Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

РОЗРОБКА МІКРОСЕРВІСНОГО ЗАСТОСУНКУ ДЛЯ РЕСТОРАНУ
(тема)

Виконав:
студент 4 курсу, групи ІТІНФ-20-1

Бойко М.К
(прізвище, ініціали)

Спеціальності 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник ст. викл. Путятіна О.Є
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Кобилін О.А.
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)

Кафедра Інформатики
(повна назва)

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Бойко Марії Костянтнівні
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка мікросервісного застосунку для ресторану

затверджена наказом університету від 18 травня 2024 року № 261 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 02 червня 2024 р.

3. Вихідні дані до роботи вебзастосунок для управління рестораном; CSV файл зі звітом про ціну покупків кожнго клієнту; окремий застосунок на Python для обчислення оцінки розташування за CSV файлами, що допомагає підрахувати середній дохід компанії за проміжок часу найточнішим з методів; програма для імітації навантаження застосунку при різній кількості користувачів.

4. Перелік питань, що потрібно опрацювати в роботі
1. Аналіз предметної області.
2. Вибір технологій та принципів проєктування.
3. Програмна реалізація.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) Рисунок 1 – Приклад архітектури мікросервісів застосунку «Mango»; Рисунок 2 – Структура хмарного сховища Azure Blob Storage; Рисунок 3 – Архітектура бази даних для реєстрації та авторизації; Рисунок 4 – Архітектура бази даних для зберігання купонів; Рисунок 5 – Архітектура бази даних для зберігання товарів; Рисунок 6 – Архітектура бази даних для керування кошиком; Рисунок 7 – Приклад коммунікації між користувачем та мікросервисами продуктів та корзини; Рисунок 8 – Архітектура проєкту «Mango.Web»; Рисунок 9 – Вміст папки Services; Рисунок 10 – Структура мікросервісу для опису купонів CouponAPI; Рисунок 11 – Структура мікросервісу Identity для реєстрації/авторізації користувачів; Рисунок 12 – Структура мікросервісу ProductAPI для описа продуктів; 13 – Структура мікросервісу SjoppingCartAPI для описа корзини; Рисунок 14 – Вміст дерікторії Sharable; Рисунок 15 – CSV файл с даними про покупки; Рисунок 16 – Результат обчислення оцінки розташування; Таблиця 1 – Результат навантаження застосунку.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	08.04.2024	
2	Аналіз завдання, підбір літератури	08.04.24-15.04.24	
3	Аналіз літератури з досліджуваної проблеми	16.04.24-18.04.24	
4	Аналіз аналогічних розробок	19.04.24-25.04.24	
5	Розробка вебзастосунку	26.04.24-14.05.24	
6	Програмна реалізація застосунку на Python	15.05.24-23.05.24	
7	Оформлення пояснювальної записки	24.05.24-26.05.24	
8	Перевірка на плагіат	28.05.24-30.05.24	
9	Рецензування	30.05.24	
10	Підготовка презентації та доповіді	01.06.24-06.06.24	
11	Занесення роботи в електронний архів	08.06.24	
12	Попередній захист кваліфікаційної роботи	12.06.24	

Дата видачі завдання 8 квітня 2024 р.

Студент _____
(підпис)

Керівник роботи _____ ст. викл. Путятіна О.Є
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка: 70с., 1 табл., 31 рис., 3 дод., 45 джерел.

МІКРОСЕРВИСНА АРХІТЕКТУРА, РЕСТОРАН, АНАЛІЗ, ЛОГІКА РОБОТИ, СТРУКТУРА ЗАСТОСУНКУ, РОЗРОБКА C#, MANGO, ТЕХНІЧНЕ ЗАВДАННЯ, МАСШТАБОВАНІСТЬ.

Проєкт присвячено розробці мікросервісного застосунку для ресторану «Mango» на мові програмування C#. Об'єктом роботи стали технічні аспекти розробки та впровадження застосунку, розпочавши з аналізу технічного завдання, в якому визначено основні вимоги до функціоналу та властивостей системи. Метою роботи є створення ефективного та функціонального інформаційно-технічного середовища, спрямованого на поліпшення обслуговування клієнтів та оптимізацію управління ресторанним бізнесом.

Важливою частиною проєкту був аналіз аналогічних розробок на ринку, їх переваг та недоліків.

Оригінальні макети та структурні рішення розроблено з урахуванням особливостей ресторанного бізнесу, забезпечуючи зручний інтерфейс для користувачів. Увага була приділена вибору технічного стеку, програмного забезпечення та обґрунтуванню вибору поліграфічного устаткування для виготовлення друкованої реклами.

MICROSERVICE ARCHITECTURE, RESTAURANT, ANALYSIS, ROBOTIC LOGIC, SUPPLEMENT STRUCTURE, C# DEVELOPMENT, MANGO, TECHNICAL DEVELOPMENT, SCALE.

The project is devoted to the development of a microservice application for the Mango restaurant in the C# programming language. The main goal of the work is to create an effective and functional information and technical environment aimed at improving customer service and optimizing restaurant business management.

The object of the work was the technical aspects of the development and implementation of the application, starting with the analysis of the technical task, in which the basic requirements for the functionality and properties of the system were defined. An important part of the project was the analysis of similar developments on the market, their advantages and disadvantages.

Original layouts and structural solutions are developed taking into account the peculiarities of the restaurant business, providing a convenient interface for users. Attention was paid to the selection of the technical stack, software and justification of the choice of printing equipment for the production of printed advertising..

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API – Application Programming Interface (Інтерфейс програмування застосунків)

C# – C-Sharp (Об’єктно-орієнтована мова програмування, розроблена Microsoft)

HTTP – Hypertext Transfer Protocol (Протокол передачі гіпертексту, використовується для передачі даних на Всесвітній павутині)

Мікросервіси – архітектурний підхід до розробки програмного забезпечення, у якому програма поділяється на невеликі, незалежні сервіси

REST – Representational State Transfer (Стиль архітектури програмного забезпечення для створення масштабованих вебсервісів)

Swagger – інструмент для створення, документування та використання API на основі специфікації OpenAPI

Visual Studio – інтегроване середовище розробки програмного забезпечення, розроблене компанією Microsoft для роботи з мовами програмування з .NET-платформи

ВСТУП

У сучасному суспільстві технологічні інновації відіграють ключову роль у вдосконаленні та оптимізації бізнес-процесів. Ресторанна галузь, переймаючи тренди сучасності, активно впроваджує інформаційні технології для покращення обслуговування та ефективного управління всім спектром ресторанного досвіду. В цьому контексті виникає необхідність у впровадженні нових технологічних рішень, спрямованих на підвищення ефективності та конкурентоспроможності ресторанів.

Розробка мікросервісного застосунку для ресторану «Mango», здійснена за допомогою мови програмування C# та базується на мікросервісній архітектурі, має на меті створення інноваційної інформаційної платформи. Ця платформа спрямована на автоматизацію та оптимізацію ключових процесів ресторанного бізнесу, роблячи його більш гнучким, відзначаючись високою масштабованістю.

Проєкт націлений на створення інформаційно-технічного середовища, що сприятиме підвищенню якості обслуговування та оптимізації управління бізнес-процесами. Задачі проєкту включають ретельний аналіз технічних аспектів розробки, вивчення аналогічних розробок на ринку, розробку структури та логіки мікросервісів, а також оцінку можливостей масштабованості, швидкодії та безпеки розробленого застосунка.

Цей застосунок має велике практичне значення для ресторанної галузі, допомагаючи у створенні сучасного та конкурентоспроможного середовища.

В умовах сучасного ритму життя та зростання вимог споживачів, ресторани виявляють потребу у використанні інноваційних інформаційних технологій. Це відкриває широкі можливості для впровадження новаторських рішень, спрямованих на поліпшення обслуговування, оптимізацію роботи персоналу та підвищення рівня задоволення клієнтів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис існуючих розробок

На поточному етапі еволюції ресторанної сфери, спостерігається багатий арсенал програмних продуктів, включаючи в себе платформи замовлення їжі, такі як UberEats та Grubhub, а також інтегровані системи управління рестораном, наприклад, Toast і Square for Restaurants. Останній категорізований продукт спрямований на оптимізацію ключових аспектів, таких як замовлення, доставка та внутрішнє управління закладом.

UberEats та Grubhub, як представники платформ для замовлення їжі, надають споживачам зручний механізм для онлайн-замовлень та доставки. З іншого боку, Toast і Square for Restaurants виступають в якості інтегрованих систем управління рестораном, охоплюючи широкий спектр функціональностей, від керування меню до обліку фінансів та статистичної звітності. Усі ці розробки направлені на спрощення та удосконалення операцій ресторанного бізнесу [1].

Представлені рішення визначають та відповідають важливим аспектам роботи закладу, таким як забезпечення швидкості обслуговування, підвищення зручності для клієнтів та управління ресурсами. Різноманітність цих продуктів створює можливість вибору оптимального рішення, залежно від специфіки та потреб конкретного закладу.

Розробки в даній категорії сприяють ефективному впровадженню сучасних технологій у ресторанний бізнес, забезпечуючи інструменти для прискорення обробки замовлень, поліпшення якості обслуговування та підвищення оперативної продуктивності.

Виокремлені рішення визначають та відповідають фундаментальним аспектам функціонування закладу, таким як максимізація ефективності обслуговування, поліпшення зручностей для клієнтів та оптимізація управлінських ресурсів.

Диверсифікація функціоналу цих продуктів створює можливість вибору оптимального рішення, враховуючи особливості та потреби конкретного закладу.

Крім того, ці інноваційні рішення визначають та відповідають фундаментальним аспектам ресторанного бізнесу, таким як підвищення рівня зручності для клієнтів через прискорення процесів замовлення та доставки [2]. Платформи замовлення їжі не лише роблять процес онлайн-замовлень легким та ефективним для клієнтів, але і сприяють залученню нових аудиторій та підвищенню конкурентоспроможності ресторанів у цифровому просторі.

З іншого боку, інтегровані системи управління рестораном глибоко занурюються в оптимізацію внутрішніх процесів. Вони не лише допомагають у веденні обліку товарів та фінансів, але і забезпечують точний аналіз різних аспектів роботи ресторану. Це включає у себе взаємодію з персоналом, відстеження популярності страв, аналіз ефективності маркетингових кампаній, та інші важливі аспекти, що дозволяють приймати обґрунтовані управлінські рішення.

Зазначені розробки, спрямовані на максимізацію ефективності та задоволення потреб клієнтів, підкреслюють важливість використання сучасних технологій у ресторанному бізнесі. Вибір оптимального рішення залежить від конкретних потреб кожного закладу, але загальна тенденція полягає в уведенні цифрових інструментів для оптимізації всіх аспектів ресторанного обслуговування та досягнення високого рівня задоволення як клієнтів, так і власників закладів.

На фоні постійного розвитку та конкуренції в ресторанній галузі, програмні продукти виявляються ключовими компонентами стратегій успіху для закладів [3–4]. Зазначені рішення не лише пристосовуються до швидкозмінюючого середовища, але й стають критично важливими для відповіді на зростаючі очікування клієнтів та оптимізації внутрішньої діяльності.

UberEats та Grubhub, розширюючи зону обслуговування ресторанів через

онлайн-платформи, відкривають нові можливості для ресторатoрів привертати клієнтів та збільшувати обсяги замовлень [5]. У той час як Toast і Square for Restaurants, у якості інтегрованих систем, дозволяють забезпечити єдиною платформою ефективне управління всіма аспектами ресторанного бізнесу, від точного обліку запасів до проведення аналітики [6].

Ці інновації не просто автоматизують рутинні процеси, але і впроваджують технологічні та аналітичні інструменти для підвищення якості обслуговування та збільшення задоволеності клієнтів. Забезпечуючи ресторанам засоби для оперативного реагування на зміни в ринкових умовах та пристосування до вимог сучасного споживача, ці технологічні рішення стають запорукою конкурентоспроможності та динамічного розвитку в галузі ресторанного бізнесу.

Описані новаторські технологічні рішення також висвітлюють важливість адаптації ресторанів до цифрової трансформації. Вони стають не тільки інструментами для автоматизації та оптимізації процесів, але і кроком у майбутнє, де взаємодія з клієнтами через мобільні застосунки та онлайн-платформи стає стандартом.

Ресторанні бізнеси, які вміло використовують ці технології, отримують можливість ефективно конкурувати на ринку, зберігаючи та привертаючи нових клієнтів. Простота онлайн-замовлень, точний облік фінансів та вдале управління персоналом стають основою для стабільної та успішної роботи ресторану в умовах сучасного глобального ринку [7].

У цьому контексті, розглядаючи різноманітність програмних продуктів для ресторанного бізнесу, слід підкреслити, що кожен із них може стати важливим інструментом для досягнення успіху, в залежності від конкретних потреб та завдань конкретного закладу. Застосування таких технологій вимагає від рестораторів не лише технічної компетентності, але й готовності до постійного вдосконалення та адаптації до змін у вимогах ринку та споживачів.

Ці програмні рішення також акцентують на важливості створення екосистеми технологій, що сприяють внутрішній взаємодії різних компонентів

ресторанного бізнесу. Вони визначають новий рівень інтеграції, де замовлення, облік, аналітика та обслуговування взаємодіють безшовно, створюючи динамічне та підтримуюче середовище.

Ці технології також відкривають можливості для збору та аналізу даних, що дозволяє ресторанам легше розуміти потреби клієнтів, прогнозувати тенденції та вдосконалювати свою стратегію. Аналітичні інструменти, що входять до складу цих систем, роблять можливим вчасне реагування на зміни в попиті та упереджене планування для оптимізації бізнес-процесів.

