Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп’ютерної інженерії та управління
(повна назва)
Кафедра Комп’ютерних інтелектуальних технологій та систем
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)
Мікросервісна архітектура для ІТ-сервісу обслуговування замовлень
(тема)

Виконав:
здобувач 2 року навчання,
групи КІТм-23-1
Куренков Б.М.
(прізвище, ініціали)

Спеціальність 123 Комп’ютерна інженерія
(код і повна назва спеціальності)
Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)
Освітня програма Комп’ютерні інтелектуальні технології
(повна назва освітньої програми)

Керівник Сердюк Н.М.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Руденко О.Г.
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет Комп’ютерної інженерії та управління

Кафедра Комп’ютерних інтелектуальних технологій та систем

Рівень вищої освіти другий (магістерський)

Спеціальність 123 Комп’ютерна інженерія
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Комп’ютерні інтелектуальні технології
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____
(підпис)

«____»_____ 20 ___ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Куренкову Богдану Михайловичу
(прізвище, ім’я, по батькові)

1. Тема роботи Мікросервісна архітектура для IT-сервісу обслуговування замовлень

затверджена наказом університету від 28 жовтня 2024 р. № 1156Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 21 січня 2025 р.

3. Вихідні дані до роботи схема організаційної структури сервісу, діаграма IDEF0 процесу «Ведення обліку доставки», діаграма IDEF0 декомпозиції процесу «Ведення обліку доставки», логічна ERD сервісу, фізична ERD сервісу

4. Перелік питань, що потрібно опрацювати в роботі аналіз предметної області та постановка задачі, аналіз існуючих архітектур, опис елементів існуючих систем забезпечення, розробка елементів систем забезпечення

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) 20 слайдів презентаційного матеріалу

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

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Аналіз предметної области та постановка задачі	28.10.2024-10.11.2024	Виконано
2	Аналіз існуючих архітектур	10.11.2024-30.11.2024	Виконано
3	Опис елементів існуючих систем забезпечення	30.11.2024-10.12.2024	Виконано
4	Розробка елементів систем забезпечення	10.12.2024-30.12.2024	Виконано
5	Оформлення пояснювальної записки	30.12.2024-10.01.2025	Виконано
6	Розробка презентації	10.01.2025-20.01.2025	Виконано
7	Попередній захист	21.01.2025	Виконано
8	Захист кваліфікаційної роботи в екзаменаційній комісії	24.01.2025	Виконано

Дата видачі завдання 28 жовтня 2024 р.

Здобувач _____
(підпис)

Керівник роботи _____ доцентка Сердюк Н.М.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи: 97 с., 8 табл., 39 рис., 1 дод., 14 джерел.

АЛГОРИТМІЧНЕ ЗАБЕЗПЕЧЕННЯ, ДИЗАЙНОВЕ ЗАБЕЗПЕЧЕННЯ, ІНФОРМАЦІЙНА СИСТЕМА, МАСШТАБОВАНІСТЬ, МІКРОСЕРВІСНА АРХІТЕКТУРА, МОДЕЛЬ С4, МОНОЛІТНА АРХІТЕКТУРА, ОБСЛУГОВУВАННЯ ЗАМОВЛЕНЬ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, СЕРВІСНО-ОРІЄНТОВАНА АРХІТЕКТУРА, СМАРТ-ПРИСТРОЇ.

Об’єкт дослідження кваліфікаційної роботи – ІТ-сервіс обслуговування замовлень.

Метою цієї кваліфікаційної роботи є розробка моделі мікросервісної архітектури інформаційної системи та елементів забезпечення задля підвищення масштабованости ІТ-сервісу обслуговування замовлень.

Для проведення дослідження використовувалися такі методи: системний підхід, методи структурного аналізу, моделювання бази даних, методологія функціонального та структурного аналізу (IDEF0), моделювання за моделлю С4.

Актуальність дослідження зумовлена зростаючою потребою у впровадженні адаптивних і надійних систем для обслуговування замовлень, які сприяють підвищенню ефективности бізнесу та задовольняють вимоги сучасних користувачів.

ABSTRACT

Explanatory note to the qualification work: 97 pages, 8 tables, 39 pictures, 1 appendice, 14 references.

ALGORITHMIC SUPPORT, C4 MODEL, DESIGN SUPPORT, INFORMATION SYSTEM, MICROSERVICE ARCHITECTURE, MONOLITHIC ARCHITECTURE, ORDER MANAGEMENT, SCALABILITY, SERVICE-ORIENTED ARCHITECTURE, SMART DEVICES, SOFTWARE.

The object of research of the qualification work is the IT service of order service.

The purpose of this qualification work is to develop a model of the microservice architecture of the information system and support elements to increase the scalability of the IT order service.

The following methods were used to conduct the research: systematic approach, methods of structural analysis, database modeling, functional and structural analysis methodology (IDEF0), modeling according to the C4 model.

The relevance of the study is driven by the growing need to implement adaptive and reliable order processing systems that improve business efficiency and meet the requirements of modern users.

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп’ютерної інженерії та управління
(повна назва)
Кафедра Комп’ютерних інтелектуальних технологій та систем
(повна назва)

АНОТАЦІЯ
Кваліфікаційної роботи

рівень вищої освіти другий (магістерський)
Мікросервісна архітектура для ІТ-сервісу обслуговування замовлень
(тема)

Виконав:
здобувач 2 року навчання,
групи КІТм-23-1
Куренков Б.М.
(прізвище, ініціали)

Спеціальність 123 Комп’ютерна інженерія
(код і повна назва спеціальності)
Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)
Освітня програма Комп’ютерні інтелектуальні технології
(повна назва освітньої програми)

Керівник Сердюк Н.М.
(посада, прізвище, ініціали)

2024 р.

АНОТАЦІЯ

Куренков Б. М. Мікросервісна архітектура для ІТ-сервісу обслуговування замовлень. – Магістерська кваліфікаційна робота.

У магістерській кваліфікаційній роботі вирішено актуальну проблему зростаючої потреби у впровадженні адаптивних і надійних систем для обслуговування замовлень, які сприяють підвищенню ефективности бізнесу та задовольняють вимоги сучасних користувачів.

Метою кваліфікаційної роботи є розробка моделі мікросервісної архітектури інформаційної системи та елементів забезпечення задля підвищення масштабованости ІТ-сервісу обслуговування замовлень.

Об’єктом дослідження кваліфікаційної роботи є сфера доставки.

Предметом дослідження кваліфікаційної роботи є модель мікросервісної архітектури інформаційної системи та елементи забезпечення.

Інформаційні технології залишаються ключовим чинником розвитку сучасного суспільства, відкриваючи нові горизонти в автоматизації бізнес-процесів. Це особливо актуально в умовах стрімкого розвитку електронної комерції та глобалізації ринку, коли оптимізація процесів обслуговування замовлень стає одним із головних завдань. Сучасні клієнти очікують від бізнесу не лише оперативности виконання замовлень, але й високої якости сервісу, адаптивности до їхніх потреб та гнучкости у взаємодії.

У відповідь на ці виклики компанії мають забезпечувати масштабованість своїх інформаційних систем, здатних швидко адаптуватися до змін ринку. Традиційна монолітна архітектура, яка свого часу була широко використовуваною, нині демонструє низку обмежень. Вона ускладнює підтримку системи, сповільнює впровадження нових функцій і не забезпечує необхідної гнучкости у масштабуванні.

Перехід до мікросервісної архітектури є ефективним рішенням цієї проблеми. Цей підхід базується на розділенні функціональних компонентів системи на незалежні сервіси, кожен з яких може розроблятися, тестуватися та масштабуватися автономно.

Під час аналізу ринку було визначено, що значна частина потенційних клієнтів надає перевагу використанню різних смарт-пристроїв для доступу до послуг. Користувачі все частіше очікують зручного і швидкого доступу до сервісів незалежно від того, яким пристроєм вони користуються в даний момент. Таким чином, наявність підтримки смарт-пристроїв стала ключовим фактором у збільшенні клієнтської бази та утриманні існуючих клієнтів.

З метою збільшення охоплення авдиторії та підвищення конкурентоспроможности на ринку, компанія прийняла стратегічне рішення забезпечити підтримку своїх сервісів на різноманітних смарт-пристроях. Це включає не лише смартфони та годинники, але й телевізори з доступом до інтернету та будь-які інші смарт-пристрої. Реалізація стратегії спрямована на забезпечення безперебійного доступу до послуг для максимально широкого кола користувачів, що повинно сприяти збільшенню кількости клієнтів та їхньої лояльности.

У процесі інтеграції нових смарт-пристроїв у сервіси компанія зіткнулася з низкою критичних проблем, що виникли через обмеження існуючої монолітної архітектури. Однією з головних складностей була інтеграція нових інтерфейсів та API. Зокрема, кожна нова платформа вимагала суттєвих змін у коді. Наприклад, додавання підтримки нового інтерфейсу для смарт-годинників потребувало глибокої інтеграції з багатьма іншими частинами системи. Це призводило до значних затримок у розробці, а також до високих витрат на тестування і верифікацію функціоналу. Крім того, розробка нових API для різних платформ потребувала великих зусиль від розробників, ускладнюючи впровадження нових функцій і сповільнюючи час їхнього виведення на ринок.

Монолітна архітектура також виявилася негнучкою в масштабуванні. Це створювало серйозні проблеми під час масштабування окремих компонентів для рі-

зних пристроїв. Наприклад, якщо один із компонентів системи потребував додаткових ресурсів для обробки запитів від годинників, масштабувати лише цей компонент було неможливо без впливу на інші частини системи. Така неефективність у використанні ресурсів збільшувала витрати на інфраструктуру та знижувала продуктивність системи загалом.

Окрім того, система виявилася вразливою до помилок. Непередбачена помилка в одному з компонентів могла спричинити збої у всій системі. Наприклад, збій у модулі обробки запитів від годинників негативно впливав на роботу інших модулів, погіршуючи користувацький досвід. Відсутність ізоляції між компонентами лише підсилювала ризик масштабних збоїв і ускладнювала процес налагодження та усунення помилок.

Існуюча монолітна архітектура значно ускладнювала інтеграцію нових функцій, уповільнювала розробку, обмежувала можливости масштабування і робила систему вразливою до збоїв. Це підкреслювало необхідність переходу на більш гнучку та стійку до помилок архітектуру, здатну відповідати сучасним вимогам масштабованости та швидкости впровадження інновацій.

З огляду на ці проблеми, очевидно, що поточні технологічні рішення не здатні забезпечити необхідну масштабованість для підтримки смарт-пристроїв. Необхідно знайти новий архітектурний підхід, який дозволив би подолати існуючі виклики та забезпечив би ефективну роботу системи на всіх платформах.

Для вирішення даної проблеми необхідно вирішити низку завдань, які забезпечать подолання обмежень монолітної архітектури та дозволять створити гнучку, масштабовану й ефективну інформаційну систему.

Для цього необхідно проаналізувати предметну область, розглянути існуючі архітектурні підходи, обґрунтувати вибір мікросервісної архітектури як оптимальної для підвищення масштабованости системи та розробити модель мікросервісної архітектури інформаційної системи, елементи алгоритмічної та дизайнової систем забезпечення, а також рекомендації щодо елементів програмної системи забезпечення.

У кваліфікаційній роботі було розроблено модель мікросервісної архітектури інформаційної системи та елементи забезпечення задля підвищення масштабованости IT-сервісу обслуговування замовлень.

На початковому етапі було виконано аналіз предметної області, що дозволило визначити основні виклики, зокрема низьку гнучкість монолітної архітектури, труднощі масштабування та інтеграції нових функцій, що ускладнювали реалізацію підтримки різноманітних платформ, включно зі смарт-пристроями. Ці проблеми стали основою для формулювання задачі роботи, яка полягала в аналізі предметної області, розгляді існуючих архітектурних підходів, обґрунтовуванні вибіру мікросервісної архітектури як оптимальної для підвищення масштабованости системи та розробці моделі мікросервісної архітектури інформаційної системи, елементів алгоритмічної та дизайнової систем забезпечення, а також рекомендацій щодо елементів програмної системи забезпечення.

В межах поглибленого аналізу існуючих архітектур інформаційних систем було досліджено монолітну, сервісно-орієнтовану та мікросервісну архітектури. Здійснено порівняння за ключовими параметрами, такими як: гнучкість, масштабованість, продуктивність і складність впровадження. Мікросервісна архітектура була визнана найефективнішим вибором завдяки її децентралізованости, незалежности компонентів та адаптивности до змін.

Детально описано елементи існуючих систем забезпечення об’єкта дослідження, а саме: схема організаційної структури, IDEF0 та ERD діаграми. Це дало змогу структурувати модель даних та підготувати основу для переходу до мікросервісної архітектури.

Завдяки описаним елементам існуючих систем забезпечення були розроблені модель мікросервісної архітектури інформаційної системи, елементи алгоритмічної та дизайнової систем забезпечення, що включали: діаграму рівня Container за моделлю C4, опис компонентів системи, її взаємодій, характеристик, інтегрованих компонентів, елементів безпеки, моніторину і керування доступом, оптимізації роботи системи, документації, схеми алгоритмів роботи вебсайту, дизайн вебсайту,

а також надані рекомендації щодо елементів програмної системи забезпечення та щодо захисту інформації системи.

Таким чином, результати роботи демонструють ефективність переходу на мікросервісну архітектуру для вирішення проблем масштабованости інформаційної системи. Розроблене рішення дозволило не лише забезпечити необхідну масштабованість системи, а й швидко адаптувати її до змін ринку та вимог користувачів. Ці результати можуть бути застосовані в інших проєктах, спрямованих на розробку сучасних інформаційних систем для обслуговування замовлень та інших бізнес-процесів.

