Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем факультету інформатики

РОЗРОБКА ASP.NET CORE ЗАСТОСУНКУ НА ОСНОВІ МІКРОСЕРВІСІВ

Текстова частина до курсової роботи

за спеціальністю „Комп’ютерні науки” 122

Керівник курсової роботи

к.т.н., Борозенний С.О.

_____

“____” _____ 2021 р.

Виконала студентка Тютюн

Марина Миколаївна

“____” _____ 2021 р.

Київ 2021

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем факультету інформатики

ЗАТВЕРДЖУЮ

старший викладач _____

Борозенний С.О. (підпис)

(підпис)

„____”_____2021 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студенту_____ факультету _____ _____курсу

ТЕМА «РОЗРОБКА ASP.NET CORE ЗАСТОСУНКУ НА ОСНОВІ МІКРОСЕРВІСІВ»

Вихідні дані: програмний застосунок.

Зміст ТЧ до курсової роботи:

Індивідуальне завдання

Вступ

1 Огляд веб-застосунків для доставки піци, аналіз архітектурних рішень.

2 Аналіз задачі та засобів розробки.

3 Розробка продукту.

Висновки

Список літератури

Додатки

Дата видачі „___” _____ 2020 р. Керівник _____ (підпис)

Завдання отримав _____ (підпис)

Календарний план

Тема: «Розробка ASP.NET Core застосунку на основі мікросервісів».

Календарний план виконання роботи:

№ п/п	Назва етапу курсової роботи	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу	21.10.2020	
2.	Огляд технічної та наукової літератури за темою роботи.	29.11.2020	
3.	Виконати аналіз аналогів системи.	06.12.2020	
4.	Розробка алгоритму та структури програми.	27.12.2020	
5.	Розробка програмного продукту.	27.01.2021	
6.	Тестування програмного продукту, заповнення бази тестових даних	06.02.2021	
7.	Виконати перевірку роботи програми на відповідність визначеним вимогам.	20.02.2021	
8.	Написання пояснювальної роботи.	28.03.2021	
9.	Створення презентації для захисту роботи.	31.03.2021	
10.	Аналіз отриманих результатів з керівником.	02.04.2021	
11	Корегування роботи згідно зауважень керівника	07.04.2021	
12.	Остаточне оформлення пояснювальної роботи та презентації.	12.04.2021	
13.	Захист курсової роботи	19.04.2021	

Зміст

Вступ

В сучасних реаліях все більше і більше бізнесу переходить в онлайн: люди замовляють одяг, техніку і навіть їжу через Інтернет. Великою популярністю користуються служби доставки піци – люди замовляють доставку страв в офіс чи додому, до простого прийому їжі чи до святкового столу. Більше того, люди стають постійними покупцями і мають все більше вимог до улюблених магазинів. Задля виділення сервісу серед конкурентів і утримання клієнтів необхідно використовувати інструменти для заохочення, наприклад, бонусні системи, накопичення тощо. Також важливим є розробка надійного та зручного застосунку для задоволення всіх вимог.

Метою курсової роботи є створення веб-застосунку для служби доставки піци на основі мікросервісів. Він полегшить роботу менеджерів та операторів сайту та зробить приємнішим досвід клієнта.

Завданням роботи є розробка застосування, що дасть можливість менеджерам додавати різноманітні акції, оновлювати каталог товарів та ресторанів, операторам швидко обробляти замовлення та скарги клієнтів, а самим клієнтам – переглядати товари, робити замовлення та відслідковувати їх статус. Під час роботи будуть розглянуті основні архітектурні стилі при розробці програмного забезпечення, досліджені аналогічні системи, виявлені виявлені ключові моменти при створенні мікросервісної архітектури та розроблена схема застосунку. В якості джерел дослідження виступатимуть наукові статті, веб-сайти аналогів, технічна документація обраного програмного забезпечення.

Робота складається з трьох розділів.

Перший розділ присвячено обґрунтуванню актуальності задачі, огляду аналогів та виявлення ключових моментів. Будуть визначені функціональні вимоги до задачі.

Другий розділ присвячено аналізу архітектурних стилів, наведені теоретичні відомості про задачу і підходу до її реалізації.

Третій розділ присвячено проектуванню власного рішення, опису алгоритму та структури проекту, наведено обґрунтування вибору засобів розробки. Описано проектування бази даних та процес розробки програми. Показані результати тестування програми та результати виконання.

Перелік умовних позначень

СКБД – система керування базами даних.

ID (від англ. – Identity Document) – унікальний ідентифікатор об’єкта, в таблицях баз даних виступає як первинний ключ.

SQL (від англ. – Structured Query Language) – декларативна мова структурованих запитів.

ANSI (від англ. – American National Standards Institute) – об'єднання американських груп для розробки стандартів, член ISO.

ORM (від англ. – Object-Relational Mapping) – об'єктно-реляційна проекція.

REST – підхід до архітектури мережевих протоколів, які надають доступ до інформаційних ресурсів.

gRPC (від англ. – Remote Procedure Calls) – система віддаленого виклику процедур з відкритим вихідним кодом.

SOAP (від англ. – Simple Object Access Protocol) – протокол обміну структурованими повідомленнями в розподілених обчислювальних системах, який базується на форматі XML.

AMQP (від англ. – Advanced Message Queuing Protocol) – відкритий стандарт протоколу прикладного рівня для проміжного програмного забезпечення, що орієнтоване на обробку повідомлень.

SPA (від англ. – single-page application) – односторінковий інтерфейс.

HTTP-протокол (Hyper Text Transfer Protocol) – протокол передачі гіпертекстових документів.

Розділ 1. Аналіз предметної області. Постановка завдання курсової роботи

1. Обґрунтування актуальності задачі

Реалії останніх років, в тому числі карантинні обмеження через поширення коронавірусної інфекції, спричинили ріст попиту на онлайн-шопінг та доставку їжі у всьому світі [1] [2]. Результатом цього є збільшення вимог користувачів даних сервісів до їх надійності та роботи: багаточисленні дослідження [3] показали, що користувачі більше задоволені тим застосунком, в чиїй надійності та безпеці вони впевнені сильніше всього. Отже, при проектуванні програм ці критерії мають бути в пріоритеті. Іншим аспектом, який необхідно враховувати для того, щоб залучати та утримувати клієнтів, є наявність різноманітних бонусних програм, знижок, подарунків. Це допомагає збільшити клієнтську лояльність та підвищити рівень продаж. Проте для реалізації цього має бути залучена велика кількість різних людей, а об’єднувати їх роботу має надійна та злагоджена інформаційна система.

Мною було обрано службу доставки піци, оскільки замовлення подібної продукції надзвичайно поширене, а веб-сайти існуючих аналогів не завжди демонструють надійність та зручність. Заклади харчування, що пропонують доставку, отримують безліч замовлень щодня, мають велику клієнтську базу, тому проектування системи, що буде відповідати запитам її користувачів, є надзвичайно актуальним питанням.

1.2 Огляд існуючих аналогів розробки

Нижче будуть розглянуті веб-сайти найпопулярніших служб доставки їжі [4] для виявлення їх переваг та типових недоліків, які допоможуть створити систему, яка буде задовольняти вимоги користувачів.

Pizza House [5] – популярний сервіс для доставки їжі у Києві та передмісті, що пропонує численні страви та напої. Сторінка каталогу містить вичерпну інформацію про продукти: назву, фото, вагу, діаметр, та ціни в залежності від розміру. Є різні способи оплати (на сайті та карткою чи готівкою при отриманні) та доставки, час якої, до речі можна обирати. Інтерфейс простий і зрозумілий, проте має певні недоліки: головна сторінка сайту занадто контрастна і у ній незручно орієнтуватися та в деяких стравах назву не видно через фотографію. Іншими недоліками є відсутність історії замовлень та наявність лише двох акції: «Друга піца безкоштовно», під якою мається на увазі придбання за 20 гривень, та сертифікат на день народження розмірі 500 гривень, проте невеликі сезонні знижки могли би збільшити кількість продаж через їх вплив на покупців [6].

Adriano pizza [7] – служба доставки піци, яка доставляє по більшій частині Києва. Інтерфейс простий і зрозумілий, проте має серйозний недолік – опис кожного розділу знаходиться внизу сторінки, написаний на сірому фоні дрібним шрифтом, що є не дуже приємним на погляд. Ще одним недоліком є відсутність особистого кабінету. Каталог містить усю необхідну інформацію про товари, а при натиску на них можна прочитати детальний опис. Наявна тільки доставка, замовлення можливо оплатити як карткою, так і готівкою. При оформленні замовлення адреса пишеться в одному текстовому полі, що не дозволяє виконати її валідацію, тому, можливо, операторам доведеться її уточнювати. Наявні акції: знижка 20% на день народження, 5% кешбек, знижки 50% на другу піцу три дні на тиждень та 1% знижки при розрахунку картою Privat Bank.

Domino’s Pizza [8] – одна із популярних в Україні служб доставки піци. Веб-сайт має приємний за зручний інтерфейс, каталог, де можна вибрати розміри та деталі піци, є сортування товарів. В особистому кабінеті є загальна інформація, збережені номери та адреси, історія замовлень. Сервіс пропонує безліч знижок: для різних днів тижня, на кожну другу піцу, знижки до дня народження та для певних

продуктів, також є можливість відстежити статус замовлення. З недоліків: наявність бокового меню, що дублює інформацію з верхньої та нижньої частин сайту і перекриває частину каталогу.

Отже, після аналізу веб-сайтів різних служб доставки можна виділити наступні необхідні складові для приємного користувацького досвіду: зручний та простий інтерфейс, гарна верстка, можливість обрати спосіб оплати, наявність особистого кабінету з історією замовлень та даними для доставки. Перевагами для вибору певного сервісу також можуть бути різноманітні акції та знижки і можливість бачити актуальний статус замовлення.

1.3 Постановка задачі

Метою курсової роботи є розробка веб-застосунку для служби доставки піци. Дана система має щодня обслуговувати велику кількість користувачів, проводячи безліч операцій та передаючи дані між її складовими, забезпечувати надійність, відмовостійкість та безпеку. При подальшому вдосконалені застосунку можуть додаватися нові компоненти, а розмір даних, що зберігається, збільшуватиметься в рази.

В даній версії системою будуть користуватися клієнти, оператори, що обробляють замовлення та скарги, і менеджери, тому необхідно в ній повинні бути окремі інтерфейси для кожного типу користувача. Також повинен бути окремий розділ для адміністраторів. В ньому зберігатимуться дані про зареєстрованих користувачів та є можливість наділяти їх ролями. Для розділення відповідальності між групами користувачів має бути створена надійна авторизація та автентифікація на основі ролей.

Клієнти повинні мати змогу переглядати товари, створювати замовлення, дивитися їх історію, переглядати та редагувати власний профіль, залишати скарги. Оператори мають обробляти вхідні замовлення, оновлювати їх стан, також мають

швидко відповідати на клієнтські скарги. Менеджери мають слідкувати за актуальністю каталогу товарів, переглядати та редагувати список клієнтів, створювати бонусні кампанії та акції та залучати в них клієнтів. Адміністратори мають можливість переглядати інформацію по всіх зареєстрованих користувачах, оновлювати їх список та надавати їм ролі.

Розділ 2. Теоретичні відомості

2.1. Вибір архітектури застосунку

Як було зазначено в розділі 1.1 для служби доставки, що використовує систему заохочень для покупців, необхідна злагоджена робота системи, яку будуть використовувати і її співробітники, і клієнти. Необхідно розглянути можливі архітектурні рішення та обрати підходяще для даної задачі.

Архітектурний стиль чи шаблон – набір принципів, що забезпечує абстрактну структуру для сімейства систем. Архітектурний стиль покращує розділення застосунку на компоненти та сприяє повторному використанню певних рішень для часто повторюваних проблем [9]. Архітектурні стилі можна поділити на категорії в залежності від їх можливостей, а саме:

1) Комунікація – належать сервісно-орієнтована архітектура, Message Bus;
2) Розгортання – клієнт-серверна архітектура, трирівнева, багаторівнева;
3) Домен – предметно-орієнтоване проектування (domain-driven design, DDD);
4) Структура – компонентно-орієнтована, об’єктно-орієнтована, багатошарова архітектури.

Нижче коротко будуть розглянуті особливості, недоліки і переваги кожної з них.

2.1.1 Клієнт-серверна архітектура

Клієнт-серверна архітектура полягає у створені системи, в якій клієнт та сервер розділені та комунікують між собою за допомогою мережі [10]. Найпростіша форма, яка називається дворівневою архітектурою – серверний застосунок, до якого прямо доступаються численні клієнти. Перевагами такого стилю є вищий рівень безпеки, централізований доступ до даних та легкість підтримання. До недоліків

можна віднести нагромадження бізнес-логіки та даних на сервері, що негативно впливає на розширення системи та масштабованість, залежність від центрального сервера, що впливає на надійність.

2.1.2 N-рівнева та багатошарова архітектура

Частковим рішенням проблем клієнт-серверної архітектури є багаторівнева (зазвичай рівнів три), в якій між клієнтом та сервером є проміжний рівень (application layer), в якому зберігається бізнес-логіка застосунку або додаткові рівні обробки даних. Рівні застосунку можуть знаходитися на одному комп’ютері або розподілені між різними. Перевагами даного підходу є абстракція, ізольованість, керованість, вища продуктивність, багаторазове використання, покращення можливостей для тестування.