Саме тому ресторани, використовуючи сучасні програмні рішення, не тільки автоматизують рутинні завдання, але й переходять на новий етап ефективного та стратегічного управління. Це стає ключовим елементом конкурентоспроможності, забезпечуючи стійкий розвиток та відповідь на виклики ринку в умовах постійної динаміки та змін.

Подібні технологічні інновації також відзначають важливість підвищення взаємодії з клієнтами. Мобільні застосунки та онлайн-платформи для замовлення їжі розширюють способи комунікації між ресторанами та клієнтами, створюючи прямий канал для взаємодії та отримання зворотного зв’язку.

Ці технології стають не лише засобами ефективної реклами та просування, але й інструментами для підтримки лояльності клієнтів. Програми лояльності та персоналізовані пропозиції, які легко впроваджуються через ці системи, дозволяють ресторанам не лише привертати нових гостей, але й утримувати існуючих.

У цьому контексті, сучасні технології стають не лише інструментами для оптимізації внутрішніх процесів ресторанного бізнесу, але і каналом для побудови та підтримки взаємовигідних відносин з клієнтами. Ресторани, які активно використовують ці можливості, не тільки підвищують ефективність своєї роботи, але й стають частинкою цифрового еко системного середовища, що сприяє їхньому успішному розвитку в еру сучасних технологій.

Ці технологічні трансформації необхідно розглядати в контексті зміни в

споживацьких патернах та очікуваннях. Клієнти сьогодення очікують не лише на смачну їжу, але й на високий рівень зручностей та персоналізованого обслуговування.

Можливість онлайн-замовлень, взаємодія з меню через мобільні застосунки та системи електронного платежу не лише спрощують життя клієнтів, але і стають критерієм вибору ресторану. Системи замовлення через мобільні застосунки, такі як UberEats чи Grubhub, роблять процес замовлення та доставки максимально зручним та ефективним.

Застосування інтегрованих систем управління рестораном, дозволяє ресторанам отримувати глибоке розуміння своєї діяльності.

1.2 Плюси і мінуси існуючих розробок

Реалізація зручностей для кінцевого користувача забезпечує сучасним клієнтам зручний та інтуїтивно зрозумілий інтерфейс для замовлення їжі та взаємодії з рестораном, що поліпшує досвід користувача, залучаючи нових клієнтів та забезпечуючи лояльність [8–9]. Інтегровані системи дозволяють ресторанам ефективно керувати своєю діяльністю, відслідковувати запаси, контролювати фінансовий потік та автоматизувати багато рутинних операцій, підвищуючи продуктивність персоналу та оптимізуючи робочі процеси [10]. Онлайн-сервіси, які забезпечують можливість замовлення та доставки їжі, різко зменшують час очікування клієнтів на обробку замовлень та отримання страв, що покращує ефективність обслуговування та задоволення клієнтських потреб [11].

Однак, використання платформ для замовлення їжі може призвести до значних витрат для ресторанів через високі комісійні ставки, що знижує їхню прибутковість та може негативно вплинути на бізнес-модель ресторану. Загальнопоширені рішення можуть неадекватно адаптуватися до унікальних вимог та особливостей конкретних ресторанів, що обмежує можливості

виокремлення себе на ринку через відсутність індивідуального підходу. Крім того, ресторани, що використовують сторонні платформи, можуть стати залежними від технічних змін, політик та умов співпраці цих постачальників, що створює ризики для стабільності та незалежності бізнесу.

1.3 Тенденції розробки мікросервісних застосунків для ресторанів

В сучасному світі ресторанний бізнес стрімко розвивається, і розробка мікросервісних застосунків стає ключовим елементом оптимізації управління. Аналіз літератури за останній рік надає цінний інсайт у тенденції та інновації в цій області.

Архітектура мікросервісів в розробці: Патерни та Практика - статті, що вивчають архітектурні патерни та практику мікросервісів, розкривають найсучасніші підходи до розробки застосунків, дозволяючи визначити оптимальні рішення для ресторанного бізнесу.

«Microservices Architecture Patterns: Best Practices in Modern Development» висвітлює різноманітні архітектурні патерни, ефективні в розробці мікросервісних застосунків, зокрема для забезпечення високої доступності, масштабованості та ефективного керування даними, надаючи конкретні приклади в контексті ресторанного бізнесу [12].

«Microservices in Practice: Real-world Examples from Leading Industries» розглядає практичне застосування мікросервісної архітектури в різних галузях, включаючи ресторанний бізнес, з акцентом на покращення ефективності управління замовленнями та обробкою платежів [13].

«Scaling Microservices: Lessons Learned from Industry Leaders» аналізує виклики та рішення при масштабуванні мікросервісів, надаючи приклади успішних рішень, що використовуються великими ресторанними мережами для управління мікросервісами, особливо при обслуговуванні великого обсягу замовлень та координації різних підсистем [14].

Ці приклади охоплюють різні аспекти архітектури мікросервісів та надають практичні приклади їх впровадження в розробці застосунків для ресторанного бізнесу.

Роль мови програмування C# в розробці мікросервісів висвітлюється в кількох статтях, присвячених використанню цієї мови для високопродуктивної та ефективної розробки.

«Maximizing Microservices Efficiency with C# Development» аналізує роль C# у високоефективній розробці мікросервісних застосунків, зосереджуючись на ключових особливостях, що сприяють зменшенню часу розробки та підвищенню продуктивності програмістів [15].

«Building Resilient Microservices with C# Exception Handling» розглядає, як C# допомагає забезпечити надійність та стійкість мікросервісів шляхом ефективного використання механізмів обробки винятків, надаючи конкретні приклади [16].

«Optimizing Microservices Communication in C# through gRPC» розкриває роль C# у оптимізації взаємодії мікросервісів, демонструючи, як використання технології gRPC може покращити продуктивність та ефективність комунікації між різними сервісами [17].

«C# and Event-Driven Architecture: Enhancing Microservices Scalability» розглядає використання C# для створення подійно-орієнтованих мікросервісів, надаючи приклади архітектурних шаблонів, що забезпечують масштабованість та ефективність розробки [18].

Ці матеріали досліджують різні аспекти використання C# у розробці мікросервісів, надаючи конкретні приклади та висвітлюючи переваги цієї мови в контексті створення ефективних застосунків.

Тенденції в ресторанному бізнесі та IT-інновації висвітлюються в дослідженнях, що надають контекст для розуміння потреб та вимог ринку, важливих при розробці застосунка.

«Revolutionizing the Dining Experience: IT Innovations in the Restaurant Industry» досліджує сучасні тенденції в ресторанному бізнесі, зосереджуючись

на ролі інформаційних технологій у вдосконаленні обслуговування та досвіду відвідувачів, розкриваючи вплив мобільних застосунків, онлайн-замовлень та систем управління столиками на ефективність та конкурентоспроможність ресторанів [19].

«Digital Menu Evolution: From Paper to Interactive Touchpoints» розглядає еволюцію цифрових меню, показуючи, як інтерактивні меню покращують взаємодію з клієнтами та оптимізують операційні процеси [20].

«Data-Driven Decision Making in the Culinary World» досліджує, як дані та аналітика стають ключовими для прийняття рішень у ресторанному бізнесі, демонструючи, як IT-інновації дозволяють ресторанам використовувати дані для вдосконалення меню, запасів та маркетингових стратегій [21].

«Enhancing Customer Engagement through Social Media Integration» обговорює роль соціальних медіа у підвищенні взаємодії з клієнтами, показуючи, як інтеграція з соціальними мережами та створення унікальних акцій допомагає залучати та утримувати клієнтів [22].

Ці дослідження розкривають тенденції в ресторанному бізнесі та IT-інновації, надаючи важливий контекст для розуміння потреб ринку та визначення ключових аспектів при розробці застосунка для ресторану «Mango».

Порівняльний аналіз існуючих систем управління ресторанами дозволяє виявити сильні та слабкі сторони конкурентів, що сприятиме ефективному проєктуванню нового застосунка.

«Evaluating the Landscape: Comparative Analysis of Restaurant Management Systems» аналізує існуючі системи управління ресторанами, зокрема популярні POS-системи та мікросервісні рішення, пропонуючи об’єктивний погляд на функціональні можливості, інтеграцію з платіжними системами, безпеку та гнучкість у використанні [23].

«From Legacy to Leading: Evolution of Restaurant Management Software» розглядає еволюцію систем управління ресторанами від традиційних до сучасних, проводячи порівняльний аналіз основних гравців ринку та

визначаючи, як вони адаптувалися до технологічних змін [24].

«User-Centric Approach: Assessing Restaurant Management Systems from the Operator’s Perspective» зосереджується на оцінці систем управління з точки зору операторів, досліджуючи, як користувальницький досвід впливає на ефективність використання та які аспекти важливі для покращення робочих процесів [25].

«Innovation in the Menu: Integrating Online Ordering and Inventory Management» порівнює системи, що інтегрують онлайн-замовлення та управління запасами, аналізуючи, як ці рішення пристосовуються до зростаючої популярності онлайн-сервісів та впливають на ефективність ресторанного бізнесу [26].

Ці дослідження детально розглядають різні аспекти існуючих систем управління ресторанами, дозволяючи отримати об’єктивний порівняльний аналіз, що стане важливим фундаментом для розробки нового застосунка у ресторанній галузі.

1.4 Постановка задачі

Об’єктом роботи стали технічні аспекти розробки та впровадження застосунку, розпочавши з аналізу технічного завдання, в якому визначено основні вимоги до функціоналу та властивостей системи. Метою роботи є створення ефективного та функціонального інформаційно-технічного середовища, спрямованого на поліпшення обслуговування клієнтів та оптимізацію управління ресторанним бізнесом.

Для досягнення мети необхідно виконати такі завдання:

– провести аналіз галузі ресторанного бізнесу та визначити основні потреби клієнтів і власників закладу;

– зібрати та підготувати дані про страви, меню, інгредієнти та інші характеристики ресторану для використання в системі;

– розробити чат на основі штучного інтелекту, в якому можна задавати питання щодо страв або отримувати консультацію;

– розробити систему дискримінації (сторона адміністратора), яка буде додавати, оновляти або видаляти страви;

– розробити систему для можливості перегляду поточних користувачів та реєстрації нових адміністраторів;

– розробити систему для можливості перегляду, видалення, оновлення поточних купонів на товари;

– розробити систему для можливості додавання, видалення товару до кошику та застосування одного з купонів;

– розробити систему для можливості оформлення замовлення;

– розробити систему для можливості авторизації та реєстрації;

– розробити систему для можливості авторизації зі сторони адміністратора;

– розробити функціонал щодо валідації даних при реєстрації, оформленні замовлення або при, додаванні, видаленні чи оновленні товару, купону або користувача;

– розробити зручний в використанні та цікавий інтерфейс застосунку;

– додати унікальні дизайнерські рішення, наприклад, анімації;

– провести тренування моделі на основі наявних даних;

– оцінити результати роботи системи, визначивши її ефективність.

2 ВИБІР ТЕХНОЛОГІЙ ТА ПРИНЦИПІВ ПРОЄКТУВАННЯ

2.1 Огляд історії мікросервісної архітектури

Мікросервісна архітектура являє собою концепцію програмного забезпечення, що складається з набору невеликих, незалежних сервісів, які взаємодіють між собою через легкий протокол. Ця архітектурна парадигма стала важливою складовою розвитку сучасного програмного забезпечення, проте її історія сягає не так далеко в минуле, як можна подумати [27–29].

Хоча ідеї, що лягли в основу мікросервісної архітектури, можна відстежити в багатьох попередніх концепціях, як от розподілена архітектура, SOA (Service-Oriented Architecture) та інших, справжній виток цієї архітектурної парадигми можна віднести до середини 2000-х років. В цей період компанії, такі як Amazon, Netflix та інші, стали стикатися з проблемами масштабування своїх систем.

З появою потреби в розширенні та збільшенні продуктивності, команди розробників почали шукати альтернативи монолітним архітектурам. Це спричинило появу поняття мікросервісів, тобто розділення програмного забезпечення на невеликі сервіси, які можуть бути розроблені, впроваджені та масштабовані незалежно один від одного.

У наступному десятилітті мікросервісна архітектура стала все більш популярною серед компаній. І це не дивно, оскільки вона пропонує ряд переваг, таких як покращена гнучкість, масштабованість, зменшення ризику та прискорення часу розробки.

Дана архітектура вже стала не просто технологічним трендом, а важливою складовою сучасного програмного забезпечення. Її історія, починаючи з невеликих початкових кроків у напрямку розподіленої архітектури, свідчить про необхідність постійного розвитку та пошуку оптимальних рішень у світі швидкозмінюваної технології.

2.2 Архітектура мікросервісів

Архітектура мікросервісів стала одним з найпопулярніших підходів у сучасному розробці програмного забезпечення. Її суть полягає в розбитті застосунку на невеликі, незалежні компоненти, які взаємодіють між собою через API. Цей підхід дозволяє досягти більшої гнучкості, швидкості вдосконалення та масштабованості.

Кожен сервіс відповідає за виконання конкретної функції, такої як авторизація, обробка замовлень або аналіз даних.

Сервіси взаємодіють між собою через API, що дозволяє їм обмінюватися даними та функціональністю. Це робить архітектуру більш гнучкою, оскільки окремі сервіси можуть бути реалізовані мовою програмування або технологією, яка найкраще відповідає їхнім потребам.

Кожен сервіс може бути розглянутий як окремий компонент, що працює автономно. Це означає, що зміни в одному сервісі не впливають на інші, що спрощує розробку та вдосконалення.