РЕКОМЕНДАЦІЇ, МОЖЛИВОСТІ, ЗАСТОСУВАННЯ, ОДЕРЖАННЯ, РЕЗУЛЬТАТИ, АЛГОРИТМІЧНЕ ЗАБЕЗПЕЧЕННЯ, ДИЗАЙНОВЕ ЗАБЕЗПЕЧЕННЯ, ІНФОРМАЦІЙНА СИСТЕМА, МАСШТАБОВАНІСТЬ, МІКРОСЕРВІСНА АРХІТЕКТУРА, МОДЕЛЬ C4, МОНОЛІТНА АРХІТЕКТУРА, ОБСЛУГОВУВАННЯ ЗАМОВЛЕНЬ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, СЕРВІСНО-ОРІЄНТОВАНА АРХІТЕКТУРА, СМАРТ-ПРИСТРОЇ

Публікації здобувача за темою роботи:

1. Куренков Б. М. The main advantages of microservice architecture. Здобутки та досягнення прикладних та фундаментальних наук XXI століття: матеріали міжнар. наук. конф., м. Черкаси, 8 груд. 2023 р. Вінниця, 2023. С. 249–250.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API (Application Programming Interface) – інтерфейс прикладного програмування, що забезпечує взаємодію комп’ютерних програм між собою

C4 Model – модель візуалізації архітектури програмного забезпечення на чотирьох рівнях: контекст, контейнер, компонент, код

ERD (Entity-Relationship Diagram) – діаграма зв’язків сутностей для моделювання структури бази даних

IDEF0 (Integration DEFinition for Function Modeling) – методологія функціонального моделювання для опису бізнес-процесів

Docker – платформа для контейнеризації програм, що дозволяє ізолювати застосунки

Kubernetes – система оркестрації контейнерів, яка автоматизує розгортання, масштабування та управління контейнеризованими застосунками

HTTP/REST – протоколи взаємодії для обміну даними між клієнтом і сервером

Load Balancer – балансувальник навантаження для розподілу запитів між серверами

MS SQL Server – система управління реляційними базами даних від Microsoft

.NET Core – платформа з відкритим кодом для розробки застосунків

Vue.js – JavaScript-фреймворк для створення користувацьких інтерфейсів

JSON (JavaScript Object Notation) – формат обміну даними, що використовує структури ключ-значення

SMTP (Simple Mail Transfer Protocol) – протокол для передавання електронної пошти між серверами

Twilio – хмарна платформа для обміну повідомленнями, дзвінків та інших комунікаційних сервісів

FCM (Firebase Cloud Messaging) – сервіс від Google для надсилання push-повідомлень на мобільні та вебзастосунки

RabbitMQ – брокер повідомлень для обміну даними між сервісами в асинхронному режимі

Redis – система управління базами даних типу «ключ-значення», що використовується для кешування та обміну повідомленнями

Swagger – інструмент для документування та тестування API

OAuth 2.0 – протокол автентифікації та авторизації для забезпечення доступу до ресурсів API

JWT (JSON Web Token) – стандарт для обміну даними у вигляді JSON-об’єктів з цифровим підписом

Nginx – вебсервер, що також може використовуватися як балансувальник навантаження та зворотний проксі

Prometheus – система моніторину та оповіщення для контейнеризованих застосунків

Grafana – інструмент для моніторину та візуалізації метрик системи

Graphviz – інструмент для візуалізації графів і діаграм програмних рішень

ВСТУП

Інформаційні технології залишаються ключовим чинником розвитку сучасного суспільства, відкриваючи нові горизонти в автоматизації бізнес-процесів. Це особливо актуально в умовах стрімкого розвитку електронної комерції та глобалізації ринку, коли оптимізація процесів обслуговування замовлень стає одним із головних завдань. Сучасні клієнти очікують від бізнесу не лише оперативности виконання замовлень, але й високої якости сервісу, адаптивности до їхніх потреб та гнучкости у взаємодії.

У відповідь на ці виклики компанії мають забезпечувати масштабованість своїх інформаційних систем, здатних швидко адаптуватися до змін ринку. Традиційна монолітна архітектура, яка свого часу була широко використовуваною, нині демонструє низку обмежень. Вона ускладнює підтримку системи, сповільнює впровадження нових функцій і не забезпечує необхідної гнучкости у масштабуванні.

Перехід до мікросервісної архітектури є ефективним рішенням цієї проблеми. Цей підхід базується на розділенні функціональних компонентів системи на незалежні сервіси, кожен з яких може розроблятися, тестуватися та масштабуватися автономно.

Метою цієї кваліфікаційної роботи є розробка моделі мікросервісної архітектури інформаційної системи та елементи забезпечення задля підвищення масштабованости ІТ-сервісу обслуговування замовлень.

Актуальність дослідження зумовлена зростаючою потребою у впровадженні адаптивних і надійних систем для обслуговування замовлень, які сприяють підвищенню ефективности бізнесу та задовольняють вимоги сучасних користувачів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТИ ТА ПОСТАНОВКА ЗАДАЧІ

У цьому розділі проведено аналіз предметної області, визначена проблема, котру треба вирішити та виконана постановка задачі.

1.1 Аналіз предметної області

Під час аналізу ринку було визначено, що значна частина потенційних клієнтів надає перевагу використанню різних смарт-пристроїв для доступу до послуг. Користувачі все частіше очікують зручного і швидкого доступу до сервісів незалежно від того, яким пристроєм вони користуються в даний момент. Таким чином, наявність підтримки смарт-пристроїв стала ключовим фактором у збільшенні клієнтської бази та утриманні існуючих клієнтів.

З метою збільшення охоплення авдиторії та підвищення конкурентоспроможности на ринку, компанія прийняла стратегічне рішення забезпечити підтримку своїх сервісів на різноманітних смарт-пристроях. Це включає не лише смартфони та годинники, але й телевізори з доступом до інтернету та будь-які інші смарт-пристрої. Реалізація стратегії спрямована на забезпечення безперебійного доступу до послуг для максимально широкого кола користувачів, що повинно сприяти збільшенню кількости клієнтів та їхньої лояльности.

1.2 Визначення проблеми

У процесі інтеграції нових смарт-пристроїв у сервіси компанія зіткнулася з низкою критичних проблем, що виникли через обмеження існуючої монолітної архітектури [1]. Однією з головних складностей була інтеграція нових інтерфейсів та API [2]. Зокрема, кожна нова платформа вимагала суттєвих змін у коді. Напри-

клад, додавання підтримки нового інтерфейсу для смарт-годинників потребувало глибокої інтеграції з багатьма іншими частинами системи. Це призводило до значних затримок у розробці, а також до високих витрат на тестування і верифікацію функціоналу. Крім того, розробка нових API для різних платформ потребувала великих зусиль від розробників, ускладнюючи впровадження нових функцій і сповільнюючи час їхнього виведення на ринок.

Ще однією серйозною проблемою став повільний цикл розробки та тестування. Монолітна архітектура вимагала ретельного тестування всієї системи навіть у випадку внесення незначних змін у будь-який модуль. Наприклад, зміни для підтримки нових методів оплати чи інтеграції з новими пристроями потребували перевірки на всіх рівнях системи, що значно уповільнювало процес розробки. До того ж висока ймовірність помилок під час інтеграції нових функцій змушувала виділяти додаткові ресурси на тестування і налагодження, що лише збільшувало витрати та затримки.

Монолітна архітектура також виявилася негнучкою в масштабуванні. Це створювало серйозні проблеми під час масштабування окремих компонентів для різних пристроїв. Наприклад, якщо один із компонентів системи потребував додаткових ресурсів для обробки запитів від годинників, масштабувати лише цей компонент було неможливо без впливу на інші частини системи. Така неефективність у використанні ресурсів збільшувала витрати на інфраструктуру та знижувала продуктивність системи загалом.

Окрім того, система виявилася вразливою до помилок. Непередбачена помилка в одному з компонентів могла спричинити збої у всій системі. Наприклад, збій у модулі обробки запитів від годинників негативно впливав на роботу інших модулів, погіршуючи користувацький досвід. Відсутність ізоляції між компонентами лише підсилювала ризик масштабних збоїв і ускладнювала процес налагодження та усунення помилок.

Існуюча монолітна архітектура значно ускладнювала інтеграцію нових функцій, уповільнювала розробку, обмежувала можливости масштабування і робила систему вразливою до збоїв. Це підкреслювало необхідність переходу на більш гнучку та стійку до помилок архітектуру, здатну відповідати сучасним вимогам масштабованости та швидкости впровадження інновацій.

З огляду на ці проблеми, очевидно, що поточні технологічні рішення не здатні забезпечити необхідну масштабованість для підтримки смарт-пристроїв. Необхідно знайти новий архітектурний підхід, який дозволив би подолати існуючі виклики та забезпечив би ефективну роботу системи на всіх платформах.

1.3 Постановка задачі

Для вирішення даної проблеми необхідно вирішити низку завдань, які забезпечать подолання обмежень монолітної архітектури та дозволять створити гнучку, масштабовану й ефективну інформаційну систему.

Метою цієї кваліфікаційної роботи є розробка моделі мікросервісної архітектури інформаційної системи та елементів забезпечення задля підвищення масштабованости ІТ-сервісу обслуговування замовлень [3].

Для цього необхідно проаналізувати предметну область, розглянути існуючі архітектурні підходи, обґрунтувати вибір мікросервісної архітектури як оптимальної для підвищення масштабованости системи та розробити модель мікросервісної архітектури інформаційної системи, елементи алгоритмічної та дизайнової систем забезпечення, а також рекомендації щодо елементів програмної системи забезпечення.

2 АНАЛІЗ ІСНУЮЧИХ АРХІТЕКТУР

У цьому розділі детально розглянуто основні архітектурні підходи, які можуть бути використані для вирішення визначеної проблеми, проаналізовані переваги та недоліки кожного з них, також обґрунтований вибір мікросервісної архітектури як оптимальної для вирішення проблеми масштабованости.

2.1 Види архітектур

Серед існуючих архітектурних підходів варто звернути увагу на такі як: монолітна архітектура, сервісно-орієнтована архітектура та мікросервісна архітектура [4]. Кожна з цих архітектур має свої особливости та застосування, що визначаються конкретними потребами та вимогами проєкту. Монолітна архітектура є традиційним підходом, який часто використовується в невеликих проєктах або на початкових етапах розробки. Сервісно-орієнтована архітектура, у свою чергу, забезпечує можливість розділення функціональности на окремі сервіси, що покращує гнучкість та масштабованість системи. Мікросервісна архітектура, яка набула значної популярности в останні роки, дозволяє створювати високоефективні, масштабовані та гнучкі системи, що складаються з незалежних мікросервісів.

На рисунку 2.1 зображена структура монолітної, сервісно-орієнтованої та мікросервісної архітектур.

Рисунок 2.1 – Структура монолітної, сервісно-орієнтованої та мікросервісної архітектур

На рисунку зображено схематичне представлення трьох різних архітектур програмного забезпечення: монолітної, сервісно-орієнтованої та мікросервісної. Кожна з цих архітектур має свої особливости, що показані на схемі.

Монолітна архітектура характеризується тим, що інтерфейс користувача взаємодіє безпосередньо з бізнес-логікою. Усі функції програми інтегровані разом у єдиному застосунку, що забезпечує цілісність бізнес-логіки. Для доступу до сховища даних використовується інтерфейс даних, причому всі дані зберігаються в одному спільному сховищі.

Сервісно-орієнтована архітектура організована таким чином, що інтерфейс користувача взаємодіє з різними сервісами через сервісно-бас, який виступає посередником для обміну повідомленнями між сервісами. Кожен сервіс є автономним компонентом, який виконує окрему функцію. Дані в такій архітектурі можуть зберігатися як в одному спільному, так і в кількох сховищах, до яких сервіси мають доступ.

Мікросервісна архітектура побудована на взаємодії інтерфейсу користувача з незалежними мікросервісами, кожен з яких відповідає за окрему частину бізнес-логіки. Кожен мікросервіс може мати власний інтерфейс даних та окреме сховище, що забезпечує автономність. Завдяки цьому мікросервіси можна розробляти, розгортати та масштабувати незалежно один від одного.

Розглянемо кожну з архітектур детальніше.

2.2 Монолітна архітектура

Монолітна архітектура – це традиційний підхід до розробки програмного забезпечення, де всі компоненти системи об’єднані в один єдиний модуль. У такій архітектурі всі функціональні частини, такі як інтерфейс користувача, бізнес-логіка та доступ до даних, інтегровані в єдиний застосунок.

2.2.1 Загальний опис монолітної архітектури

Монолітна архітектура передбачає створення одного великого застосунку, що містить усі функціональні компоненти системи. Всі модулі, включаючи інтерфейс користувача, бізнес-логіку та доступ до даних, об’єднані в одну кодову базу. Коли користувач запускає застосунок, все його програмне забезпечення завантажується водночас. Всі модулі та компоненти доступні з моменту запуску. Всі компоненти застосунку взаємодіють безпосередньо один з одним через внутрішні методи викликів. Наприклад, якщо користувач виконує дію на інтерфейсі користувача, ця дія напряму викликає методи бізнес-логіки, які потим звертаються до бази даних для отримання або збереження даних. Кожен запит користувача обробляється через один вхідний пункт застосунку. Всі бізнес-правила, перевірки та доступ до даних відбуваються всередині цього моноліту. Монолітний застосунок розгортається як

єдиний блок. Це означає, що навіть якщо зміни внесені лише в одну невелику частину застосунку, весь застосунок необхідно перекомпілювати та розгортати знову.

2.2.2 Приклади систем із монолітною архітектурою

Монолітну архітектуру доцільно використовувати в кількох типах предметних областей та застосунків. Наприклад, системи малого бізнесу, зокрема для невеликих компаній з обмеженими ресурсами, часто будуються за монолітною архітектурою. Невеликий інтернет-магазин може мати єдиний застосунок, що охоплює управління товарами, обробку замовлень та взаємодію з клієнтами. Також монолітні архітектури популярні серед стартових проєктів, оскільки їх простота та швидкість розробки дозволяють швидко вивести продукт на ринок та перевірити його життєздатність. Окрім цього, монолітна архітектура добре підходить для систем з простим функціоналом, які не потребують частих оновлень або масштабування. Наприклад, внутрішні системи управління для невеликих організацій можуть ефективно функціонувати в межах монолітного підходу, забезпечуючи облік працівників чи відстеження робочих процесів.

2.2.3 Приклади реалізації монолітної архітектури

Нижче наведено декілька прикладів систем, реалізованих за допомогою монолітної архітектури. Ці приклади демонструють різноманітні типи проєктив, які використовують монолітну архітектуру для досягнення своїх цілей. Спільним для всіх цих прикладів є те, що вони використовують монолітну архітектуру для забезпечення простоти розробки, підтримки та розгортання. Усі компоненти систем інтегровані в єдиний кодовий базис, що дозволяє швидко вносити зміни та впроваджувати новий функціонал.

