користувач має обрати дату та час доставки, якщо вона необхідна та підтвердити замовлення. Перед відправкою даних на сторону сервера всі параметри замовлення валідуються і у випадку знаходження помилок користувачу висвітлюється повідомлення про помилку.
15) orders-and-instances-component – компонент, що відображає коротку інформацію про всі замовлення та екземпляри користувача.
16) profile-component – компонент, що відображає всю інформацію користувача. Має кнопку переходу на сторінку зміни інформації.
17) registration-component – компонент, що забезпечує можливість реєстрації користувача. Має всі необхідні поля для реєстрації. Після заповнення даних користувача вони валідуються і кнопка реєстрації розблоковується, якщо дані невалідні, то висвітлюється повідомлення про помилку та кнопка залишається заблокованою

Також на клієнтській частині використовуються сервіси, які використовуються компонентами для роботи. Розглянемо основні сервіси:

1) instance-service – сервіс, що зберігає методи для комунікації з мікросервісом order-service для отримання даних про екземпляри продуктів.
2) offers-serivce – сервіс, що зберігає методи для комунікації з мікросервісом offers-service для отримання даних про пропозиції та знижки.
3) order-service – сервіс, що зберігає методи для комунікації з мікросервісом order-service для отримання даних про замовлення та їх створення.
4) resource-service – сервіс, що зберігає методи для комунікації з мікросервісом resource-service для отримання даних про ресурси.
5) user-service – сервіс, що зберігає методи для комунікації з мікросервісом user-service для реєстрації користувачів, отримання їх даних та зміни.
6) auth-guard – сервіс, що зберігає токен авторизації в пам’ять браузера та підтримує авторизації користувача.
7) jwt-interceptor – сервіс, що перехоплює всі запити, що надсилають вищевказані сервіси, та вставляє токен, якщо користувач автризований.

В цьому розділі були розглянути основні моменти реалізації клієнтської частини проекту.

ВИСНОВКИ ДО РОЗДІЛУ 3

В даному розділі було розглянуто реалізацію розроблюваної системи, як на серверній частині, так і клієнтській. Також було розглянуто загальну структуру проекту для кращого розуміння архітектури та архітектуру трьох баз даних.

Для реалізації серверної частини використовується мікросервісна архітектура, що дає змогу повноцінно розділити функціонально незалежні частини системи на окремі компоненти, що називаються мікросервіси. Всього було створено 5 мікросервісів, що, в свою чергу, використовують 3 бази даних, для підтримки яких використовуються PostgreSQL та MongoDB у якості СУБД. Для зручної роботи мікросервісів використовуються інструменти Spring Cloud фрейморку, а саме – Spring Cloud Netflix Zuul, Spring Cloud Netflix Eureka, Spring Cloud OpenFeign. Також використовується фреймворк Spring Security та JWT для реалізації аутентифікації та авторизації користувачів. Окрім цього був використаний брокер повідомлень RabbitMQ, для реалізації асинхронної взаємодії клієнта та сервіса. Були наповну використані концепції ООП та шаблон проектування Ланцюжок Відповідальності.

Для реалізації клієнтської частини був використаний веб-фреймворк Angular, що надає багатий інструментарій для розробки веб-додатків. Було створено багато компонентів, що реалізують незалежну функціональність, та сервісів для взаємодії з серверною частиною, та підтримання авторизації тощо.

РОЗДІЛ 4. ІНСТРУКЦІЯ КОРИСТУВАЧА

Наглядно покажемо всю реалізовану функціональність системи від реєстрації нового користувача до перегляду активних екземплярів продуктів.

Титульна сторінка системи є сторінкою з відображенням пропозицій:

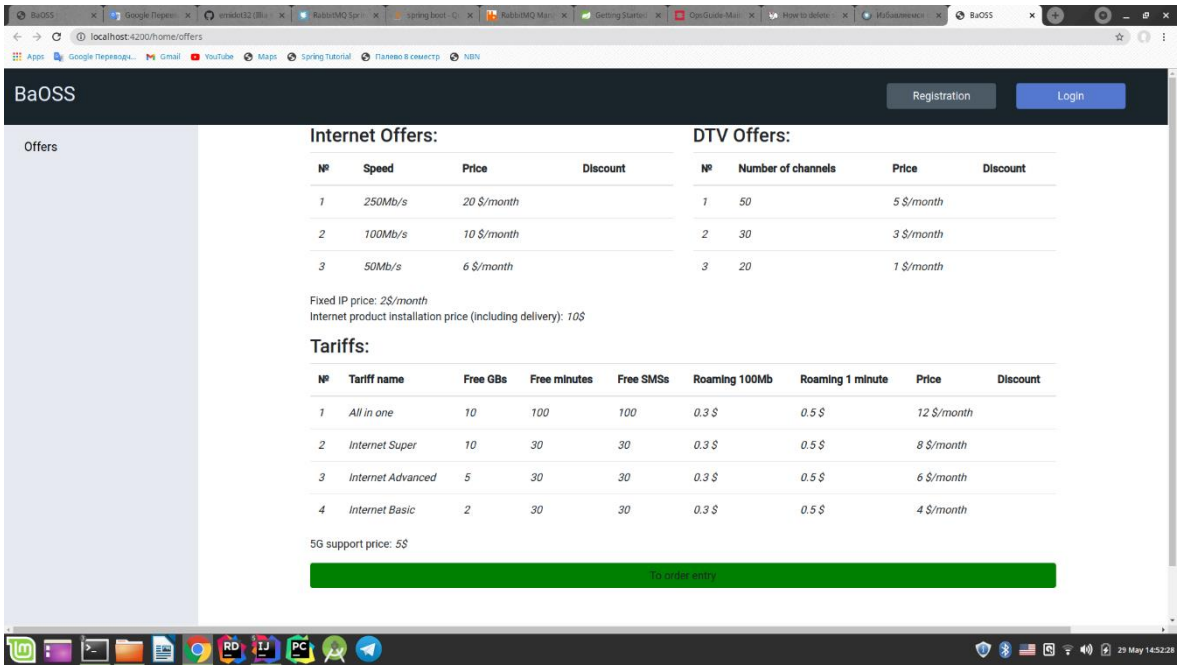


Рис. 4.1 – Сторінка пропозицій

Зліва бачимо меню з одним посиланням на дану сторінку. Зверху бачимо заголовок з кнопками реєстрації та авторизації. Для продовження роботи в системі потрібно зареєструватися та авторизуватися.

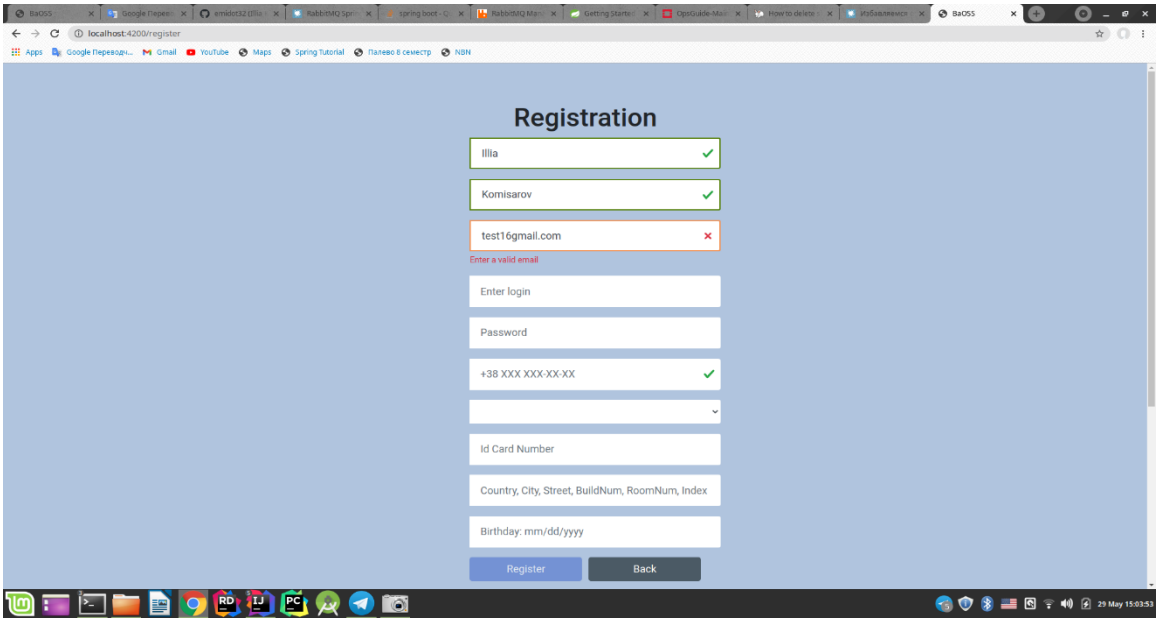


Рис. 4.2 – Сторінка реєстрації

Як бачимо при введенні неправильної пошти, висвітлюється повідомлення про помилку та блокується кнопка підтвердження реєстрації.

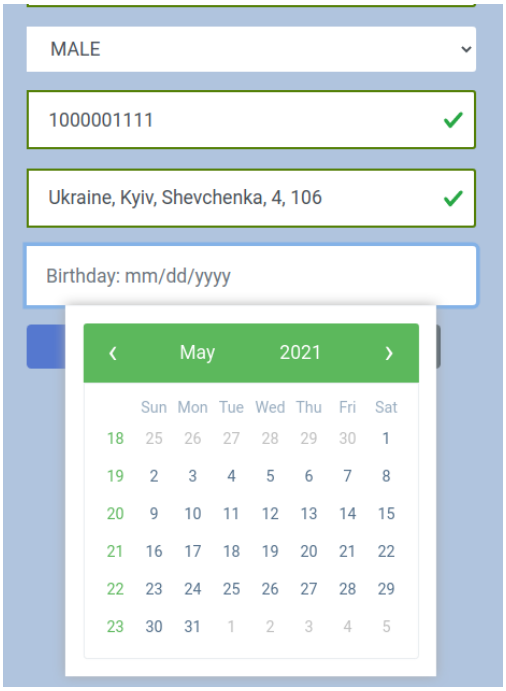


Рис. 4.3 – Введення всіх даних

Після підтвердження реєстрації користувач переходить на сторінку авторизації.