Для застосунку «Mango» було обрано саме мікросервісну архітектуру, що демонструє взаємодію різних компонентів системи через API (рис 2.1). Клієнтські частини застосунку поділяються на Web, що працює через браузер, який функціонує на мобільних пристроях. API забезпечує взаємодію між клієнтськими застосунками і мікросервісами, приймаючи запити від клієнтів та перенаправляючи їх до відповідних сервісів.

Мікросервіси включають Products, який відповідає за управління продуктами та обробляє запити, пов’язані з їх створенням, читанням, оновленням та видаленням; Identity, що займається автентифікацією та авторизацією користувачів, обробляючи запити на реєстрацію, вхід, управління ролями та правами доступу; Coupons, що керує купонами та обробляє запити на їх створення, читання, оновлення та видалення; і Cart, що відповідає за управління кошиком покупок, обробляючи запити на додавання продуктів до кошика, перегляд вмісту та оформлення замовлень.

Кожен мікросервіс має свою власну базу даних, яка зберігає дані, пов’язані з його функціональністю, забезпечуючи незалежність та автономність роботи.

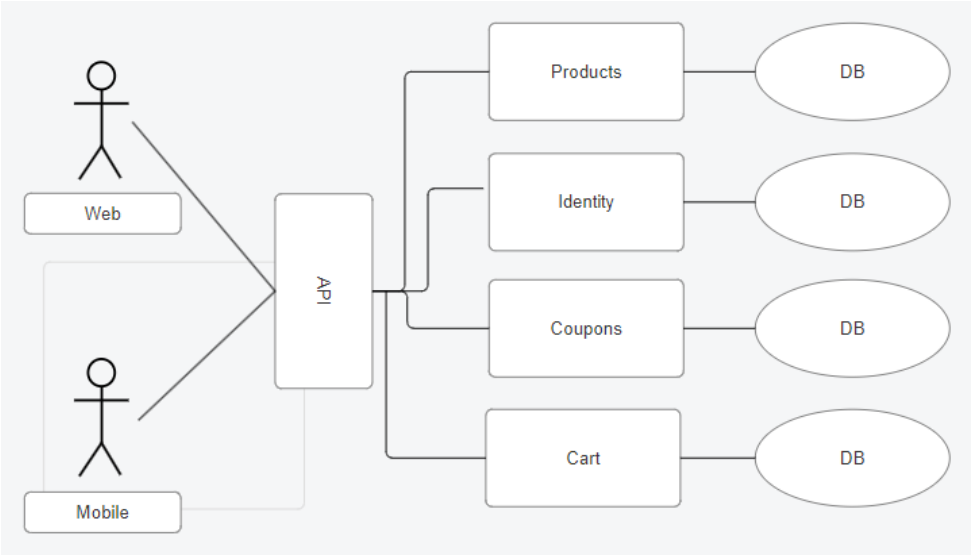

Рисунок 2.1 – Приклад архітектури мікросервісів застосунку «Mango»

Як це працює: клієнти Web та Mobile надсилають запити через API для взаємодії з системою, наприклад, для отримання списку продуктів, додавання продукту до кошика або застосування купона. API приймає ці запити та перенаправляє їх до відповідних мікросервісів. Кожен мікросервіс обробляє запити та взаємодіє зі своєю базою даних для отримання або оновлення даних. Наприклад, мікросервіс Cart може звернутися до своєї бази даних для додавання нового продукту до кошика користувача. Після обробки запиту мікросервіс надсилає відповідь назад через API до клієнта.

Ця архітектура мікросервісів забезпечує високу гнучкість, масштабованість та незалежність компонентів системи. Кожен мікросервіс виконує свою конкретну функцію та має власну базу даних, що дозволяє легко вносити зміни та масштабувати окремі частини системи без впливу на інші. Використання API забезпечує зручну взаємодію між клієнтами та мікросервісами, підвищуючи загальну ефективність та надійність системи.

Використання архітектури мікросервісів має ряд переваг, таких як

гнучкість, здатність швидко вносити зміни та вдосконалення в окремі сервіси; масштабованість, можливість масштабувати окремі сервіси відповідно до потреб; резистентність до відмов, випадок відмови одного сервісу не призводить до загального збою системи; технологічна різноманітність, можливість використання різних технологій для кожного сервісу в залежності від його потреб.

Архітектура мікросервісів є потужним інструментом для розробки сучасного програмного забезпечення. Її гнучкість та масштабованість дозволяють компаніям швидше реагувати на зміни на ринку та вдосконалювати свої продукти, а зростаюча популярність свідчить про їхню важливість у сучасній IT-індустрії.

2.3 Опис загальної концепції мікросервісної архітектури

Кожен мікросервіс відповідає за виконання конкретної функції або набору функцій і має власну логіку та базу коду. Ця архітектурна парадигма ґрунтується на принципах розділення великої системи на більш маневрені та керовані окремими модулями, які можуть бути розроблені, масштабовані, випущені та підтримувані незалежно один від одного. Основні характеристики мікросервісної архітектури включають розділення функціональності, незалежність розгортання та масштабування, реактивність і гнучкість, стійкість до відмов, а також командну автономію [30–31].

Розділення функціональності передбачає, що кожен мікросервіс виконує лише певну функцію, що зменшує складність кожного окремого сервісу і полегшує їхнє розроблення та підтримку. Незалежність розгортання та масштабування дозволяє мікросервісам бути розгорнутими та масштабованими окремо один від одного, що забезпечує ефективне використання ресурсів і можливість швидко реагувати на змінні потреби без необхідності розгортання всієї системи [32]. Реактивність та гнучкість

архітектури дозволяє створювати системи, які легко адаптуються до змін у вимогах або умовах використання, а кожен мікросервіс може бути розроблений, випущений та масштабований окремо, що полегшує зміни та підтримку.

Стійкість до відмов означає, що відмова одного сервісу не призводить до відмови всієї системи, адже кожен мікросервіс може бути проєктований для самостійного відновлення після відмови або неполадки. Командна автономія забезпечується тим, що кожен мікросервіс може бути розроблений та підтримуваний незалежною командою, що сприяє високій швидкості розробки та реакції на зміни, а також полегшує комунікацію та співпрацю між різними групами розробників.

Дана архітектура стала важливою складовою розвитку сучасних програмних систем, дозволяючи компаніям створювати більш гнучкі, масштабовані та стійкі до відмов застосунки. Цей підхід продовжує набирати популярність, особливо серед компаній, які стикаються з високими вимогами до масштабованості, стійкості та ефективності своїх програмних систем.

У проєкті було обрано мікросервісну архітектуру для створення системи, оскільки вона надає багато переваг у порівнянні з монолітною архітектурою. Кожен описаних мікросервісів, таких як Products, Identity, Cart і Coupon, відповідає за конкретну функціональність у системі. Наприклад, у мікросервісі Products було реалізовано CRUD операції для управління товарами та була інтегрована система чату для підтримки клієнтів.

За допомогою стандартних протоколів комунікації, таких як HTTP/REST або messaging, мікросервіси можуть обмінюватися даними та взаємодіяти між собою. Моя архітектура також забезпечує масштабованість, оскільки кожен мікросервіс може бути масштабований незалежно залежно від навантаження. Це дозволяє оптимізувати використання ресурсів та забезпечує ефективну роботу системи під високими навантаженнями.

Загалом, мікросервісна архітектура дозволяє створити гнучку, модульну та розширювану систему, яка відповідає потребам застосунку.

2.4 Основні принципи мікросервісної архітектури

Мікросервісна архітектура є комплексним підходом до розробки програмного забезпечення, який базується на розбитті застосунку на невеликі, самодостатні сервіси, які працюють разом із взаємодією через мережу.

Мікросервіси повинні бути невеликими, самодостатніми компонентами, які виконують конкретні функції. Це дозволяє їм бути легко зрозумілими, підтримуваними та масштабованими незалежно один від одного.

Кожен мікросервіс повинен бути незалежним від інших. Це означає, що вони можуть бути розроблені, вдосконалені та масштабовані незалежно один від одного, що спрощує процес розробки та розгортання.

Комунікація між мікросервісами повинна бути легкою та ефективною. Зазвичай вона здійснюється через API або інші зручні механізми взаємодії.

Автоматизація є ключовим принципом мікросервісної архітектури. Це означає використання інструментів для автоматизації процесів розгортання, моніторингу, масштабування та управління сервісами.

Кожен мікросервіс повинен бути відповідальним за свої функції та даних. Це означає, що він повинен бути самостійним і не залежати від інших сервісів для свого функціонування.

Мікросервіси повинні бути здатні надавати інформацію про свій стан та працездатність. Це дозволяє забезпечити відповідне моніторинг та реагування на проблеми.

Ці принципи допомагають створити гнучкі, масштабовані та надійні системи на основі мікросервісної архітектури. Вони є основою успішного впровадження цього підходу у розробці програмного забезпечення.

2.5 Технології та принципи проєктування

2.5.1 Фреймворки та хмарні сховища

Для розробки мікросервісів на мові програмування C# існують кілька потужних вебфреймворків, які дозволяють створювати невеликі та незалежні сервіси, але в рамках цього проєкту використовується саме ASP.NET Core. ASP.NET Core є відкритою та крос-платформенною версією вебфреймворку ASP.NET, що надає потужні можливості для створення мікросервісів за допомогою вбудованих бібліотек для створення HTTP-серверів, маршрутизації, обробки запитів та інших функцій. Він також підтримує контейнеризацію, що робить його ідеальним вибором для розгортання мікросервісів у хмарних середовищах [33–35].

Хмарні технології стали невід’ємною частиною сучасної розробки програмного забезпечення, забезпечуючи високу доступність, масштабованість та безпеку даних. Одним з провідних постачальників хмарних послуг є платформа Microsoft Azure, яка надає широкий спектр інструментів для зберігання та обробки даних. Було обрано використання хмарного сховища Azure для зберігання всіх зображень, що використовуються в застосунку. Azure Blob Storage є однією з основних служб зберігання даних в Azure, яка дозволяє зберігати великі обсяги неструктурованих даних, таких як зображення, відео, аудіо та інші мультимедійні файли. Ця служба пропонує кілька важливих можливостей: масштабованість, яка автоматично адаптується до зростаючих потреб у зберіганні, забезпечуючи ефективну роботу навіть при великому обсязі даних; висока доступність даних завдяки глобально розподіленій інфраструктурі Azure, що дозволяє отримувати доступ до даних з будь-якої точки світу; та безпека, яка включає шифрування даних у стані спокою та під час передачі, а також контроль доступу на основі ролей.

Було додано зберігання зображень у хмарному сховищі Azure, що забезпечує кілька ключових переваг. По-перше, централізоване зберігання всіх зображень проєкту в одному місці спрощує управління та доступ до

даних, що особливо важливо при роботі з великими обсягами зображень. По-друге, використання Azure забезпечує швидкий доступ до зображень завдяки глобально розподіленій мережі серверів, що дозволяє користувачам застосунку швидко завантажувати зображення незалежно від їхнього місцезнаходження. Хмарне сховище Azure надає високий рівень безпеки для збережених даних, включаючи шифрування та резервне копіювання, що забезпечує надійний захист даних від втрати або несанкціонованого доступу.

Було використано структуру хмарного сховища Azure Blob Storage, яка використовується для зберігання даних у вигляді блобів (рис. 2.2). Обліковий запис сховища в Azure, який є основним контейнером для всіх даних, називається pbaccount і може містити кілька контейнерів. Один з таких контейнерів, prodvhds, використовується для зберігання віртуальних жорстких дисків (VHD) для продуктивних середовищ, а інший контейнер, testvhds, призначений для тестових середовищ. У контейнерах зберігаються сторінкові блоби, такі як os1.vhd, os2.vhd, os3.vhd, os4.vhd, що часто використовуються для зберігання великих файлів, таких як віртуальні жорсткі диски.

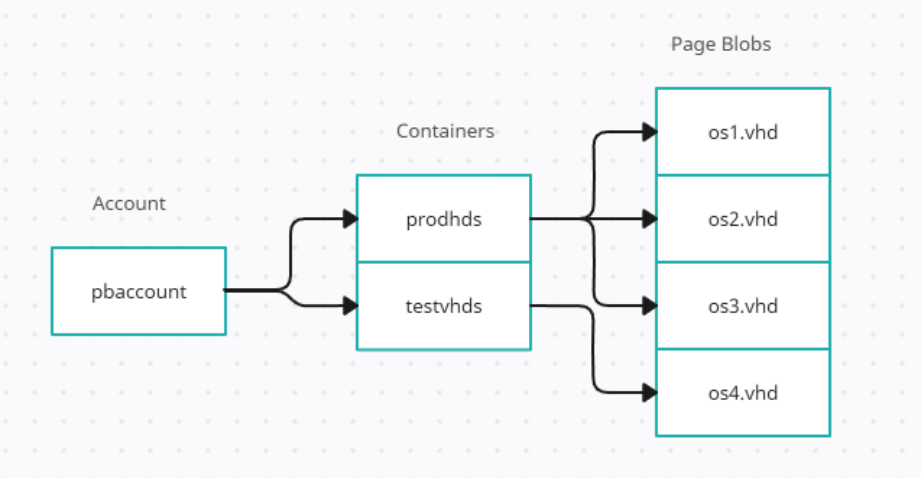

Рисунок 2.2 – Структура хмарного сховища Azure Blob Storage

В обліковому записі сховища може бути кілька контейнерів, кожен з яких містить кілька блобів. У цьому прикладі контейнери prodvhds і testvhds містять сторінкові блоби, які представляють собою віртуальні жорсткі диски. У поточному проєкті хмарне сховище Azure Blob Storage використовується

для зберігання всіх зображень. Було створено контейнер, images для зберігання всіх зображень проєкту, оскільки блоби в цьому контейнері, забезпечать централізоване, безпечне та масштабоване сховище для мультимедійних файлів.