2.2.3.1 Сервіс замовлення мила

Вебсторінка миловарні «Натхнення» є типовим прикладом застосування монолітної архітектури. У цьому застосунку інтерфейс користувача взаємодіє з бізнес-логікою, яка включає всі функції застосунку, об’єднані в одному кодовому базисі. Користувач може переглядати асортимент продукції, читати опис товарів та відгуки, додавати мило до кошика та оформлювати замовлення. Кожен з цих компонентів – від управління кошиком до обробки замовлень та взаємодії з базою даних – є частиною єдиної системи. Це означає, що зміни або оновлення, внесені до будь-якої частини застосунку, впливають на весь застосунок, що може ускладнювати підтримку та масштабування, але забезпечує швидкість розробки та простоту інтеграції всіх функцій.

На рисунку 2.2 зображено вебсторінку миловарні «Натхнення» реалізовану за допомогою монолітної архітектури.

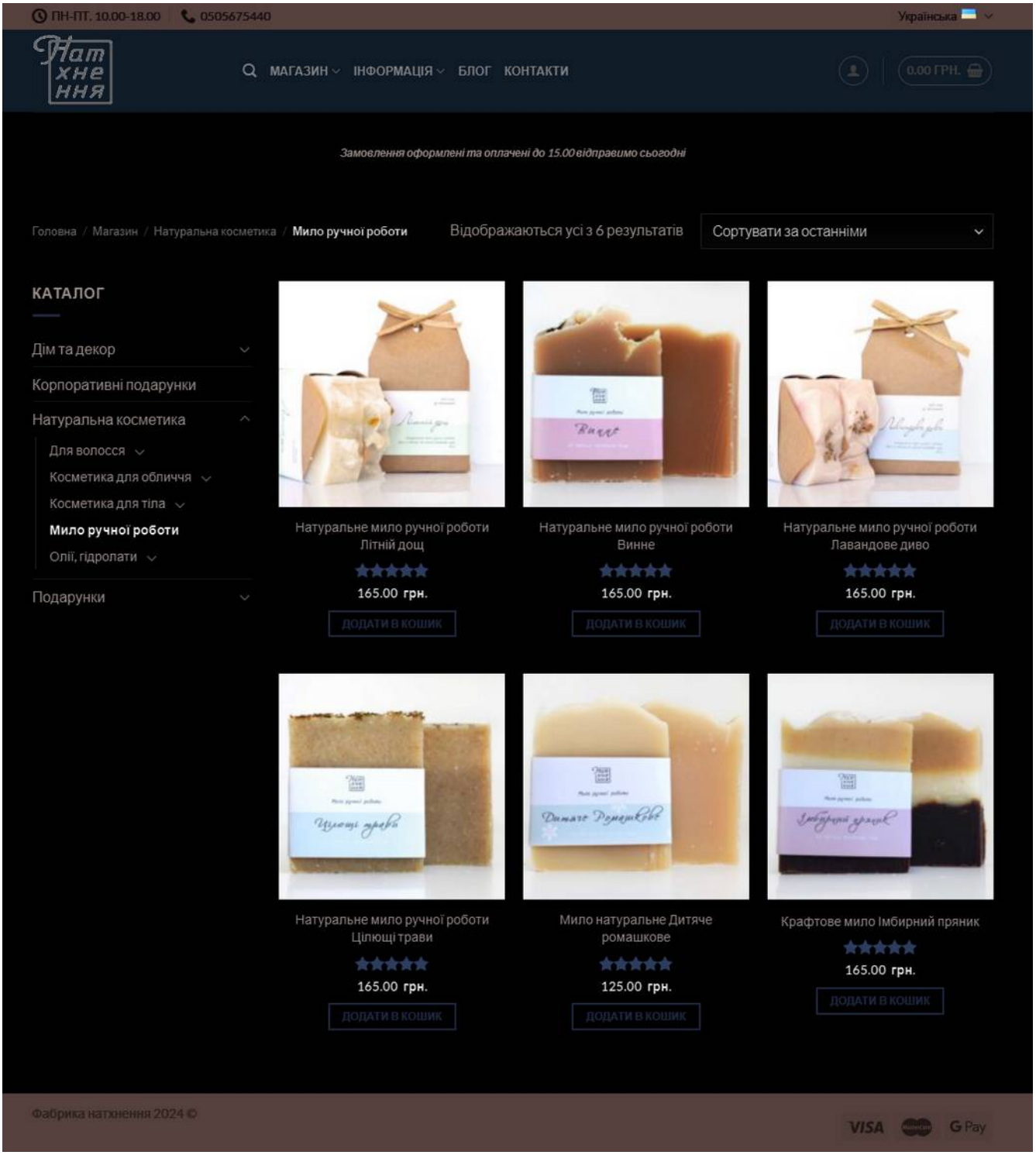

Рисунок 2.2 – Вебсторінка миловарні «Натхнення»

2.2.3.2 Сервіс стартапу

Стартап Advin, що спеціалізується на контролі наявности товарів на полицях, також використовує монолітну архітектуру. Інтерфейс користувача, управління даними та бізнес-логіка, такі як алгоритми відстеження та аналізу наявности товарів, функціонують як єдиний кодовий базис.

На рисунку 2.3 зображено вебсторінку стартапу Advin реалізовану за допомогою монолітної архітектури.

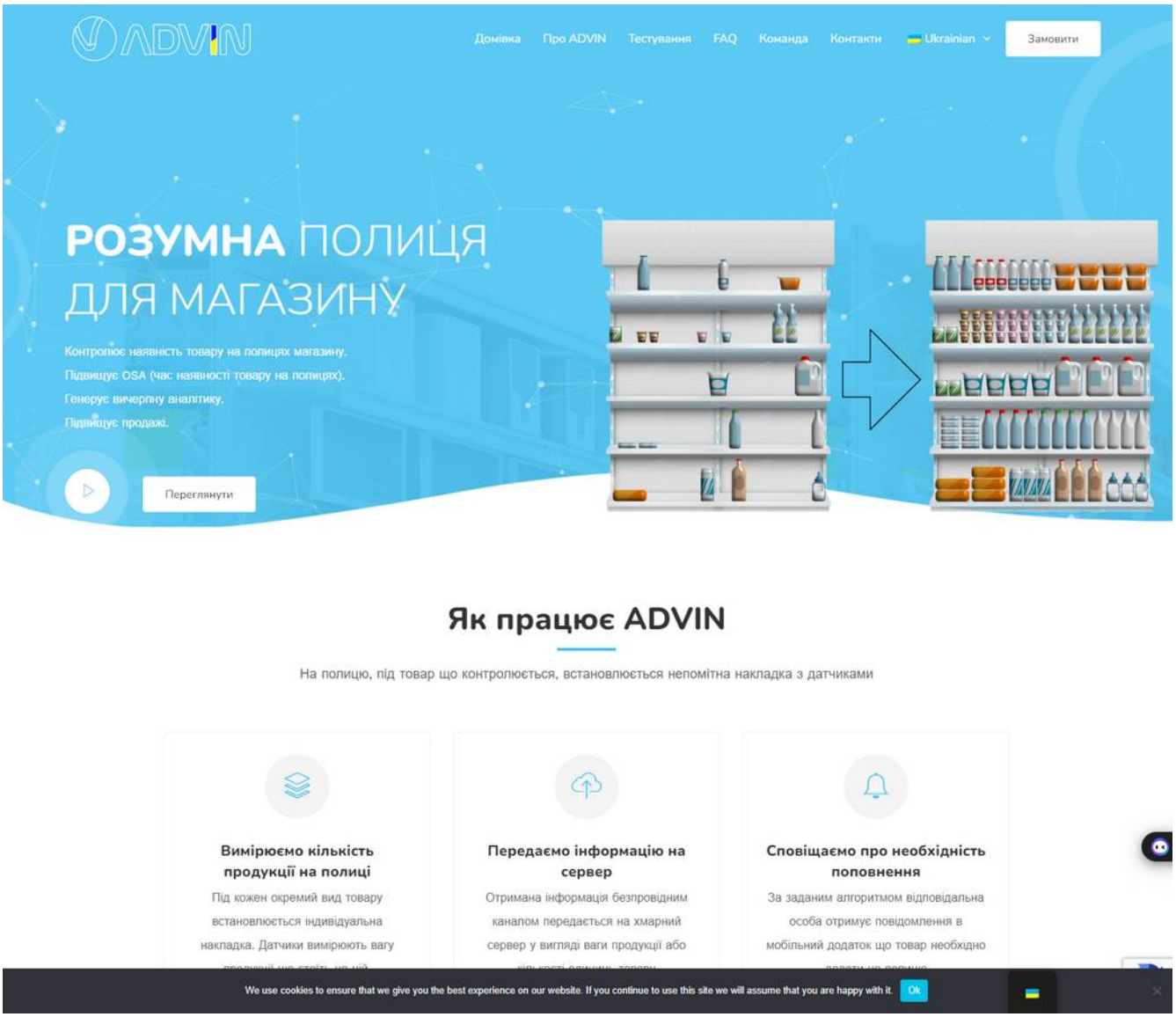

Рисунок 2.3 – Вебсторінка стартапу Advin

2.2.3.3 Сервіс з завантаження відейо з YouTube

Сервіс uniDownloader реалізований у за допомогою монолітної архітектури. Користувачі можуть ввести URL відейо, вибрати формат і натиснути кнопку завантаження. Весь цей процес є частиною єдиного застосунку.

На рисунку 2.4 зображено вебсторінку сервісу uniDownloader реалізовану за допомогою монолітної архітектури.

Рисунок 2.4 – Вебсторінка сервісу uniDownloader

2.2.4 Переваги та недоліки монолітної архітектури

Монолітна архітектура має низку переваг, серед яких виділяється простота розробки. На початкових етапах створення такі системи легше розробляти та розгортати. Цілісність коду забезпечується тим, що всі компоненти зберігаються в одному кодовому базисі, що спрощує управління залежностями. Розгортання також є відносно простим процесом, оскільки весь код зібраний в одному місці, що дозволяє швидко налаштовувати нові середовища та розгортати оновлення. Ще однією перевагою є єдине управління версіями: всі компоненти застосунку мають одну версію, що забезпечує сумісність між частинами системи та спрощує процес управління версіями.

Однак монолітна архітектура має й недоліки. Внесення змін або додавання нового функціоналу може ускладнюватися, оскільки потрібно змінювати весь застосунок, що значно ускладнює розробку та тестування. Масштабування також є проблемою, адже для масштабування доводиться розгортати весь застосунок, навіть якщо зміни потрібні лише в одному його компоненті. Крім того, монолітні застосунки можуть бути нестабільними, оскільки помилка в одному модулі може призвести до збою всієї системи. Процес розгортання таких систем може займати значний час через інтегрованість усіх компонентів в єдину структуру, особливо коли необхідно розгортати всю систему для внесення змін у її невелику частину.

2.3 Сервісно-орієнтована архітектура

Сервісно-орієнтована архітектура – це підхід до розробки програмного забезпечення, де функціональні можливости системи розбиваються на окремі сервіси, які взаємодіють через визначені інтерфейси. Кожен сервіс є самостийною одиницею з чітко визначеною функціональністю.

2.3.1 Загальний опис сервісно-орієнтованої архітектури

Сервісно-орієнтована архітектура будується на принципі розділення функціональних компонентів на незалежні сервіси, які можуть взаємодіяти один з одним через стандартизовані інтерфейси, такі як вебсервіси. Система розбивається на окремі сервіси, кожен з яких виконує конкретну функцію. Наприклад, може бути окремий сервіс для обробки платежів, управління користувачами або обробки замовлень. Сервіси взаємодіють один з одним через стандартизовані інтерфейси, такі як SOAP. Це дозволяє сервісам бути незалежними від технології, на яких вони реалізовані. Коли користувач надсилає запит, фронтенд застосунку звертається до відповідного сервісу через його інтерфейс. Наприклад, запит на отримання інформації про продукт може викликати сервіс управління продуктами. Більш складні бізнес-процеси можуть бути реалізовані через оркестрацію (централізоване управління процесами) або хореографію (децентралізоване управління, де сервіси взаємодіють напряму). Кожен сервіс розгортається окремо. Це дозволяє розгортати оновлення для окремих сервісів без необхідности перекомпілювати та розгортати всю систему.

2.3.2 Приклади систем із сервісно-орієнтованою архітектурою

Сервісно-орієнтовану архітектуру доцільно використовувати в різних предметних областях та застосунках. Наприклад, у фінансових системах, таких як банківські системи, обробка платежів, управління активами та інші фінансові послуги, цей підхід дозволяє інтегрувати різні сервіси, включаючи обробку транзакцій, управління портфелем та звітність. У телекомунікаційних системах сервісно-орієнтована архітектура сприяє ефективному управлінню абонентами, обробці даних трафіку та підтримці послуг. Також урядові інформаційні системи широко використовують цей підхід для об’єднання різних державних служб. На-

приклад, архітектура може забезпечувати інтеграцію між податковою службою, службою соціального забезпечення та реєстраційними службами, покращуючи ефективність і взаємодію між ними.

2.3.3 Приклади реалізації сервісно-орієнтованої архітектури

Нижче наведено декілька прикладів систем, реалізованих за допомогою сервісно-орієнтованої архітектури. Ці приклади демонструють різноманітні типи проєктів, які використовують сервісно-орієнтовану архітектуру для досягнення своїх цілей. Спільним для всіх цих прикладів є те, що вони використовують сервісно-орієнтовану архітектуру.

2.3.3.1 Сервіс банку

Система банку monobank розроблена з використанням сервісно-орієнтованої архітектури. У цьому застосунку користувач взаємодіє з елементами, котрі реалізовані як окремі сервіси, такі як сервіс авторизації, сервіс транзакцій, або сервіс валютних операцій.

На рисунку 2.5 зображено вебсторінку банку monobank реалізовану за допомогою сервісно-орієнтованої архітектури.