2.1.3 Message Bus

Архітектура Message Bus полягає у використанні програмної системи, що може надсилати і отримувати повідомлення (найчастіше асинхронно) за допомогою одного чи більше каналів зв’язку [11]. Найбільш популярні реалізації це маршрутизатор повідомлень чи шаблон Publish/Subscribe, який реалізується чергою повідомлень. Переваги стилю наступні: розширюваність, низька складність, гнучкість, слабка зв’язність, масштабованість, невеликий розмір застосунку.

2.1.4 Сервісно-орієнтована архітектура

Сервісно-орієнтована архітектура полягає у розробці функціоналу як набору сервісів, які слабо пов’язані між собою та комунікують за допомогою повідомлень. Перевагами є можливість повторного використання загальних сервісів, абстракція, відкритість, сумісність [12].

2.1.5 Предметно-орієнтоване проектування

Предметно-орієнтоване проектування – об’єктно-орієнтований підхід до розробки програмного забезпечення, що базується на бізнес-домені, його елементам, поведінці та відношеннях між ними [13]. Модель домену може розглядатися як фреймворк. Перевагами є легкість комунікації, розширюваність, легкість тестування.

2.1.6 Компонентно-орієнтований стиль

Компонентно-орієнтована архітектура – підхід до створення програмного забезпечення, що зосереджується на декомпозиції дизайну на окремі логічні чи функціональні частини, що мають чітко визначені інтерфейси, та не зосереджує увагу на протоколах зв’язку та спільному стані [14]. Переваги: легкість розгортання, зменшення вартості розробки і підтримки, повторне використання компонентів, послаблення технічної складності.

2.1.7 Об’єктно-орієнтований стиль

Об’єктно-орієнтована архітектура – це парадигма проектування, заснована на поділі системи на окремі самодостатні об’єкти, які можна повторно використовувати та кожен з яких містить власні дані та поведінку. До переваг цього підходу можна віднести легкість розуміння, багаторазовість використання, легкість тестування, розширюваність, висока згуртованість.

Отже, після перегляду деталей про основні архітектурні стилі програмного забезпечення було обрано, зважаючи на предметну область та користувачів застосунку, сервісно-орієнтований підхід, а саме його сучасну інтерпретацію – мікросервіси. В цій архітектурі сервіси – це процеси, що взаємодіють між собою через мережу для досягнення мети через різноманітні протоколи [15]. Проте у реальному житті часто не виходить використовувати лише один стиль, тому у

роботі мікросервісна архітектура буде поєднуватися з компонентно-орієнтованою, об’єктно-орієнтованою та Message Bus для взаємодії частин системи.

2.2 Особливості розробки мікросервісів

При розробці необхідно враховувати ряд правил, що дозволять побудувати автономні, слабо пов’язані та швидкі мікросервіси, які разом складатимуть великі комплексні системи з високою масштабованістю.

Кожен сервіс повинен мати власну концептуальну модель даних та власну бізнес-логіку. Так, одні і ті ж моделі даних можуть бути різними доменними моделями в різних частинах системи. Дані можуть зберігатися в окремих для кожного сервісу базах, які, в свою чергу, можуть бути різних типів – і реляційними, і нереляційними, проте інколи доцільніше використовувати одну спільну нормалізовану базу даних. Дані між сервісами можуть передаватися синхронно – через API (використовуючи REST, gRPC, SOAP тощо), чи асинхронно через повідомлення (AMQP чи подібні протоколи).

Необхідно визначити підхід, як буде здійснюватися комунікація між серверною частиною – мікросервісами, та клієнтом, тобто, фронтендом. Один вид такої взаємодії є пряма комунікація, яка зображена на рисунку 2.1.

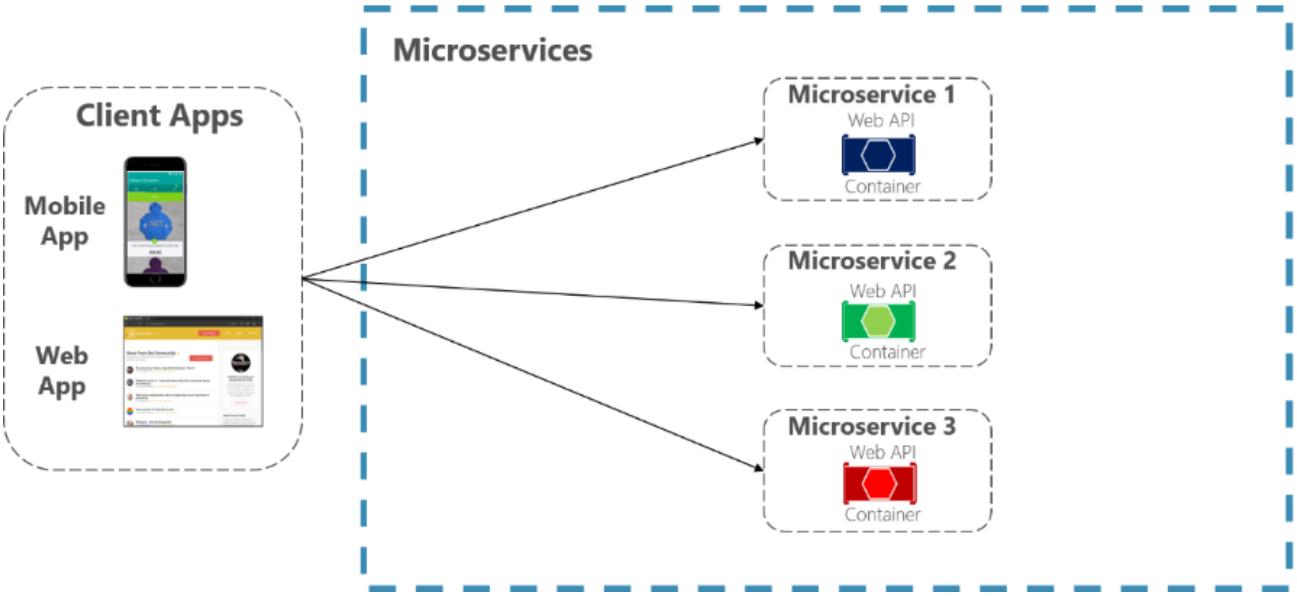


Рисунок 2.1 – Пряма взаємодія між клієнтом та мікросервісами

В цьому підході кожен сервіс має власний унікальний URL, який в продуктовому середовищі буде співставлятися з балансувальником навантаження у кластері, який, в свою чергу, розподіляє запити по мікросервісам. Пряма взаємодія між клієнтом та мікросервісами добре підходить для невеликих серверних веб-застосунків, як от ASP.NET MVC. Проте у великих складних застосунках (особливо у мобільних чи SPA) цей підхід може викликати ряд проблем: надмірна кількість запитів до мікросервісів, проблеми з авторизацією та безпекою, неможливість використання клієнтом не Інтернет-протоколів.

Вищевказану проблему можливо вирішити за допомогою підходу «API Gateway», що полягає в побудуванні проміжного рівня між клієнтом та мікросервісами, так званим «бекендом для фронтенду», принцип якого зображено на рисунку 2.2.

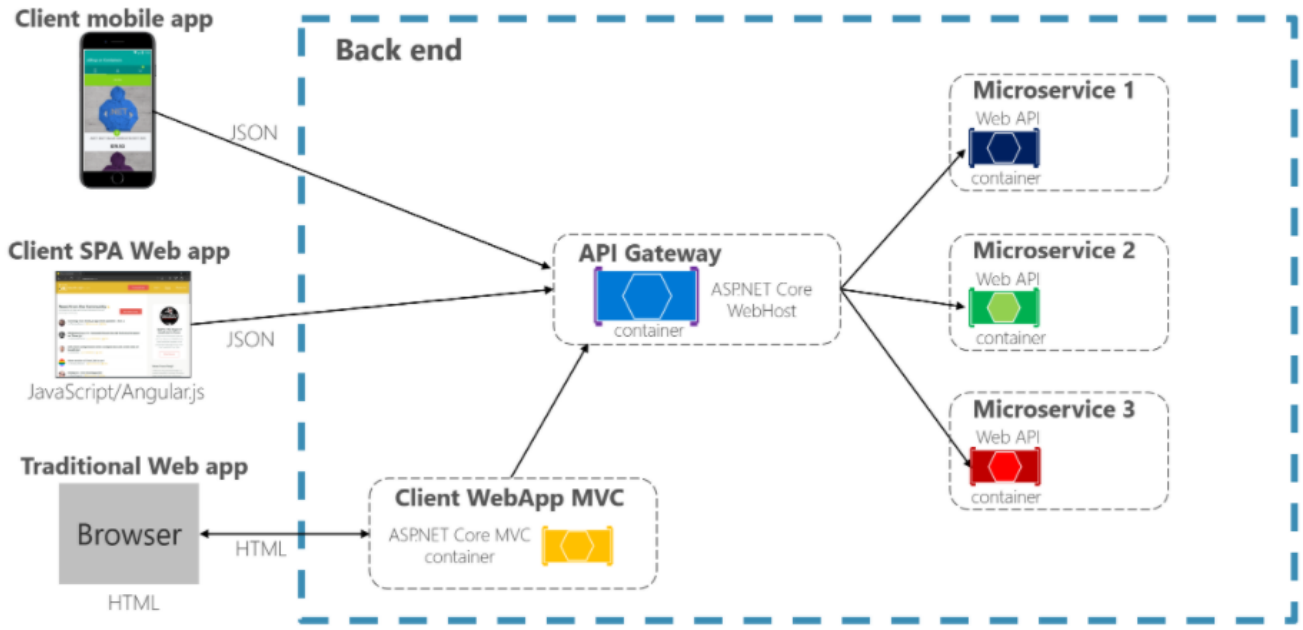


Рисунок 2.2 – API Gateway

Даний рівень діє як зворотній проксі, переправлячи запити з клієнтів на сервіси. Також в ньому можна реалізувати такий функціонал як автентифікацію, кешування тощо. При необхідності застосунок може мати декілька API Gateway рівнів.

Іншим важливим пунктом при розробці мікросервісів є сумісність спільних бібліотек, програмного забезпечення та контрактів даних, що передаються. Такі розбіжності можуть спричинити втрату можливих даних чи помилки у роботі застосунку.

Отже, для вдалої побудови застосунку необхідно проаналізувати предметну область, визначити доменні моделі даних, обрати варіанти їх зберігання, розділити систему на компоненти та визначити спосіб їх комунікації і вибрати підходяще програмне забезпечення.

Розділ 3. Опис реалізації програмного продукта

3.1 Аналіз технічного завдання

Мета курсової роботи – розробка веб-застосунку на основі мікросервісної архітектури. Для реалізації цієї задачі необхідно визначатися із користувачами, дозволеним їм функціоналом, розробити структуру проекту та бази даних.

В застосунку наявні наступні групи користувачів: клієнти, оператори, менеджери та адміністратори. Клієнти мають інтерфейси для перегляду та редагування профілю, перегляду каталогу товарів, здійснення замовлення з можливістю вибрати способи оплати та доставки; можуть переглядати свої минулі замовлення, залишати скарги і вести подальшу переписку з оператором. Оператори можуть переглядати скарги, призначати на себе і вести подальше листування з клієнтом один на один та можуть оновлювати стан замовлення на надсилати дані клієнту. Менеджери мають можливість наповнювати сайт контентом, редагувати його, переглядати та редагувати список клієнтів і список ресторанів мережі, створювати бонусні кампанії та акції і залучати в них користувачів. Адміністратори можуть бачити список всіх зареєстрованих користувачів, оновлювати їх і видаляти, надавати їм ролі та має доступ до всіх дій вищевказаних користувачів.

Після визначення технічних вимог користувачів до функціональності системи важливо виокремити основні сутності та визначити вимоги до них. Мною були визначені наступні типи даних:

- Продукт

Про продукт зберігаються унікальний номер, назва, його опис, специфічні дані та інгредієнти. Під специфічними даними маються на увазі розмір, ціна, що залежить від розміру, тип тіста та бортика, наявність алкоголю тощо. Інформацію про продукт додає, змінює та видаляє менеджер, клієнт має право тільки на перегляд.

- Користувач

Про нього зберігається унікальний номер, прізвище, ім’я, по-батькові, стать, дата народження, роль, логін та пароль. Повний доступ до цієї сутності має тільки адміністратор.

- Клієнт

В моделі клієнта дублюються всі дані, що і про користувача, окрім паролю та ролі. Право на перегляд інформації про клієнта має він сам та менеджер.

- Адреса доставки

Містить в собі унікальний номер, унікальний номер клієнта, місто, район, будинок, під’їзд, поверх та квартиру. Адреса доставки пов’язана з клієнтом і не може існувати окремо. Клієнт може мати одну чи більше адресу. Право на перегляд та оновлення адрес доставки має лише клієнт.

- Замовлення

Містить в собі унікальний номер, дату замовлення, дату та тип доставки, адресу доставки (ресторану або будинку клієнта), поточний стан готовності, дату останнього оновлення стану, сума замовлення, сума можливої знижки і остаточна сума. Замовлення пов’язане з лише одним клієнтом і не може існувати окремо. Право на перегляд замовлення та його редагування до початку виконання має лише клієнт.