Архітектура рішення включає завантаження зображень до Azure Blob Storage за допомогою спеціально розробленого інтерфейсу користувача у застосунку. Після завантаження зображення отримує унікальний URL, який зберігається в базі даних проєкту. Для відображення зображень у застосунку використовується цей URL, що дозволяє швидко завантажувати та відображати зображення у користувацькому інтерфейсі. Адміністратори застосунку можуть керувати зображеннями, використовуючи інструменти Azure для перегляду, оновлення або видалення зображень з хмарного сховища.

Використання хмарного сховища Azure для зберігання зображень у даному проєкті забезпечує високу ефективність, доступність та безпеку даних. Це дозволяє зосередитися на розробці функціональності застосунку, не турбуючись про проблеми зі зберіганням та доступом до мультимедійних файлів.

2.5.2 Принципи проєктування баз даних для мікросервісів

При проєктуванні баз даних для окремих мікросервісів було проаналізовано кілька ключових понять та принципів. Перш за все, важливо розуміти принцип супутності, тобто кожен мікросервіс має свою власну базу даних, яка повністю відповідає його функціональності. Це дозволяє забезпечити велику незалежність між сервісами та уникнути зайвих зв’язків, що можуть призвести до складнощів при масштабуванні або зміні [36–39].

Було враховано принцип єдиної відповідальності, який означає, що кожен мікросервіс має бути відповідальним лише за певну частину бізнес-логіки або функціональність. Це допомагає забезпечити чистоту та простоту

коду, а також полегшує розуміння та підтримку системи в цілому.

Крім того, враховано принцип надійності, що означає забезпечення доступності та стійкості бази даних. Для цього можуть використовуватися різноманітні стратегії реплікації, резервного копіювання та моніторингу для виявлення та усунення проблем. Наприклад, при проєктуванні бази даних для мікросервісів було використано принципи супутності, єдиної відповідальності та надійності, щоб забезпечити ефективну та стійку архітектуру системи.

Нормальні форми в базах даних є важливими для забезпечення їхньої ефективності та структурованості. У контексті проєктування баз даних для Identity Server4 та інших проаналізовано принципами трьох основних нормальних форм (1НФ, 2НФ, 3НФ) та зв’язками між таблицями.

Перша нормальна форма (1НФ) означає, що кожен атрибут в таблиці повинен бути атомарним, тобто неділеним на більші елементарні частини. У контексті, наприклад, Identity Server4 це означає, що кожен атрибут користувача або клієнта (наприклад, ім’я, електронна пошта, пароль) має бути представлений як окремий стовпець у таблиці користувачів чи клієнтів.

Друга нормальна форма (2НФ) вимагає, щоб кожен атрибут був пов’язаний з унікальним ключем, тобто не повинно бути часткових залежностей атрибутів від ключа. У випадку Identity Server4 це означає, що інформація про користувачів та їх додаткові атрибути (наприклад, ролі або дозволи) збережені в окремих таблицях і пов’язані з основною таблицею користувачів за допомогою зовнішніх ключів.

Третя нормальна форма (3НФ) вимагає усунення транзитивних залежностей між атрибутами, тобто кожен атрибут повинен залежати від ключа, а не від інших атрибутів. У випадку Identity Server4 це означає, що дані про користувачів та їх атрибути мають бути декомпозовані таким чином, щоб уникнути зберігання зайвої інформації та забезпечити консистентність даних.

2.5.3 Опис структур баз даних

Як основну СУБД, було обрано SSMS.

База даних для Identity Server4 включає в себе набір таблиць, які зберігають інформацію про користувачів, клієнтів, згоди на доступ та інші пов’язані дані, необхідні для автентифікації, авторизації та управління доступом у системі (рис. 2.3). Основні складові бази даних Identity Server4 включають таблицю Користувачів, яка зберігає інформацію про користувачів, таку як ідентифікатори, ім’я, електронну пошту, хеш паролю та інші атрибути, необхідні для ідентифікації та аутентифікації. Таблиця Клієнтів містить дані про клієнтські застосунки або сервіси, які використовують Identity Server4 для аутентифікації та отримання доступу до ресурсів, включаючи ідентифікатори, секретні ключі, типи клієнтів та інші атрибути. Таблиця Згод зберігає дані про згоди користувачів на доступ до їхніх особистих даних з боку різних клієнтських застосунків, включаючи ідентифікатори користувачів та клієнтів, обсяги доступу та інші відомості. Таблиці Ролей та Дозволів використовуються для зберігання інформації про ролі користувачів та дозволи на доступ до ресурсів, що дозволяє налаштовувати права доступу користувачів на основі їхніх ролей та дозволів. Крім того, можуть бути створені додаткові допоміжні таблиці для зберігання інформації про сесії, журналізацію подій, аудит доступу та інші допоміжні дані. Усі ці таблиці утворюють структуру бази даних Identity Server4, яка дозволяє здійснювати ефективну автентифікацію та авторизацію користувачів, керувати доступом до ресурсів і забезпечувати безпеку у системі.

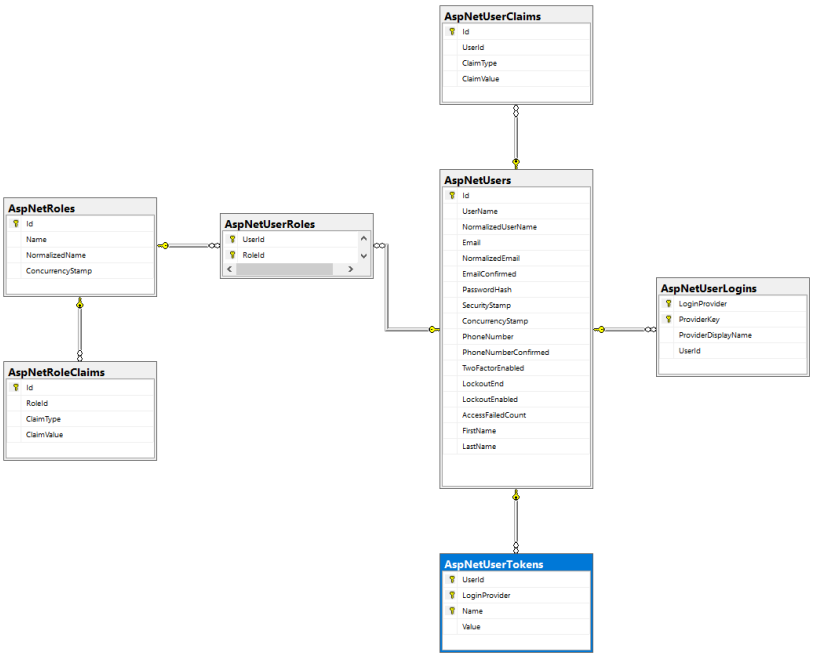

Рисунок 2.3 – Архітектура бази даних для реєстрації та авторизації

Кожен мікросервис має свою окрему базу даних, тому крім бази Identity існує ще декілька.

База даних Coupons містить дві таблиці: __EFMigrationsHistory та Coupons (рис. 2.4).

Таблиця __EFMigrationsHistory включає поле MigrationId, яке є унікальним ідентифікатором міграції та використовується для відстеження історії міграцій бази даних, а також поле ProductVersion, що вказує версію Entity Framework Core, яка використовувалася для створення міграції.

Таблиця Coupons містить поля CouponId, що є автоматично збільшуваним цілочисельним ідентифікатором для однозначної ідентифікації купона; CouponCode, текстове поле з унікальним кодом купона; та DiscountAmount, числове поле, яке вказує розмір знижки, що надається за допомогою цього купона. Ці таблиці використовуються для зберігання та керування купонами, що надаються користувачам, а також для відстеження історії міграцій бази даних, забезпечуючи консистентність схеми даних.

Рисунок 2.4 – Архітектура бази даних для зберігання купонів

База даних Products (рис. 2.5), містить дві таблиці: __EFMigrationsHistory та Products.

Таблиця __EFMigrationsHistory використовується для відстеження міграцій, застосованих до бази даних під час розробки за допомогою Entity Framework, і містить поля MigrationId (унікальний ідентифікатор міграції) та ProductVersion (версія Entity Framework, яка була використана для виконання міграції).

Таблиця Products містить інформацію про продукти з полями Id (унікальний ідентифікатор продукту, що генерується автоматично), Name (назва продукту), CategoryName (назва категорії, до якої відноситься продукт), Price (ціна продукту), Description (опис продукту) та ImageUrl (посилання на зображення продукту).

Обидві таблиці мають визначений первинний ключ для унікальності записів, а також встановлені обмеження щодо деяких атрибутів для забезпечення правильності та ефективності даних.

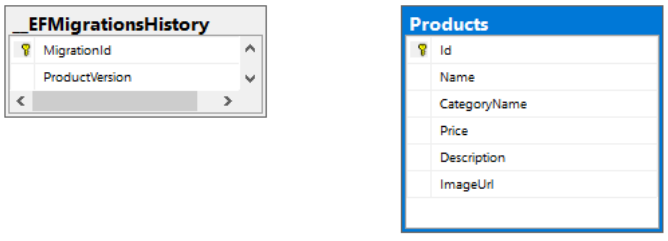

Рисунок 2.5 – Архітектура бази даних для зберігання товарів

База даних Cart використовується для зберігання даних про кошик (рис 2.6).

Таблиця __EFMigrationsHistory використовується схемою міграції Entity Framework для відстеження та керування версіями міграцій бази даних, зберігаючи ідентифікатор міграції та версію продукту.

Таблиця CartHeaders містить інформацію про заголовки кошиків, зберігаючи унікальні ідентифікатори кошиків, ідентифікатори користувачів, які їх створили, та можливі купонні коди, що можуть бути застосовані до кошика.

Таблиця CartDetails містить деталі кошиків, зберігаючи інформацію про товари, додані до конкретного кошика, такі як ідентифікатори товарів, кількість одиниць товарів та посилання на відповідний заголовок кошика. У таблиці CartDetails встановлені зовнішні ключі, які забезпечують зв’язок з таблицею CartHeaders, що гарантує цілісність даних, дозволяючи кожній деталі кошика посилатися на відповідний заголовок.

Ця база даних призначена для управління кошиками покупок у вебзастосунку, і її структура дозволяє ефективно зберігати та керувати інформацією про товари, що додаються до кошиків користувачів.

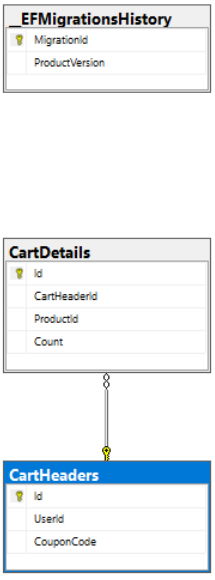

Рисунок 2.6 – Архітектура бази даних для керування кошиком

2.5.4 Методи комунікації між мікросервісами

Методи комунікації між мікросервісами грають важливу роль у забезпеченні взаємодії та обміну даними між різними компонентами системи [40–41].

HTTP і REST (Representational State Transfer) є одними з найпоширеніших протоколів та архітектур для комунікації між мікросервісами. Кожен мікросервіс надає HTTP API, яке інші сервіси можуть використовувати для виконання запитів та обміну даними. RESTful архітектура встановлює стандарти щодо організації ресурсів та використання HTTP методів (GET, POST, PUT, DELETE) для взаємодії з ними.

Підходи до мікросервісної архітектури використовують повідомлення та черги повідомлень для комунікації. Кожен мікросервіс публікує повідомлення в черзі, яке інші сервіси підписують та споживають. Цей метод дозволяє розділити сервіси в часі та забезпечити асинхронну взаємодію [42–43].

Було проаналізовано ієрархію та прийняте рішення щодо важливості використання RESTful API. Цей підхід дозволяє ефективно обмінюватися даними між окремими сервісами за допомогою стандартних HTTP-запитів та відповідей.

Скрипти, що створюють таблиці у базі даних – є основою для мікросервісів. Однак для забезпечення взаємодії між мікросервісами та фронтендом, було обрано RESTful API [44–45].

Відправлений запит з фронтенду надходить до конкретного мікросервісу через HTTP-протокол. Мікросервіс отримує цей запит і виконує відповідні операції з базою даних, відповідно до запиту. Наприклад, якщо потрібні дані про купони, запит поступає до таблиці Coupons. Після обробки запиту, мікросервіс генерує відповідь, яка може бути у форматі JSON або іншому зрозумілому форматі для фронтенду, і надсилає її назад через HTTP-відповідь. На фронтенді програма отримує цю відповідь і обробляє отримані дані відповідно до потреб користувача. Ця стратегія взаємодії між мікросервісами

та фронтендом дозволяє створювати ефективні та зручні застосунки.

Обробляють продуктів та кошику у системі починається з клієнтського застосунку або інтерфейсу, що відправляє запити до системи (рис. 2.7). API виступає посередником між користувачем та мікросервісами, приймаючи запити та перенаправляючи їх до відповідних мікросервісів. Мікросервіс продуктів відповідає за управління продуктами, обробляючи запити, пов'язані з отриманням інформації про продукти з бази даних продуктів. Мікросервіс кошика, у свою чергу, відповідає за управління кошиком покупок, обробляючи запити, пов’язані з додаванням продуктів до кошика та взаємодією з базою даних кошика.