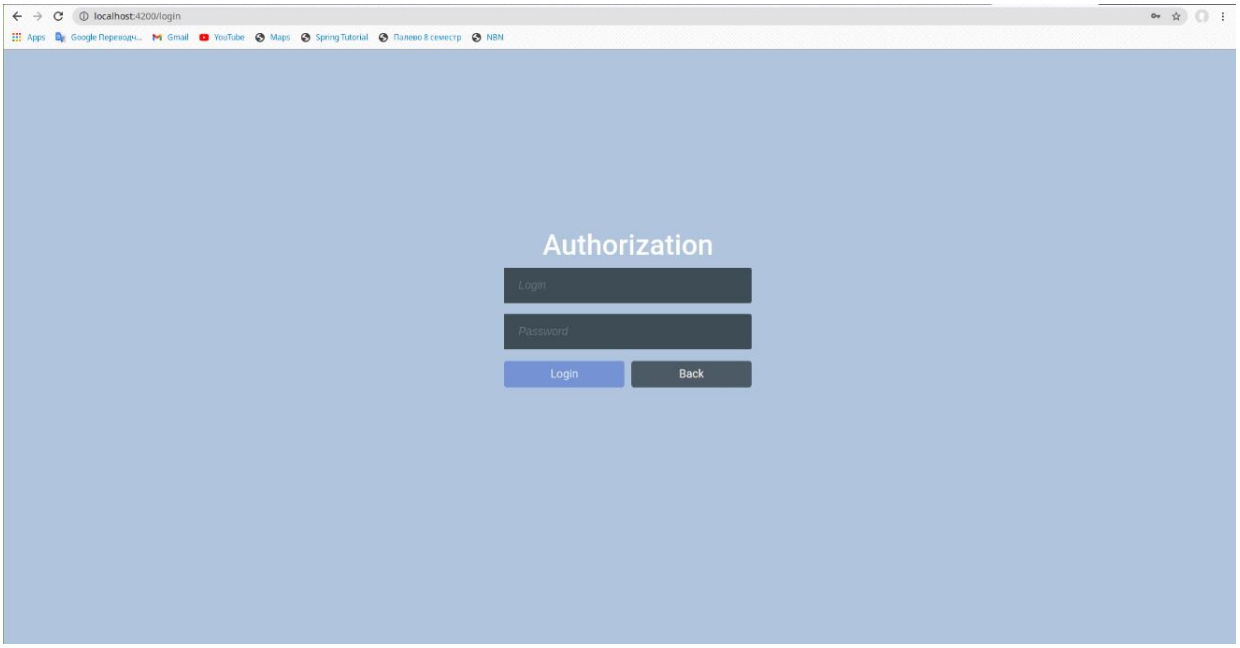


Рис. 4.4 – Сторінка авторизації

Після введення правильного логіну та паролю користувач переходить на персональну сторінку з відображенням власних даних.

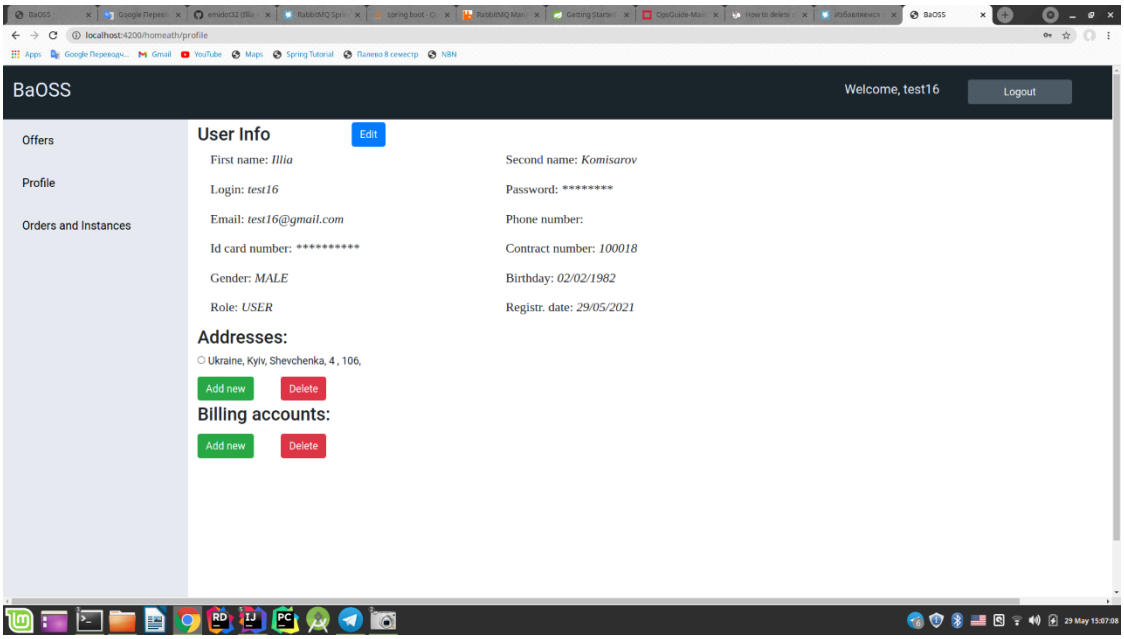


Рис. 4.5 – Персональна сторінка користувача

На цій сторінці також користувач може додати інші адреси проживання та білінг акаунти.

Рис. 4.6 – Створення білінг акаунту

Після натискання на кнопку Edit користувач переходить на сторінку, де він може відредагувати свої дані.

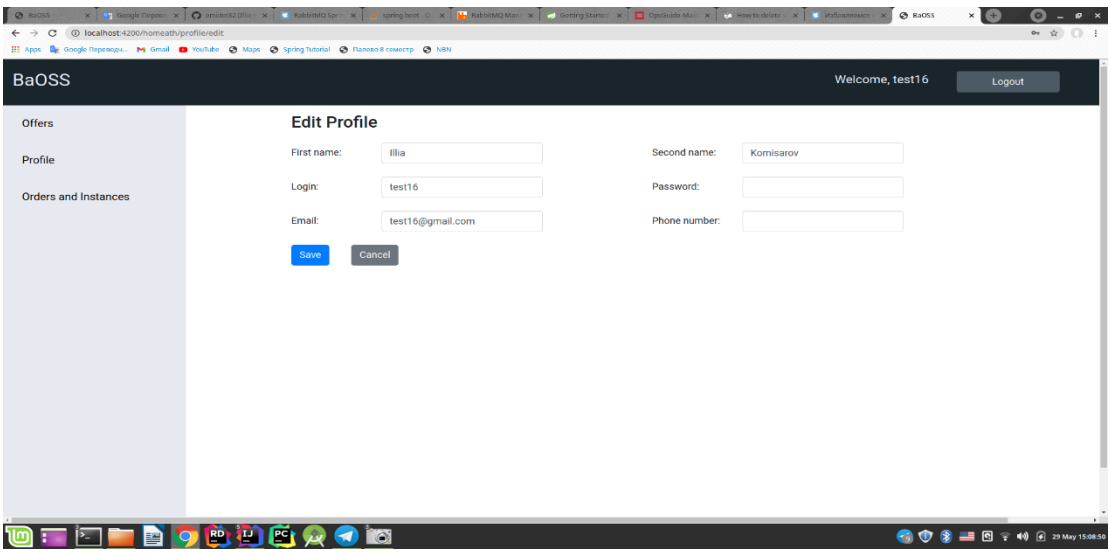


Рис. 4.7 – Сторінка редагування персональних даних користувача

Відредагуємо ім’я користувача та підтвердимо зміни.

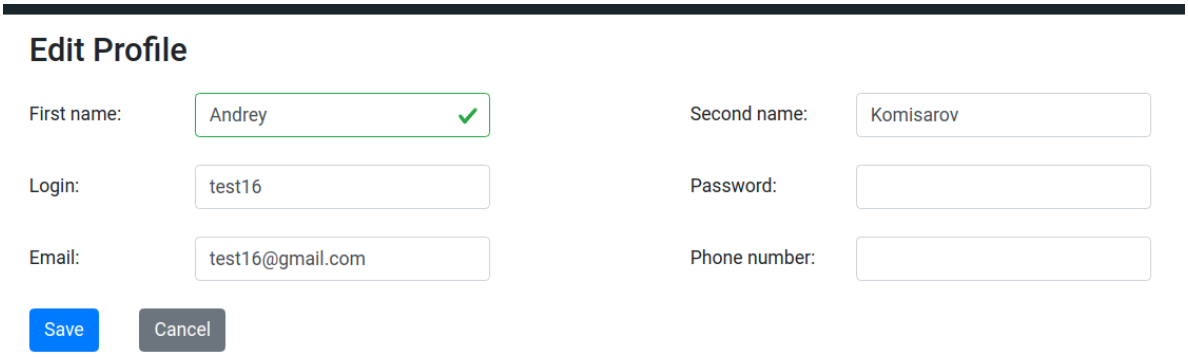


Рис. 4.8 – Редагування імені користувача

Як бачимо, ім’я було успішно зменено.

Рис. 4.9 – Перевірка зміни імені користувача

Перейдемо до сторінки пропозицій. При наявності певних знижок пропозиції мають наступний вигляд:

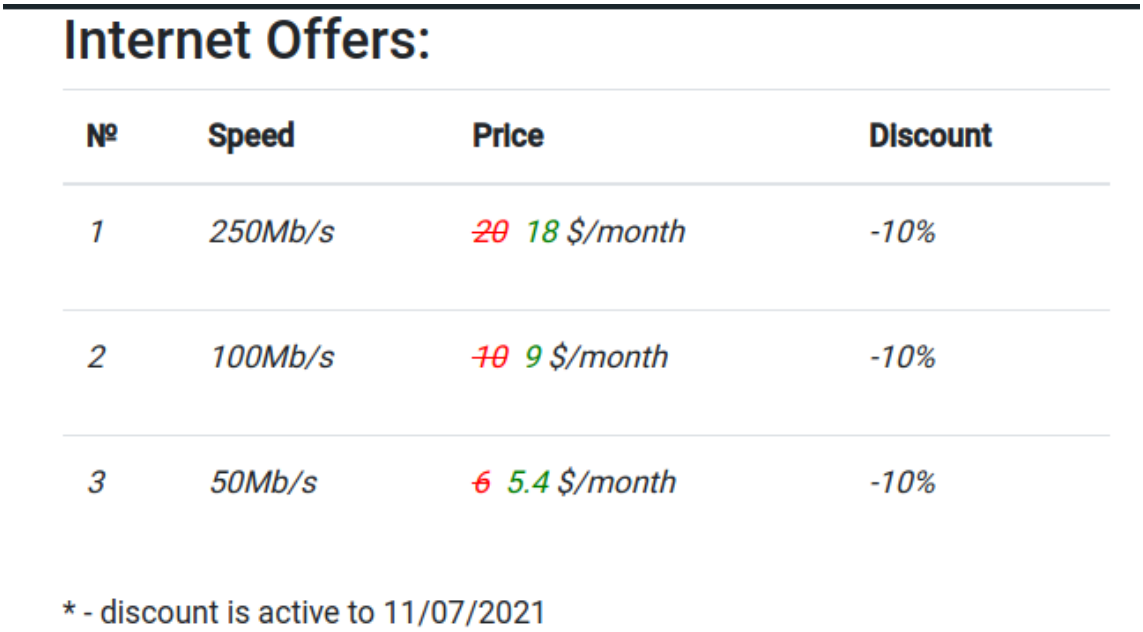


Рис. 4.10 – Пропозиції при наявності знижки

Перейдемо на сторінку створення замовлення та оберемо Інтернет та DTV продукти.