- Рядок замовлення

Містить в собі унікальний номер, номер замовлення, номер продукту, кількість одиниць продукту та загальну суму. Рядок замовлення пов'язаний лише з одним замовленням, проте в замовленні може бути більше одного рядка. Право на перегляд має лише клієнт

- Ресторан

Містить в собі унікальний номер, місто, район, та номер будівлі. Право на перегляд має клієнт та менеджер, на оновлення – тільки менеджер.

- Оновлення замовлення

В даній моделі зберігається унікальний номер, номер замовлення, його поточний стан та остання дата зміни стану. Право на перегляд та оновлення цієї сутності має лише оператор, сервіс клієнта отримує інформацію про ці зміни і відповідно оновлює саме замовлення.

- Користувацька скарга

Містить в собі унікальний номер, текст скарги, номер автора, індикатор чи була вона оброблена, якщо так – номер оператора, що відгукнувся. Право на перегляд скарг мають клієнти, що їх створили, та оператори

- Приватне повідомлення

Містить в собі унікальний номер, номер скарги, номери відправника та отримувача, текст повідомлення та дату надсилання. Повідомлення пов’язане зі скаргою не може існувати окремо. Оскільки щодо однієї теми клієнт та оператор можуть спілкуватися багато, сутність скарга та повідомлення мають зв'язок «один до багатьох».

- Бонусна кампанія

Містить в собі унікальний номер, назву, опис, відсоток знижки, індикатор, чи кампанія активна, та правила відповідно до типу кампанії. Кампанії можуть надавати знижки після певної кількості замовлень, на день народження клієнта, у визначений день тижня, у певний період, за безготівкову оплату або через використання промокоду. Право на перегляд та редагування кампаній мають менеджери.

- Користувацький прогрес в кампанії

Містить номер клієнта, номер кампанії, ідентифікатор того, чи прогрес активний. Прогрес може бути пов'язаний лише з одним клієнтом та однією кампанію та служить проміжною сутність для зв’язку «багато до багатьох» між ними.

Далі, після визначення основних сутностей даних треба обрати їх сховище. В його якості було обрано реляційну СКБД через їх розповсюдженість та легкість використання. Далі має бути розроблена схема даних, що відповідатиме всім вимогам до сутностей, створені таблиці, зв'язки між ними, визначені обмеження домену, створені необхідні представлення та процедури.

Для зменшення залежності між програмними компонентами та слабої зв'язності використовується шаблон «посередник» та використовується механізм Dependency Injection. Також для інкапсуляції роботи з базою даних використовується шаблон data access object (DAO). Такі дії, як обробка API-запитів, валідація даних, реалізація бізнес-логіки, та запис в базу даних, не пов'язані між собою та мають міститися в окремих класах. Отже, проект має обов'язково містити наступні компоненти:

1) моделі даних, що відповідатимуть сутностям в БД,
2) API-контролери для обробки HTTP-запитів,
3) Класи-репозиторії для інкапсуляції доступу до даних
4) Класи запити, обробники запитів та відповіді, які реалізують шаблон «посередник» та бізнес-логіку застосунку.
5) Класи-виробники для відправлення повідомлень іншим сервісам за допомогою брокера повідомлень
6) Класи-споживачі для читання повідомлень, переданих брокером з інших сервісів
7) Класи- HTTP-клієнти для передачі даних між сервісами за допомогою API

Отже, при наявності вимог до реалізації, необхідно розробити структуру та алгоритм програми, провести аналіз та вибір засобів розробки.

3.2 Обґрунтування алгоритму та структури проекту

Після аналізу завдання та вимог до реалізації необхідно створити алгоритм роботи програми та її структуру, що буде найточніше їм відповідати.

Згідно висновків розділу 2.1 застосунок має складатися з мікросервісів для кожного типу користувача, які мають власне сховище та обмінюються даними між собою. Схема проекту виглядатиме наступним чином (рисунок 3.1).

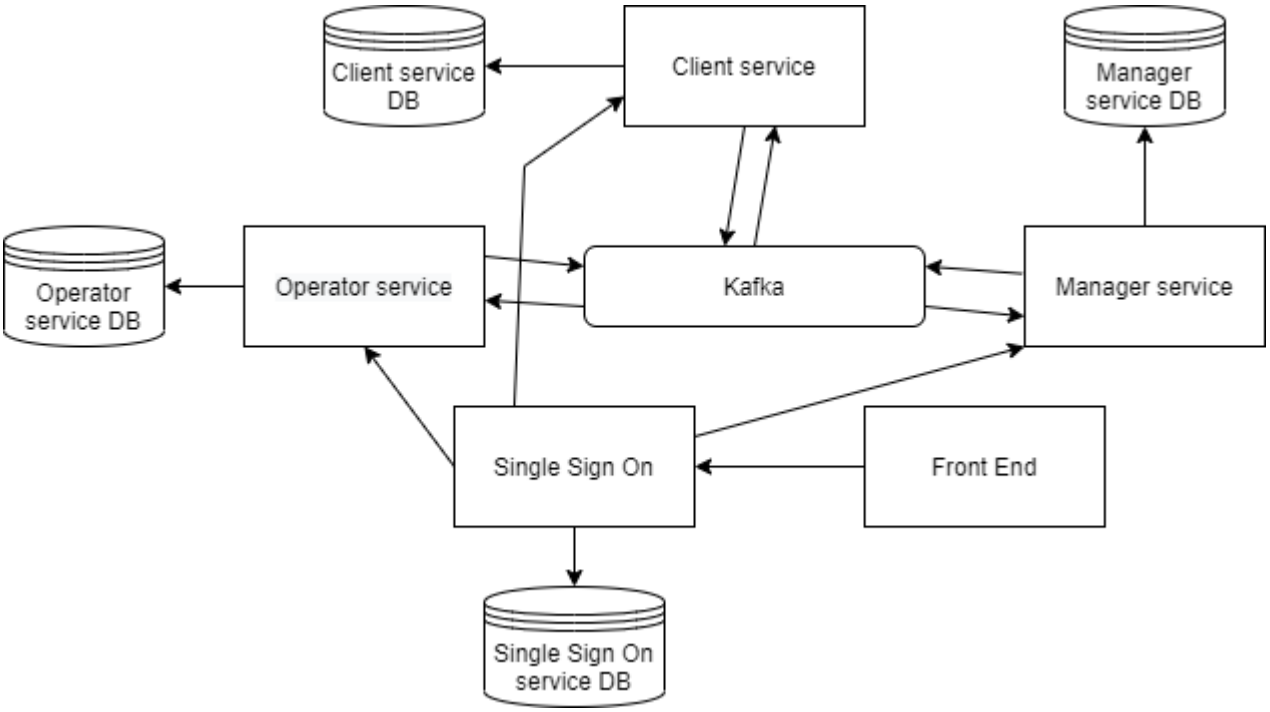


Рисунок 3.1 – Схема проекту

Кожен back end сервіс матиме доступ лише до власної бази даних, а для зв’язку один із одним обмінюватиметься повідомленнями, які зберігатимуться на кластері Kafka. Авторизація проходитиме через сервіс Single Sign On (далі – SSO). Front end показуватиме лише ті інтерфейси, які доступні для ролі автентифікованого користувача.

Далі необхідно визначити основні алгоритми, що мають бути в програмі. Оскільки даний застосунок – платформа для служби доставки піци, найголовніший workflow, що треба реалізувати – процес виконання замовлення клієнтом, який

зображено на рисунку 3.2. Він в собі поєднує взаємодію клієнта з сайтом, роботу операторів, що слідкують за процесом виконання замовлення та оновлюють статус, а також роботу менеджерів, що створюють бонусні кампанію та залучають в них клієнтів.

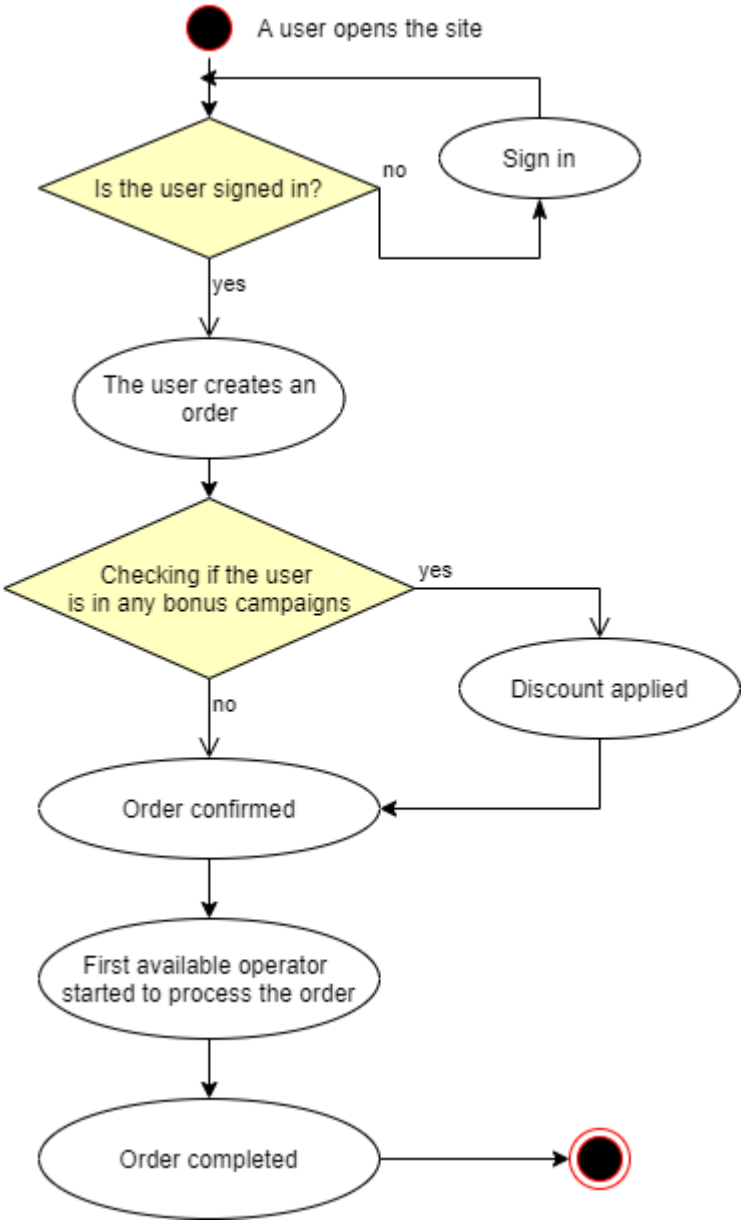


Рисунок 3.2 – Створення замовлення клієнтом

Іншою взаємодією клієнта та оператора є створення скарг та їх обробка, алгоритм роботи наведено нижче (рисунок 3.3).

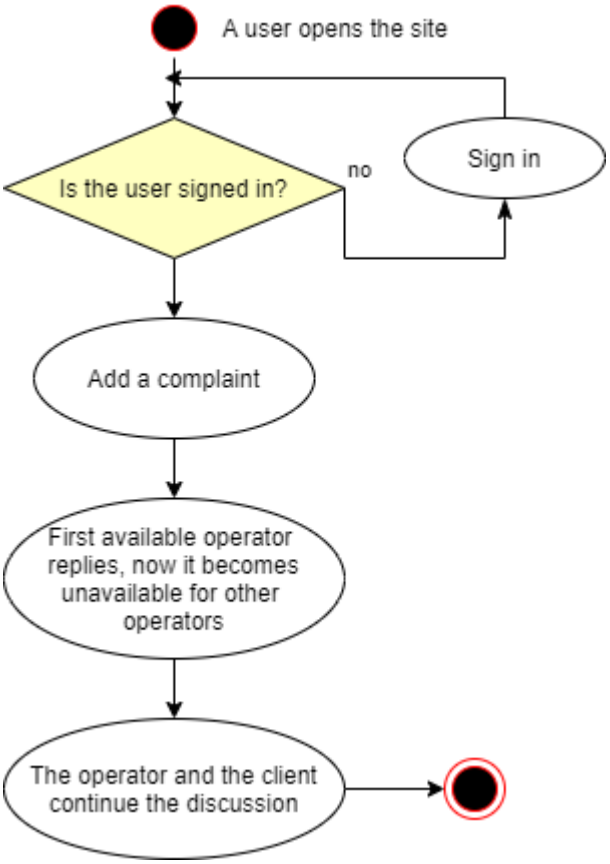


Рисунок 3.3 – Обробка скарг

3.3 Обґрунтування вибору засобів розробки

Для розробки програмного продукту було обрано платформу .NET (раніше відому як .NET Core), а саме її складову ASP.NET, базу даних PostgreSQL, та розподілений програмний брокер повідомлень Apache Kafka.

3.3.1 .NET

ASP.NET — технологія створення веб-застосунків і веб-сервісів від компанії Microsoft. Вона була вперше випущена в 2002 році з версією 1.0 .NET Framework і є спадкоємцем технології Active Server Pages від Microsoft. ASP.NET побудований на Common Language Runtime (CLR), що дозволяє програмістам писати код

ASP.NET, використовуючи будь-яку підтримувану мову .NET. В даному застосунку використовуються наступні програмні моделі ASP.NET:

1) ASP.NET Web Pages (Razor) – спеціальний синтаксис для додавання динамічного коду та доступу до даних безпосередньо всередині розмітки HTML.