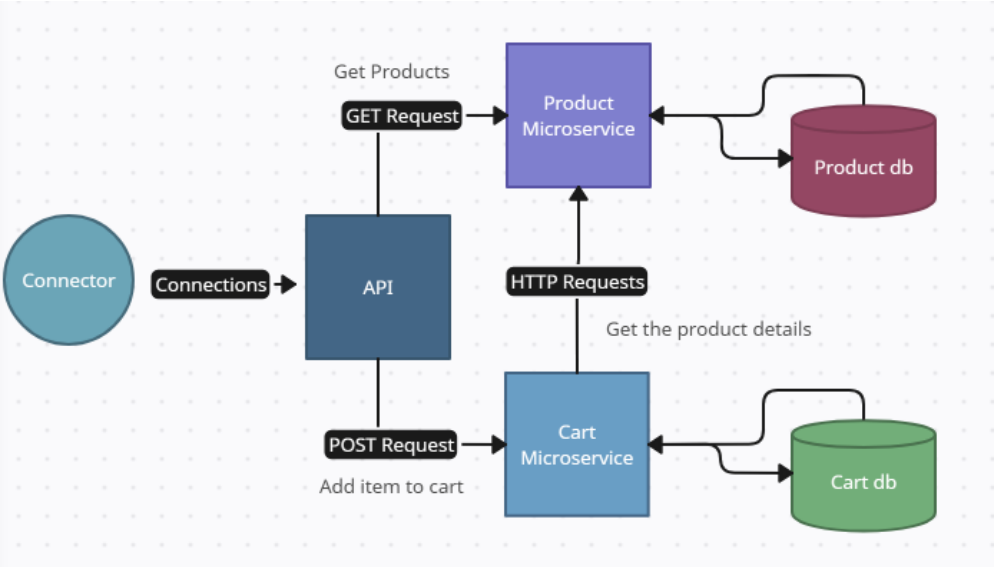

Рисунок 2.7 – Приклад коммунікації між користувачем та мікросервисами продуктів та корзини

Процес отримання продуктів починається з того, що користувач надсилає запит на отримання списку продуктів до API, який перенаправляє цей запит до мікросервісу продуктів. Мікросервіс продуктів звертається до бази даних продуктів для отримання необхідної інформації та надсилає отримані дані назад через API до користувача.

Додавання продукту до кошика починається з запиту користувача до API, який перенаправляє його до мікросервісу кошика. Мікросервіс кошика

може звернутися до мікросервісу продуктів для отримання деталей продукту, а потім додати продукт до кошика та оновити інформацію в базі даних кошика.

Після успішного додавання продукту, мікросервіс кошика надсилає підтвердження назад через API до користувача.

Діаграма демонструє, як клієнтські запити проходять через API до відповідних мікросервісів, які взаємодіють з базами даних для виконання запитів. Така архітектура забезпечує гнучкість, масштабованість та модульність системи, дозволяючи кожному мікросервісу відповідати за свою конкретну функціональність і працювати незалежно від інших компонентів.

2.5.5 Авторизація та реєстрація

Алгоритм реєстрації та авторизації в Identity Server4 може виглядати наступним чином. Реєстрація першого адміністратора здійснюється вручну через адміністративний інтерфейс або скрипт, при цьому встановлюється спеціальний флаг або роль, яка вказує на статус адміністратора. Перший адміністратор отримує можливість реєструвати інших адміністраторів через адміністративний інтерфейс або API, причому інші адміністратори також можуть мати право реєструвати нових адміністраторів. Звичайні користувачі можуть реєструватися через стандартну реєстраційну форму або API, після чого їм призначається роль користувача зі звичайними правами доступу.

Авторизація користувачів здійснюється після успішної аутентифікації, наприклад, за допомогою логіна та пароля, після чого Identity Server4 видає токен доступу, який використовується для доступу до захищених ресурсів у системі. Такий алгоритм дозволяє ефективно керувати реєстрацією та авторизацією користувачів, включаючи адміністраторів, а також передбачає можливість іншим адміністраторам реєструвати нових адміністраторів, тоді як звичайним користувачам доступна лише реєстрація стандартних користувацьких облікових записів.

У даній системі необхідно пройти авторизацію перед покупкою товарів, що забезпечує додатковий рівень безпеки та контролю доступу до функціоналу системи. Авторизація перед покупкою передбачає, що користувачі повинні бути авторизовані у системі, щоб отримати доступ до функціоналу покупки товарів, включаючи введення логіна та пароля або використання інших методів аутентифікації. Контроль доступу до покупки означає, що система перевіряє авторизацію користувача перед дозволом на покупку товарів, що включає перевірку прав доступу або ролей користувачів для підтвердження їхньої здатності до здійснення покупок. Система також чітко повідомляє користувачів про необхідність авторизації перед покупкою товарів. Це підвищує рівень безпеки та контролю над транзакціями в системі, а також забезпечує кращий досвід користувача шляхом надання можливості придбати товари лише дозволеним користувачам.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Вибір мови та інструментальних засобів

C# – це сучасна, об’єктно-орієнтована мова програмування, розроблена компанією Microsoft, яка об’єднує потужність C++ з простотою Visual Basic. Вона широко використовується для розробки різних типів застосунків, включаючи вебзастосунки, настільні застосунки, ігри, мобільні застосунки та хмарні сервіси. Як строго об’єктно-орієнтована мова, C# організовує всі елементи коду, включаючи змінні, функції та структури даних, у класи та об’єкти, що сприяє повторному використанню коду та модульності. Мова має сильну типізацію, яка забезпечує визначення типу для кожної змінної, що допомагає запобігати помилкам на ранніх етапах розробки, підвищуючи надійність коду.

C# підтримує відкритий код, оскільки компілятор та багато бібліотек .NET є проєктами з відкритим кодом, що забезпечує прозорість і можливість внесення змін спільнотою розробників. Завдяки .NET Core (тепер просто .NET), застосунки на C# можуть запускатися на різних операційних системах, включаючи Windows, macOS та Linux. Мова також має доступ до великої стандартної бібліотеки .NET, яка включає багато класів для роботи з файлами, введенням та виведенням, мережевим програмуванням, базами даних та іншими загальними задачами.

Однією з переваг C# є автоматичне управління пам’яттю за допомогою збирача сміття (garbage collector), який автоматично керує виділенням та звільненням пам’яті, зменшуючи кількість помилок, пов’язаних з неправильним управлінням пам’яттю, таких як витоки пам’яті. Мова підтримує сучасні мовні конструкції, такі як LINQ (Language Integrated Query), асинхронне програмування, анонімні методи та лямбда-вирази.

C# є ідеальною мовою для розробки мікросервісних застосунків завдяки підтримці .NET Core, який є крос-платформенною версією .NET, що дозволяє

створювати масштабовані та високопродуктивні застосунки на різних операційних системах. ASP.NET Core, фреймворк для побудови вебзастосунків та сервісів на основі .NET, оптимізований для розробки мікросервісів завдяки високій продуктивності, легкій ваговій моделі та підтримці сучасних стандартів вебпрограмування. C# також має відмінну інтеграцію з хмарними сервісами, такими як Microsoft Azure, що дозволяє легко розгортати та масштабувати мікросервіси у хмарному середовищі.

Асинхронне програмування в C# підтримується через ключові слова async та await, що дозволяє створювати високопродуктивні застосунки, які ефективно використовують ресурси та можуть обробляти велику кількість одночасних запитів. Microsoft надає потужні інструменти для розробки на C#, такі як Visual Studio та Visual Studio Code, які забезпечують розширені можливості налагодження, автозаповнення коду, інтеграцію з системами контролю версій та багато іншого.

У поточному проєкті було використано такі бібліотеки: Duende.IdentityServer для реалізації аутентифікації та авторизації, Microsoft.AspNetCore.Authentication для базової аутентифікації, Microsoft.AspNetCore.Authentication.OpenIdConnect для інтеграції з провайдерами аутентифікації, Newtonsoft.Json для роботи з JSON, System.IdentityModel.Tokens.Jwt для управління JWT токенами, Microsoft.EntityFrameworkCore для роботи з базами даних, AutoMapper для мапінгу об’єктів, а також Swashbuckle.AspNetCore для автоматичного генерування документації API на основі Swagger.

Razor, синтаксис для створення динамічних вебсторінок в ASP.NET, розроблений Microsoft, використовується в ASP.NET MVC та ASP.NET Core для створення вебзастосунків. Він забезпечує зручний спосіб написання HTML з вбудованим C# кодом, що дозволяє створювати потужні та інтерактивні вебінтерфейси. Razor відзначається легкістю у використанні, інтеграцією з ASP.NET, зручним синтаксисом, підтримкою IntelliSense у

Visual Studio, та безпекою завдяки автоматичному екрануванню вихідних даних, що захищає від атак типу XSS.

Razor використовується для створення представлень (views) в ASP.NET Core MVC та сторінок у Razor Pages. Представлення Razor є частиною архітектури MVC і використовуються для відображення даних, що передаються від контролерів, зазвичай зберігаючись у папці Views. Razor Pages поєднують логіку та представлення в одному файлі, спрощуючи структуру проєкту, та зберігаються у папці Pages.

Razor є потужним і гнучким інструментом для створення динамічних вебсторінок в ASP.NET Core. Він забезпечує легкий у використанні синтаксис для інтеграції C# коду з HTML, підтримує всі можливості фреймворків ASP.NET та забезпечує високу безпеку. Використання Razor дозволяє розробникам швидко та ефективно створювати сучасні вебзастосунки з інтерактивними інтерфейсами.

3.2 Архітектура проєкту. Використання N-Tier та MVC

Проєкт «MangoRestaurant» має комплексну структуру, що складається з декількох мікросервісів та загальних компонентів. Така структура забезпечує модульність, масштабованість та зручність у підтримці і розширенні проєкту.

Проєкт «Mango.Web» має фронтенд частину, яка відповідає за інтерфейс користувача та взаємодію з користувачами через вебзастосунок (рис. 3.2). Проєкт включає різноманітні компоненти, що забезпечують його функціональність.

Connected Services інтегрує зовнішні сервіси, APIModes містить моделі, що використовуються для передачі даних між клієнтом та сервером через API, а CONST зберігає константи, що використовуються у застосунку.

Controllers обробляють HTTP-запити та взаємодіють з моделями та представленнями, тоді як Extensions містить розширення для застосунку.

Models представляють дані, що використовуються у застосунку, Services забезпечують бізнес-логіку, а ViewModels передають дані між контролерами та представленнями.

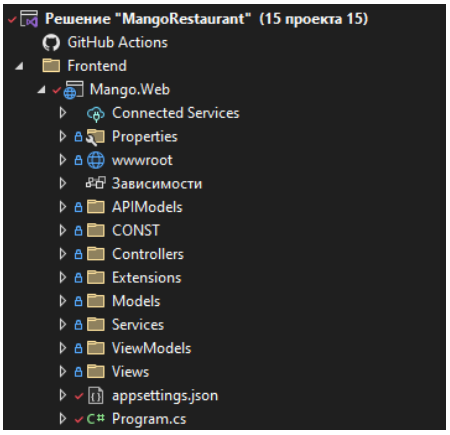

Рисунок 3.2 – Архітектура проєкту «Mango.Web»

Views відповідають за відображення даних користувачу за допомогою Razor views.

Конфігураційний файл appsettings.json зберігає налаштування застосунку, а Program.cs є головним файлом для запуску застосунку.

Секція Services містить окремі мікросервіси, що відповідають за різні функціональні частини системи (рис. 3.3).

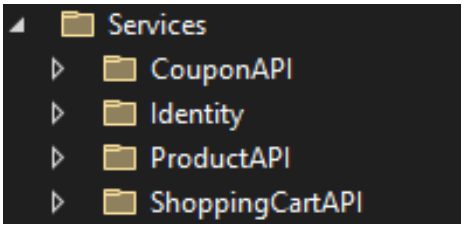

Рисунок 3.3 – Вміст папки Services

Мікросервіси, включені до цієї секції, кожен з яких відповідає за свою специфічну функціональність, забезпечують модульність і гнучкість системи, дозволяючи ефективно розробляти, тестувати та розгортати окремі компоненти без впливу на інші частини системи. Кожен мікросервіс має власну базу даних і незалежно виконувати свої задачі, обробляючи відповідні

запити від користувачів або інших сервісів.

Архітектура включає в себе проєкт CouponAPI, який відповідає за обробку запитів, пов’язаних з купонами (рис. 3.4). Цей проєкт включає кілька ключових компонентів, кожен з яких виконує специфічні функції для забезпечення роботи сервісу.

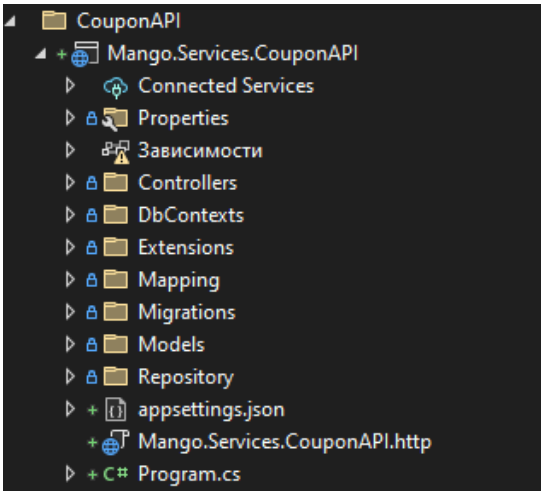

Рисунок 3.4 – Структура мікросервісу для опису купонів CouponAPI

Connected Services забезпечує інтеграцію з зовнішніми сервісами, а Properties містить налаштування проєкту. Конфігураційний файл appsettings.json зберігає налаштування сервісу.