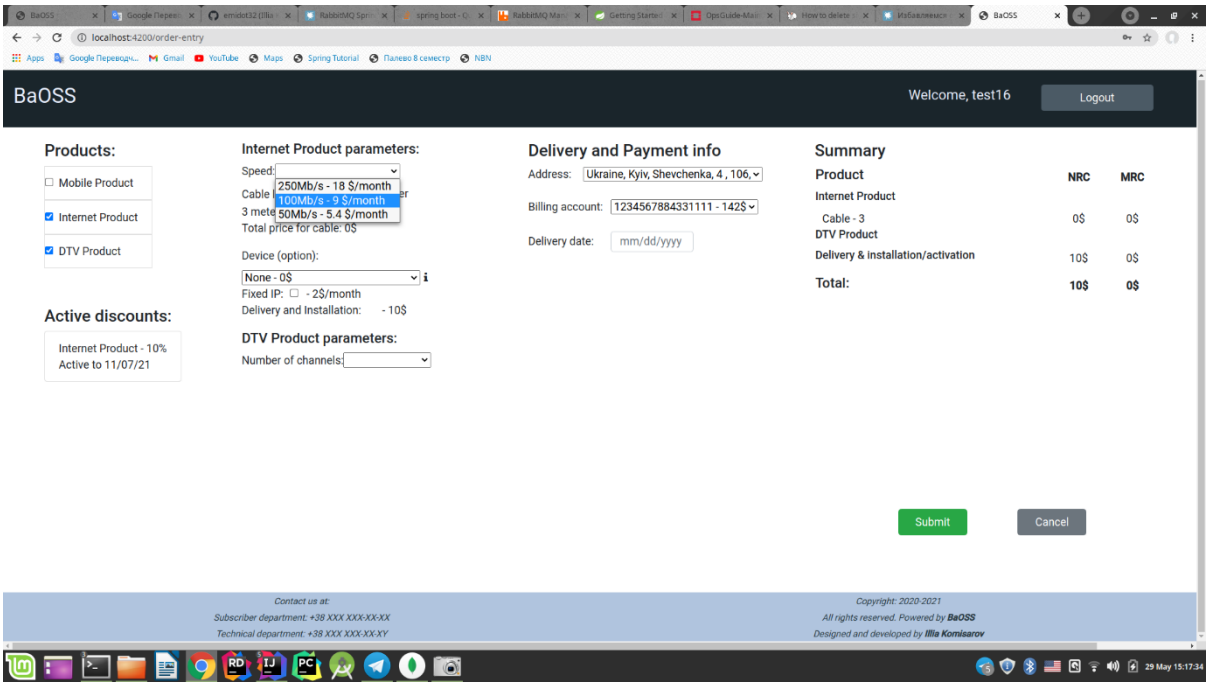


Рис. 4.11 – Сторінка створення замовлення

Введемо всі параметри, а саме – швидкість інтернету, довжину кабелю, маршрутизатор, кількість каналів, адресу, білінг акаунт, дату та час підключення послуг на стороні користувача. Після цього підтвердимо замовлення. Користувач переходить на сторінку всіх замовлень та екземплярів продуктів:

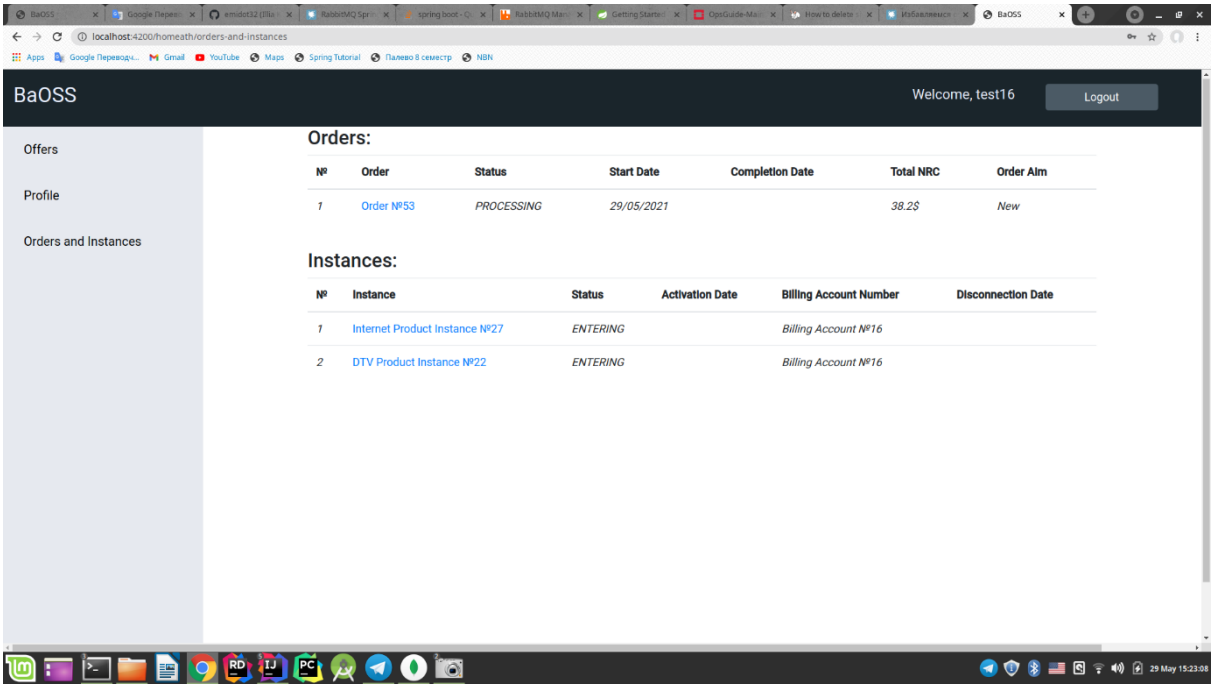


Рис. 4.12 – Сторінка зі всіма замовленнями та екземплярами продуктів

Як бачимо замовлення в статусі виконання, а екземпляри ще не активовані. Переглянемо детальніше стан замовлення та стан задач, які були створені на сторінці замовлення, як бачимо деякі задачі виконались, але зараз потік виконання очікує підтвердження виконання робіт на стороні клієнта для підключення послуг.

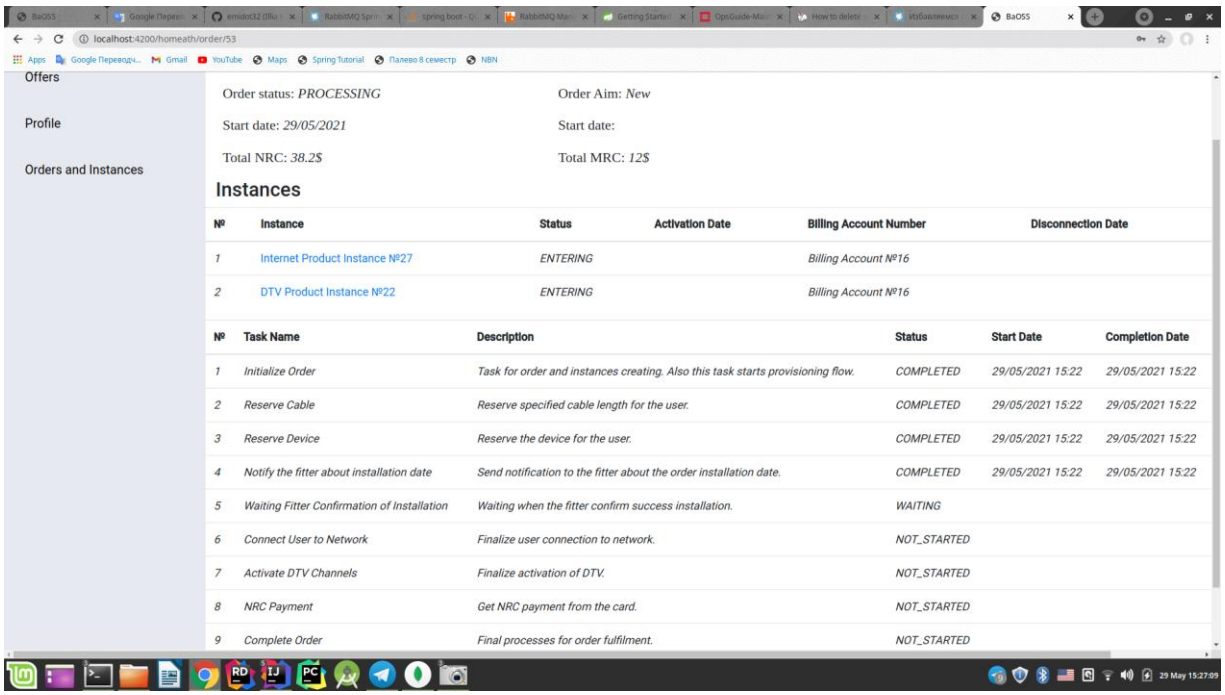


Рис. 4.13 – Сторінка з параметрами замовлення та задач

Для цього використовується сторінка доставок для співробітників компанії.