2) ASP.NET Web API – фреймворк для побудови HTTP сервісів.

Платформа .NET була обрана мною через її популярність, поширеність та ряд наступних переваг:

1) Перевага ASP.NET в швидкості у порівняні з іншими технологіями, що засновані на скриптах;

2) Ефективний доступ до даних – набір компонентів ADO.NET представляє ефективний доступ до реляційних баз даних та інших джерел даних, файлових систем та каталогів.

3) Сумісність з Windows, Linux и macOS.

4) Поділ коду. Платформа .NET змінила спосіб поділу коду між додатками, ввівши компіляцію збірки замість традиційних бібліотек DLL.

5) Підвищена безпека – кожна збірка може містити в собі інформацію, яка описує, яким категоріям користувачів або процесів які методи чи класи дозволено використовувати та інші.

3.3.2 PostgreSQL

PostgreSQL – відкрита об'єктно-реляційна система управління базами даних (СКБД), існує в реалізаціях для багатьох UNIX-подібних платформ та Microsoft Windows.

PostgreSQL базується на мові SQL та підтримує більшість можливостей стандарту SQL:2011. Перевагами PostgreSQL є наступні:

1) відсутність обмежень на максимальний розмір бази даних та максимальну кількість індексів в таблиці, максимальний розмір таблиці – 32 ТБ,

поля – 1 ГБ, максимум полів у записі – 250-1600 [16], що є доволі великими значеннями.

2) Підтримка принципів ACID, відповідність стандартам ANSI SQL-92, SQL-99, SQL:2003, SQL:2011.

3) високопродуктивні і надійні механізми транзакцій і реплікації;

4) розширювана система вбудованих мов програмування: в стандартній поставці підтримуються PL/pgSQL, PL/Perl, PL/Python, PL/Tcl, додатково можна використовувати PL/Java, PL/PHP, PL/Py, PL/R, PL/Ruby, PL/Scheme, PL/sh и PL/V8, також є підтримка завантаження модулів розширення на C. [17]

5) можливість індексування геометричних об'єктів і наявність розширення PostGIS;

6) вбудована підтримка слабоструктурованих даних в форматі JSON з можливістю їх індексації;

7) розширюваність (можливість створювати нові типи даних, типи індексів, мови програмування, модулі розширення, підключати будь-які зовнішні джерела даних).

3.3.3 Apache Kafka

Apache Kafka – розподілений програмний брокер повідомлень, розроблений в рамках фонду Apache. Написаний на мовах програмування Java і Scala.

Брокер повідомлень Kafka – розподілена система. Його сервери об'єднуються в кластери. Зберігання та пересилання повідомлень йде паралельно на різних серверах, а це дає більшу надійність і відмовостійкість. Навіть при виході з ладу кількох машин, повідомлення все ще будуть пересилатися і оброблятися. Ще одною перевагою Kafka є консистентність даних: записи зберігаються у вигляді журналу комітів, що дає надійність та простоту зміни станів. Іншими його

властивостями є зберігання повідомлень на диску, горизонтальна масштабованість, реплікація.

Apache Kafka може застосовуватися для різноманітних задач розробки, а саме:

1) Зв'язку мікросервісів один між одним: сервіси можуть слати та/або отримувати повідомлення один від одного і відповідним чином їх обробляти.

2) Організація потоків даних – наприклад, при постійній передачі певних подій, які треба передавати по ланцюгу та обробляти на кожному кроці, можна використати механізм роутингу повідомлень.

3) Агрегація записів – швидкість запису даних в Apache Kafka набагато швидший, ніж у звичайну базу даних, що дозволяє організувати збір певних метрик, рахувати агрегаційні значення, і вже їх писати в БД.

4) Збір логів – є можливість зберігати повідомлення протягом певного часу, що дозволяє розвантажити базу даних та повільні системи логування.

3.3.4 Інше програмне забезпечення

Для реалізації проекту було обрано середовище розробки Microsoft Visual Studio компанії Microsoft, яка дозволяє створювати різноманітні застосунки – консольні, з графічним інтерфейсом, веб-сайти, веб-служби тощо, та включає такі компоненти: Visual C#, Visual C++, Visual Basic .NET, Visual F#. Visual Studio включає в себе редактор вихідного коду з технологією доповнення коду IntelliSense та можливістю рефакторингу, відлагоджувач рівня вихідного коду та машинного рівня. Також наявні інструменти для редагування форм, веб-редактор, дизайнер класів та схеми бази даних, використання систем контролю версій.

Для взаємодії з базою даних PostgreSQL використовується графічний клієнт для роботи з сервером pgAdmin. Він надає зручний інтерфейс для створення, редагування та видалення баз даних [18].

Для зручної взаємодії брокером повідомлень використовується Offset Explorer (раніше Kafka Tool) – інструмент з інтуїтивним користувацьким інтерфейсом, що дозволяє швидко переглядати обєкти всередині кластера та повідомлення, які зберігаються в топіках[19].

3.4 Розробка бази даних

Як зазначено вище, кожен сервіс має окрему базу даних. Почнемо з сервісу Client (додаток А), який призначено для користування клієнтами служби доставки піци.

Таблиця «Адреси доставки» представляє собою збережені адреси клієнтів для доставки, містить ID, ID клієнта та дані про поштову адресу.

Таблиця 3.1 – Адреси доставки

DeliveryAddresses				
Ключ	Ім’я атрибуту	Тип атрибуту	NULL?	Примітки
PK	Id	integer	NOT NULL	Генерується за замовчуванням як ідентифікатор
	ClientId	integer	NOT NULL	Індекс
	City	text	NOT NULL	
	District	text	NOT NULL	
	Street	text	NOT NULL	
	Building	integer	NOT NULL	
	Entrance	integer	NULL	
	Floor	integer	NULL	
	Apartment	integer	NULL	

Таблиця «Замовлення» містить загальну інформацію про замовлення клієнта, а саме ID, ID клієнта, час замовлення, тип та час доставки, поточний стан та дату його останньої зміни, тип оплати (готівкою чи картою), сума замовлення, можлива знижка та фінальна сума.

Таблиця 3.2 – Замовлення

Orders				
Ключ	Ім’я атрибуту	Тип атрибуту	NULL?	Примітки
PK	Id	integer	NOT NULL	Генерується за замовчуванням як ідентифікатор
	ClientId	integer	NOT NULL	Індекс
	AddressId	integer	NOT NULL	
	OrderTime	timestamp without time zone	NOT NULL	
	DeliveryType	text	NOT NULL	
	DeliveryTime	timestamp without time zone	NULL	
	OrderState	text	NOT NULL	
	StateChanged	timestamp without time zone	NOT NULL	
	PaymentType	text	NOT NULL	
	DiscountSum	numeric	NULL	

	OrderSum	numeric	NOT NULL	
	FinalSum	numeric	NOT NULL	

Таблиця «Рядки замовлення» показує дані про товари у замовленні, ID, ID замовлення, ідентифікатор продукту, кількість та суму рядку.

Таблиця 3.3 – Рядки замовлення

OrderLines				
Ключ	Ім’я атрибуту	Тип атрибуту	NULL?	Примітки
PK	Id	integer	NOT NULL	Генерується за замовчуванням як ідентифікатор
FK	OrderId	integer	NOT NULL	On update no action, On delete cascade. Індекс
	FoodItemId	integer	NOT NULL	
	Quantity	bigint	NOT NULL	Значення за замовчуванням – 1
	OrderLineSum	numeric	NOT NULL	

Перейдемо до сервісу Operator (додаток Б), який використовується операторами служби доставки, які обробляють замовлення та скарги.

Таблиця «Оновлення замовлення» дозволяє операторам заносити дані про актуальний стан замовлення. Містить ID, ID клієнта, інформацію про стан та час зміни.

Таблиця 3.4 – Оновлення замовлення

OrderUpdates				
Ключ	Ім’я атрибуту	Тип атрибуту	NULL?	Примітки
PK	Id	integer	NOT NULL	Генерується за замовчуванням як ідентифікатор
	OrderId	integer	NOT NULL	Індекс
	State	text	NOT NULL	
	StateChanged	timestamp without time zone	NOT NULL	

Таблиця «Скарги користувачів» містить інформацію про скарги, які залишають клієнти, містить ID, ID клієнта, текст скарги, та в разі обробки – інформацію, хто на неї відповів.

Таблиця 3.5 – Скарги користувачів

UserComplaints				
Ключ	Ім’я атрибуту	Тип атрибуту	NULL?	Примітки
PK	Id	integer	NOT NULL	Генерується за замовчуванням як ідентифікатор
	Text	text	NOT NULL	

	CreatedBy	integer	NOT NULL	Ідентифікатор клієнта
	IsProcessed	boolean	NOT NULL	
	ProcessedBy	ProcessedBy	NULL	

Таблиця «Повідомлення у приватному чаті» містить подальшу переписку оператора з клієнтом в разі обробки скарги, поля наступні: ID, ID скарги, ID відправника та отримувача, текст та час повідомлення.

Таблиця 3.6 – Повідомлення у приватному чаті

PrivateChatMessages				
Ключ	Ім’я атрибуту	Тип атрибуту	NULL?	Примітки
PK	Id	integer	NOT NULL	Генерується за замовчуванням як ідентифікатор
FK	UserComplaintId	integer	NOT NULL	On update no action, on delete cascade. Індекс
	FromId	integer	NOT NULL	
	ToId	integer	NOT NULL	
	Text	text	NOT NULL	
	MessageSent	timestamp without time zone	NOT NULL	

Перейдемо до сервісу Manager (додаток В), що використовується менеджерами служби доставки, які оновлюють контент сайту та додають промо-акції.

Таблиця «Ресторани» містить інформацію про закладу самовивозу, складається з ID та поштової адреси.

Таблиця 3.7 – Ресторани

Restaurants				
Ключ	Ім’я атрибуту	Тип атрибуту	NULL?	Примітки
PK	Id	integer	NOT NULL	Генерується за замовчуванням як ідентифікатор
	City	text	NOT NULL	
	District	text	NOT NULL	
	Street	text	NOT NULL	
	Building	integer	NOT NULL	

Таблиця «Продукти» призначена для детальної інформації про товари, складається з ID, назви продукту, короткого опису, типу, специфічних деталей, таких, як розмір, вид тіста тощо, та списку інгредієнтів.

Таблиця 3.8 – Продукти

FoodItems				
Ключ	Ім’я атрибуту	Тип атрибуту	NULL?	Примітки
PK	Id	integer	NOT NULL	Генерується за замовчуванням як ідентифікатор

	Name	text	NOT NULL	
	Description	text	NULL	
	FoodType	text	NOT NULL	
	SizeBasedData	jsonb	NOT NULL	
	AdditionalData	jsonb	NOT NULL	
	Ingredients	jsonb	NOT NULL	

Таблиця «Клієнти» містить інформацію для клієнтів, якою можуть оперувати менеджери. Складається з ID, ПІБ, статі та дня народження.

Таблиця 3.9 – Клієнти

Clients				
Ключ	Ім’я атрибуту	Тип атрибуту	NULL?	Примітки
PK	Id	integer	NOT NULL	Генерується за замовчуванням як ідентифікатор
	LastName	text	NOT NULL	
	FirstName	text	NOT NULL	
	Patronymic	text	NULL	
	Gender	text	NOT NULL	
	Birthday	timestamp without time zone	NULL	

Таблиця «Бонусні кампанії» представляє сутність, в якій можна зберігати правила нарахування бонусів клієнтам. Містить ID, назву, опис, розмір знижка, стан

(активний/неактивний) та правила нарахування, наприклад – знижка до дня народження, сезонна знижка, бонус при оплаті карткою тощо.

Таблиця 3.10 – Бонусні кампанії

BonusCampaigns				
Ключ	Ім’я атрибуту	Тип атрибуту	NULL?	Примітки
PK	Id	integer	NOT NULL	Генерується за замовчуванням як ідентифікатор
	Name	text	NOT NULL	
	Description	text	NOT NULL	
	BonusDiscountInPercent	double precision	NOT NULL	
	IsActive	boolean	NOT NULL	
	Rule	jsonb	NOT NULL	

Таблиця «Прогрес клієнта в кампанії» показує приналежність клієнта до кампанії. Складається з ID клієнта, ID кампанії та чи активний прогрес.

Таблиця 3.11 – Прогрес клієнта в кампанії

ClientCampaignProgresses				
Ключ	Ім’я атрибуту	Тип атрибуту	NULL?	Примітки

PK, FK	BonusCampaignId	integer	NOT NULL	On update no action, on delete cascade
PK, FK	ClientId	text	NOT NULL	On update no action, on delete cascade. Індекс
	IsActive	boolean	NOT NULL	

Перейдемо до сервісу SSO, який використовується адміністраторами веб-сайту та містить дані про зареєстрованих користувачів. Сервіс складається з однією таблиці «Користувачі», що м

Таблиця 3.11 – Користувачі

ClientCampaignProgresses				
Ключ	Ім’я атрибуту	Тип атрибуту	NULL?	Примітки
PK	Id	integer	NOT NULL	Генерується за замовчуванням як ідентифікатор
	Role	text	NOT NULL	
	LastName	text	NOT NULL	
	FirstName	text	NOT NULL	
	Patronymic	text	NULL	
	Gender	text	NOT NULL	
	Birthday	timestamp without time zone	NULL	
	Login	text	NOT NULL	

	Password	text	NOT NULL	

Обмеження на типи даних в атрибутах, можливість NULL значень чи значень за замовчуванням та визначення взаємозв’язків між таблицями дає можливість забезпечити вимоги бізнес-логіки та цілісність даних.