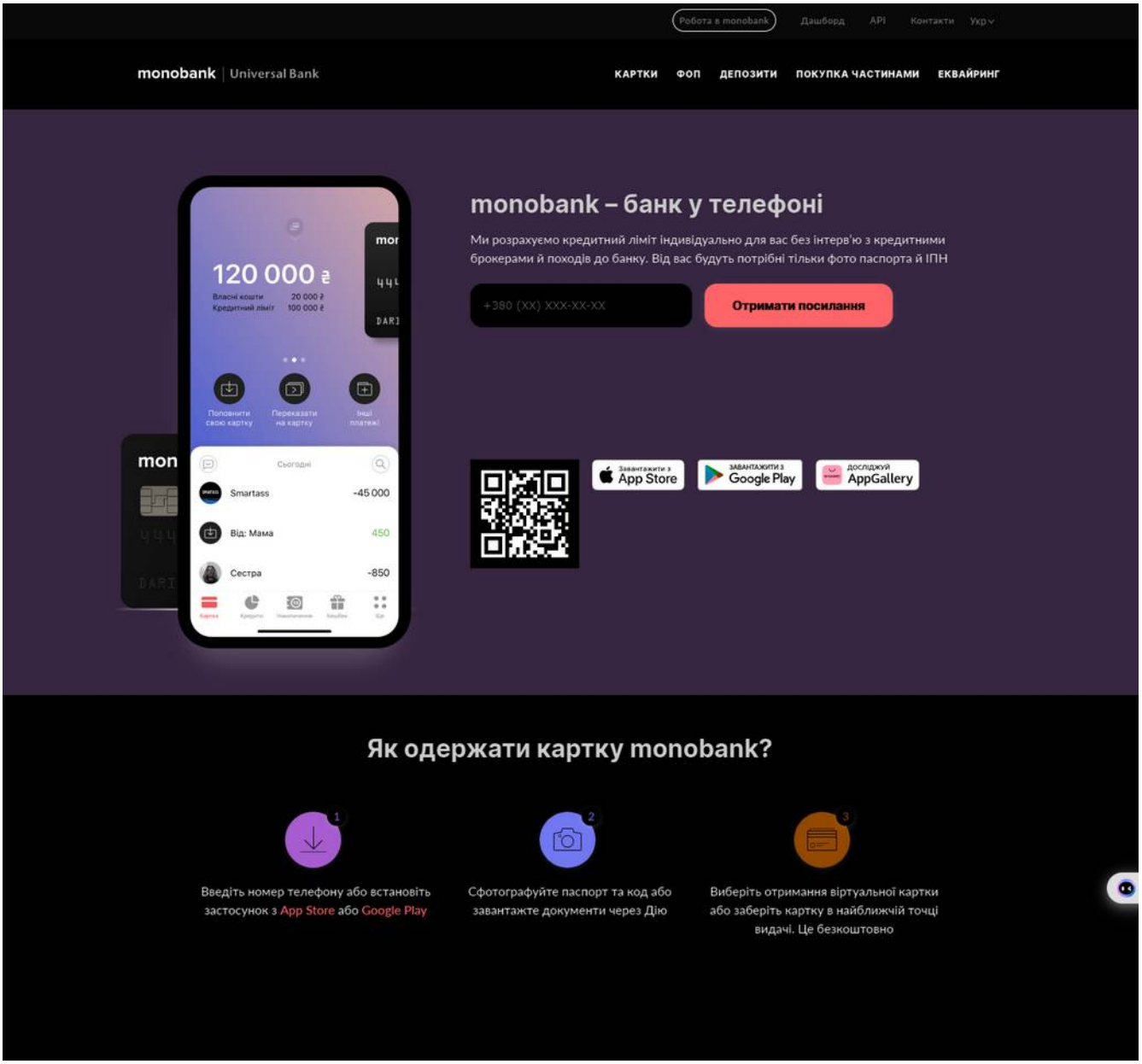

Рисунок 2.5 – Вебсторінка банку monobank

2.3.3.2 Сервіс мобільного оператору

Система мобільного оператору Vodafone надає послуги зв’язку у стандартах GSM (EDGE), UMTS (HSPA+) та LTE. Система розроблена як набір окремих сервісів.

На рисунку 2.6 зображено вебсторінку мобільного оператору Vodafone реалізовану за сервісно-орієнтованої архітектури.

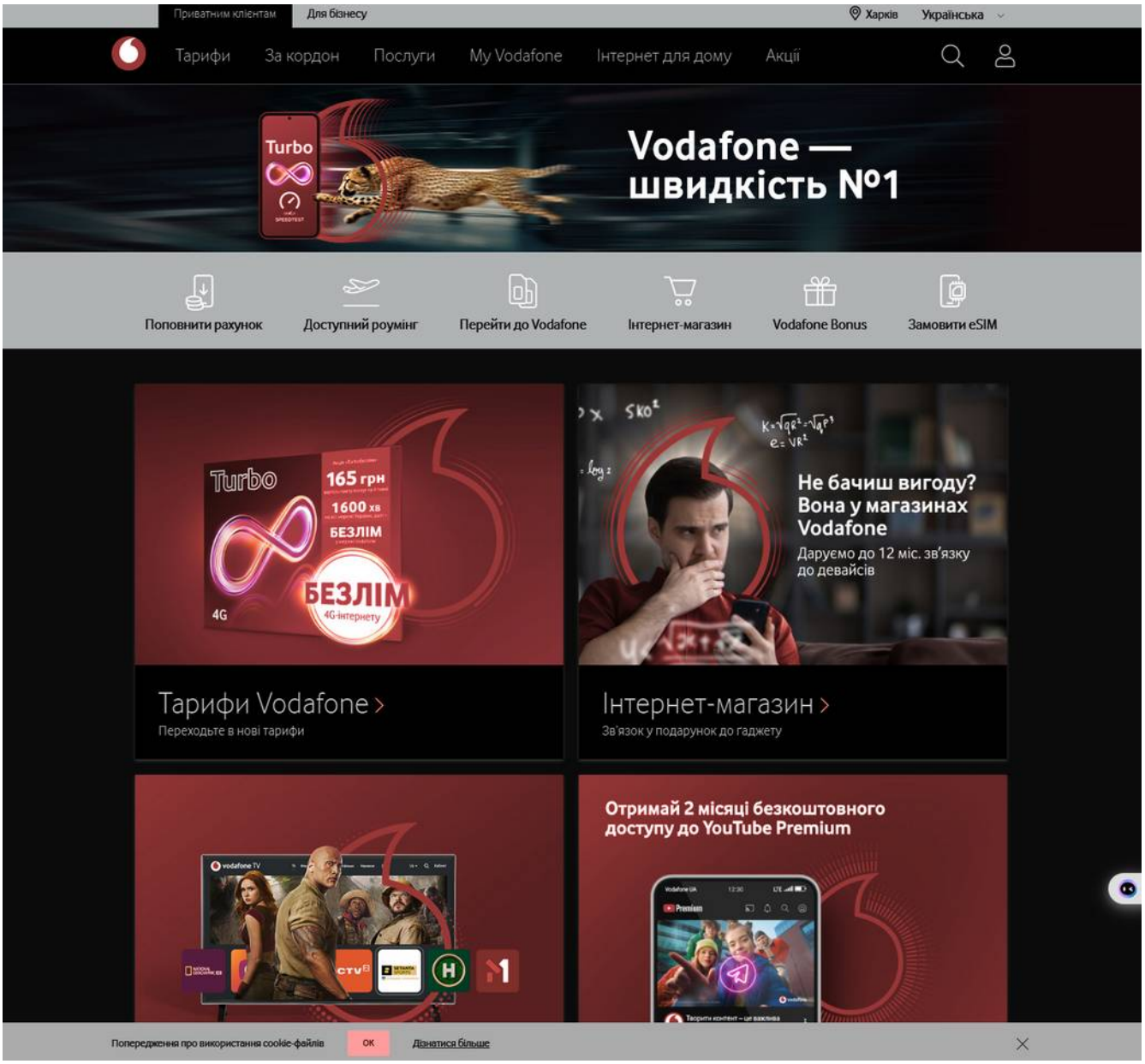

Рисунок 2.6 – Вебсторінка мобільного оператору Vodafone

2.3.3.3 Сервіс державних послуг

Сервіс «Дія» дає змогу зберігати водійське посвідчення, внутрішній і закордонний паспорти й інші документи в смартфоні, а також передавати їхні копії в різних життєвих ситваціях.

На рисунку 2.7 зображено вебсторінку сервісу державних послуг «Дія» реалізовану за допомогою сервісно-орієнтованої архітектури.

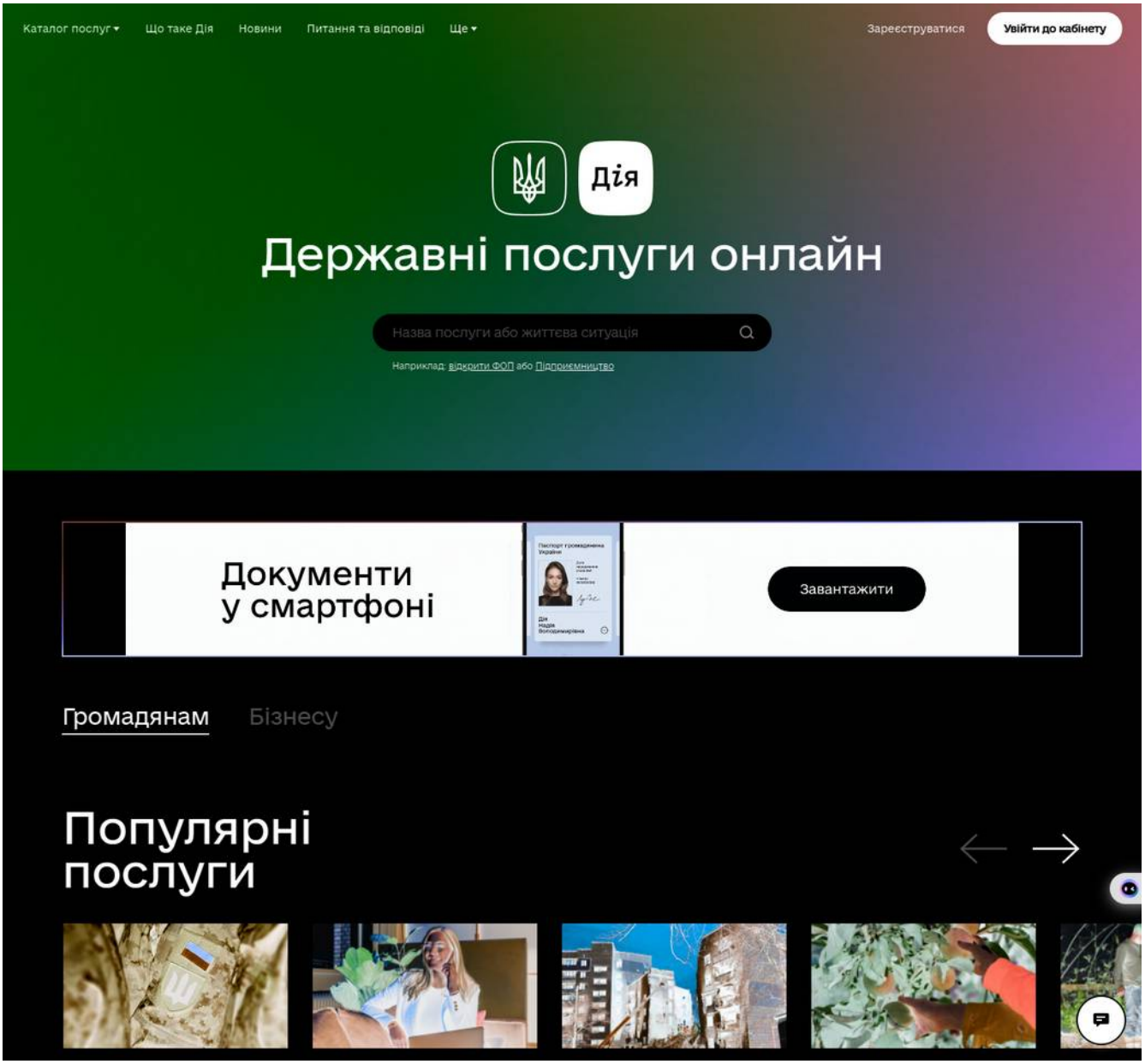

Рисунок 2.7 – Вебсторінка сервісу державних послуг «Дія»

2.3.4 Переваги та недоліки сервісно-орієнтованої архітектури

Сервісно-орієнтована архітектура має низку переваг, які роблять її привабливим вибором для багатьох систем. Однією з ключових переваг є те, що сервіси фу-

нкціонують як автономні компоненти. Кожен сервіс є самодостатнім і може працювати незалежно від інших, взаємодіючи через чітко визначені інтерфейси, зазвичай за допомогою протоколів, таких як SOAP. Ще однією перевагою є можливість повторного використання сервісів: сервіси, створені для однієї програми, можна без змін застосовувати в інших, що знижує витрати на розробку нових застосунків. Завдяки стандартизованим протоколам і інтерфейсам сервіси в сервісно-орієнтованій архітектурі можуть взаємодіяти незалежно від платформи або мови програмування, забезпечуючи високу інтероперабельність. Окрім цього, архітектура забезпечує гнучкість і масштабованість, дозволяючи легко змінювати або розширювати систему шляхом додавання нових сервісів чи модифікації існуючих без необхідности втручатися в роботу всієї системи.

Проте сервісно-орієнтована архітектура має й недоліки. Управління великою кількістю автономних сервісів може бути складним завданням, яке вимагає добре організованої інфраструктури та інструментів моніторину. Взаємодія між сервісами через мережу може спричиняти затримки в комунікаціях, що впливає на загальну продуктивність системи. Також одним із викликів є забезпечення безпеки: взаємодія між сервісами потребує додаткових заходів і ресурсів для захисту даних. Крім того, впровадження сервісно-орієнтованої архітектури вимагає значних інвестицій у створення інфраструктури, навчання персоналу та розробку нових сервісів, що може бути дороговартісним.

2.4 Мікросервісна архітектура

Мікросервісна архітектура – це підхід, де система розбивається на невеликі, незалежні сервіси, кожен з яких виконує конкретну функцію і може бути розгорнутий окремо. Ці сервіси взаємодіють за допомогою стандартизованих API через легковагові протоколи, такі як HTTP/REST.