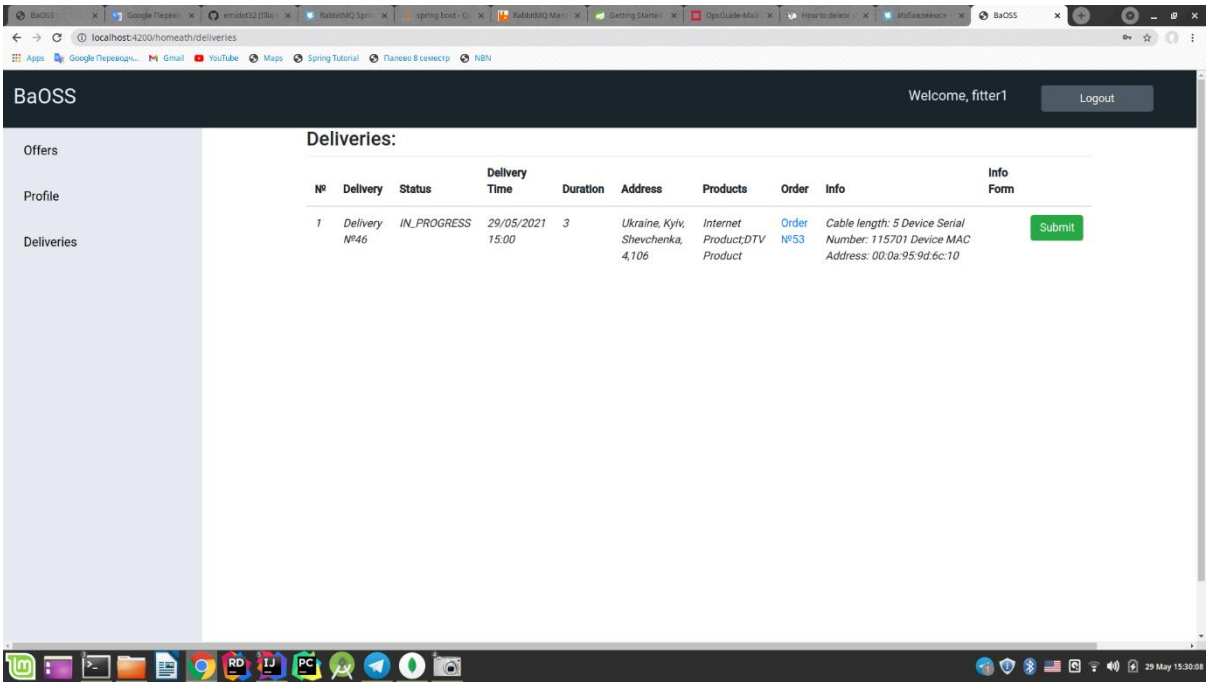


Рис. 4.14 – Сторінка доставок для співробітника компанії

Для перегляду цієї сторінки потрібно авторизуватися з акаунту співробітника компанії, який відповідний за підключення послуг клієнту. Після підтвердження підключення на сторінці замовлення бачимо, що замовлення було успішно виконане, а статус екземплярів став активним.

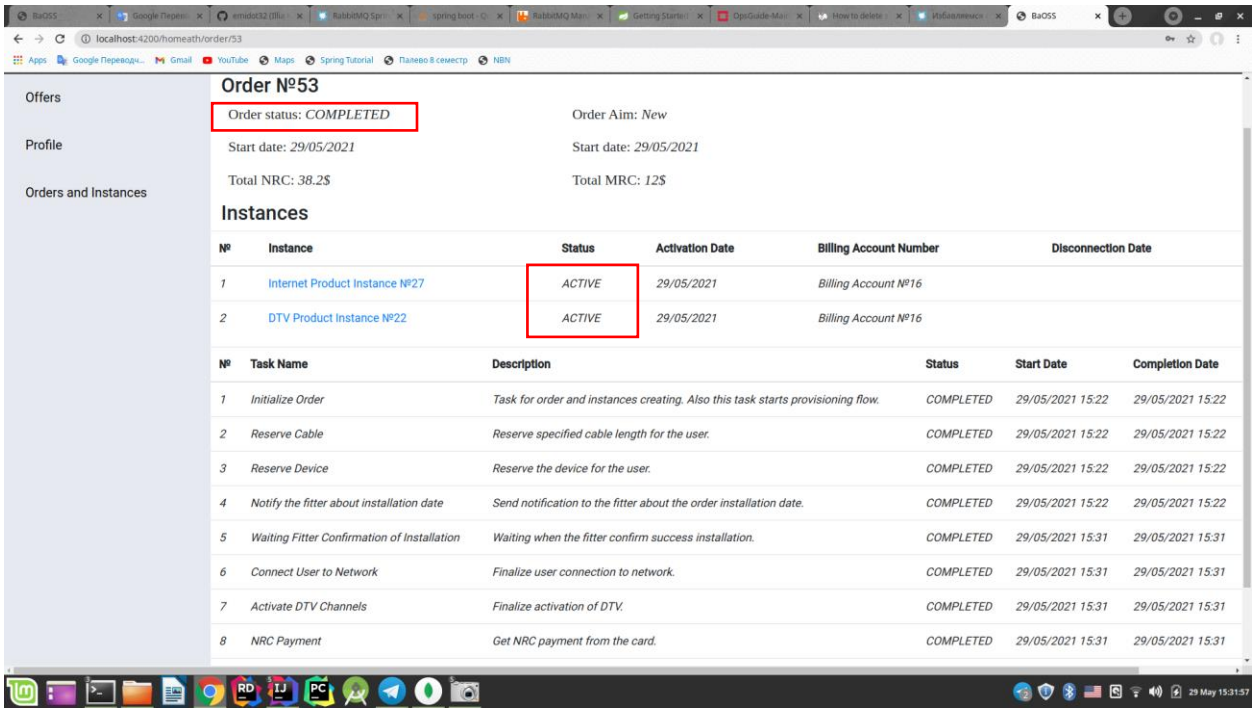


Рис. 4.15 – Сторінка виконанного замовлення

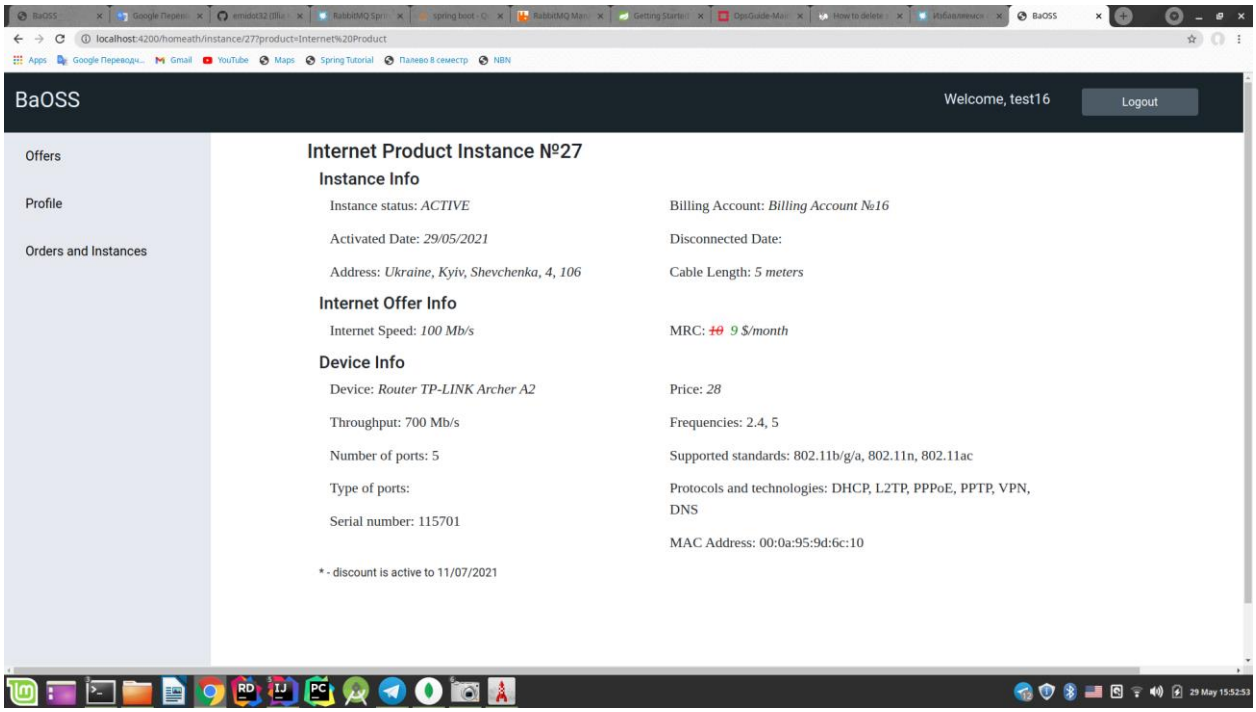


Рис. 4.16 – Сторінка екземпляру інтернет продукту

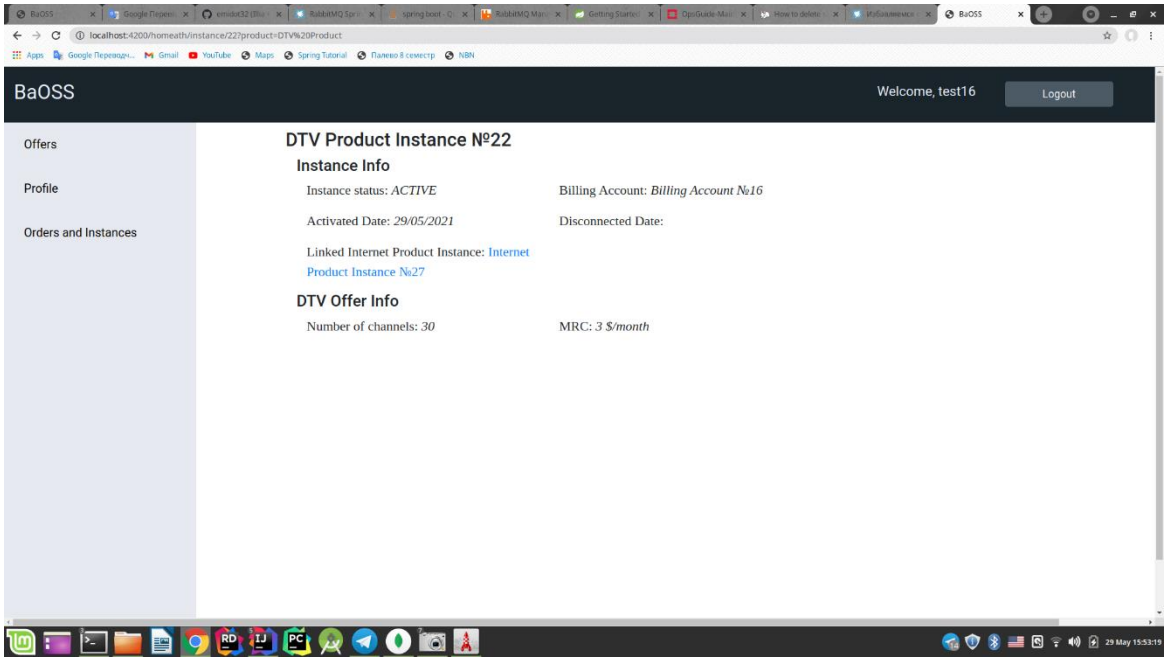


Рис. 4.17 – Сторінка екземпляру DTV продукту

Тепер користувач може переглянути параметри активованих продуктів на сторінках екзмеплярів, що зображені на рисунках 4.16 та 4.17

ВИСНОВКИ ДО РОЗДІЛУ 4

В даному розділі було показано інструкцію користувача по використанню розробленої системи. Було проведено E2E тестування системи, фактичні результати повністю зійшлись з очікуваними. Було розглянуто таку функціональність системи: реєстрацію, авторизацію, перегляд пропозицій, перегляд пропозицій зі знижкою, перегляд персональної сторінки користувача, зміна даних користувача, додавання білінг акаунту користувача, створення замовлення, перегляд сторінки всіх замовлень та екземплярів продуктів, перегляд сторінки замовлення, задач, перегляд сторінки доставок та підтвердження підключення послуг інтернету та DTV, перевірка стану замовлення, екземпляру.