3.5 Опис розробки

C# – універсальна, мультипарадигмальна мова програмування, що включає в себе і об’єктно-орієнтовану парадигму, засоби якої використовуються при розробці системи. Вся програма представляє собою множину об’єктів, які взаємодіють між собою, тому розробка класів є вирішальним моментом для досягнення поставленої мети.

Наш веб-застосунок складається з проектів двох типів: ASP.NET Web API проекту та бібліотеки класів. Основні складові Web API проекту:

- Program.cs – точка входу в програму, де створюється об’єкт IHost, в рамках якого розгортається веб-застосунок. Створення цього об’єкту виконується за допомогою метода CreateDefaultBuilder(args), який також виконує ряд задач:
- Встановлює кореневий каталог.
- Встановлює конфігурацію хоста.
- Встановлює конфігурацію застосунку (з файлів appsettings.json та appsettings.{Environment}.json).
- Додає провайдери логування
- Забезпечує валідацію сервісів для проекту в статусі розробки.
- Startup.cs – клас, в якому мають бути реалізовані метод Configure(), що визначає, як застосунок оброблюватиме запити, та опційно метод ConfigureServices(), де реєструються всі сервіси застосунку.

- API контролери, в якому кожен публічний метод має реалізовувати HTTP запит.

Оскільки застосунок використовує шаблон «посередник», необхідно розробити класи, що будуть його реалізовувати. Мною було обрано бібліотеку MediatR, що містить для цього всі засоби. Для використання її необхідно спершу зареєструвати в методі ConfigureServices наступним чином:

```
services.AddMediatR(AppDomain.CurrentDomain.GetAssemblies());
```

Для використання в контролері використовується механізм Dependency Injection:

```
[Route("api/client")]
    public class ClientController : ApiController
    {
        private readonly IMediator _mediator;

        public ClientController(IMediator mediator)
        {
            _mediator = mediator;
        }

     // api methods
}
```

Для запитів необхідно сконструювати три типи класів:

- Клас запиту, що реалізує інтерфейс IRequest та параметризований класом відповіді.
- Клас відповіді
- Клас обробника відповіді, що реалізує інтерфейс IRequestHandler, а саме метод Handle(), та параметризований класами запиту та відповіді.

Для отримання відповіді необхідно створити об’єкт запиту та передати його в якості параметра при виклику методу IMediator.Send():

```
[HttpGet]
[ProducesResponseType(typeof(GetClientsResponse), StatusCodes.Status200OK)]
```

```
[ProducesResponseType(StatusCodes.Status401Unauthorized)]
public async Task<IActionResult> GetClients([FromQuery] int page = 1, int size = 10)
{
     var request = new GetClientsRequest { Page = page, Size = size };
     var response = await _mediator.Send(request);
     return Ok(response);
}
```

Вищевказаний GET запит в разі успішної автентифікації повертає список клієнтів із застосованою пагінацією.

Якщо застосунок взаємодіє з іншим сервісом через HTTP, має бути створений клас клієнта з об'єктом HttpClient, який і буде відсилати запити на вказані адреси. Також необхідно вказати URL сервісу в appsettings.json та зареєструвати клієнта:

```
services.AddHttpClient<ManagerServiceClient>();
var managerServiceUrl =
Configuration.GetSection("ServiceClientUrls").GetValue<string>(nameof(ManagerServiceClient));
services.Configure<ManagerClientSettings>(o => o.BaseUrl = managerServiceUrl);
```

При взаємодії застосунків один із одним за допомогою брокера повідомлень Kafka необхідно встановити бібліотеку Confluent.Kafka, додати налаштування для топіків (topic), серверів, на які будуть відправлятися повідомлення, та групу споживачів (consumer group), що будуть повідомлення отримувати у appsettings.json. Налаштування виконуються за допомогою створення об'єктів класів ConsumerConfig та ProducerConfig, а читання та надсилання – ConsumerBuilder та ProducerBuilder відповідно. При отриманні повідомлення його треба десереалізувати до певної моделі, провести валідацію та при її успіху оновити відповідний запис у базі даних. Також, створені класи необхідно зареєструвати як Singleton чи HostedService в Startup для подальшого використання.

Мною бібліотеки класів було створено для двох типів проектів: проектів Domain, що містять всі доменні сутності, та проектів DAL, що містять всі класи для роботи з базою даних.

Під доменними сутностями маються на увазі моделі, що відображають структуру даних застосунку та співвідносяться з однойменними реляціями. В якості прикладу нижче наведено клас OrderLine, що відповідає таблиці OrderLines і показує дані про товари в замовленні.

```
public class OrderLine

{
    public int Id { get; set; }
    public int OrderId { get; set; }
    public int FoodItemId { get; set; }
    public uint Quantity { get; set; }
    public decimal OrderLineSum { get; set; }
}
```

Також в проекті Domain містяться перелічення (enumerations), які дозволяють уникнути багаторазового написання одних і тих же рядкових значень. Наприклад, нижче наведено можливі опції для доставки – доставка з ресторану або самовивіз.

```
[JsonConverter(typeof(JsonStringEnumConverter))]
public enum DeliveryType
{
    PickUp,
    Delivery
}
```

Проекти DAL, як було зазначено вище, призначені для взаємодії з базою даних. Для уникнення написання чистого SQL та взаємодії з рядками таблиць як з об’єктами в об’єктно-орієнтованому програмуванні використовується Object-Relational Mapping (ORM) фреймворк Entity Framework Core. EF Core дозволяє абстрагуватися від бази даних і працювати з даними незалежно від типу сховища. Фреймворк підтримує безліч систем керування баз даних, в тому числі має вбудований провайдер для PostgreSQL: для цього необхідно встановити бібліотеку Npgsql.EntityFrameworkCore.PostgreSQL.

Для наповнення бази таблицями використовується підхід Code first: спочатку створюються класи моделей даних, а далі Entity Framework генерує базу даних за цією схемою. Для цього необхідно створити клас контексту, що наслідуватиметься від DbContext, вказати, які набори даних будуть створивши колекції DbSet<TEntity>, та перевизначити метод OnModelCreating(), в якому явно

вказати первинні та зовнішні ключі, індекси, значення за замовчуванням, способи запису деяких полів тощо. Це виглядає наступним чином:

```
modelBuilder.Entity<OrderLine>()
    .ToTable("OrderLines")
    .HasKey(o => o.Id);

modelBuilder.Entity<OrderLine>()
    .Property(o => o.Id)
    .ValueGeneratedOnAdd()
    .IsRequired();

modelBuilder.Entity<OrderLine>()
    .Property(o => o.Quantity)
    .HasDefaultValue(1)
    .IsRequired();

modelBuilder.Entity<OrderLine>()
    .HasOne(t => t.Order)
    .WithMany(order => order.OrderLines)
    .HasForeignKey(t => t.OrderId)
    .OnDelete(DeleteBehavior.Cascade);

modelBuilder.Entity<OrderLine>()
    .HasIndex(o => o.OrderId);
```

Для кожної зміни схеми БД необхідно додати міграцію: для цього в Package Manager Console необхідно ввести команду Add-Migration Migration_Name, результатом виконання якої буде створення файлу, похідного від класу Migration, що міститиме ряд команд для бази даних, та оновлення класу DbContextModelSnapshot, що походить від класу ModelSnapshot та містить знімок всієї схеми даних. Для застосування всіх змін необхідно виконати команду Update-Database, проте для уникнення її ручного вводу необхідно забезпечити можливість виконання останніх міграцій при запуску застосунку. Мною було створено клас DatabaseMigrationHostedService, що реалізує інтерфейс IHostedService, який в свою чергу дозволяє виконувати довгі фонові операції. В його методі StartAsync викликається метод контексту MigrateToLatestVersion, який оновлює базу даних до останньої міграції.

Для налаштування бази даних необхідно вказати рядок підключення в настройках та передавати його значення в якості параметра методу UseNpgsql.

Також при додаванні контексту можливо налаштувати рівень логування, способи відстеження результатів запитів тощо.

Для реалізацію шаблону DAO були створені класи класи-репозиторії – окремий для кожної доменної сутності. В репозиторіях містяться CRUD-операції над даними та при необхідності додаткові запити, наприклад, у BonusCampaignRepository є перевірка, чи існує бонусна кампанія з такими параметрами.

Для зручної взаємодії з бекенд сервісами використовується OpenAPI – специфікація і екосистема багатьох інструментів для опису REST API, а саме фреймворк Swagger. Інструмент дозволяє створити веб-сторінку з детальною документацією API-сервісу, яка включає всі HTTP-методи з параметрами, статус-коди та моделі відповідей, детальні описи тощо. Для підключення Swagger використовуються бібліотеки Swashbuckle.AspNetCore.Swagger, Swashbuckle.AspNetCore.SwaggerGen та Swashbuckle.AspNetCore.SwaggerUI, викликаються відповідні методи у класі Startup та додаються анотації у контролерах, як показано нижче.

```
/// <summary>
/// Returns product catalog
/// </summary>
/// <param name="types">Food types</param>
/// <param name="page">Page</param>
/// <param name="size">Size</param>
/// <returns></returns>
```

3.6 Тестування програми та результати її виконання

Поточна версія програми містить повністю готові до інтеграції з фронтендом бекенд сервіси для клієнтів, операторів, менеджерів та адміністратора. За допомогою Swagger до кожного сервісу наявна інтерактивна документація, за допомогою якої можна провести тестування бекенду та побачити результати запитів.

Даний застосунок передбачає авторизацію на основі ролей, тому доцільно почати із сервісу, в якому передбачено управління користувачами та їх доступами – сервісу для адміністраторів. Його функціонал показано на рисунку 3.4.

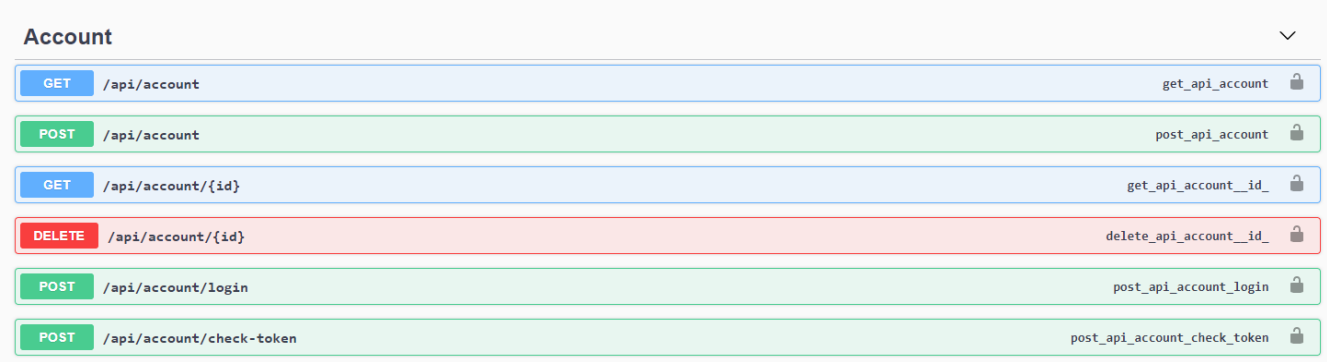


Рисунок 3.4 – Функціонал сервісу адміністраторів

GET-метод /api/account має фільтрацію за ролями та пагінацію і повертає список зареєстрованих користувачів, а POST-метод з такою адресою приймає на вхід модель User і додає чи оновлює запис у базі даних. Методи GET і DELETE /api/account/{id} відповідно повертають і видаляють користувачів з певним ID. Метод /api/account/login використовується для автентифікації, приймає на вхід логін та пароль і повертає JWT-токен, який надалі використовується для запитів у сервісах. Для перевірки особистості користувача при запитах викликається метод /api/account/check-token, що в разі успішної автентифікації повертає дані про користувача. Зареєструємо користувача з роллю «Адміністратор», викликавши метод api/account. Результат виклику – інформація про користувача з зашифрованими вразливими даними, а саме паролем, зображена на рисунку 3.5.