2.4.1 Загальний опис мікросервісної архітектури

Мікросервісна архітектура базується на ідеї розбиття системи на невеликі, незалежні сервіси, кожен з яких виконує одну конкретну функцію. Кожен мікросервіс є незалежним і автономним, що дозволяє їм розгортатися та масштабуватися окремо один від одного. У мікросервісній архітектурі кожен сервіс відповідає за конкретний аспект бізнес-логіки, наприклад, обробка замовлень, управління користувачами або обробка платежів. Ці сервіси взаємодіють один з одним через легковагові API, зазвичай використовуючи HTTP/REST. Це дозволяє кожному сервісу бути незалежним від технології, на якій він реалізований, та легко змінювати або замінювати окремі сервіси без впливу на всю систему. Запити користувачів обробляються через спеціалізовані сервіси, що відповідають за конкретні функції. Наприклад, запит на створення нового замовлення може викликати кілька мікросервісів: один для перевірки наявности товару, інший для обробки платежу, і ще один для створення запису в базі даних. Розгортання мікросервісів відбувається окремо. Це дозволяє масштабувати окремі сервіси в залежности від навантаження, що робить систему більш гнучкою та ефективною в плані використання ресурсів. Крім того, оновлення можуть бути внесені в конкретний мікросервіс без необхідности зупинки всієї системи.

2.4.2 Приклади систем із мікросервісною архітектурою

Мікросервісна архітектура ідеально підходить для багатьох предметних областей та застосунків. Зокрема, її використовують великі інтернет-платформи, такі як Amazon, Netflix та eBay. Для забезпечення масштабованости та надійности своїх систем вони реалізовують кожну бізнес-функцію, наприклад, обробку замовлень, управління інвентарем чи стримінові сервіси, у вигляді окремих мікросервісів. У медичних інформаційних системах мікросервіси застосовуються для

управління електронними медичними записами, обробки страхових заявок, управління ліками та аналізу медичних даних. Такий підхід забезпечує високу гнучкість і надійність, що є важливим для роботи в галузі охорони здоров’я. Крім того, мікросервісна архітектура є оптимальним рішенням для систем інтернету речей (IoT). Вона дозволяє інтегрувати та обробляти дані з великої кількості різних пристроїв, забезпечуючи при цьому легке масштабування та високу продуктивність обробки інформації.

2.4.3 Приклади реалізації мікросервісної архітектури

Нижче наведено декілька прикладів систем, реалізованих за допомогою мікросервісної архітектури. Ці приклади демонструють різноманітні типи проєктів, які використовують мікросервісну архітектуру для досягнення своїх цілей. Спільним для всіх цих прикладів є те, що вони використовують мікросервісну архітектуру для забезпечення високої масштабованости, гнучкости та відмовостійкости.

2.4.3.1 Сервіс потокового мультимедіа

Сервіс megogo пропонує послуги потокового мультимедіа безпосередньо споживачам в однойменному OTT-медіасервісі для перегляду телебачення, кіно, спортивних трансляцій та прослуховування авдіо на різних платформах.

На рисунку 2.8 зображено вебсторінку сервісу потокового мультимедіа megogo реалізовану за допомогою мікросервісної архітектури.

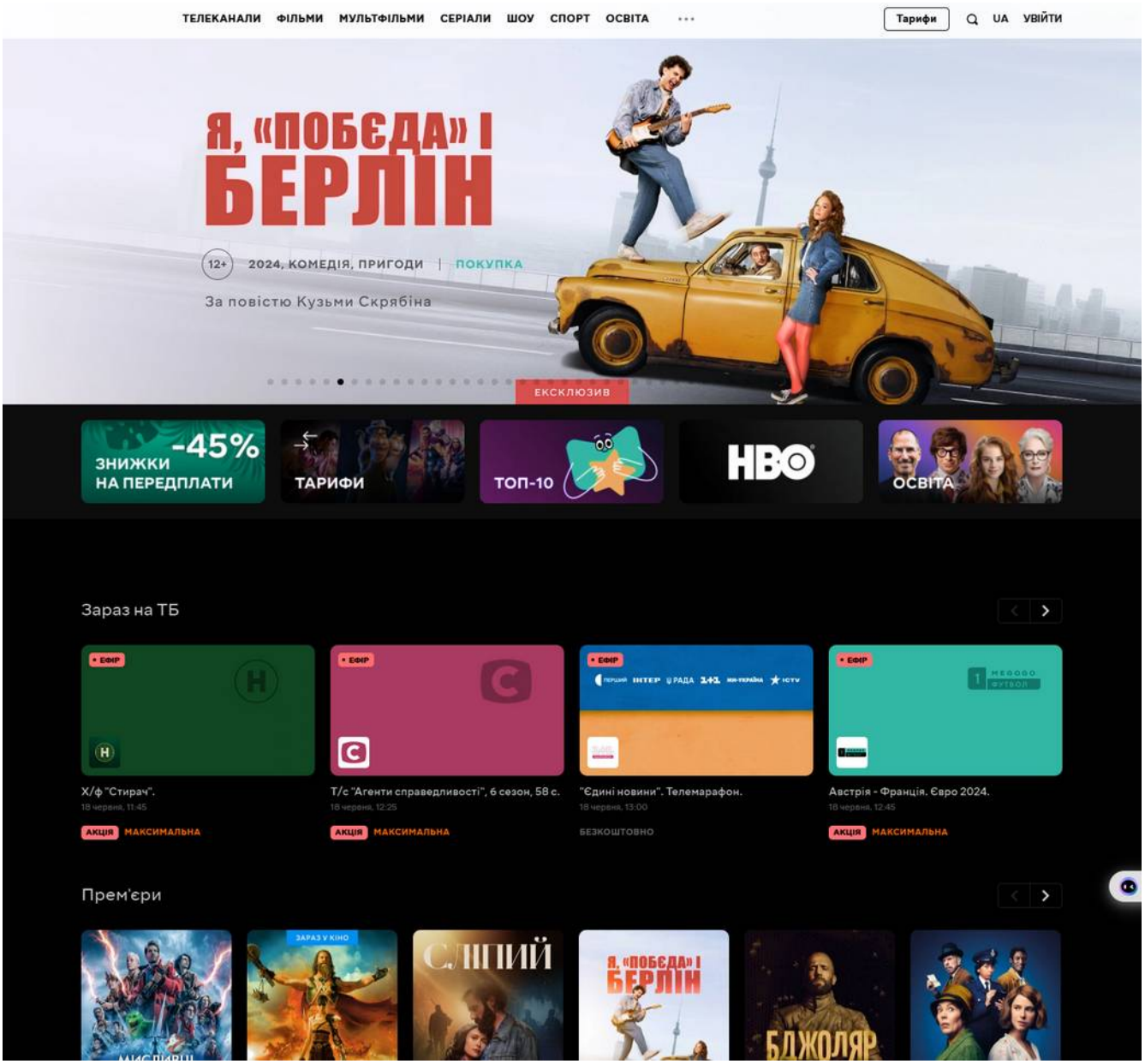

Рисунок 2.8 – Вебсторінка сервісу потокового мультимедіа megogo

2.4.3.2 Сервіс медичних інформаційних систем

Сервіс медичних інформаційних систем onclinic надає змогу реєстраторам записувати пацієнтів на прийом, лікарям вести медичну картку пацієнтів, а пацієнтам мати доступ до свого профілю здоров’я.

На рисунку 2.9 зображено вебсторінку сервісу медичних інформаційних систем onclinic реалізовану за допомогою мікросервісної архітектури.

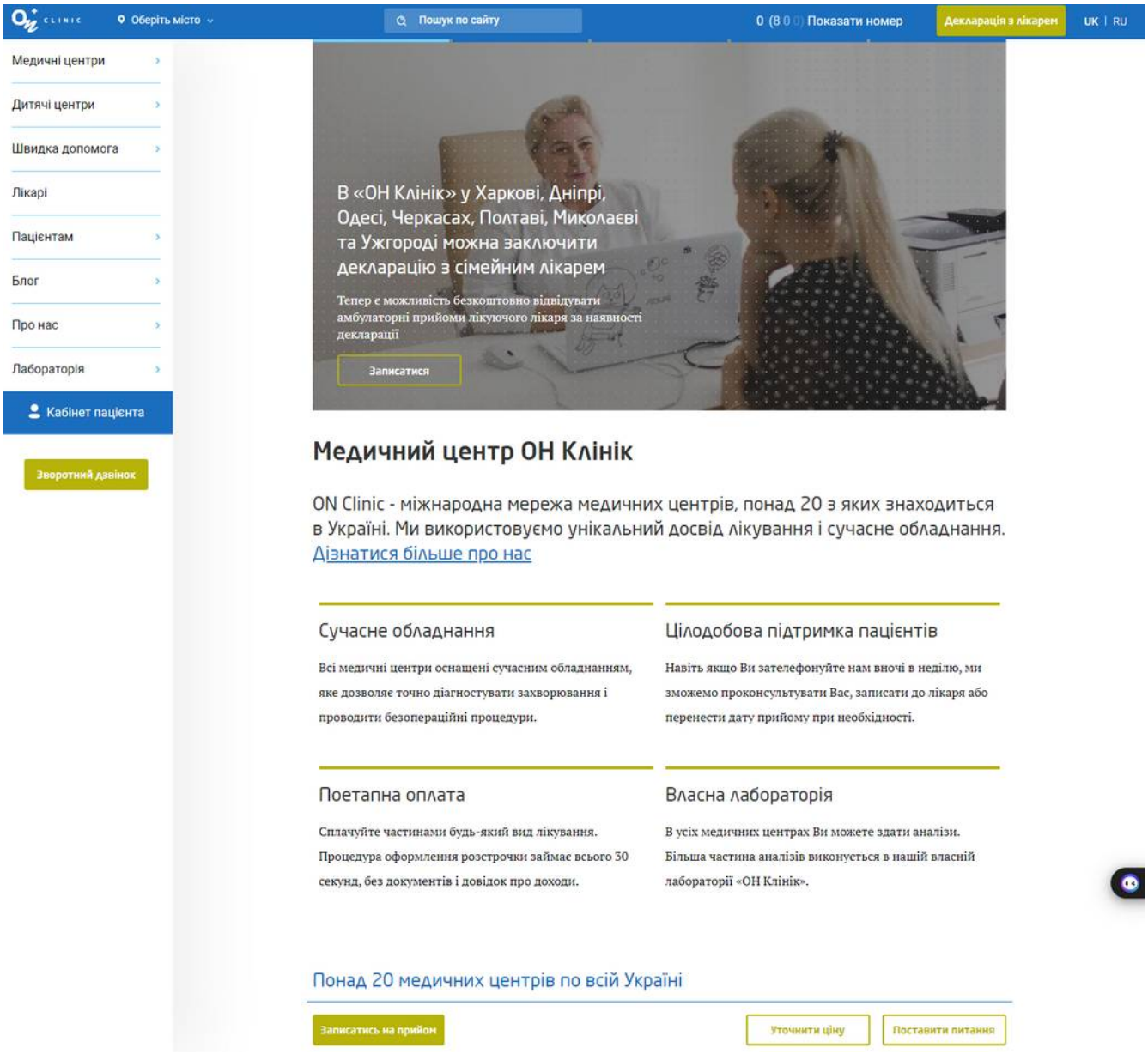

Рисунок 2.9 – Вебсторінка стартапу Advin

2.4.3.3 Сервіс розумного будинку

Сервіс розумного будинку IQdim пропонує послуги з автоматизації будинку, квартири чи комерційного приміщення в інтерактивному варіанті для користувачів сервісу.

На рисунку 2.10 зображено вебсторінку сервісу розумного будинку IQdim реалізовану за допомогою мікросервісної архітектури.

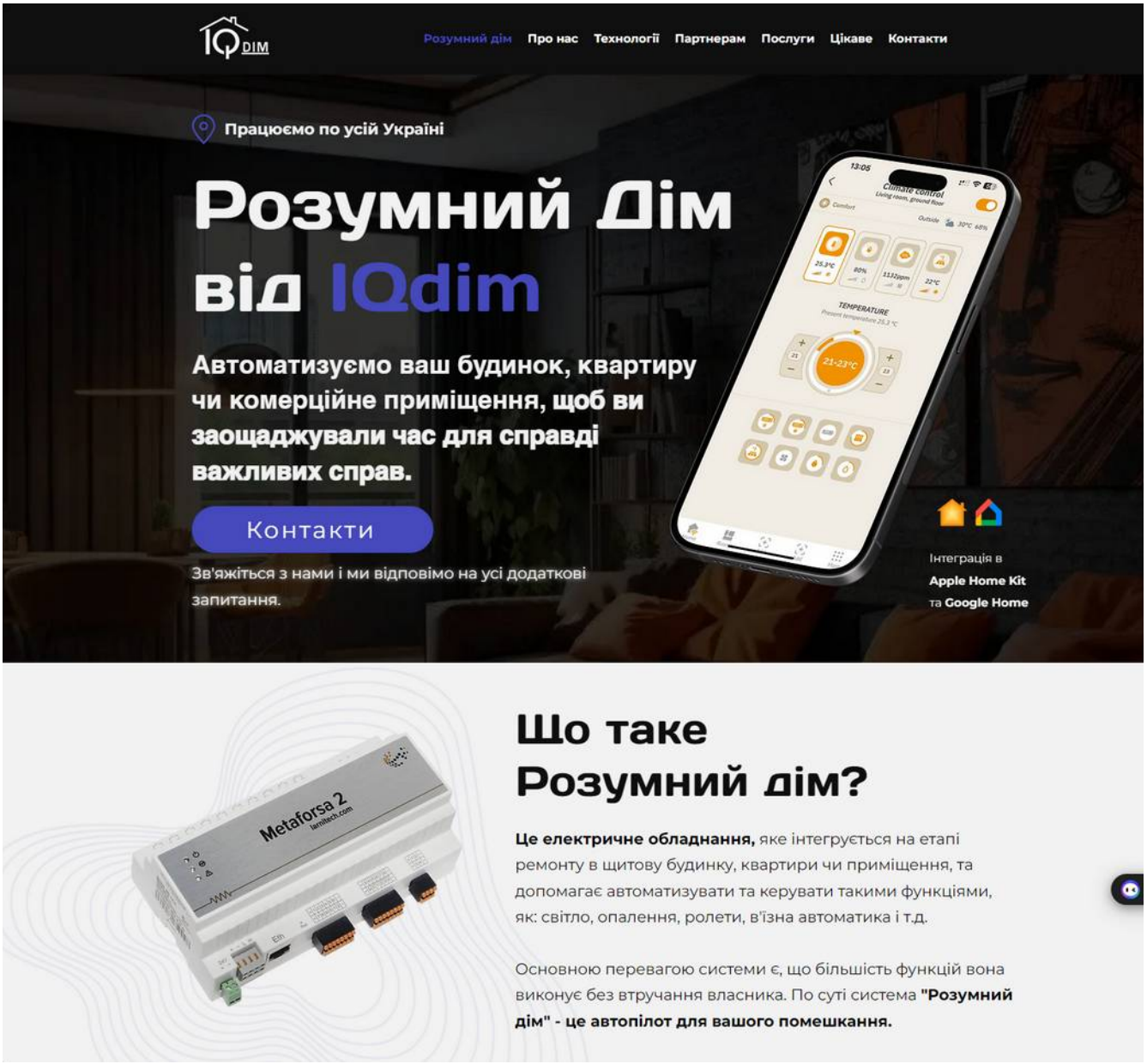

Рисунок 2.10 – Вебсторінка сервісу розумного будинку Iqdim

2.4.4 Переваги та недоліки мікросервісної архітектури

Переваги мікросервісної архітектури можна охарактеризувати через її гнучкість та швидкість розробки. Завдяки мікросервісам розробники мають змогу

оновлювати окремі компоненти системи, не впливаючи на інші її частини. Це значно пришвидшує процес розробки та зменшує ризик помилок. Додатково, архітектура відзначається масштабованістю, оскільки окремі мікросервіси легко масштабуються, що сприяє ефективному використанню ресурсів та забезпеченню високої продуктивности системи. Надійність також є важливим аспектом: у разі помилки в одному мікросервісі інші частини системи продовжують працювати стабільно. Кожен мікросервіс функціонує як відокремлений модуль, що спрощує управління та моніторин системи. До того ж, дана архітектура дозволяє використовувати різноманітні технології та обирати найбільш ефективні інструменти для кожного конкретного завдання, підвищуючи ефективність розробки.