Розроблена система була повноцінно протестована. Результати показують, що система може використовуватись користувачами, так як було реалізовано всі вимоги, що ставились на початку проектування.

ВИСНОВКИ

Виконана бакалаврська робота, в якій була реалізована система, що автоматизує процеси телекомунікаційних компаній, такі як створення та виконання замовлення та управління ресурсами. Такі системи називаються BSS/OSS, розроблена система має урізану функціональність в порівнянні з комерційними аналогами, але розроблювалась з використанням чинних технологій та архітектур. Система покриває основний спектр функціональності таких систем, а саме – реєстрацію, авторизацію користувачів, відображення та зміна їх даних, перегляд пропозицій, створення замовлення, автоматизація процесу його виконання, оплата рахунків та управління ресурсами.

В даній роботі були використані такі архітектури: клієнт-серверна, що розділяє систему на 2 частини – клієнтську та серверну, багатошарова, що розділяє зони абстракції системи та мікросервісна, що розділяє систему на окремі компоненти, що реалізують конкретну функціональність. Такий набір архітектур забезпечує стабільність, відмовостійкість, надійність та полегшує довготривалу розробку системи.

В даній роботі були використані такі технології: Java як мова програмування для серверної частини, Spring та його компоненти як основний фреймворк для серверної частини, RabbitMQ як брокер повідомлень, Maven як засіб автоматизації збірки проекту, Angular як основний веб-фреймворк для клієнтської частини. Такий набір технологій забезпечує надійність, стабільність, відмовостійкість та полегшує довготривалу розробку системи.

Розроблена система відповідає всім вимогам, що закладались перед початком проектування системи і має можливість до розширення функціональності та масштабування.

СПИСОК ЛІТЕРАТУРИ

1. Invest Implications: 'Magic Quadrant for Integrated Revenue and Customer Management for CSPs' [Електронний ресурс]. – Режим доступу: https://www.gartner.com/en/documents/3156622 – Дата доступу: 07.05.2021 – Назва з екрану
2. Netcracker BSS Portfolio [Електронний ресурс]. – Режим доступу: https://www.netcracker.com/portfolio/products/digital-bss/ – Дата доступу: 07.05.2021 – Назва з екрану
3. Netcracker OSS Portfolio [Електронний ресурс]. – Режим доступу: https://www.netcracker.com/portfolio/products/digital-oss/ – Дата доступу: 07.05.2021 – Назва з екрану
4. Amdocs Company [Електронний ресурс]. – Режим доступу: https://ru.wikipedia.org/wiki/Amdocs – Дата доступу: 07.05.2021 – Назва з екрану
5. Amdocs OSS Portfolio Presentation [Електронний ресурс]. – Режим доступу: http://solutions.amdocs.com/rs/647-OJR-802/images/Beyond-Next-generation-OSS-whitepaper.pdf – Дата доступу: 07.05.2021 – Назва з екрану
6. Oracle OSS [Електронний ресурс]. – Режим доступу: https://www.tadviser.ru/index.php/%D0%9F%D1%80%D0%BE%D0%B4%D1%83%D0%BA%D1%82:Oracle_Communications_Operations_Support_Systems_(OSS) – Дата доступу: 07.05.2021 – Назва з екрану
7. Oracle OSS Portfolio Image [Електронний ресурс]. – Режим доступу: https://www.google.com/url?sa=i&url=https%3A%2F%2Fenterprisearchitectview.wordpress.com – Дата доступу: 07.05.2021 – Назва з екрану
8. Gartner Overview of Oracle OSS [Електронний ресурс]. – Режим доступу: https://www.gartner.com/reviews/market/operations-support-systems/vendor/oracle – Дата доступу: 07.05.2021 – Назва з екрану

9. Huawei OSS [Електронний ресурс]. – Режим доступу: https://carrier.huawei.com/en/technical-topics/service/SolutionTopic/OSS/OSS – Дата доступу: 07.05.2021 – Назва з екрану
10. Gartner Overview of Huawei OSS [Електронний ресурс]. – Режим доступу: https://www.gartner.com/reviews/market/operations-support-systems/vendor/huawei – Дата доступу: 07.05.2021 – Назва з екрану
11. Herbert Schildt. Java: The Complete Reference, Ninth edition / Herber Schildt – McGraw-Hill Education, 2014. – 1312 p.
12. Irakli Nadareishvili, Ronnie Mitra, and Mike Amundsen Microservice Architecture Published by O'Reilly Media, Inc. June 2016: First Edition 697p
13. Craig Walls. Spring in Action, Third Edition / Craig Walls – Wiley India Pvt. Limited, 2011. – 424 p.
14. Joshua Bloch. Effective Java, Second Edition / Joshua Bloch – Addison-Wesley Professional, 2008. – 375 p.
15. Robert C. Martin. Clean Architecture: A Craftsman's Guide to Software Structure and Design / Robert C. Martin – Prentice Hall, 2017. – 432 p.
16. Leonard Richardson and Mike Amundsen with a Foreword by Sam Ruby RESTful Web APIs Published by O'Reilly Media, Inc. September 2013: First Edition 854 p.
17. Spring Docs [Електронний ресурс]. – Режим доступу: https://spring.io/projects/ – Дата доступу: 07.05.2021 – Назва з екрану
18. Angular Docs [Електронний ресурс]. – Режим доступу: https://angular.io/docs/ – Дата доступу: 07.05.2021 – Назва з екрану

Додатки

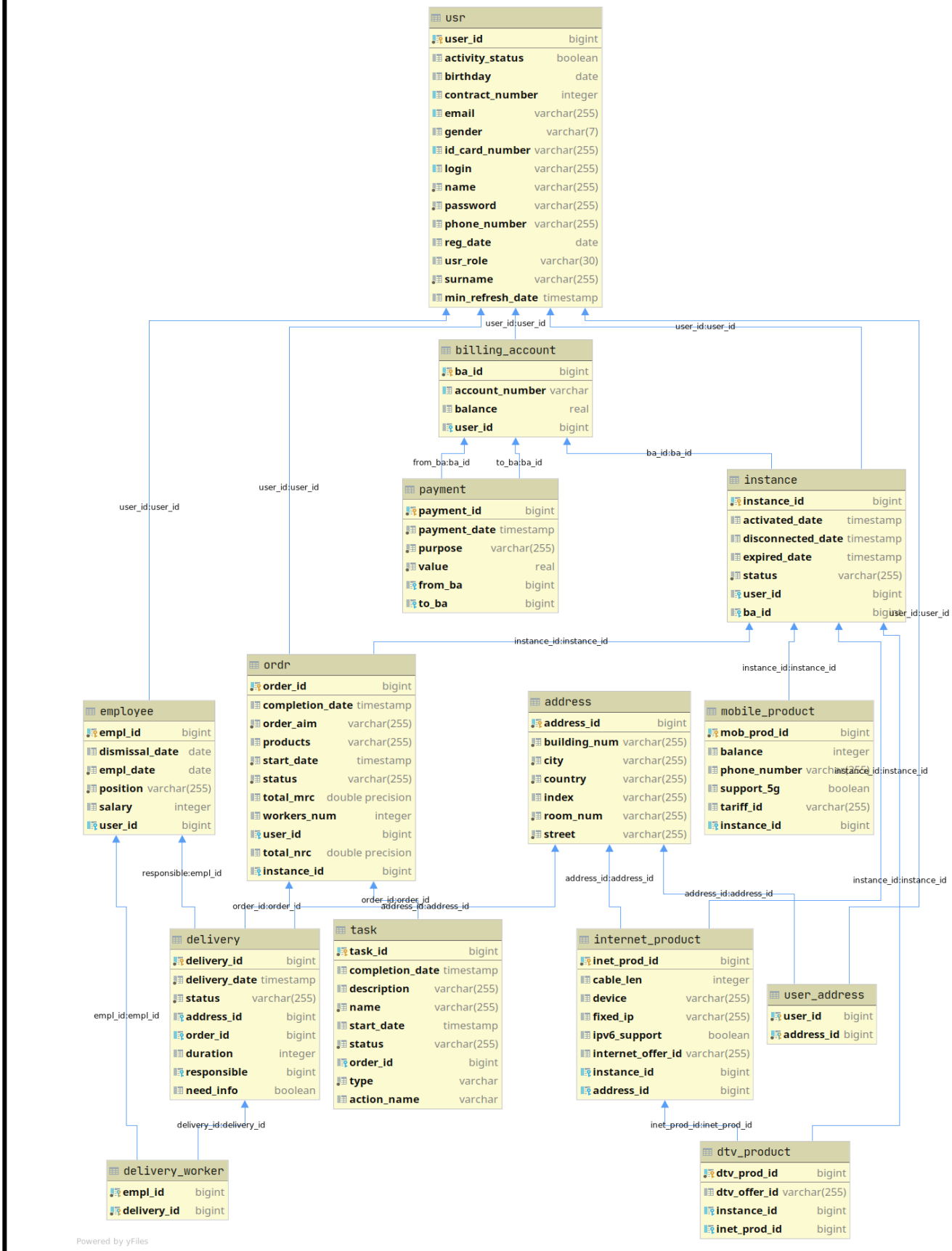
ДОДАТОК 1

Автоматизована система управління замовленнями та ресурсами для телекомунікаційних компаній

Схема структурна: схема бази даних

Аркушів 1

2021



					ІАЛЦ.467100.003 Д1	
Змн	Арк.	№ докум.	Підпис	Дата	Схема структурна Схема бази даних	Літ. / Маса. / Масш.
Розроб.		Комісаров І.А.				
Перевір.		Ружевський М.С.				Аркуш 1 Аркушів 1
Т. Контр.						
Н. Контр.		Сімоненко В.П.			Дипломна робота	НТУУ “КПІ” ФІОТ ІО-71
Затверд.		Стіренко С.Г.				