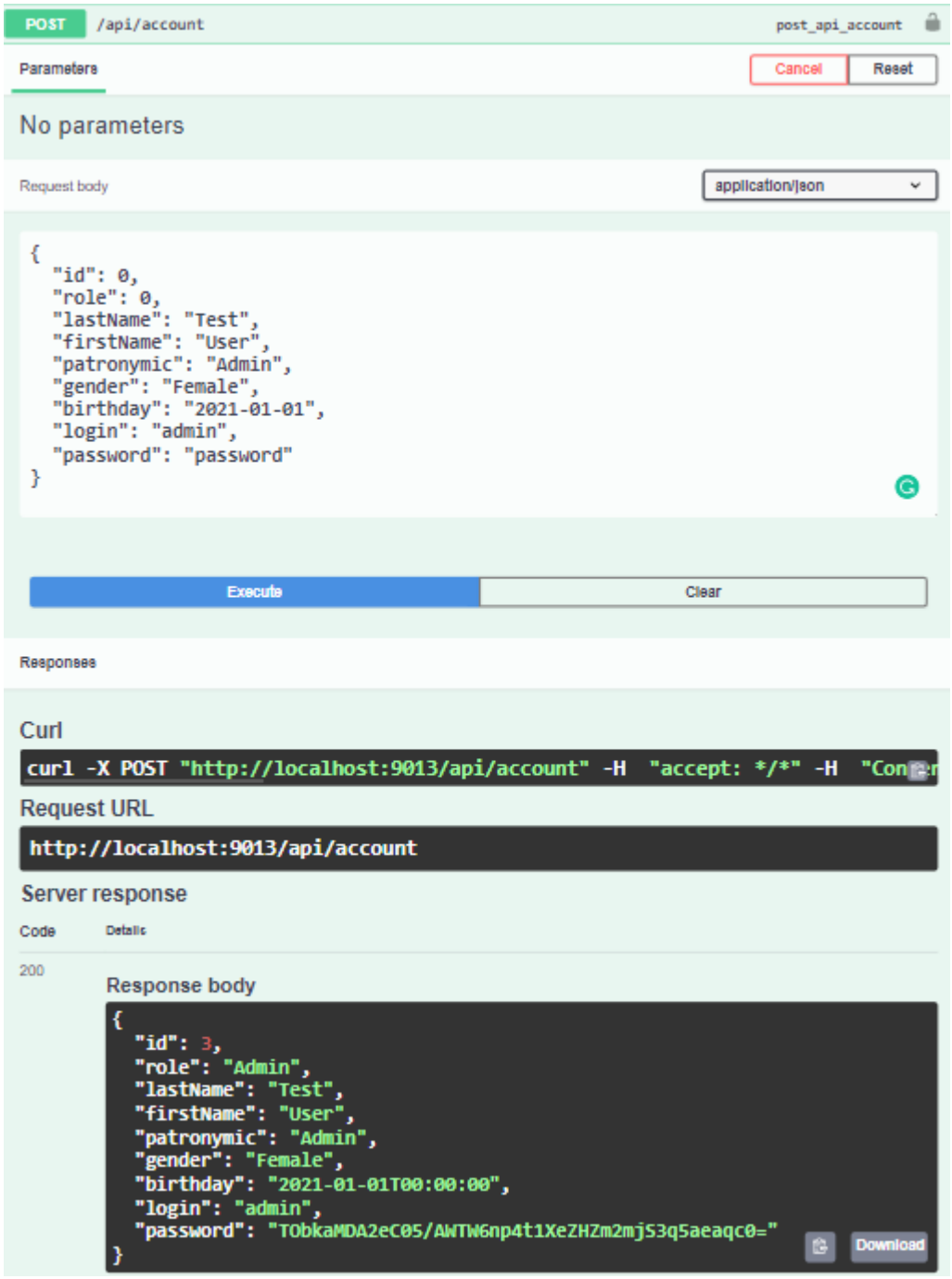


Рисунок 3.5 – Створення користувача

При автентифікації вищевказаного користувача отримуємо токен (рис.3.6), який можна перевірити викликавши метод check-token.

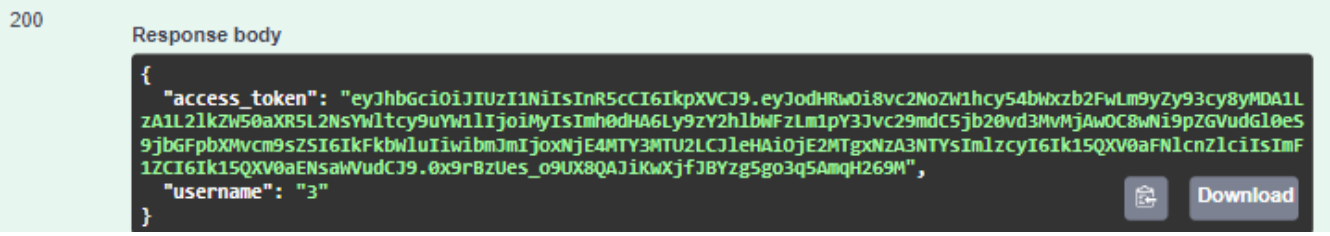


Рисунок 3.6 – Відповідь методу login

При успішній перевірці отриманий результат буде схожим на результат на рис. 3.7.

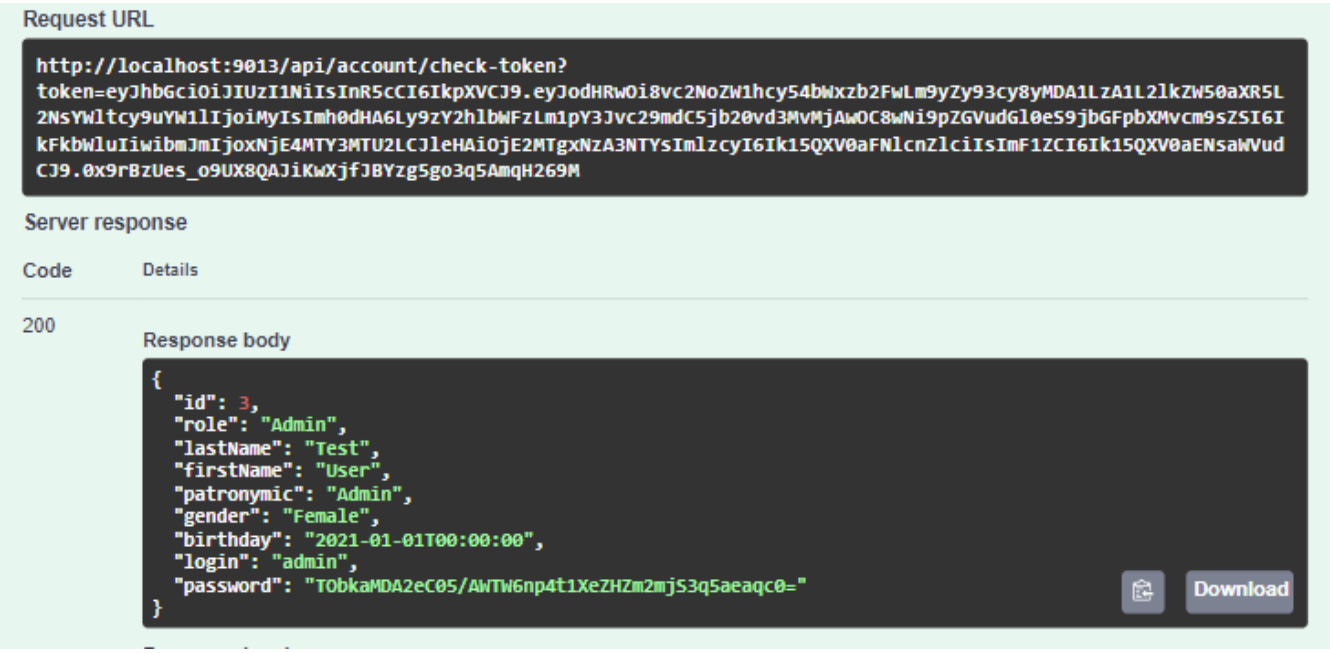


Рисунок 3.7 – Перевірка токену

Перейдемо до сервісу менеджерів, функціонал якого показано на додатку Г. Є можливість додавати, редагувати, видаляти та переглядати ресторани, продуктовий каталог, клієнтів та бонусні кампанії, також є можливість додавати клієнтів у кампанію. Наприклад, клієнт, що доданий в кампанію, зображену на рисунку 3.8, зробивши замовлення з 12/04/2021 по 18/04/2021, отримає знижку 5 відсотків.

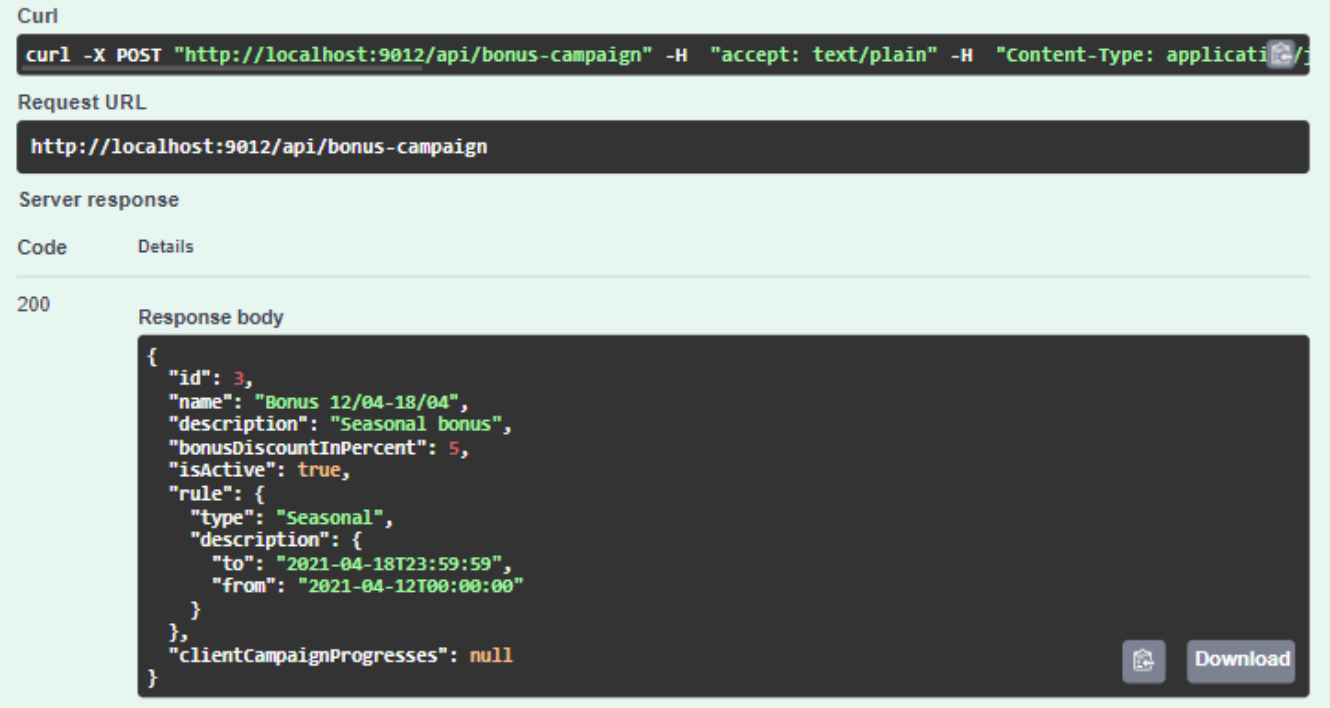


Рисунок 3.8 – Створення бонусної кампанії

Доступ до функціоналу даного сервісу мають менеджери та адміністратори, проте клієнти теж можуть переглядати його певну частину, а саме список продуктів, щоб їх обирати і додавати до замовлення.

Перейдемо до функціоналу сервісу для клієнтів (додаток Ґ), який передбачає створення, редагування, видалення та перегляд власних замовлень клієнта; перегляд та оновлення профілю, додавання, редагування, перегляд та видалення адрес доставки. Також цей сервіс взаємодіє з іншими за допомогою брокеру повідомлень Kafka. Створимо замовлення зі статусом «Новий» і трьома продуктами в чеку (рисунок 3.9).

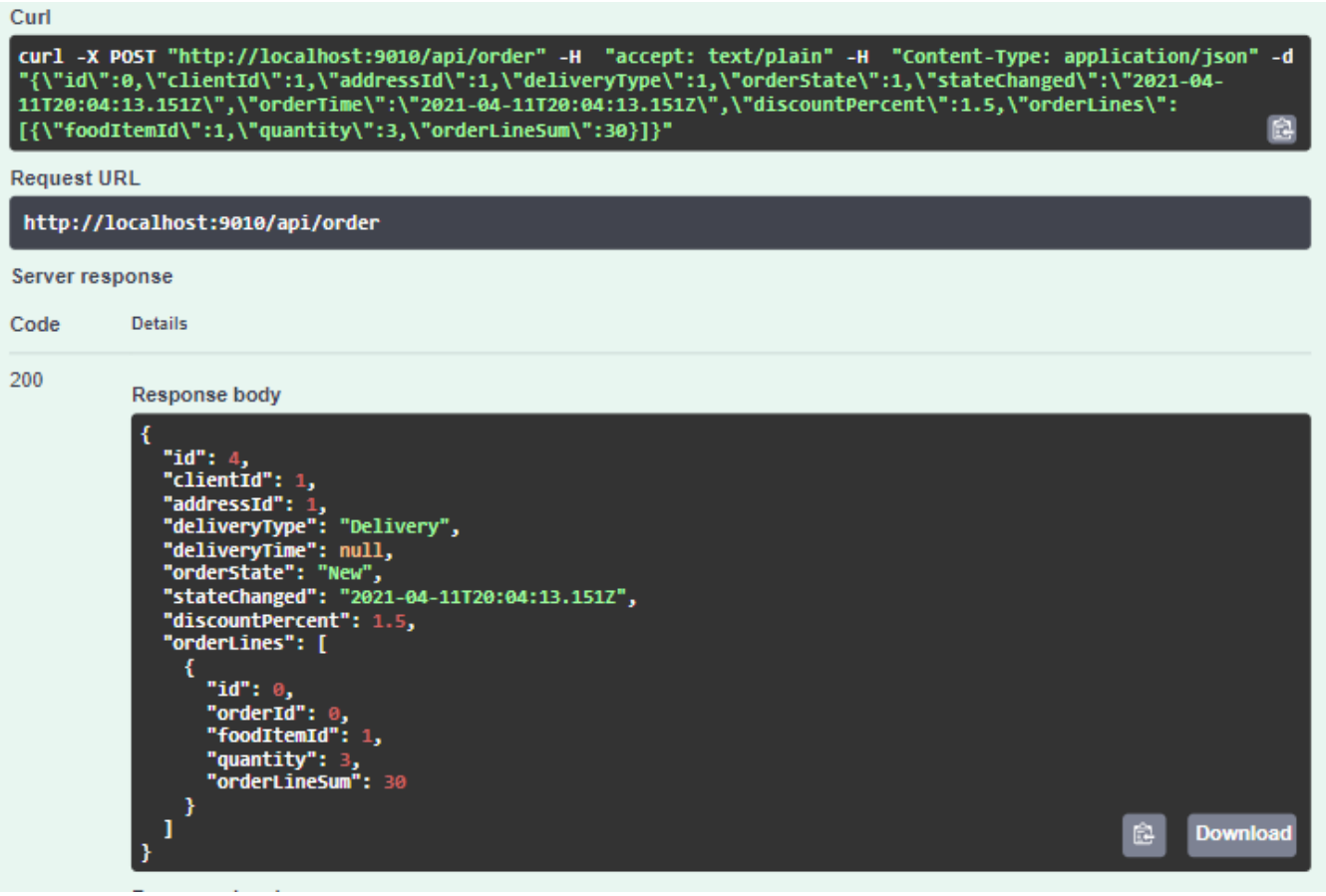


Рисунок 3.9 – Створення замовлення

Окрім записів в таблиці Orders та OrderLines, дані про замовлення відправляються до сервісу операторів. Дані у форматі JSON заносяться в topic, який далі читатиме група OperatorService.API, як зображено на рисунку 3.10.