Недоліки мікросервісної архітектури також варто зазначити. Розгортання великої кількости мікросервісів потребує добре організованої інфраструктури та спеціалізованих інструментів управління, що ускладнює процес. Взаємодія між мікросервісами через мережу може створювати додаткові затримки та викликати проблеми з продуктивністю. Забезпечення безпеки у процесі взаємодії мікросервісів є складним завданням, яке потребує додаткових заходів та ресурсів. Також управління залежностями між сервісами вимагає ретельного планування та може стати викликом для команди розробників.

2.5 Порівняння архітектур

2.5.1 Порівняння монолітної та сервісно-орієнтованої архітектури

У монолітних системах внесення змін у функціонал ускладнюється через тісний зв’язок між компонентами, що змушує розробників працювати з великим обсягом коду навіть для невеликих доопрацювань, тоді як у сервісно-орієнтованій архітектурі кожен сервіс є незалежною одиницею, завдяки чому модифікації виконуються локально, без впливу на інші частини системи.

Розгортання всього застосунку під час масштабування в монолітній архітектурі спричиняє значні витрати ресурсів, адже неможливо виділити окремі компоненти, натомість сервісно-орієнтовані системи дозволяють масштабувати лише ті сервіси, які потребують додаткових ресурсів, що значно підвищує ефективність використання інфраструктури.

Автономність сервісів у сервісно-орієнтованій архітектурі забезпечує високу надійність, адже вихід з ладу одного сервісу не впливає на інші, тоді як у монолітній системі помилка в одному модулі здатна порушити роботу всього застосунку.

Процес інтеграції нових технологій чи зовнішніх систем у монолітній архітектурі часто ускладнюється через взаємозалежність компонентів, водночас сервісно-орієнтована архітектура спрощує цей процес завдяки модульному підходу та стандартизованим інтерфейсам, які дозволяють швидко долучати додаткові рішення.

2.5.2 Порівняння монолітної та мікросервісної архітектури

У монолітній архітектурі внесення змін у функціонал потребує втручання в увесь застосунок, оскільки всі його компоненти тісно пов’язані, тоді як у мікросервісній архітектурі кожен мікросервіс функціонує незалежно, що дозволяє оновлювати окремі частини системи без впливу на інші.

Масштабування в монолітних системах є трудомістким і передбачає одночасне розгортання всього застосунку навіть для мінімальних змін, у той час як мікросервіси можна масштабувати окремо, спрямовуючи ресурси лише туди, де вони потрібні, і таким чином підвищуючи ефективність використання інфраструктури.

Помилка в одному модулі монолітної системи здатна викликати збій усього застосунку, що знижує надійність системи, натомість у мікросервісній архітектурі збої обмежуються окремими мікросервісами, забезпечуючи стабільну роботу решти компонентів.

Процес розгортання у монолітних системах потребує значних зусиль, адже всі компоненти запускаються одночасно, тоді як мікросервіси можна розгортати чи оновлювати індивідуально, що скорочує час і ресурси, необхідні для впровадження змін.

2.5.3 Порівняння сервісно-орієнтованої та мікросервісної архітектури

Модульність є важливою характеристикою обох архітектур, але її реалізація відрізняється: у сервісно-орієнтованій архітектурі сервіси є незалежними, але часто потребують централізованого контролю, тоді як мікросервіси розробляються, розгортаються і масштабуються автономно без необхідности в централізованому управлінні.

Гнучкість у мікросервісних системах значно вища, оскільки кожен мікросервіс може використовувати свої власні технології, що забезпечує більшу свободу в розробці, на відміну від сервісно-орієнтованих систем, де інтеграція різних технологій може вимагати додаткових зусиль через необхідність дотримання стандартів.

Організація управління в сервісно-орієнтованих системах зазвичай централізована, що забезпечує узгодженість сервісів, тоді як у мікросервісах кожен компонент є автономним і відповідає за свою функціональність, що сприяє незалежности та мінімізації ризику відмови через єдину точку.

Масштабування в обох архітектурах дозволяє розширювати окремі компоненти, але в сервісно-орієнтованих системах потрібно враховувати взаємодію між сервісами, що ускладнює процес, тоді як у мікросервісах кожен мікросервіс може масштабуватися незалежно, підвищуючи ефективність використання ресурсів.

Взаємодія між сервісами в сервісно-орієнтованих архітектурах зазвичай реалізується через стандартизовані протоколи, такі як SOAP, що забезпечує сумісність, але може бути складним у налаштуванні, тоді як у мікросервісах часто

використовуються більш легкі протоколи, наприклад HTTP/REST, що робить взаємодію швидшою і простішою.

2.6 Підсумки порівнянь

Монолітна архітектура, хоча і проста у реалізації на початкових етапах, має свої значні обмеження. Складність внесення змін, низька масштабованість, нестабільність та тривалий час розгортання роблять її менш привабливою. Ці обмеження стають ще більш очевидними, коли мова йде про підтримку різноманітних смарт-пристроїв, де кожне нове пристосування вимагатиме значних змін у всій системі, що може призвести до збільшення витрат та часу на розробку.

Сервісно-орієнтована архітектура є кроком вперед порівняно з монолітною, надаючи можливість розділення функціоналу на окремі сервіси. Вона забезпечує певну гнучкість і незалежність розробки, але її складність у інтеграції та управлінні сервісами, а також залежність від складних механізмів комунікації, таких як ESB (Enterprise Service Bus), можуть створювати додаткові проблеми. У нашому контексті, де потрібна висока швидкість реакції та незалежність сервісів, серверно-орієнтована архітектура може виявитися занадто громіздкою та складною в управлінні.

Мікросервісна архітектура, на відміну від серверно-орієнтованої архітектури, орієнтована на максимальну декомпозицію системи на малі, незалежні сервіси, які легко масштабуються і розгортаються. Це дозволяє нам ефективно впроваджувати підтримку різноманітних смарт-пристроїв без значних змін у всій системі. Кожен мікросервіс може бути відповідальним за конкретний функціонал або підтримку певного типу пристрою, що значно підвищує гнучкість та швидкість розробки.

Аналізуючи різні архітектурні підходи, зрозуміло, що мікросервісна архітектура підходить найліпше для вирішення нашої проблеми. Вона забезпечить необхідну масштабованість, що дозволить швидко і ефективно розробити підтри-

мку різноманітних смарт-пристроїв для збільшення конкурентоспроможности та охоплення авдиторії.

3 ОПИС ЕЛЕМЕНТІВ ІСНУЮЧИХ СИСТЕМ ЗАБЕЗПЕЧЕННЯ

У цьому розділі описані елемети існуючих систем забезпечення.

3.1 Опис предметної области

Об’єктом дослідження предметної области є сфера доставки. Основні процеси, механізми взаємодій суб’єктів з середовищем, правила обліку замовленнями, внутрішні організаційні структури сфери та інше зазначені в даному розділі. Для початку розглянемо початкові, але фундаментальні терміни цієї сфери та їх особливости.

Інтернет доставка пакунків (англ. Internet parcel delivery) – процес доставки товарів та пакунків споживачам через Інтернет-платформу. Цей сервіс надає можливість замовити та отримати пакунок безпосередньо до дверей споживача. В процесі використовуються Інтернет-технології для замовлення, відстеження та спілкування з кур’єрами.

До впровадження змін, система мала вхідні дані, зазначені нижче. Після переліку елементів існуючих систем забезпечення, для зручности і наочности будуть розроблені елементи алгоритмічної та дизайнової систем забезпечення, а також надані рекомендації щодо розробки елементів програмної системи забезпечення для вебзастосунку, оскільки він дозволяє найкраще показати всі етапи інтеграції мікросервісів. Це зумовлено зручністю розробки, тестування та відлагодження такої системи в контексті вебтехнологій, а також можливістю продемонструвати ключові принципи архітектури.

3.2 Схема організаційної структури сервісу

Схема організаційної структури сервісу приведена на рисунку 3.1 [5].

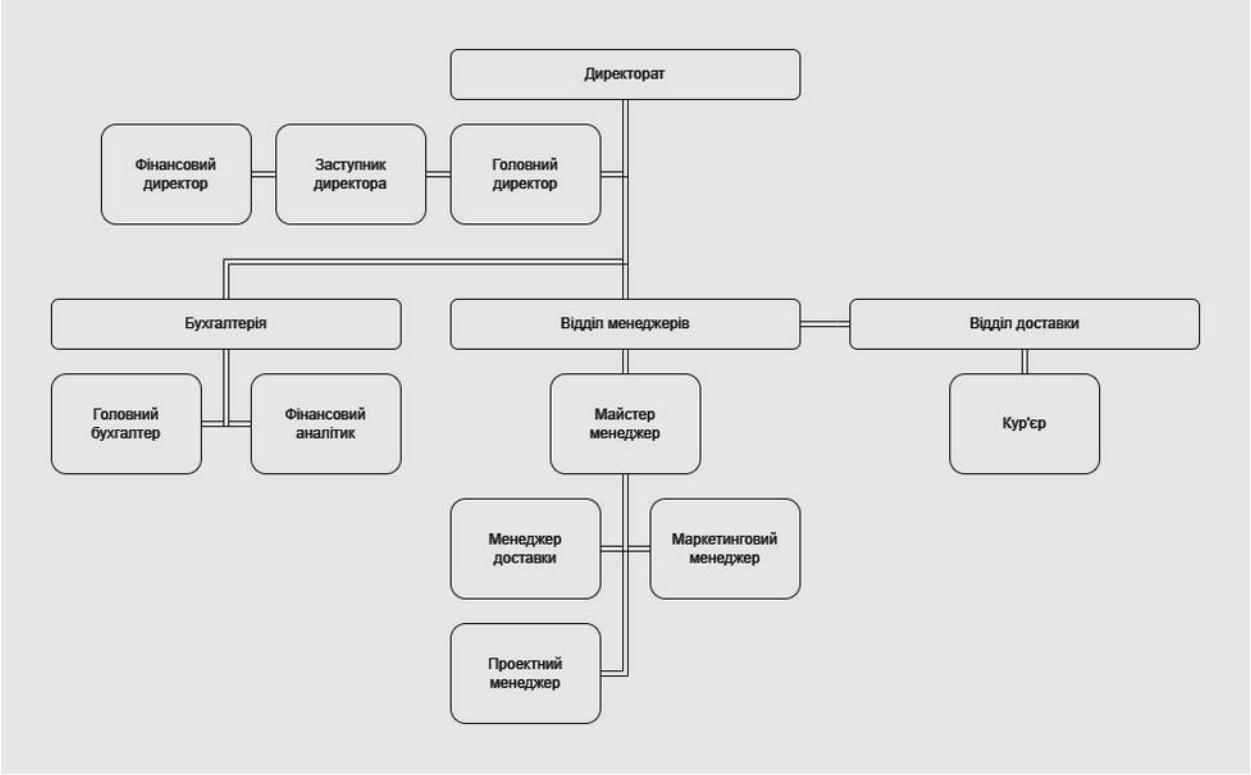

Рисунок 3.1 – Схема організаційної структури сервісу

Опис підрозділів системи та їх функцій:

– директорат;

1) стратегічне управління. Розробка стратегій підприємства та визначення його довгострокових цілей, уникнення ризиків;

2) кадрова політика. Найм, звільнення, компенсація, розвиток кадрів, оцінка праці;

3) представництво підприємства. Ведення переговорів, укладання угод, підтримка стосунків та відстоювання інтересів підприємства перед зацікавленими сторонами, такими як інвестори, партнери, клієнти, регулюючі органи тощо;

– бухгалтерія;

1) фінансовий облік. Ведення детального обліку фінансових транзакцій підприємства, таких як прибутки, витрати, активи та зобов’язання;

2) фінансові звіти. Готування фінансових звітв, таких як: звіт про прибутки та збитки, баланс тощо;

3) оподаткування та податковий облік. Коректне розраховування, звітність та сплата податків;

4) бюджетування та прогнозування. Складання бюджету підприємства, аналіз фінансових даних, прогнозування фінансових результатів та ризиків;

– відділ менеджерів;

1) управління та контроль за функціональними підрозділами. Координація діяльности різних функціональних підрозділів, контроль за їх функціонуванням;

2) керівництво та мотивація. Надання напрямку та надихання команди до досягнення високих результатів, створення сприятливого середовища для розвитку та ефективної роботи співробітників;

– відділ доставки;

1) взаємодія з клієнтами. Відповідання на питання пов’язані з доставкою, надання інформації про статус доставки, узгодження умов доставки;

2) доставка пакунків. Упаковка пакунків, вибір оптимальних маршрутів доставки, забезпечення безпеки пакунків під час транспортування.