Controllers обробляють HTTP-запити для купонів, тоді як DbContexts містять контексти бази даних, які використовуються EF Core для доступу до бази даних. Extensions містить розширення для сервісу, а Mapping - конфігурації для мапінгу об’єктів, наприклад, з використанням AutoMapper.

Migrations містить міграції для бази даних, що дозволяє управляти змінами в схемі бази даних. Models представляють дані, що використовуються у сервісі.

Repository включає репозиторії для доступу до даних, забезпечуючи абстракцію над операціями з базою даних та спрощуючи управління даними.

Головний файл Program.cs відповідає за запуск сервісу, налаштування середовища та конфігурацію всіх необхідних компонентів.

Секція Identity структурована для кращої організації та чіткості, розділена на дві основні частини: BusinessLogic та DataAccess (рис. 3.5).

У частині BusinessLogic включені Extensions, які містять розширення для бізнес-логіки, та Services, що реалізують бізнес-логіку для аутентифікації та авторизації, забезпечуючи основні функції безпеки та управління доступом у системі.

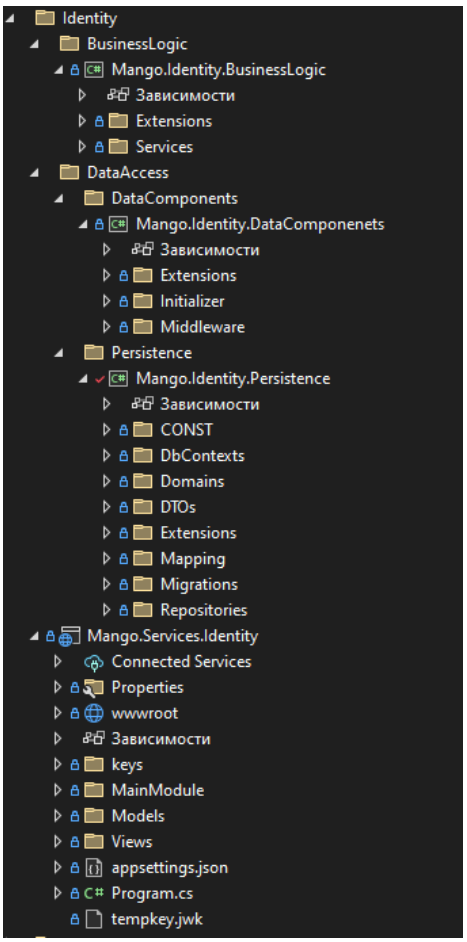

Рисунок 3.5 - Структура мікросервісу Identity для реєстрації/авторізації користувачів

Частина DataAccess включає кілька компонентів. DataComponents містять Extensions для доступу до даних, Initializer для налаштування початкових даних і Middleware для обробки HTTP-запитів, що забезпечують ефективну роботу з даними та інтеграцію різних компонентів системи.

Секція Persistence включає CONST, де зберігаються константи для роботи з базою даних, DbContexts, які містять контексти бази даних, та Domains, які представляють сутності бізнес-логіки.

DTOs використовуються для передачі даних між різними частинами системи, а Extensions забезпечують розширення для роботи з базою даних.

Migrations містить міграції для бази даних, дозволяючи керувати змінами в схемі бази даних, зберігаючи цілісність даних під час оновлень.

Repositories включають репозиторії для доступу до даних, що забезпечують абстракцію над операціями з базою даних, спрощуючи управління даними.

Ця структура дозволяє чітко організувати та розділити різні аспекти управління і доступу до даних, забезпечуючи модульність і легкість в управлінні системою. Такий підхід допомагає підтримувати високу якість коду, спрощує процес розробки та полегшує інтеграцію з іншими компонентами системи.

Представлена структура проєкту ProductAPI, яка аналогічна до структури CouponAPI і включає дві основні частини: BusinessLogic та DataAccess, що забезпечують управління продуктами у системі (рис. 3.6).

Частина BusinessLogic містить сервіси, які реалізують бізнес-логіку для управління продуктами, обробляючи відповідні запити та забезпечуючи основні функції продуктового сервісу, а також розширення для бізнес-логіки, що додають додаткову функціональність і допомагають організувати код більш ефективно.

Частина DataAccess включає DataComponents, до яких належать розширення для доступу до даних, що полегшують роботу з базою даних і забезпечують інтеграцію з іншими компонентами, ініціалізатори для налаштування початкових даних у базі даних, що допомагають підготувати середовище для роботи з даними, та проміжне ПЗ для обробки HTTP-запитів, що забезпечує обробку запитів і відповіді на них, покращуючи взаємодію між клієнтом і сервером.

Секція Persistence включає константи для роботи з базою даних, що визначають налаштування і параметри, необхідні для взаємодії з даними, контексти бази даних, які використовуються для доступу до даних за допомогою EF Core, домени або сутності бізнес-логіки, що представляють структуру даних продуктів, об’єкти передачі даних, які використовуються для обміну даними між різними частинами системи, розширення для роботи з базою даних, що додають додаткові функції для ефективної взаємодії з даними, міграції для бази даних, що забезпечують керування змінами у схемі бази даних, та репозиторії для доступу до даних, що забезпечують абстракцію над операціями з базою даних, спрощуючи управління даними.

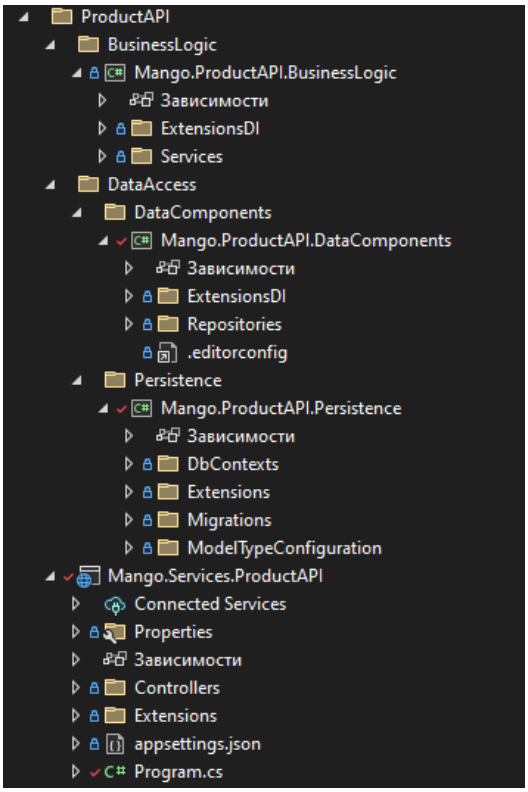

Рисунок 3.6 – Структура мікросервісу ProductAPI для описа продуктів

Структура проєкту ProductAPI забезпечує модульність, гнучкість та зручність у розробці, підтримці та масштабуванні застосунку, дозволяючи ефективно працювати з даними продуктів та інтегрувати їх з іншими компонентами системи.

Представлена структура проєкту ShoppingCartAPI, яка аналогічна до структури ProductAPI і включає дві основні частини: BusinessLogic та DataAccess, що забезпечують управління кошиками покупок у системі (рис. 3.7). Частина BusinessLogic містить сервіси, які реалізують бізнес-логіку для управління кошиками, обробляючи відповідні запити та забезпечуючи основні функції сервісу, а також розширення для бізнес-логіки, що додають додаткову функціональність і допомагають організувати код більш ефективно.

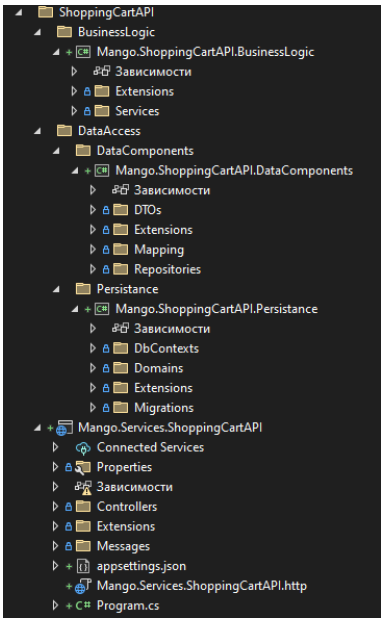

Рисунок 3.7 – Структура мікросервису ShoppingCartAPI для описа корзини

Частина DataAccess включає DataComponents, до яких належать розширення для доступу до даних, що полегшують роботу з базою даних і забезпечують інтеграцію з іншими компонентами, ініціалізатори для налаштування початкових даних у базі даних, що допомагають підготувати середовище для роботи з даними, та проміжне ПЗ для обробки HTTP-запитів, що забезпечує обробку запитів і відповіді на них, покращуючи взаємодію між клієнтом і сервером.

Секція Persistence включає константи для роботи з базою даних, що визначають налаштування і параметри, необхідні для взаємодії з даними, контексти бази даних, які використовуються для доступу до даних за допомогою EF Core, домени або сутності бізнес-логіки, що представляють

структуру даних кошиків, об’єкти передачі даних, які використовуються для обміну даними між різними частинами системи, розширення для роботи з базою даних, що додають додаткові функції для ефективної взаємодії з даними, міграції для бази даних, що забезпечують керування змінами у схемі бази даних, та репозиторії для доступу до даних, що забезпечують абстракцію над операціями з базою даних, спрощуючи управління даними.

Ця структура проєкту ShoppingCartAPI забезпечує модульність, гнучкість та зручність у розробці, підтримці та масштабуванні застосунку, дозволяючи ефективно працювати з даними кошиків покупок та інтегрувати їх з іншими компонентами системи.

Модуль Sharable містить спільні компоненти, що використовуються різними сервісами (рис. 3.8). Цей модуль включає такі компоненти, як Domains, які представляють спільні доменні моделі, DTO для передачі даних між різними частинами системи, та Mapping для конфігурації мапінгу об’єктів.

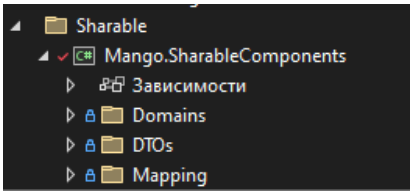

Рисунок 3.8 – Вміст директорії Sharable

Причини вибору такої структури базуються на кількох ключових аспектах. По-перше, модульність дозволяє розробляти, тестувати та розгортати сервіси незалежно один від одного, що підвищує гнучкість і масштабованість системи. Чіткість забезпечується тим, що кожен сервіс відповідає за певну функціональність, роблячи код більш зрозумілим та легким у підтримці. Розширюваність досягається за рахунок можливості легко додавати нові сервіси або модулі без суттєвих змін у існуючому коді. Повторне використання спільних компонентів у модулі Sharable дозволяє уникати дублювання коду та спрощує його підтримку.

Ця структура забезпечує високу якість розробки програмного забезпечення, зручність у підтримці та можливість масштабування системи відповідно до зростаючих потреб бізнесу. Проєкт «MangoRestaurant» побудований за принципами мікросервісної архітектури, яка передбачає розбиття великого монолітного застосунку на невеликі, незалежні сервіси, кожен з яких виконує окрему бізнес-функцію. Кожен мікросервіс відповідає за конкретний аспект системи, наприклад, управління користувачами, продуктами, кошиком покупок тощо.

Модульність, яка реалізована через мікросервісну архітектуру, має ряд переваг. По-перше, легкість у підтримці забезпечується модульною структурою, яка дозволяє розділяти систему на окремі частини, кожна з яких може бути розроблена та підтримувана незалежно. Це зменшує складність підтримки системи в цілому. Масштабованість досягається тим, що кожен мікросервіс можна масштабувати незалежно від інших, залежно від навантаження. Це дозволяє ефективно використовувати ресурси та забезпечувати високу продуктивність системи. Незалежність розробки дозволяє різним командам працювати над різними мікросервісами одночасно, не заважаючи одна одній, що прискорює процес розробки та впровадження нових функціональностей. Гнучкість у виборі технологій забезпечується тим, що різні мікросервіси можуть бути написані на різних мовах програмування та використовувати різні технології, що дозволяє вибирати найкращі інструменти для кожного конкретного завдання.

Покращена ізоляція збоїв забезпечує, що збої в одному мікросервісі не впливають на роботу інших, підвищуючи загальну надійність та стійкість системи до збоїв. Спрощене тестування досягається тим, що мікросервіси можна тестувати окремо, що спрощує процес виявлення та виправлення помилок, підвищуючи якість програмного забезпечення та зменшуючи кількість дефектів у продуктивному середовищі. Прозорість та спрощене оновлення забезпечуються тим, що оновлення мікросервісів можна виконувати незалежно, що дозволяє вводити нові функції та виправляти

помилки без зупинки всієї системи, забезпечуючи безперервність бізнес-процесів та мінімізуючи час простою.

Мікросервісна архітектура у поєднанні з модульністю забезпечує високу гнучкість, масштабованість та надійність системи. Поточна проєктна структура дозволяє ефективно розподіляти навантаження, швидко адаптуватися до змін та забезпечувати високу якість обслуговування користувачів. Такий підхід дозволяє розробляти сучасні та продуктивні застосунки, що відповідають вимогам сучасного бізнесу.

Директорія BusinessLogic у кожному мікросервісі відповідає за бізнес-логіку застосунку, реалізуючи основні правила та операції, пов’язані з бізнес-процесами. Вона містить залежності, необхідні для роботи бізнес-логіки, такі як сервіси, репозиторії та інші компоненти, методи-розширення та інші утиліти, що полегшують роботу з бізнес-логікою, а також сервіси, що реалізують основні бізнес-процеси та правила.