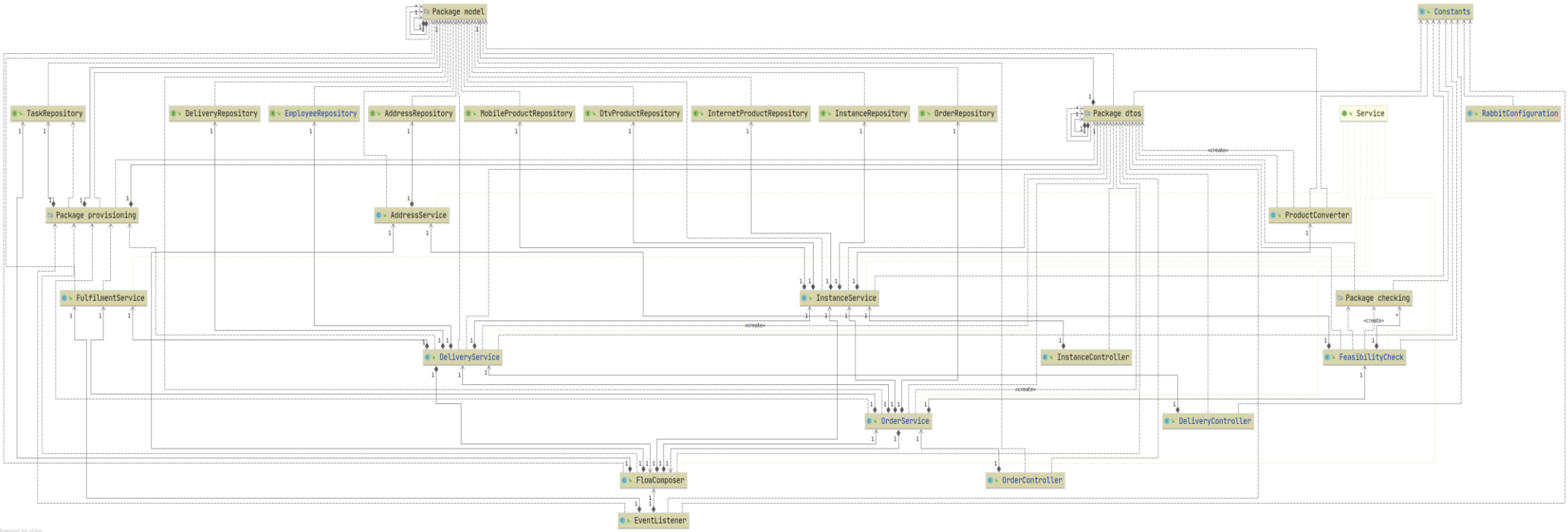
ДОДАТОК 2

Автоматизована система управління замовленнями та ресурсами для телекомунікаційних компаній

Схема функціональна: діаграма класів

Аркушів 1

2021



					ІАЛЦ.467100.004 Д2			
						Літ.	Маса.	Масш.
Змн	Арк.	№ докум.	Підпис	Дата	Схема функціональна Діаграма класів			
Розроб.		Комісаров І.А.				Аркуш	1	Аркушів 1
Перевір.		Ружевський М.С						
Т. Контр.								
Н. Контр.		Сімоненко В.П.			Дипломна робота	НТУУ “КПІ” ФІОТ ІО-71		
Затверд.		Стіренко С.Г.						

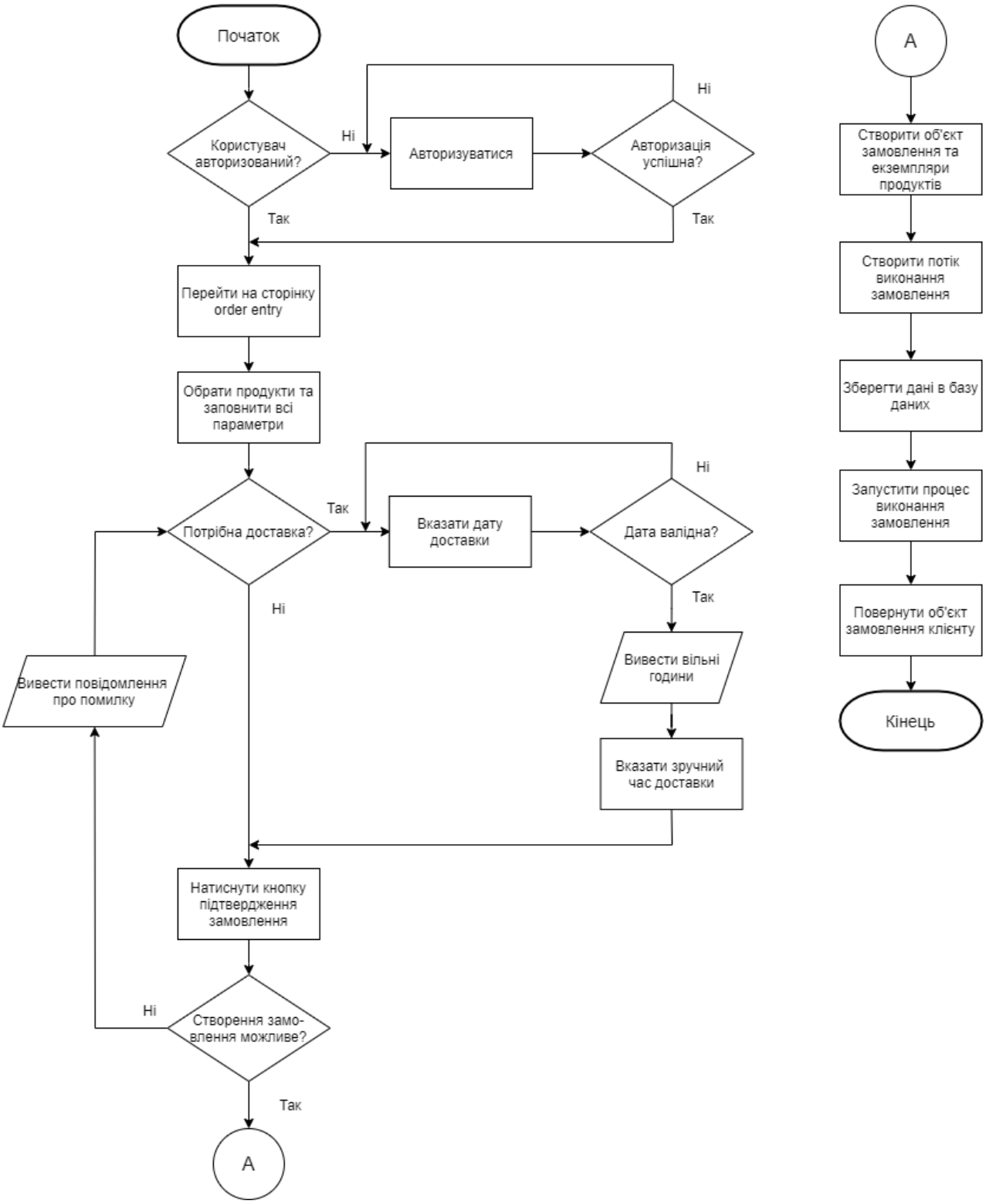
ДОДАТОК З

Автоматизована система управління замовленнями та ресурсами для телекомунікаційних компаній

Схема принципова: схема алгоритму створення замовлення

Аркушів 1

2021



					ІАЛЦ.467100.005 ДЗ			
						Літ.	*Маса.*	*Масш.*
Змн	Арк.	№ докум.	*Підпис*	*Дата*	Схема принципова Схема алгоритму створення замовлення			
Розроб.		Комісаров І.А.				*Аркуш*	*1*	*Аркушів* 1
Перевір.		Ружевський М.С.						
Т. Контр.								
Н. Контр.		Сімоненко В.П.			Дипломна робота	НТУУ “КПІ” ФІОТ ІО-71		
Затверд.		Стіренко С.Г.						

ДОДАТОК 4

Автоматизована система управління замовленнями та ресурсами для телекомунікаційних компаній

Лістинг програми

Аркушів 20

2021

order-service

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

import edu.baoss.orderservice.actions.provisioning.ProvisioningAction;
import edu.baoss.orderservice.dtos.DeliveryDto;
import edu.baoss.orderservice.dtos.OrderValue;
import edu.baoss.orderservice.exceptions.NoEmployeeFoundException;
import edu.baoss.orderservice.model.*;
import edu.baoss.orderservice.model.enums.DeliveryStatus;
import edu.baoss.orderservice.model.enums.OrderStatus;
import edu.baoss.orderservice.repositories.DeliveryRepository;
import edu.baoss.orderservice.repositories.EmployeeRepository;
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.core.AmqpTemplate;
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.services.Constants.onlyDateFormat;

@Service
public class DeliveryService {
    @Autowired
    DeliveryRepository deliveryRepository;
    @Autowired
    EmployeeRepository employeeRepository;
    @Autowired
    InstanceService instanceService;
    @Autowired
    AmqpTemplate amqpTemplate;

    @Autowired
    FulfilmentService fulfilmentService;
    @Autowired
    FlowComposer flowComposer;

    final int[] HOURS =  IntStream.rangeClosed(10, 18).toArray();
    SimpleDateFormat dateHourFormat = new SimpleDateFormat("dd/MM/yyyy HH");
    SimpleDateFormat onlyHourFormat = new SimpleDateFormat("HH");

    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 = dateHourFormat.parse(orderValue.getDeliveryDateStr() + " " +
hour);
        List<Employee> allEmployees = employeeRepository.findAllByPosition("Fitter");
        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(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);

            System.out.println(employee);
            Date now = new Date();
            List<DeliveryDto> deliveriesForEmployee = new ArrayList<>();
            System.out.println(employee.getDeliveries());
            for (Delivery delivery : new ArrayList<>(employee.getDeliveries())) {
                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);
                    }
                    System.out.println(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("Fitter");
            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);
            //amqpTemplate.convertAndSend(Constants.CONTINUE_ORDER_FULFILMENT_QUEUE,
orderValue);
            List<ProvisioningAction> executionFlowAfterDelivery =
flowComposer.getActionsAfterDelivery(orderValue);
            fulfilmentService.continueFulfilment(executionFlowAfterDelivery);
            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(onlyHourFormat.format(plannedDate));
        }

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


        private 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 String getAdditionalInfo(Delivery delivery) {
            Instance instance = delivery.getOrder().getInstance();
            return instanceService.getSimCardNumberIfPresentMobileProduct(instance) +
                    instanceService.getCableLengthIfPresentInternetProduct(instance) +
                    instanceService.getDeviceSerialNumberIfPresentInternetProduct(instance);