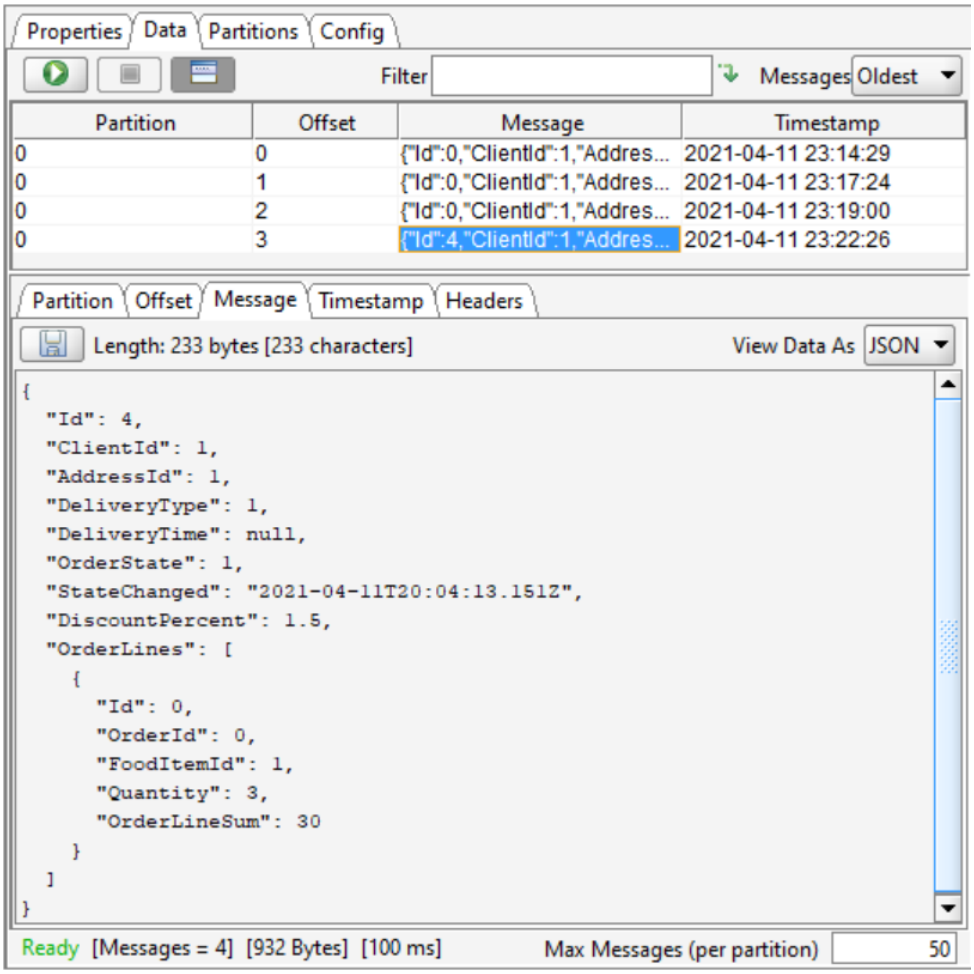


Рисунок 3.10 – Повідомлення про замовлення у Kafka

Також, клієнт може додати скаргу, повідомлення з даними якої теж надсилатимуться в Kafka (рисунок 3.11).

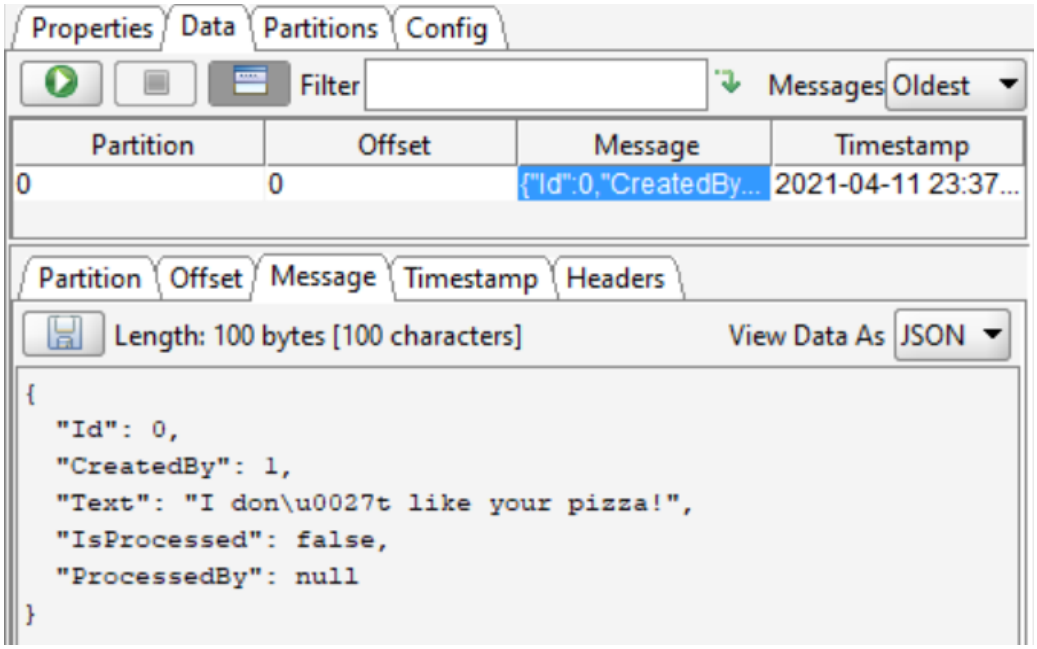


Рисунок 3.11 – Повідомлення про скарги у Kafka

Перейдемо до сервісу операторів, функціонал яких представлено на рисунку 3.12.

Рисунок 3.12 – Функціонал сервісу операторів

Оператори мають змогу оновлювати статус замовлення та переглядати дані по замовленням та можуть відповідати на скарги користувачів. API-сервіс читає повідомлення про оновлення статусу замовлення та про додані скарги. Наприклад,

нижче на рисунку можна побачити дані про замовлення у базі даних сервісу Operator, які були відправлені сервісом Client і показані на рисунку 3.13.

public.OrderUpdates/OperatorService/postgres@localhost

Query Editor | Query History

```
1  SELECT * FROM public."OrderUpdates"
2  ORDER BY "Id" ASC
```

Data Output | Explain | Messages | Notifications

	Id [PK] integer	OrderId integer	State text	StateChanged timestamp without time zone
1	1	0	InCart	2021-04-11 20:04:13.151
2	2	0	InCart	2021-04-11 20:04:13.151
3	3	0	InCart	2021-04-11 20:04:13.151
4	4	4	InCart	2021-04-11 20:04:13.151

Рисунок 3.13 – Записи про оновлення замовлення

Доступ до функціоналу даного сервісу мають оператори та адміністратори, проте клієнти теж можуть переглядати його певну частину, а саме список своїх скарг та переписки з технічною підтримкою сайту.

Також в сервісі фронтенду створені та налаштовані HTTP-клієнти та додані всі моделі даних для інтеграції з бекендом. Залишилось додати та оформити HTML-сторінки, які будуть відображати частини системи для клієнтів, менеджерів, операторів та адміністраторів.

Висновки

Метою курсової роботи є створення ASP.NET Core застосунку для служби доставки піци на основі мікросервісів. В ході роботи був проведений аналіз аналогічних систем, визначені типові недоліки та можливий функціонал. Були розглянуті основні архітектурні стилі та ключові моменти при створенні мікросервісів та з урахуванням всіх деталей розроблена програма та визначені способи її реалізації.

Після визначення технічних вимог, алгоритму та структури програми були виокремлені її основні об’єкти, підходящі засоби розробки, зберігання та обміну даних, що привело до створення бекенду застосунку для служби доставки піци для різних груп користувачів на основі мікросервісів.

Дана система, незважаючи на відповідність вимогам, має безліч можливостей для вдосконалення. По-перше, необхідно розробити фронтенд і зінтегрувати готові сервіси. Гарним доповненням буде розділ для збережених товарів – так клієнти зможуть швидко здійснювати замовлення улюблених продуктів. Зручним функціоналом може бути відстеження геолокації кур’єра та відображення його номеру телефона задля уточнення деталей доставки. Також можливо додати до списків розширену фільтрацію, сортування та пошук.

Список використаної літератури

1. COVID-19 and the Demand for Online Food Shopping Services: Empirical Evidence from Taiwan [Електронний ресурс] / Hung-Hao Chang Chad D. Meyerhoefer – 2020 – Режим доступу до ресурсу: https://onlinelibrary.wiley.com/doi/full/10.1111/ajae.12170
2. Impact of COVID-19 on consumer buying behavior toward online shopping in Iraq. Economic Studies Journal [Електронний ресурс] / Ali, B. J. – 2020 – №18 – с. 267–280 – Режим доступу до ресурсу: https://ssrn.com/abstract=3729323
3. Examining user satisfaction and stickiness in social networking sites from a technology affordance lens: uncovering the moderating effect of user experience [Електронний ресурс] / Shao, Z., Zhang, L., Chen, K. and Zhang, C. – 2020 – №7 – с. 1331-1360 – Режим доступу до ресурсу: https://doi.org/10.1108/IMDS-11-2019-0614
4. Електронний ресурс: https://blog.pokupon.ua/prosto-privezi-mne-edy-top-servisov-dostavki-v-kieve-i-prijatnyj-bonus/
5. Електронний ресурс: https://pizzahouse.ua/uk/
6. Kim, Jungkeun. (2017). The impact of different price promotions on customer retention. Journal of Retailing and Consumer Services. 46. 10.1016/j.jretconser.2017.10.007.
7. Електронний ресурс: https://adriano.com.ua/uk/pizza-sets
8. Електронний ресурс: https://dominos.ua/uk/kyiv/
9. Електронний ресурс: Chapter 3: Architectural Patterns and Styles. Msdn.microsoft.com.
10. Oluwatosin H. S. Client-server model //IOSRJ Comput. Eng. – 2014. – Т. 16. – №. 1. – С. 2278-8727.
11. Електронний ресурс: https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ff647328(v=pandp.10)?redirectedfrom=MSDN

12. Електронний ресурс: "SOA Source Book - What Is SOA?". collaboration.opengroup.org

13. Електронний ресурс: https://docs.microsoft.com/en-us/archive/msdn-magazine/2009/february/best-practice-an-introduction-to-domain-driven-design

14. Rainer Niekamp. "Software Component Architecture" (PDF). Gestión de Congresos - CIMNE/Institute for Scientific Computing, TU Braunschweig. p. 4. Retrieved 2011-07-29. The modern concept of a software component largely defined by Brad Cox of Stepstone, => Objective-C programming language

15. Dragoni, Nicola; Giallorenzo, Saverio; Alberto Lluch Lafuente; Mazzara, Manuel; Montesi, Fabrizio; Mustafin, Ruslan; Safina, Larisa (2016). "Microservices: yesterday, today, and tomorrow". arXiv:1606.04036v1 [cs.SE].

16. Електронний ресурс: www.postgresql.org.