Директорія DataAccess відповідає за доступ до даних та включає два основні підрозділи: DataComponents та Persistence. DataComponents містить компоненти, що полегшують доступ до даних та взаємодію з ними, такі як методи-розширення для маніпуляції об’єктами або колекціями даних, класи репозиторіїв для абстрагування доступу до бази даних, ініціалізатори для налаштування початкових даних та проміжне програмне забезпечення для обробки HTTP-запитів. Persistence включає константи для роботи з базою даних, класи контекстів бази даних для роботи з базою даних за допомогою Entity Framework, доменні моделі, що представляють основні сутності бізнес-логіки, об’єкти передачі даних для передачі даних між різними шарами застосунку або між мікросервісами, методи-розширення для роботи з доменними моделями або контекстами бази даних, класи міграцій для оновлення структури бази даних та конфігурації моделей для Entity Framework.

Проєкт «MangoRestaurant» використовує комбінацію MVC та N-Tier архітектур. У папці Frontend («Mango.Web») реалізовано архітектуру MVC, що

дозволяє ефективно організувати роботу вебзастосунку, зокрема через моделі, представлення (Razor Views), контролери та моделі представлення. Кожен мікросервіс у проєкті реалізує N-Tier архітектуру, включаючи бізнес-логіку, доступ до даних та доменні компоненти.

CouponAPI реалізує N-Tier архітектуру з окремими шарами бізнес-логіки, доступу до даних та доменних компонентів. ProductAPI має аналогічну структуру з N-Tier архітектурою, що забезпечує чітке розділення відповідальності між бізнес-логікою, доступом до даних та доменними компонентами. ShoppingCartAPI також використовує N-Tier архітектуру для організації коду, що полегшує підтримку та масштабування.

Проєкт «MangoRestaurant» використовує комбінацію MVC та N-Tier архітектур, що забезпечує високу модульність, масштабованість та легкість у підтримці. MVC архітектура у фронтенді дозволяє ефективно організувати взаємодію з користувачами, тоді як N-Tier архітектура у мікросервісах забезпечує чітке розділення бізнес-логіки, доступу до даних та інших компонентів, що робить систему більш гнучкою та легкою у розвитку.

3.3 Використовування принципів програмування

При розробці проєкту «MangoRestaurant» особлива увага приділялася дотриманню принципів Clean Code, SOLID, DRY та KISS. Ці принципи забезпечують високу якість коду, його зрозумілість, легкість у підтримці та розширюваність.

Принципи Clean Code (Чистий код) – це набір практик, які допомагають писати зрозумілий, читабельний та підтримуваний код. Основні принципи Clean Code включають такі аспекти: зрозумілість, де код повинен бути зрозумілим і читабельним, з самодокументуючими іменами змінних, методів та класів, що відображають їх призначення; простота, де код повинен бути максимально простим і не містити зайвої складності, уникаючи складних

структур і методів; однорідність, де використовуються однакові шаблони і підходи у всьому проєкті, що зменшує кількість помилок та спрощує розуміння коду; модульність, де код розділяється на незалежні модулі, які можна легко тестувати, змінювати та повторно використовувати.

Принципи SOLID, які є набором принципів об’єктно-орієнтованого програмування, сприяють створенню гнучкого, зрозумілого та легко підтримуваного коду. Вони включають принцип єдиної відповідальності (SRP), де кожен клас повинен мати лише одну відповідальність; принцип відкритості/закритості (OCP), де програмні сутності повинні бути відкритими для розширення, але закритими для змін; принцип підстановки Ліскова (LSP), де об’єкти повинні бути взаємозамінними з їх підтипами без порушення роботи програми; принцип розділення інтерфейсів (ISP), де інтерфейси повинні бути специфічними до клієнта, а клієнти не повинні залежати від інтерфейсів, які вони не використовують; і принцип інверсії залежностей (DIP), де вищі модулі не повинні залежати від нижчих модулів, а обидва повинні залежати від абстракцій.

Принцип DRY (Don’t Repeat Yourself) забороняє дублювання коду, вимагаючи, щоб кожна частина коду була представлена лише один раз у системі. Принцип KISS (Keep It Simple, Stupid), наголошує на необхідності тримати код простим та зрозумілим, де складні рішення слід використовувати лише тоді, коли це абсолютно необхідно.

При розробці проєкту «MangoRestaurant» були використані наступні принципи: Clean Code, де імена змінних, методів та класів чітко відображають їх призначення, кожен клас і метод мають одну відповідальність, а код розділений на модулі, що полегшує його тестування та підтримку; SOLID, де кожен мікросервіс має чітко визначені відповідальності, абстракції використовуються для зменшення залежностей між компонентами, і використовуються інтерфейси для поділу функціональності та зменшення залежностей; DRY, де спільні сервіси та утиліти використовуються для уникнення дублювання коду, а загальні функціональності винесені в окремі

класи та методи; KISS, де весь код був написаний так просто, наскільки це можливо, а складні рішення використовуються лише тоді, коли це абсолютно необхідно.

Використання принципів Clean Code, SOLID, DRY та KISS у проєкті «MangoRestaurant» дозволило створити чистий, зрозумілий та легко підтримуваний код. Ці принципи допомогли забезпечити високу якість програмного забезпечення, спростити його розробку та забезпечити можливість легкого розширення та масштабування системи.

3.4 Асистент на основі штучного інтелекту

В даному проєкті була створена інтеграція чата з GPT як особистого асистента на сайті. Це дозволяє користувачам отримувати допомогу та відповіді на їхні запитання в режимі реального часу, не покидаючи застосунок.

Інтерфейс чата на сайті включає вікно чата, де користувачі можуть вводити свої запитання та спілкуватися з асистентом, який буде намагатися надати відповіді на основі натренованих моделей GPT. Асистент може допомогти в різних сферах, від відповідей на загальні запитання до надання конкретних порад або інформації про продукти та послуги.

Через аналіз запитань користувачів та їхніх відповідей асистент може покращувати свої навички та надавати більш точні та корисні відповіді з часом. Чат з асистентом забезпечує можливість отримати відповіді в реальному часі, що дозволяє користувачам отримувати необхідну допомогу без затримок.

Ця інтеграція чата з GPT додає значну цінність до застосунку, забезпечуючи користувачам зручний інструмент для отримання допомоги та відповідей на їхні запитання.

3.5 Методи обчислення оцінки розташування

У даному проєкті всі ціни замовлень записуються у CSV файл (рис. Б.1), який потім обробляється додатковим десктопним застосунком на Python з використанням бібліотеки Tkinter (рис. Б.2). Метою цього застосунку є обчислення оцінки розташування для визначення середнього місячного доходу ресторану якнайточніше. Програма обробляє дуже великі файли, що містять більше 120000 даних, і робить це дуже швидко завдяки ефективним алгоритмам і оптимізаціям. Залежно від відсотка викидів у даних, які вводяться користувачем, застосовуються різні методи обчислення оцінки розташування.

$$\frac{1}{n}\sum_{i=1}^{n} x_i \,. \tag{3.1}$$

Середнє арифметичне використовується, коли відсоток викидів становить 0%. Це стандартний метод для обчислення середнього значення набору даних, де всі значення додаються, а потім діляться на кількість значень. Проте середнє арифметичне не завжди є найкращим способом розрахунку середнього, оскільки воно чутливе до викидів. Наприклад, якщо ми маємо набір даних з доходами ресторану, де більшість місячних доходів складають близько 5000 грн, але є одне значення 50000 грн, середнє арифметичне буде значно завищене, що не відображає реальну ситуацію.

Усічене середнє арифметичне застосовується, коли відсоток викидів становить 10%. Усічене середнє виключає певний відсоток найменших і найбільших значень перед обчисленням середнього, що допомагає зменшити вплив викидів. Усічене середнє краще використовувати, коли є підозра, що в даних можуть бути незначні викиди, які не сильно впливають на загальну тенденцію.

$$\frac{1}{n-2k}\sum_{i=k+1}^{n-k} x_i. \tag{3.2}$$

Зважене середнє арифметичне використовується, коли відсоток викидів становить 20%, і якщо є другий показник для ваги кожного значення. Зважене середнє враховує різні ваги для кожного. Зважена медіана застосовується, коли відсоток викидів становить 30%, і якщо є другий показник для ваги кожного значення.

$$\frac{\sum_1^n w_i x_i}{\sum_{i=1}^n w_i}. \tag{3.3}$$

Зважена медіана враховує ваги і знаходить середнє значення таким чином, щоб сума ваг зліва і справа від медіани була приблизно рівною.

$$\sum_{i=1}^{n} w_i \geq \sum_{i=1}^{n} w_i. \tag{3.4}$$

Медіана використовується, коли відсоток викидів більше 30%. Медіана – це середнє значення упорядкованого набору даних, що не піддається впливу викидів. Медіана є кращим варіантом, коли дані містять значну кількість викидів, оскільки вона менш чутлива до екстремальних значень.

$$\begin{cases} x_{\left(\frac{n+1}{2}\right),} \\ \frac{x_{\left(\frac{n}{2}\right)}+x_{\left(\frac{n}{2}+1\right)}}{2} \end{cases}. \tag{3.5}$$

Викиди – це значення, які значно відрізняються від інших значень у наборі даних. Вони можуть суттєво впливати на результати аналізу, тому важливо їх враховувати.

Межквартильний розмах (IQR) є мірою варіації даних і використовується для визначення границь викидів. IQR розраховується як

різниця між третьою (Q3) та першою (Q1) квартилями. Використовуючи IQR, можна визначити границі викидів.

$$IQR = Q_3 - Q_1, \tag{3.6}$$

$$L = Q_1 - 1.5 \times IQR, \tag{3.7}$$

$$U = Q_3 + 1.5 \times IQR, \tag{3.8}$$

$$Q_1 = \frac{n+1}{4}, \tag{3.9}$$

$$Q_3 = \frac{3(n+1)}{4}, \tag{3.10}$$

Цей десктопний застосунок на Python використовує відповідний метод обчислення оцінки розташування на основі відсотка викидів у даних, які вводяться користувачем, забезпечуючи точне визначення середнього місячного доходу ресторану. Усі відсотки є відносними і залежать від аналізу, тому користувач вводить їх самостійно. Програма здатна ефективно обробляти великі файли, що містять понад 120000 даних, забезпечуючи швидке та надійне обчислення завдяки оптимізованим алгоритмам і використанню сучасних бібліотек для обробки даних. Результат обробки CSV файлів показано у Додатку Б.

3.6 Підготовка даних

Підготовка даних є критичним етапом у розробці програмного забезпечення, особливо коли мова йде про мікросервісні архітектури та застосунки, що працюють з великим обсягом даних. У проєкті «MangoRestaurant» підготовка даних включає кілька ключових аспектів:

налаштування баз даних, ініціалізація початкових даних, міграції баз даних та обробка даних для тестування.

Для кожного мікросервісу в проєкті було створено окрему базу даних, яка відповідає за зберігання даних, специфічних для цього мікросервісу. Використання окремих баз даних для кожного мікросервісу підвищує автономність сервісів та зменшує ризик виникнення конфліктів даних.

Ініціалізація бази даних – це процес завантаження початкових даних, необхідних для роботи застосунку. Це можуть бути довідники, налаштування, дані для тестування тощо. У проєкті «MangoRestaurant» для цього використовується спеціальний клас Initializer, який відповідає за завантаження початкових даних у базу даних.

Міграції баз даних використовуються для керування змінами структури бази даних. Це дозволяє розробникам легко оновлювати структуру бази даних у відповідності до змін у моделях даних. «MangoRestaurant» використовує міграції Entity Framework Core для автоматичного створення та оновлення таблиць у базі даних.

Для забезпечення якості програмного забезпечення необхідно мати дані для тестування.

Підготовка даних є важливою частиною процесу розробки проєкту. Вона включає налаштування баз даних, ініціалізацію початкових даних, міграції баз даних та створення даних для тестування. Це забезпечує коректне функціонування застосунку, його стабільність та можливість проведення якісного тестування. Завдяки ретельній підготовці даних, проєкт ефективно працює та задовольняє потреби користувачів.

3.7 Аналіз результатів

У проєкті «MangoRestaurant» процес аналізу включає оцінку досягнутих цілей, перевірку відповідності вимогам, тестування продуктивності та

надійності, а також виявлення потенційних областей для поліпшення.

Перше, що потрібно проаналізувати, це чи були досягнуті цілі проєкту, поставлені на початку розробки.

Створення модульного мікросервісного застосунку включало реалізацію мікросервісної архітектури з використанням декількох незалежних сервісів, таких як ProductAPI, CouponAPI, ShoppingCartAPI, кожен з яких відповідає за окрему функціональність. Принципи Clean Code, SOLID, DRY та KISS були впроваджені на всіх етапах розробки для забезпечення зрозумілості, підтримуваності та розширюваності коду. Інтеграція з хмарними сервісами, такими як Azure для зберігання зображень та інших ресурсів, забезпечила масштабованість і надійність системи. Забезпечення високої продуктивності та надійності здійснювалося через тестування та моніторинг застосунку, що дозволило виявити та виправити потенційні проблеми, забезпечуючи стабільну роботу системи. Демонстрація застосунку наведена у Додатку В.