        }

        private boolean hasProduct(String product, String[] products) {
            for (String product_: products) {
                if (product_.equals(product)) return true;
            }
            return false;
        }

        private boolean isDeliveryStarted(Delivery delivery) {
            Date now = new Date();
            return delivery.getStatus().equals(DeliveryStatus.NOT_STARTED) &&
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 onlyDateFormat.format(date1).equals(onlyDateFormat.format(date2));
        }
    }
```

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

    import edu.baoss.orderservice.dtos.DtvProductInstance;
    import edu.baoss.orderservice.dtos.InternetProductInstance;
    import edu.baoss.orderservice.dtos.MobileProductInstance;
    import edu.baoss.orderservice.feign.OfferServiceFeignProxy;
    import edu.baoss.orderservice.feign.ResourceServiceFeignProxy;
    import edu.baoss.orderservice.model.DtvProduct;
    import edu.baoss.orderservice.model.Instance;
    import edu.baoss.orderservice.model.InternetProduct;
    import edu.baoss.orderservice.model.MobileProduct;
    import org.springframework.beans.factory.annotation.Autowired;
    import org.springframework.stereotype.Service;

    @Service
    public class ProductConverter {
        @Autowired
        ResourceServiceFeignProxy resourceServiceFeignProxy;
        @Autowired
        OfferServiceFeignProxy offerServiceFeignProxy;


        public MobileProductInstance convertMobileProductToProductInstance(MobileProduct
mobileProduct, boolean isShort) {
            Instance instance = mobileProduct.getInstance();
            MobileProductInstance mobileProductInstance = new MobileProductInstance();
            mobileProductInstance.setId(mobileProduct.getId());
            mobileProductInstance.setProduct(Constants.MOBILE_PRODUCT_STR);
            mobileProductInstance.setAllParams(instance);
            if (isShort) {
                return mobileProductInstance;
            }

mobileProductInstance.setPhoneNumber(resourceServiceFeignProxy.getPhoneNumber(mobileProduct.ge
tPhoneNumber()));

mobileProductInstance.setTariff(offerServiceFeignProxy.getTariffById(mobileProduct.getTariffId
()));
            mobileProductInstance.setSupport5g(mobileProduct.isSupport5g());
            mobileProductInstance.setBalance(mobileProduct.getBalance());
            return mobileProductInstance;
        }

        public InternetProductInstance convertInternetProductToProductInstance(InternetProduct
internetProduct, boolean isShort) {
            Instance instance = internetProduct.getInstance();
            InternetProductInstance internetProductInstance = new InternetProductInstance();
            internetProductInstance.setId(internetProduct.getId());
            internetProductInstance.setProduct(Constants.INTERNET_PRODUCT_STR);
            internetProductInstance.setAllParams(instance);
            if (isShort) {
                return internetProductInstance;
            }
            internetProductInstance.setAddress(internetProduct.getAddress());
            internetProductInstance.setDevice(internetProduct.getDeviceId() == null
                    ? null : resourceServiceFeignProxy.getDetailDeviceById(
                    internetProduct.getDeviceId(), internetProduct.getAddress().getId()));
            internetProductInstance.setCableLen(internetProduct.getCableLen());
            internetProductInstance.setFixedIp(internetProduct.getFixedIp());

internetProductInstance.setInternetOffer(offerServiceFeignProxy.getInternetOfferById(internetP
roduct.getInternetOfferId()));
            return internetProductInstance;
        }
```

```
        public DtvProductInstance convertDtvProductToProductInstance(DtvProduct dtvProduct,
    boolean isShort) {
            Instance instance = dtvProduct.getInstance();
            DtvProductInstance dtvProductInstance = new DtvProductInstance();
            dtvProductInstance.setId(dtvProduct.getId());
            dtvProductInstance.setProduct(Constants.DTV_PRODUCT_STR);
            dtvProductInstance.setAllParams(instance);
            if (isShort) {
                return dtvProductInstance;
            }

    dtvProductInstance.setDtvOffer(offerServiceFeignProxy.getDtvOfferById(dtvProduct.getDtvOfferId
    ()));
            dtvProductInstance.setInternetProductId(dtvProduct.getInternetProduct().getId());
            return dtvProductInstance;
        }
    }
```

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

    import edu.baoss.orderservice.actions.provisioning.AbstractProvisioningAction;
    import edu.baoss.orderservice.actions.provisioning.ProvisioningAction;
    import edu.baoss.orderservice.dtos.OrderDto;
    import edu.baoss.orderservice.dtos.OrderValue;
    import edu.baoss.orderservice.dtos.OrderWithInstances;
    import edu.baoss.orderservice.dtos.TaskDto;
    import edu.baoss.orderservice.model.Instance;
    import edu.baoss.orderservice.model.Order;
    import edu.baoss.orderservice.model.Task;
    import edu.baoss.orderservice.model.User;
    import edu.baoss.orderservice.model.enums.OrderStatus;
    import edu.baoss.orderservice.repositories.OrderRepository;
    import org.springframework.amqp.core.AmqpTemplate;
    import org.springframework.beans.factory.annotation.Autowired;
    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.Set;
    import java.util.stream.Collectors;

    @Service
    public class OrderService implements Serializable {
        @Autowired
        FeasibilityCheck feasibilityCheck;
        @Autowired
        AmqpTemplate amqpTemplate;
        @Autowired
        FlowComposer flowComposer;
        @Autowired
        OrderRepository orderRepository;
        @Autowired
        DeliveryService deliveryService;
        @Autowired
        InstanceService instanceService;
        @Autowired
        FulfilmentService fulfilmentService;

        @Transactional
        public Order orderEntryPoint(OrderValue orderValue) throws ParseException {
            feasibilityCheck.feasibilityCheck(orderValue);
            Order order = createOrderAndInstance(orderValue);
            orderValue.setOrder(order);
```

```
            //throw new RuntimeException();
            //amqpTemplate.convertAndSend(Constants.START_ORDER_FULFILMENT_QUEUE, orderValue);
            List<ProvisioningAction> 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> getAllOrdersForUser(long userId) {
            return 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.dtos.*;
    import edu.baoss.orderservice.feign.ResourceServiceFeignProxy;
    import edu.baoss.orderservice.model.*;
    import edu.baoss.orderservice.model.enums.InstanceStatus;
    import edu.baoss.orderservice.repositories.DtvProductRepository;
    import edu.baoss.orderservice.repositories.InstanceRepository;
```

```
import edu.baoss.orderservice.repositories.InternetProductRepository;
import edu.baoss.orderservice.repositories.MobileProductRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

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

@Service
public class InstanceService {
    @Autowired
    InstanceRepository instanceRepository;
    @Autowired
    MobileProductRepository mobileProductRepository;
    @Autowired
    InternetProductRepository internetProductRepository;
    @Autowired
    DtvProductRepository dtvProductRepository;
    @Autowired
    ProductConverter productConverter;
    @Autowired
    ResourceServiceFeignProxy resourceServiceFeignProxy;

    @Transactional
    Instance createInstances(OrderValue orderValue) {
        Instance instance = instanceRepository.save(Instance.builder()
                .userId(orderValue.getUserId())
                .status(InstanceStatus.ENTERING)
                .billingAccountId(orderValue.getSelectedAccount().getId())
                .build());
        System.out.println(instance);
        InternetProduct internetProduct = null;
        for (String product : orderValue.getSelectedProducts()) {
            if (Constants.MOBILE_PRODUCT_STR.equals(product))
                mobileProductRepository.save(MobileProduct.builder()
                        .instance(instance)
                        .balance(100)
                        .support5g(orderValue.isSupport5g())
                        .phoneNumber(orderValue.getSelectedPhoneNumber().getPhoneNumber())
                        .tariffId(orderValue.getSelectedTariff().getId())
                        .build());
            if (Constants.INTERNET_PRODUCT_STR.equals(product))
                internetProduct = internetProductRepository.save(
                        InternetProduct.builder()
                        .cableLen(orderValue.getCableLength())
                        .internetOfferId(orderValue.getSelectedSpeed().getId())
                        .address(orderValue.getSelectedAddress())
                        .instance(instance)
                        .build());
            if (Constants.DTV_PRODUCT_STR.equals(product)) {
                dtvProductRepository.save(DtvProduct.builder()
                        .dtvOfferId(orderValue.getSelectedChannelNumber().getId())
                        .instance(instance)
                        .internetProduct(internetProduct)
                        .build());
            }
        }
        return instance;
    }

    public List<ProductInstance> getAllProductInstancesForUser(long userId) {
        return instanceRepository.getInstancesByUserId(userId).stream()
                .flatMap(instance -> getProductInstancesForInstance(instance,
true).stream())
                .collect(Collectors.toList());
    }
```

```
        public List<ProductInstance> getProductInstancesForInstance(Instance instance, boolean
isShort) {
            List<ProductInstance> productInstances = new ArrayList<>();
            mobileProductRepository.getMobileProductByInstance(instance)
                    .ifPresent(mobileProduct -> productInstances.add(productConverter
                            .convertMobileProductToProductInstance(mobileProduct, isShort)));
            internetProductRepository.getInternetProductByInstance(instance)
                    .ifPresent(internetProduct -> productInstances.add(productConverter
                            .convertInternetProductToProductInstance(internetProduct,
isShort)));
            dtvProductRepository.getDtvProductByInstance(instance)
                    .ifPresent(dtvProduct -> productInstances.add(productConverter
                            .convertDtvProductToProductInstance(dtvProduct, isShort)));
            return productInstances;
        }

        public String getSimCardNumberIfPresentMobileProduct(Instance instance) {
            MobileProduct mobileProduct =
mobileProductRepository.getMobileProductByInstance(instance).orElse(null);
            if (mobileProduct == null) {
                return "\n";
            }
            return "SIM Card Number: "+
resourceServiceFeignProxy.getPhoneNumber(mobileProduct.getPhoneNumber()).getSimCardNumber() +
"\n";
        }

        public String getDeviceSerialNumberIfPresentInternetProduct(Instance instance) {
            InternetProduct internetProduct =
internetProductRepository.getInternetProductByInstance(instance).orElse(null);
            if (internetProduct == null || internetProduct.getDeviceId() == null) {
                return "\n";
            }
            UserDevice device =
resourceServiceFeignProxy.getReservedDeviceByDeviceId(internetProduct.getDeviceId());
            return "Device Serial Number: " + device.getSerialNumber() + "\n" +
                   "Device MAC Address: " + device.getMacAddress() + "\n";
        }

        public String getCableLengthIfPresentInternetProduct(Instance instance) {
            InternetProduct internetProduct =
internetProductRepository.getInternetProductByInstance(instance).orElse(null);
            if (internetProduct == null) {
                return "\n";
            }
            return "Cable length: " + internetProduct.getCableLen() + "\n";
        }

        public MobileProductInstance getMobileProductInstance(long mobileProductId) {
            return mobileProductRepository.findById(mobileProductId)
                    .map(mobileProduct ->
productConverter.convertMobileProductToProductInstance(mobileProduct, false))
                    .orElse(null);
        }

        public InternetProductInstance getInternetProductInstance(long internetProductId) {
            return internetProductRepository.findById(internetProductId)
                    .map(internetProduct ->
productConverter.convertInternetProductToProductInstance(internetProduct, false))
                    .orElse(null);
        }

        public DtvProductInstance getDtvProductInstance(long dtvProductId) {
            return dtvProductRepository.findById(dtvProductId)
                    .map(dtvProduct ->
productConverter.convertDtvProductToProductInstance(dtvProduct, false))
                    .orElse(null);
```