3.3 Діаграма IDEF0 процесу «Ведення обліку доставки»

На рисунку 3.2 зображено діаграму IDEF0 процесу «Вести облік доставки» з усіма вхідними та вихідними даними, обмеженнями та діючими особами [6].

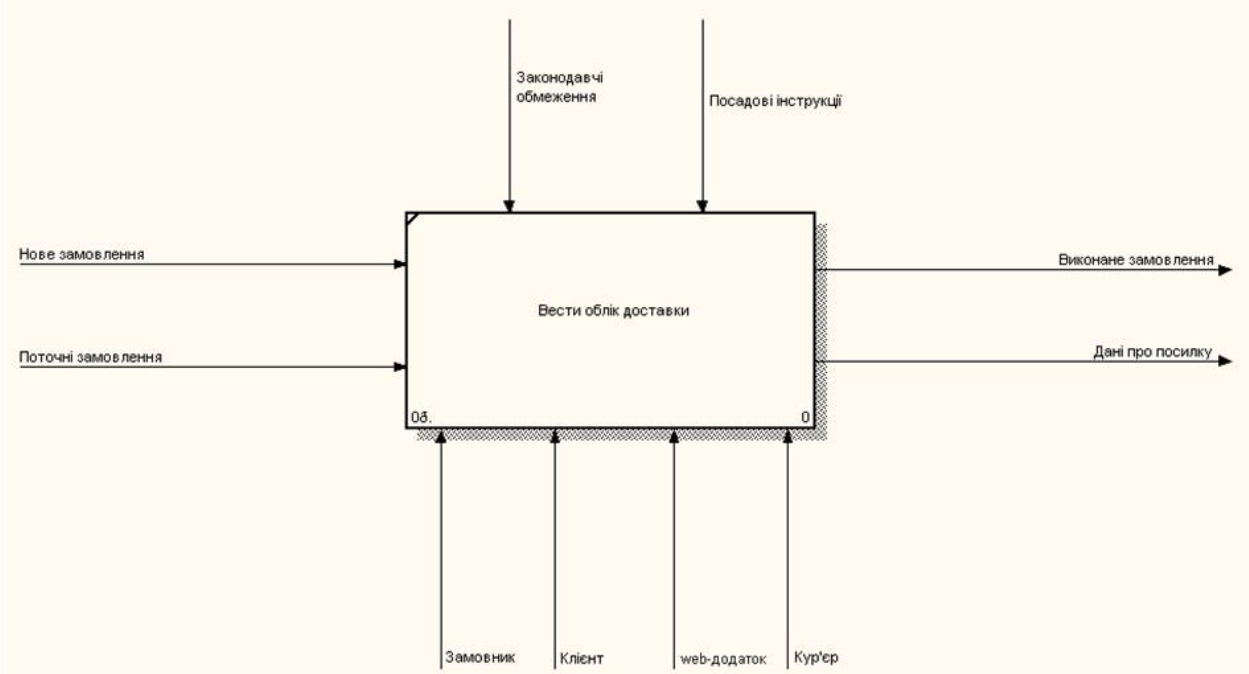

Рисунок 3.2 – Діаграма IDEF0 процесу «Ведення обліку доставки»

Процес «Вести облік доставки» включає вхідні дані, такі як нове замовлення, що містить всю необхідну інформацію про замовлення, а також поточні замовлення, що допомагає відстежувати статус вже створених замовлень. В результаті цього процесу генеруються вихідні дані, серед яких виконане замовлення, інформація про успішно виконану доставку, яка потребує оновлення в системі, а також дані про посилку, що містять специфікацію замовлення. Врахований ряд обмежень, таких як законодавчі вимоги, що регулюють обробку і збереження даних про доставку відповідно до чинного законодавства, а також посадові інструкції, що визначають конкретні вимоги та процедури для менеджера з доставки та кур’єра. Зазначені кілька ключових осіб, зокрема замовник, який розміщує нове замовлення і надає відповідну інформацію, клієнт – отримувач посилки, вебзастосунок, що відповідає за організацію доставки, та кур’єр, який здійснює фактичну доставку посилки від місця відправлення до місця призначення.

Ця діаграма допомагає уявити послідовність дій та взаємодію осіб із залученням вхідних та вихідних даних у процесі обліку доставки, а також врахування обмежень, що впливають на процес.

3.4 Діаграма IDEF0 декомпозиції процесу «Ведення обліку доставки»

На риснку 3.3 зображена діаграма IDEF0 декомпозиції процесу «Вести облік доставки».

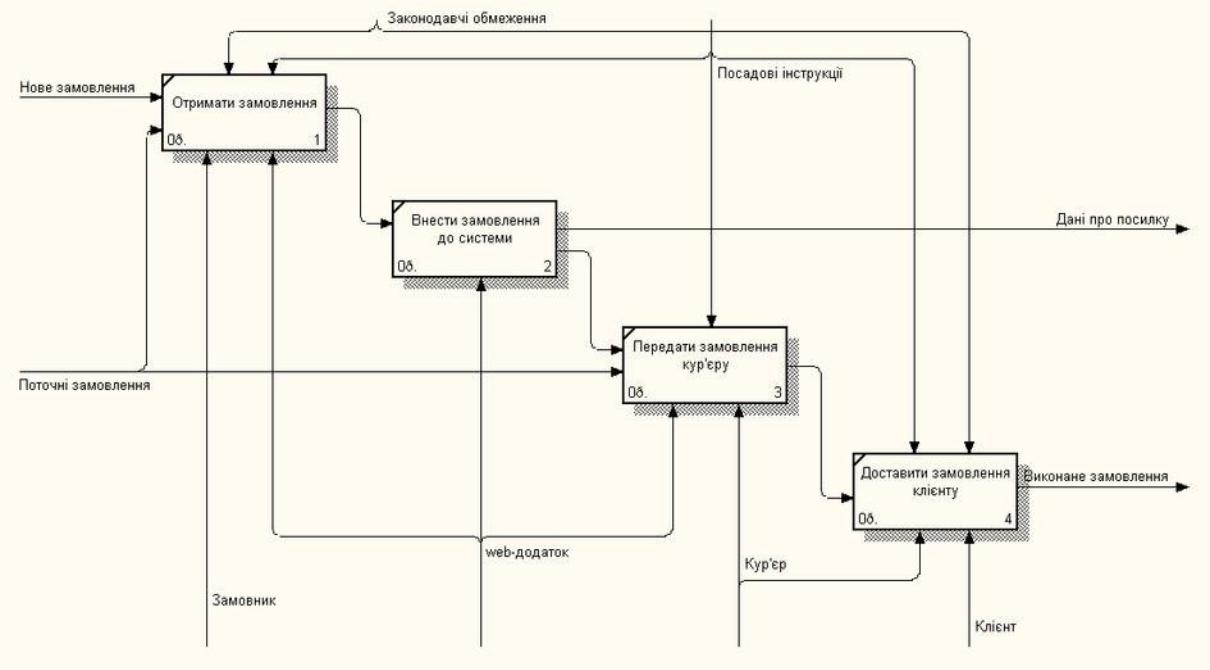

Рисунок 3.3 – Діаграма IDEF0 декомпозиції процесу «Ведення обліку доставки»

Перший процес, «Отримати замовлення», передбачає отримання нового замовлення від замовника. Вебзастосунок приймає замовлення та включає його до поточних замовлень, що забезпечує необхідний облік і документацію замовлення в системі. Другий процес, «Внести замовлення до системи», полягає у внесенні отриманого замовлення до системи, де вебзастосунок вводить всі необхідні дані про замовлення. Після цього наступає третій процес, «Передати замовлення кур’єру», у якому підготовлене замовлення передається кур’єру. Після внесення замовлення до системи, система визначає поточні замовлення, готові до доставки, і передає їх за допомогою вебзастосунку кур’єру для подальшої доставки клієнтам. Четвертий процес, «Доставити замовлення клієнту», полягає в доставці замовлення клієнту.

Кур’єр забирає замовлення і доставляє його до клієнта, а після успішної доставки процес завершується, і інформація про виконане замовлення фіксується в системі.

3.5 Логічна ERD сервісу

Логічна ERD демонструє концептуальну структуру бази даних, що відображає сутности, їх атрибути та зв’язки між ними [7].

Логічна ERD сервісу зображена на рисунку 3.4.

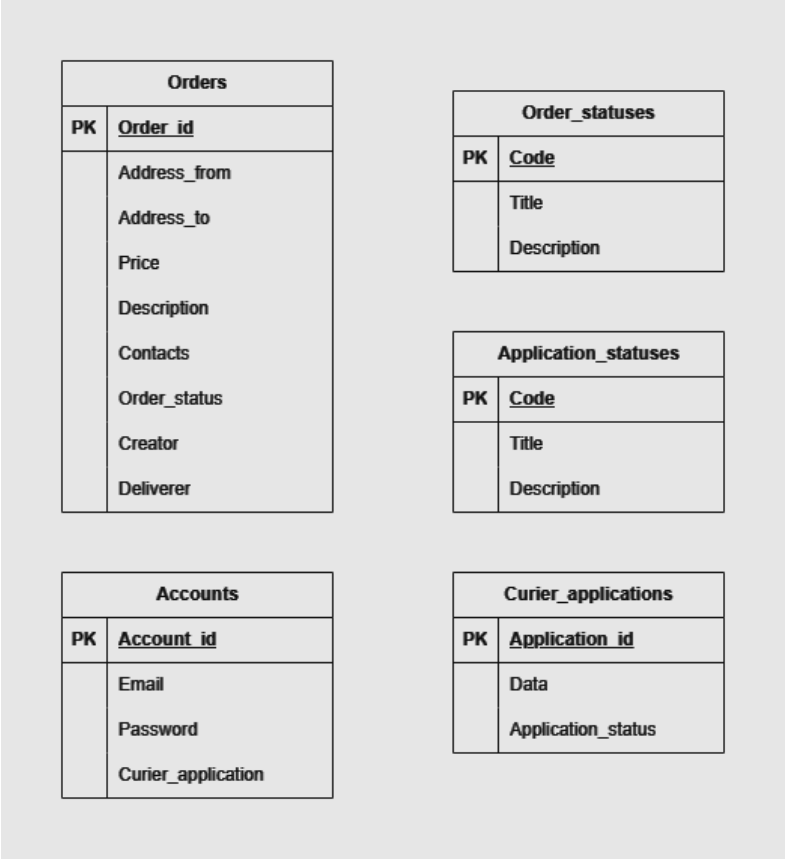

Рисунок 3.4 – Логічна ERD сервісу

3.6 Фізична ERD сервісу

Фізична ERD, у свою чергу, визначає реалізацію бази даних на рівні таблиць, стовпців та зв’язків.

Фізична ERD сервісу зображена на рисунку 3.5.

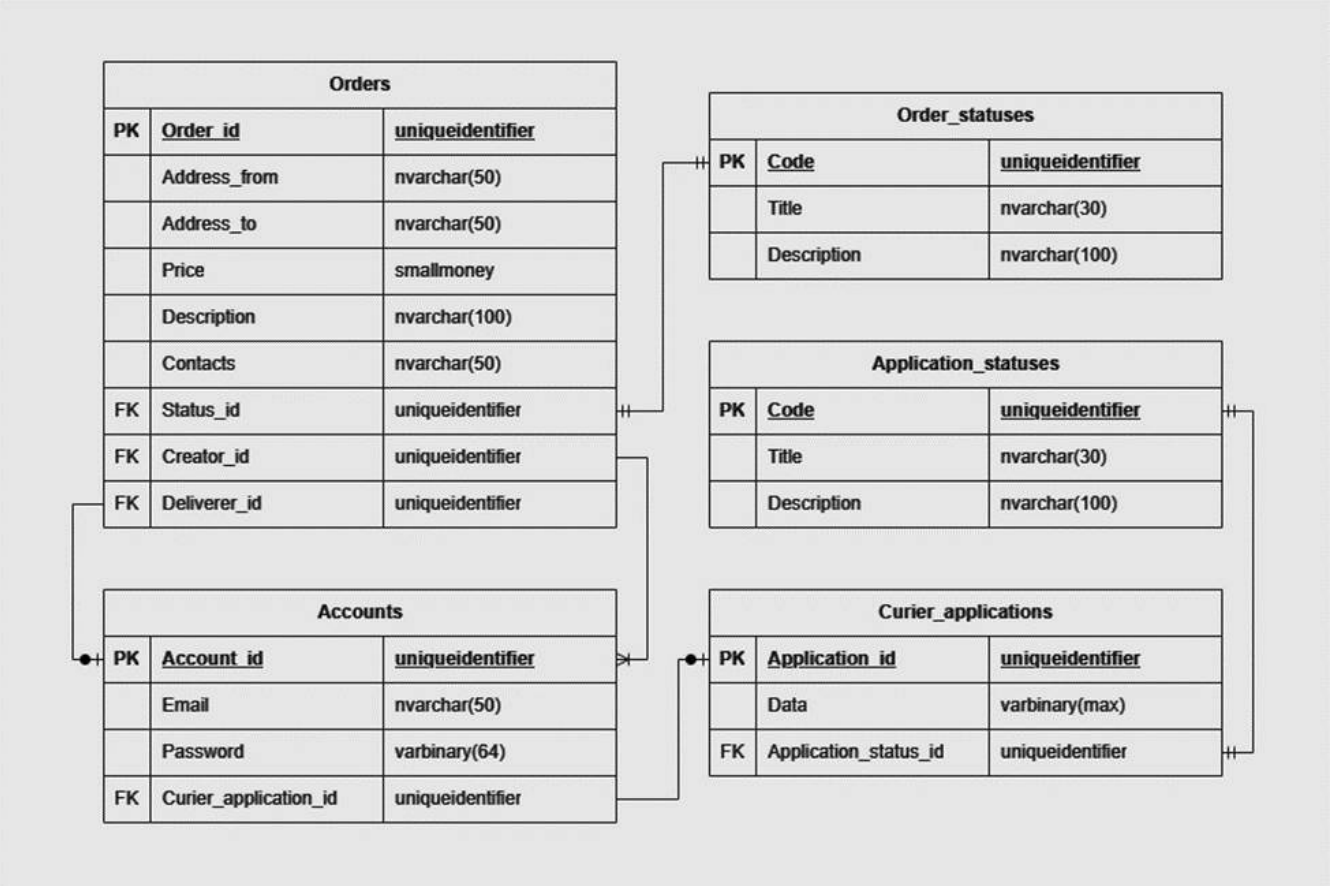

Рисунок 3.5 – Фізична ERD сервісу

У таблицях 3.1, 3.2, 3.3, 3.4, 3.5 представлений детальний опис таблиць фізичної ERD інформаційної системи забезпечення.