17. PostgreSQL: Documentation: 11: Procedural Languages. www.postgresql.org.

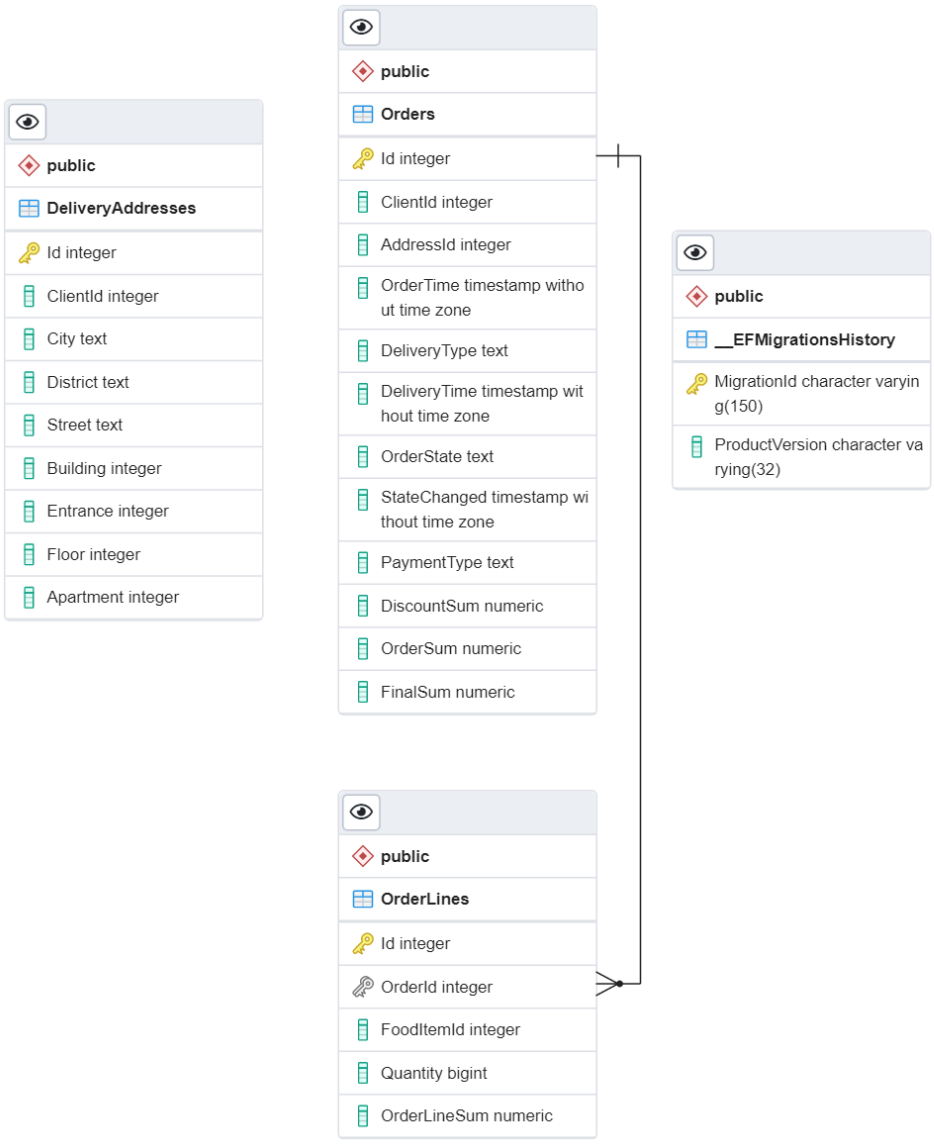
18. Електронний ресурс: https://www.pgadmin.org/docs/pgadmin4/latest/index.html

19. Електронний ресурс: https://www.kafkatool.com/

Додаток А

(обов’язковий)

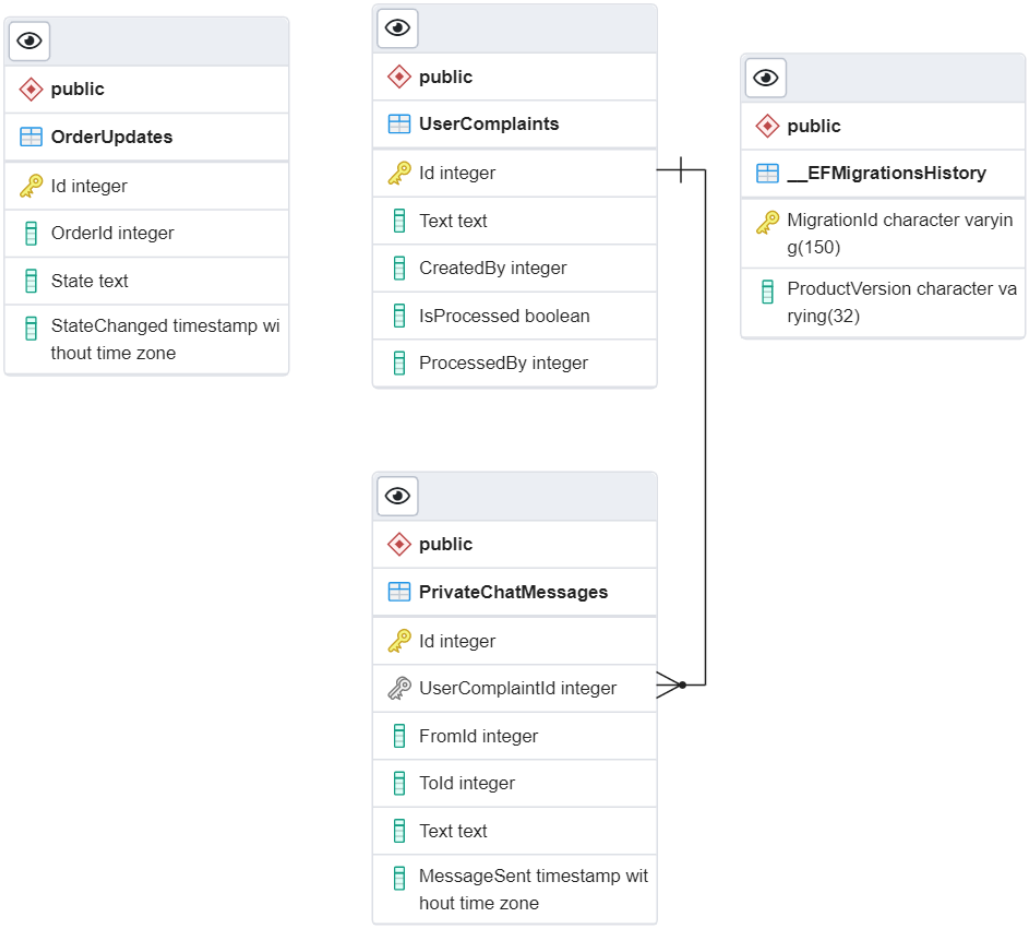
Схема бази даних сервісу для клієнтів



Додаток Б

(обов’язковий)

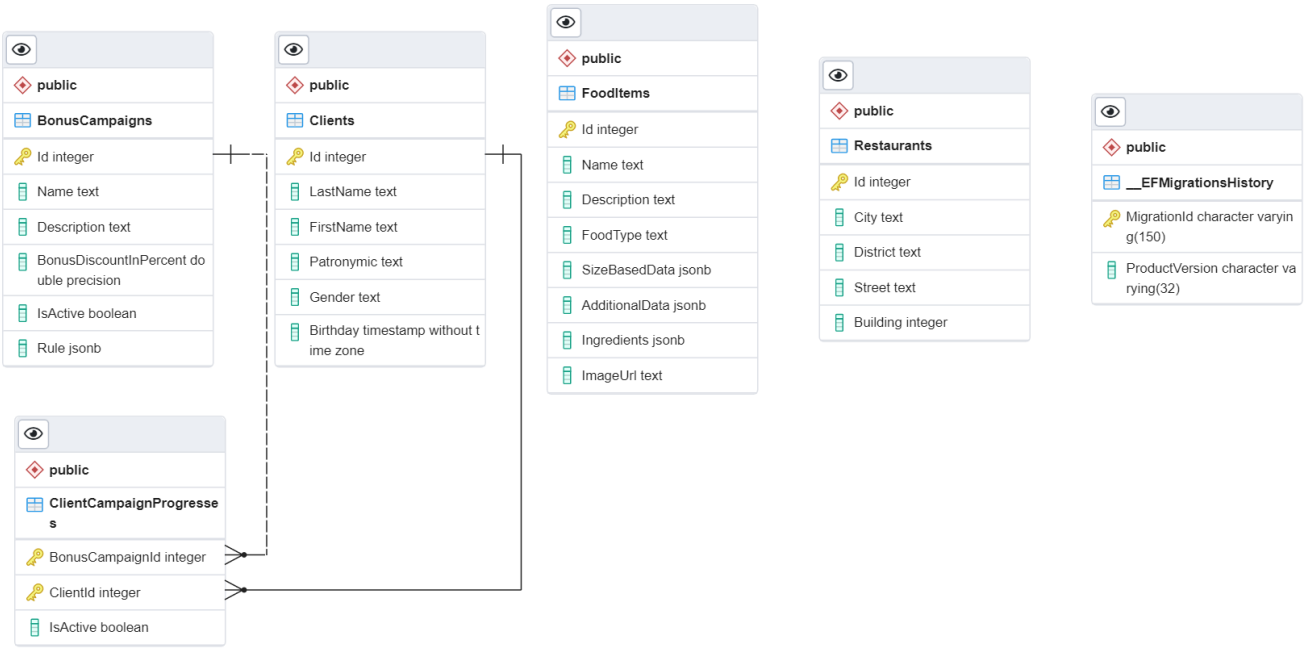
Схема бази даних сервісу для операторів



Додаток В

(обов’язковий)

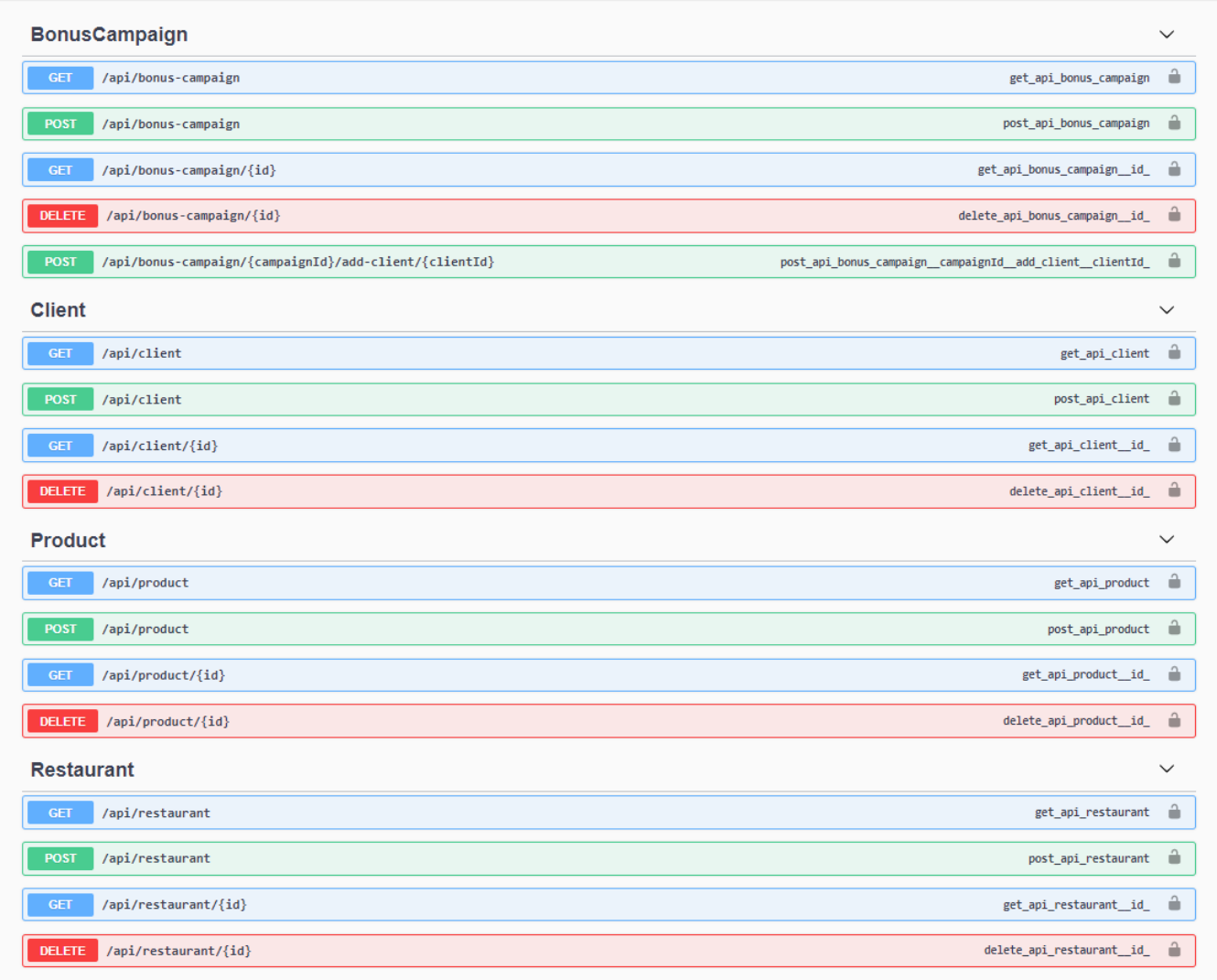
Схема бази даних сервісу для менеджерів



Додаток Г

(обов’язковий)

Функціонал сервісу для менеджерів



Додаток Ґ

(обов'язковий)

Функціонал сервісу для операторів