```
        }

        public void setDeviceIdToInternetProduct(OrderValue orderValue) {
internetProductRepository.getInternetProductByInstance(orderValue.getOrder().getInstance())
                    .ifPresent(internetProduct -> {
                        System.out.println(internetProduct);
                        internetProduct.setDeviceId(orderValue.getSelectedDevice().getId());
                        internetProductRepository.save(internetProduct);
                    });
        }

        public void setFixedIpToInternetProduct(String fixedIp, OrderValue orderValue) {
internetProductRepository.getInternetProductByInstance(orderValue.getOrder().getInstance())
                    .ifPresent(internetProduct -> {
                        internetProduct.setFixedIp(fixedIp);
                        internetProductRepository.save(internetProduct);
                    });
        }

        public Instance saveInstance(Instance instance) {
            return instanceRepository.save(instance);
        }
    }
```

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


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

    import java.util.List;

    @Service
    public class FulfilmentService {
        public void startFulfilment(List<ProvisioningAction> executionFlow) {
            for (ProvisioningAction action: executionFlow) {
                Task task = action.getTask();
                if (isOneOf(task.getType(),
                            TaskType.PENDING_FITTER_DELIVERYMAN_CONFIRMATION_TASK,
                            TaskType.PENDING_MAC_ADDRESS_TASK,
                            TaskType.PENDING_USER_CONFIRMATION_TASK)) {
                    task.setStatus(TaskStatus.WAITING);
                    action.saveTask();
                    break;
                }
                action.execute();
            }
        }

        public void continueFulfilment(List<ProvisioningAction> executionFlowAfterDelivery) {
            for (ProvisioningAction action: executionFlowAfterDelivery) {
                action.execute();
            }
        }

        private boolean isOneOf(TaskType type, TaskType ... taskTypes) {
            for (TaskType taskType: taskTypes) {
                if (taskType.equals(type))
                    return true;
            }
            return false;
        }
    }
```

```
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.WaitingDelivery manConfirmationAction;
import edu.baoss.orderservice.dtos.OrderValue;
import edu.baoss.orderservice.feign.BillingServiceFeignProxy;
import edu.baoss.orderservice.feign.ResourceServiceFeignProxy;
import edu.baoss.orderservice.model.Task;
import edu.baoss.orderservice.model.enums.TaskType;
import edu.baoss.orderservice.repositories.TaskRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

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

@Service
public class FlowComposer {
    @Autowired
    TaskRepository taskRepository;
    @Autowired
    ResourceServiceFeignProxy resourceServiceFeignProxy;
    @Autowired
    BillingServiceFeignProxy billingServiceFeignProxy;
    @Autowired
    InstanceService instanceService;
    @Autowired
    OrderService orderService;
    @Autowired
    AddressService addressService;

    public List<ProvisioningAction> createExecutionFlow(OrderValue orderValue) {
        List<ProvisioningAction> executionFlow = new ArrayList<>();
        for (ProvisioningAction action: createAllActions(orderValue)) {
            if (action.instantiationCondition()) {
                executionFlow.add(action);
                action.setTask(action.saveTask());
            }
        }
        return executionFlow;
    }

    public List<ProvisioningAction> getActionsAfterDelivery(OrderValue orderValue) {
        List<ProvisioningAction> executionFlowAfterDelivery = new ArrayList<>();
        for (ProvisioningAction action: createActionsAfterDelivery()) {
            System.out.println(orderValue.getOrder().getTasks());
            for (Task task: orderValue.getOrder().getTasks()) {
                if (task.getActionName().equals(action.getClass().getName())) {
                    action.setTask(task);
                    action.setOrderValue(orderValue);
                    executionFlowAfterDelivery.add(action);
                    break;
                }
            }
        }
        return executionFlowAfterDelivery;
    }

    private List<ProvisioningAction> createAllActions(OrderValue orderValue) {
        List<ProvisioningAction> allActions = new ArrayList<>();
        allActions.add(new InitializeOrderAction(orderValue, taskRepository));
```

```
            allActions.add(new ReserveSimCardAction(orderValue, taskRepository,
resourceServiceFeignProxy));
            allActions.add(new ReserveCableAction(orderValue, taskRepository,
resourceServiceFeignProxy));
            allActions.add(new ReserveDeviceAction(orderValue, taskRepository,
resourceServiceFeignProxy, instanceService));
            allActions.add(new NotifyUserAction(orderValue, taskRepository));
            allActions.add(new NotifyFitterOrDeliverymanAction(orderValue, taskRepository));
            allActions.add(new Activate5GAction(orderValue, taskRepository));
            allActions.add(new WaitingFitterConfirmationAction(orderValue, taskRepository,
resourceServiceFeignProxy, addressService));
            allActions.add(new WaitingDeliverymanConfirmationAction(orderValue, taskRepository,
resourceServiceFeignProxy, addressService));
            allActions.add(new WaitingMacAddressAction(orderValue, taskRepository,
resourceServiceFeignProxy, addressService));
            allActions.add(new ConnectToNetworkAction(orderValue, taskRepository));
            allActions.add(new ActivateDtvChannelsAction(orderValue, taskRepository));
            allActions.add(new ProvideFixedIpAction(orderValue, taskRepository
,resourceServiceFeignProxy, instanceService));
            allActions.add(new GetPaymentAction(orderValue, taskRepository,
billingServiceFeignProxy));
            allActions.add(new CompleteOrderAction(orderValue, taskRepository, orderService,
instanceService));
            return allActions;
        }

        public List<ProvisioningAction> createActionsAfterDelivery() {
            List<ProvisioningAction> actions = new ArrayList<>();
            actions.add(new WaitingFitterConfirmationAction(taskRepository,
resourceServiceFeignProxy, addressService));
            actions.add(new WaitingDeliverymanConfirmationAction(taskRepository,
resourceServiceFeignProxy, addressService));
            actions.add(new WaitingMacAddressAction(taskRepository, resourceServiceFeignProxy,
addressService));
            actions.add(new ConnectToNetworkAction(taskRepository));
            actions.add(new ActivateDtvChannelsAction(taskRepository));
            actions.add(new ProvideFixedIpAction(taskRepository ,resourceServiceFeignProxy,
instanceService));
            actions.add(new GetPaymentAction(taskRepository, billingServiceFeignProxy));
            actions.add(new CompleteOrderAction(taskRepository, orderService,
instanceService));
            return actions;
        }


        private boolean isOneOf(TaskType type, TaskType ... taskTypes) {
            for (TaskType taskType: taskTypes) {
                if (taskType.equals(type))
                    return true;
            }
            return false;
        }
    }

    package edu.baoss.orderservice.services;

    import edu.baoss.orderservice.actions.checking.CheckAction;
    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.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.dtos.OrderValue;
    import edu.baoss.orderservice.feign.ResourceServiceFeignProxy;
```

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

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

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

@Service
public class FeasibilityCheck {
    @Autowired
    AddressService addressService;
    @Autowired
    ResourceServiceFeignProxy resourceServiceFeignProxy;

    List<CheckAction> mobileCheckActionList = Arrays.asList(
            new CommonMobileCheckAction(), new SimCardAvailabilityCheckAction());
    List<CheckAction> internetCheckActionList = new ArrayList<>(Arrays.asList(
            new CommonInternetCheckAction(), new DeviceAvailabilityCheckAction()));
    List<CheckAction> commonCheckActionList = new ArrayList<>(Arrays.asList(
            new CommonCheckAction(), new PriceCheckAction(), new
NrcPaymentAvailabilityCheckAction()));

    public void feasibilityCheck(OrderValue orderValue) {
        internetCheckActionList.add(new HomeConnectionToNetworkCheckAction(addressService,
resourceServiceFeignProxy));
        for (String product: orderValue.getSelectedProducts()) {
            if (MOBILE_PRODUCT_STR.equals(product)) {
                commonCheckActionList.addAll(mobileCheckActionList);
            }
            if (INTERNET_PRODUCT_STR.equals(product)) {
                commonCheckActionList.addAll(internetCheckActionList);
            }
            if (DTV_PRODUCT_STR.equals(product)) {
                commonCheckActionList.add(new CommonDtvCheckAction());
            }
        }
        checkForProduct(commonCheckActionList, orderValue);
    }

    private void checkForProduct(List<CheckAction> checkActionList, final OrderValue
orderValue) {
        for (CheckAction checkAction: checkActionList) {
            checkAction.check(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.IOffer;
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 IOffer> offers) {
            Discount totalDiscount = discountService.getTotalDiscount(getProductName());
            return offers.stream()
                    .sorted(Comparator.comparingDouble(IOffer::getRent).reversed())
                    .map(offer -> getOfferDto(offer, totalDiscount))
                    .collect(Collectors.toList());
        }

        public OfferDto getOfferDto(IOffer 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()));
            return offerDto;
        }

        public abstract String getProductName();

        public abstract OfferDto createConcreteDto(IOffer offer);

    }
```

order-entry-component.ts

```
    import {Component, OnInit, Pipe, PipeTransform} from '@angular/core';
    import {
        Address,
        BillingAccount,
        Cable,
        ConstantPrices,
        Device,
        Discount,
        DtvOffer,
        InternetOffer, OrderValue,
        PhoneNumber,
        Tariff
    } from '../_models/interface';
    import { constants } 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: constants.MOBILE_PRODUCT, active: false, check: false},
        {id: 1, name: constants.INTERNET_PRODUCT, active: false, check: false},
        {id: 2, name: constants.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 = false;
    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;
    billingAccounts: BillingAccount[];
    selectedAccount: BillingAccount;
    homeDelivery = true;
    storeAddress = 'Kyiv, Kopernika, 1';
    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 => this.toaster.error(error.error.message));
            this.userService.getBillingAccountForUser(this.authService.currentUserValue.login)
                .subscribe(data => {
                    if (data.length === 0) {
                        this.noBillingAccountAction('You don\'t have any billing accounts.\n
Please add some account.');
                    } else {
                        this.billingAccounts = data;
                        this.selectedAccount = data[0];
                    }},
                    error => this.noBillingAccountAction(error.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,
                                    selectedAccount: this.selectedAccount,
                                    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.router.navigate(['/homeath/orders-and-instances']),
                        error => this.toaster.error(error.error.message));
            }
        }

        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 (let item of selectedProducts) {
                if (item == constants.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 == constants.INTERNET_PRODUCT) {
                    if (this.selectedSpeed == null) { return 'Please select speed of Internet';
}
                    if (this.selectedAddress == null) { return 'Please select address for
installation'; }
```

```
                }
                if (item == constants.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 (let 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();
        }
    }
```