Наступним кроком було перевірити, чи відповідає розроблене рішення вимогам, визначеним на початку проєкту, які включають як функціональні, так і нефункціональні аспекти. Функціональні вимоги включають можливість управління продуктами (створення, редагування, видалення), управління купонами та їх застосування до замовлень, а також управління кошиком покупок (додавання, видалення товарів, оформлення замовлень). Нефункціональні вимоги охоплюють продуктивність, де система повинна швидко обробляти запити користувачів, надійність, що передбачає стабільну роботу застосунку без збоїв, і безпеку, яка забезпечує безпечне зберігання та передачу даних.

Тестування продуктивності включало оцінку швидкості обробки запитів, часу відгуку системи та здатності обробляти великий обсяг одночасних запитів. Наприклад, час відгуку для запитів до ProductAPI становив середній час 100 мс, для CouponAPI — 120 мс, а для ShoppingCartAPI — 110 мс. Тестування надійності включало перевірку стабільності системи під навантаженням та її стійкості до збоїв, де система витримала навантаження у

1000 одночасних користувачів без збоїв, що було перевірено завдяки тестовому застосунку для симуляції навантаження. Жоден з мікросервісів не зазнав збоїв під час тривалого тестування (8 годин).

Після аналізу досягнень та тестування важливо було виявити області, де система може бути покращена. Це можуть бути як функціональні аспекти, так і нефункціональні. Оптимізація продуктивності може включати зменшення часу відгуку для певних запитів за рахунок оптимізації запитів до бази даних або покращення алгоритмів обробки даних, а також впровадження кешування для зменшення навантаження на базу даних. Покращення користувацького досвіду може включати розширення функціональності інтерфейсу користувача для зручності використання, оптимізацію навігації та швидкості завантаження сторінок. Забезпечення безпеки може передбачати впровадження додаткових заходів безпеки, таких як двофакторна автентифікація, а також регулярне оновлення бібліотек та компонентів для захисту від нових загроз. Масштабованість може бути забезпечена підготовкою до можливого зростання кількості користувачів шляхом впровадження горизонтального масштабування.

Аналіз результатів проєкту «MangoRestaurant» показав, що основні цілі були досягнуті, а система відповідає встановленим вимогам. Тестування продуктивності та надійності підтвердило здатність системи ефективно обробляти запити користувачів. Однак, були виявлені деякі області для поліпшення, що забезпечить ще більшу ефективність та зручність використання застосунку. Подальші зусилля будуть спрямовані на оптимізацію продуктивності, покращення користувацького досвіду, підвищення безпеки та забезпечення масштабованості системи. Результат роботи застосунку при різній кількості користувачів наведено у Додатку А.

ВИСНОВКИ

У рамках кваліфікаційної роботи був розроблений і реалізований метод створення модульного мікросервісного застосунку для ресторану «MangoRestaurant». Використання мікросервісної архітектури дозволило розробити кілька незалежних сервісів, таких як ProductAPI, CouponAPI, ShoppingCartAPI, Identity, кожен з яких відповідає за окрему функціональність, забезпечуючи гнучкість та масштабованість системи.

Принципи Clean Code, SOLID, DRY та KISS були застосовані на всіх етапах розробки, що сприяло створенню зрозумілого, підтримуваного та розширюваного коду. Інтеграція з хмарними сервісами, такими як Azure для зберігання зображень та інших ресурсів, забезпечила надійність та можливість легкого масштабування системи.

Був розроблений додатковий десктопний застосунок на Python з використанням бібліотеки Tkinter для обробки даних про ціни замовлень, записаних у CSV файл. Цей застосунок обчислює оцінку розташування для визначення середнього місячного доходу ресторану, використовуючи різні методи залежно від відсотка викидів, які вводяться користувачем. Програма здатна ефективно обробляти великі файли, що містять понад 120000 даних, забезпечуючи точне і надійне обчислення завдяки оптимізованим алгоритмам.

Результати проєкту підтвердили, що розроблена система відповідає встановленим вимогам. Вона забезпечує можливість управління продуктами, купонами та кошиком покупок, а також відповідає критеріям продуктивності, надійності та безпеки. Виявлені області для поліпшення дозволяють оптимізувати продуктивність, покращити користувацький досвід, підвищити безпеку та забезпечити подальшу масштабованість системи.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Torrese, S. (2021). *Business model innovation in the food delivery industry: The increasing roles of delivery models and sustainability* (Doctoral dissertation, Politecnico di Torino).

2. Dedhia, P., Shanmuganathan, S., & Livis, W. (2021). Business Plan and Analysis for uplifting Small Business (SB).

3. Tan, T. F., & Netessine, S. (2020). At your service on the table: Impact of tabletop technology on restaurant performance. *Management Science*, *66*(10), 4496-4515.

4. Alt, R. (2021). Digital transformation in the restaurant industry: Current developments and implications. *Journal of smart tourism*, *1*(1), 69-74.

5. Dedhia, P., Shanmuganathan, S., & Livis, W. (2021). Business Plan and Analysis for uplifting Small Business (SB).

6. Chen, P. H., Gao, S., Wang, Y. J., Xu, A. D., Li, Y. S., & Wang, D. (2012). Classifying ischemic stroke, from TOAST to CISS. *CNS neuroscience & therapeutics*, *18*(6), 452-456.

7. Pagaldiviti, S. R., & Roy, B. K. (2023). The future of restaurants: How technology is changing the way we dine out. In *Impactful Technologies Transforming the Food Industry* (pp. 63-74). IGI Global.

8. Sundqvist, I., & Zurawska, K. J. (2023). Low-impact user interfaces: How can we create a climate-smart approach toward front-end mobile application design?: Evaluation and proposed improvement of existing guidelines–Focused on Online Food Delivery sector."Microservices in Action" - Morgan Bruce, Paulo A. Pereira

9. Sungboonlue, P., Thanakaew, S., Rangseepanya, K., Tangpatong, T., & Siriborvornratanakul, T. (2022). A study of redesigning food delivery application in Thailand. *Telkomnika (Telecommunication Computing Electronics and Control)*, *20*(5), 1073-1082.

10. Lo, C. Y., Lin, C. T., & Tsai, C. L. (2011). Mobile restaurant information

system integrating reservation navigating and parking management. *International Journal of Engineering and Technology*, *3*(2), 173-181.

11. Du, Z., Fan, Z. P., & Chen, Z. (2023). Implications of on-time delivery service with compensation for an online food delivery platform and a restaurant. *International Journal of Production Economics*, *262*, 108896.

12. Velepucha, V., & Flores, P. (2023). A survey on microservices architecture: Principles, patterns and migration challenges. *IEEE Access*.

13. Fritzsch, J., Bogner, J., Wagner, S., & Zimmermann, A. (2019, September). Microservices migration in industry: intentions, strategies, and challenges. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)* (pp. 481-490). IEEE.

14. Xu, M., Song, C., Ilager, S., Gill, S. S., Zhao, J., Ye, K., & Xu, C. (2022). Coscal: Multifaceted scaling of microservices with reinforcement learning. *IEEE Transactions on Network and Service Management*, *19*(4), 3995-4009.

15. Milić, M., & Makajić-Nikolić, D. (2022). Development of a quality-based model for software architecture optimization: a case study of monolith and microservice architectures. *Symmetry*, *14*(9), 1824.

16. Pacheco, V. F. (2018). *Microservice Patterns and Best Practices: Explore patterns like CQRS and event sourcing to create scalable, maintainable, and testable microservices*. Packt Publishing Ltd.

17. Oreshchuk, H. (2023). Methodology and communication patterns of microservices in cloud systems.

18. Cabane, H., & Farias, K. (2024). On the impact of event-driven architecture on performance: An exploratory study. *Future Generation Computer Systems*, *153*, 52-69.

19. Milton, T. (2024). Artificial Intelligence Transforming Hotel Gastronomy: An In-depth Review of AI-driven Innovations in Menu Design, Food Preparation, and Customer Interaction, with a Focus on Sustainability and Future Trends in the Hospitality Industry. *International Journal for Multidimensional Research Perspectives*, *2*(3), 47-61.

20. Zhu, Y. (2023). The Role of Touch, Touchscreens, and Haptic Technology in Interactive Marketing: Evolution from Physical Touch to Digital Touch. In *The Palgrave Handbook of Interactive Marketing* (pp. 867-891). Cham: Springer International Publishing.

21. Arunachalam, D., & Kumar, N. (2018). Benefit-based consumer segmentation and performance evaluation of clustering approaches: An evidence of data-driven decision-making. *Expert Systems with Applications*, *111*, 11-34.

22. Hollebeek, L. D. (2019). Developing business customer engagement through social media engagement-platforms: An integrative SD logic/RBV-informed model. *Industrial Marketing Management*, *81*, 89-98.

23. Yang, N., Li, F., Liu, Y., Dai, T., Wang, Q., Zhang, J., ... & Yu, B. (2022). Environmental and economic life-cycle assessments of household food waste management systems: a comparative review of methodology and research progress. *Sustainability*, *14*(13), 7533.

24. Vu, O. T. K., Alonso, A. D., Martens, W., Ha, L. D. T., Tran, T. D., & Nguyen, T. T. (2022). Hospitality and tourism development through coffee shop experiences in a leading coffee-producing nation. *International Journal of Hospitality Management*, *106*, 103300.

25. Bu, L., Chen, C. H., Ng, K. K., Zheng, P., Dong, G., & Liu, H. (2021). A user-centric design approach for smart product-service systems using virtual reality: A case study. *Journal of Cleaner Production*, *280*, 124413.

26. Milton, T. (2024). Artificial Intelligence Transforming Hotel Gastronomy: An In-depth Review of AI-driven Innovations in Menu Design, Food Preparation, and Customer Interaction, with a Focus on Sustainability and Future Trends in the Hospitality Industry. *International Journal for Multidimensional Research Perspectives*, *2*(3), 47-61.

27. Milton, T. (2024). Artificial Intelligence Transforming Hotel Gastronomy: An In-depth Review of AI-driven Innovations in Menu Design, Food Preparation, and Customer Interaction, with a Focus on Sustainability and Future Trends in the Hospitality Industry. *International Journal for Multidimensional Research*

Perspectives, *2*(3), 47-61.

28. Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., & Safina, L. (2017). Microservices: yesterday, today, and tomorrow. *Present and ulterior software engineering*, 195-216.

29. Salah, T., Zemerly, M. J., Yeun, C. Y., Al-Qutayri, M., & Al-Hammadi, Y. (2016, December). The evolution of distributed systems towards microservices architecture. In *2016 11th International Conference for Internet Technology and Secured Transactions (ICITST)* (pp. 318-325). IEEE.

30. Bucchiarone, A., Dragoni, N., Dustdar, S., Lago, P., Mazzara, M., Rivera, V., & Sadovykh, A. (2020). Microservices. *Science and Engineering. Springer*.

31. Bakshi, K. (2017, March). Microservices-based software architecture and approaches. In *2017 IEEE aerospace conference* (pp. 1-8). IEEE.

32. Hasselbring, W., & Steinacker, G. (2017, April). Microservice architectures for scalability, agility and reliability in e-commerce. In *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)* (pp. 243-246). IEEE.

33. Hoffman, K. (2017). *Building Microservices with ASP. NET Core: Develop, Test, and Deploy Cross-platform Services in the Cloud*. " O'Reilly Media, Inc.".

34. Weil, A. (2018). *Learn Microservices-ASP. NET Core and Docker*. Lulu. com.

35. Hoffman, K. (2017). *Building Microservices with ASP. NET Core: Develop, Test, and Deploy Cross-platform Services in the Cloud*. " O'Reilly Media, Inc.".

36. Kumar, S. S., & Shastry, P. M. (2017, August). Database-per-service for e-learning system with micro-service architecture. In *2017 International Conference On Smart Technologies For Smart Nation (SmartTechCon)* (pp. 705-708). IEEE.

37. Viennot, N., Lécuyer, M., Bell, J., Geambasu, R., & Nieh, J. (2015, April). Synapse: a microservices architecture for heterogeneous-database web applications. In *Proceedings of the tenth european conference on computer systems* (pp. 1-16).

38. Mikkelsen, A., Grønli, T. M., Tamburri, D. A., & Kazman, R. (2020, January). Architectural Principles for Autonomous Microservices. In *HICSS* (pp. 1-10).

39. Zimmermann, O., Lübke, D., Zdun, U., Pautasso, C., & Stocker, M. (2020, July). Interface responsibility patterns: processing resources and operation responsibilities. In *Proceedings of the European Conference on Pattern Languages of Programs 2020* (pp. 1-24).

40. Yellavula, N. (2020). *Hands-On RESTful Web Services with Go: Develop Elegant RESTful APIs with Golang for Microservices and the Cloud*. Packt Publishing Ltd.

41. Biehl, M. (2016). RESTful API Design: Best Practices in API Design with REST.

42. Lähtevänoja, V. (2021). *Communication Methods and Protocols Between Microservices on a Public Cloud Platform* (Master's thesis).

43. Batista, C., Morais, F., Cavalcante, E., Batista, T., Proença, B., & Rodrigues Cavalcante, W. B. (2024). Managing asynchronous workloads in a multi-tenant microservice enterprise environment. *Software: Practice and Experience*, *54*(2), 334-359.

44. Yellavula, N. (2017). *Building RESTful Web services with Go: Learn how to build powerful RESTful APIs with Golang that scale gracefully*. Packt Publishing Ltd.

45. Kotstein, S., & Bogner, J. (2021). Which restful api design rules are important and how do they improve software quality? a delphi study with industry experts. In *Service-Oriented Computing: 15th Symposium and Summer School, SummerSOC 2021, Virtual Event, September 13–17, 2021, Proceedings 15* (pp. 154-173). Springer International Publishing.