Таблиця 3.1 – Опис таблиці «Orders»

Тип поля	Назва змінної	Тип даних	Опис
Primary Key	Order_id	uniqueidentifier	Унікальний айдентифікатор замовлення
Value Field	Address_from	nvarchar(50)	Адреса пакунку
Value Field	Address_to	nvarchar(50)	Адреса доставки пакунку
Value Field	Price	smallmoney	Ціна за доставку
Value Field	Description	nvarchar(100)	Опис замовлення
Value Field	Contacts	nvarchar(50)	Контакти отримувача
Foreign Key	Status_id	uniqueidentifier	Зовнішній ключ на сутність із таблиці статусів замовлення
Foreign Key	Creator_id	uniqueidentifier	Зовнішній ключ на сутність із таблиці акавнтів
Foreign Key	Delivered_id	uniqueidentifier	Зовнішній ключ на сутність із таблиці акавнтів

Таблиця 3.2 – Опис таблиці «Accounts»

Тип поля	Назва змінної	Тип даних	Опис
Primary Key	Account_id	uniqueidentifier	Унікальний айдентифікатор акавнта
Value Field	Email	nvarchar(50)	Пошта акавнта
Value Field	Password	varbinary(64)	Пароль акавнта
Foreign Key	Curier_application_id	uniqueidentifier	Зовнішній ключ на сутність із таблиці заяв на становлення кур’єром

Таблиця 3.3 – Опис таблиці «Curier_applications»

Тип поля	Назва змінної	Тип даних	Опис
Primary Key	Application_id	uniqueidentifier	Унікальний айдентифікатор заяви на становлення кур’єром
Value Field	Data	varbinary(max)	Зображення з айдентифікаційними даними особи
Foreign Key	Application_status_id	uniqueidentifier	Зовнішній ключ на сутність із таблиці статусів заяв

Таблиця 3.4 – Опис таблиці «Application_statuses»

Тип поля	Назва змінної	Тип даних	Опис
Primary Key	Code	uniqueidentifier	Унікальний айдентифікатор статусу заяви
Value Field	Title	nvarchar(30)	Назва статусу заяви
Value Field	Description	nvarchar(100)	Опис статусу заяви

Таблиця 3.5 – Опис таблиці «Order_statuses»

Тип поля	Назва змінної	Тип даних	Опис
Primary Key	Code	uniqueidentifier	Унікальний айдентифікатор статуса замовле-ння
Value Field	Title	nvarchar(30)	Назва статусу замовлення
Value Field	Description	nvarchar(100)	Опис статусу замовлення

4 РОЗРОБКА ЕЛЕМЕНТІВ СИСТЕМ ЗАБЕЗПЕЧЕННЯ

У цьому розділі розроблені модель мікросервісної архітектури інформаційної системи, алгоритмічна та дизайнова системи забезпечення, а також надані рекомендації щодо розробки елементів програмної системи забезпечення.

4.1 Розробка моделі мікросервісної архітектури інформаційної системи

4.1.1 Концепція та використання моделі C4

Модель C4 (Context, Container, Component, Code) є підходом до створення архітектурних діаграм, який дозволяє структуровано представляти зв’язки компонентів у системі. Вона розроблена для полегшення аналізу та візуалізації складних систем і забезпечення прозорости їхньої архітектури [8].

У таблиці 4.1 зазначено принципи та використання моделі C4 в системі «Order Management System».

Таблиця 4.1 – Принципи та використання моделі C4 в системі «Order Management System».

Назва компоненту	Принцип компоненту	Використання компоненту
1	2	3
Context (Контекст)	Найвищий рівень моделі, що визначає систему як чорний ящик у її середовищі.	Визначає основних користувачів (User) та зовнішні системи. Показує, як ці елементи взаємодіють із системою загалом.

Продовження таблиці 4.1

1	2	3
Container (Контейнер)	Представляє програмну систему як набір контейнерів – основних елементів виконання, таких як веб-додатки, бази даних або мікросервіси.	Включає контейнери, такі як Client Application, API Gateway, мікросервіси (Order Service, Payment Service тощо) та зовнішні сервіси (SMTP, Twilio, FCM).
Component (Компонент)	Фокусується на внутрішній структурі контейнера, показуючи, як він організований у вигляді функціональних частин.	Включає компоненти, контролери, сервіси бізнес-логіки та інші модулі, відповідальні за конкретну функціональність контейнера.
Code (Код)	Найнижчий рівень деталізації, який показує реалізацію конкретного компонента, включаючи класи, методи, структури.	Пояснює внутрішню логіку обраного компонента, наприклад, як OrderManager взаємодіє з базою даних MS SQL Server або викликає API інших сервісів.

Модель C4 має низку переваг, які роблять її корисною для проєктування та розуміння архітектури програмних систем. Однією з головних переваг є візуалізація складних систем: модель дозволяє отримати огляд на всіх рівнях архітектури, починаючи від загального контексту і завершуючи деталями реалізації. Ця модель також забезпечує гнучкість, що дозволяє легко модифікувати, додавати або змінювати компоненти системи відповідно до потреб. Завдяки прозорости модель C4 дає змогу зрозуміти роль кожного компонента та характер їхньої взаємодії. Крім того, підтримка модульности допомагає розділити систему на логічні блоки, що значно спрощує процеси тестування та розгортання.

4.1.2 Діаграма рівня Container за моделлю C4

На рисунку 4.1 зображено діаграму рівня Container за моделлю C4.

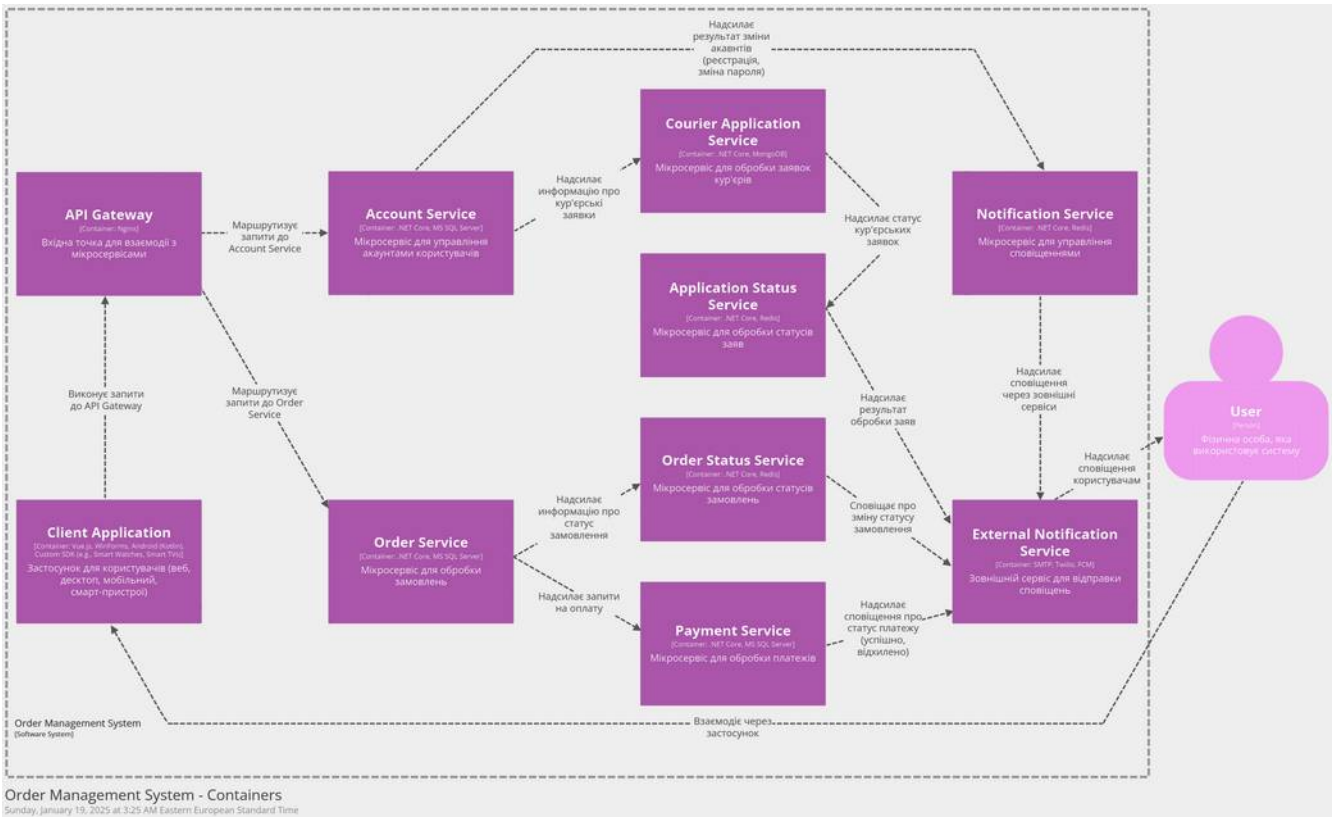

Рисунок 4.1 – Діаграма рівня Container за моделлю C4

Важливо зазначити що у нашому випадку модель C4 завершується на рівні Container, оскільки метою є концептуальне представлення архітектури системи, що зосереджується на основних компонентах і їхніх зв’язках, без детального занурення в імплементацію, яка буде визначена на етапі розробки.

4.1.3 Компоненти системи

Система «Order Management System» побудована за мікросервісною архітектурою, яка передбачає розподіл логіки на кілька незалежних компонентів (сервісів), кожен із яких відповідає за виконання певних функцій. Головна мета такої архітектури – забезпечення масштабованости, гнучкости в розробці, тестуванні та експлуатації.

Модель системи побудована за принципами мікросервісної архітектури, де кожен мікросервіс відповідає за окрему бізнес-логіку та працює незалежно.

У таблиці 4.2 зазначено компоненти системи «Order Management System».

Таблиця 4.2 – Компоненти системи «Order Management System»

Назва	Опис	Функція	Технології
1	2	3	4
Користувач (User)	Фізична особа, яка використовує систему	Взаємодія з клієнтським застосунком	–
Client Application	Репрезентує веб, десктоп, мобільні, а також смарт-пристрійні застосунки	Зв’язує користувача із системою; працює на різних платформах (веб, десктоп, мобільні)	Vue.js, WinForms, Kotlin, Custom SDK
API Gateway	Централізована точка входу	Маршрутизація запитів між клієнтами та мікросервісами	Nginx

Продовження таблиці 4.2

1	2	3	4
Order Service	Мікросервіс управління замовленнями	Обробка замовлень, взаємодія з базою даних і іншими сервісами для оновлення статусів	.NET Core, MS SQL Server
Account Service	Мікросервіс управління акавнтами	Управління Інформацією про користувачів	.NET Core, MS SQL Server
Courier Application Service	Мікросервіс для кур’єрських заявок	Обробка заявок кур’єрів і пере-дача їх статусів	.NET Core, MongoDB
Order Status Service	Мікросервіс статусів замовлень	Управління статусами замовлень, використовуючи кеш Redis	.NET Core, Redis
Application Status Service	Мікросервіс статусів заяв	Обробка результатів заяв і інформування Notification Service	.NET Core, Redis
Payment Service	Мікросервіс обробки платежів	Управління платежами та інте-грація з базою даних	.NET Core, MS SQL Server
Notification Service	Мікросервіс управління спові-щеннями	Надсилання сповіщень до External Notification Service	.NET Core, Redis
External Notification Service	Репрезентує мі-кросервіси для доставки спові-щень	Доставка повідомлень через ка-нали (SMS, електронна пошта, push-нотифікації)	SMTP, Twilio, FCM

4.1.4 Взаємодії системи

Взаємодії між компонентами системи «Order Management System» відображають передачу даних та запитів для забезпечення коректної роботи кожного з модулів у межах загальної архітектури.

У таблиці 4.3 зазначені взаємодії системи «Order Management System»

Таблиця 4.3 – Взаємодії системи «Order Management System»

Компонент-відправник	Тип та зміст взаємодії	Компонент-отримувач
1	2	3
User	Надсилає запити через застосунок для взаємодії з системою	Client Application
Client Application	Передає запити для маршрутизації до мікросервісів	API Gateway
API Gateway	Маршрутизує запити для створення, оновлення або отримання інформації про замовлення	Order Service
API Gateway	Маршрутизує запити для управління акавнтами користувачів	Account Service
API Gateway	Маршрутизує запити для ініціації сповіщень	Notification Service
Order Service	Надсилає Інформацію для оновлення статусу замовлення	Order Status Service
Order Service	Надсилає запити на оплату для обробки платежів	Payment Service
Order Status Service	Надсилає сповіщення про зміну статусу замовлення	Notification Service

Продовження таблиці 4.3

1	2	3
Account Service	Передає інформацію про заявки кур’єрів	Courier Application Service
Account Service	Надсилає результат зміни акавнтів (реєстрація, зміна пароля)	Notification Service
Courier Application Service	Надсилає статуси кур’єрських заявок для обробки	Application Status Service
Application Status Service	Інформує про результати обробки заявок для пере-дачі користувачам	Notification Service
Payment Service	Надсилає сповіщення про статус платежу (успішно, відхилено).	Notification Service
Notification Service	Надсилає повідомлення через зовнішні сервіси (SMS, email, push-сповіщення)	External Notification Service
External Notification Service	Доставляє повідомлення через доступні канали (SMS, email, push-нотифікації)	User

«Компонент-відправник» є тим, хто ініціює взаємодію, «Компонент-отримувач» – тим, хто приймає або обробляє запит, а «Тип та зміст взаємодії» визначають, що саме передається між компонентами та з якою метою.

Зв’язок між компонентами системи зображений відповідно направленими пунктирними стрілочками, де стрілочка вказує на «Компонент-отримувач» з таблиці 4.3, а відповідний підпис має скороченене пояснення взаємодії з пункту таблиці 4.3 «Тип та зміст взаємодії».

4.1.5 Характеристики системи

Система «Order Management System» характеризується високою масштабованістю, оскільки завдяки горизонтальному масштабуванню через додавання нових інстансів мікросервісів, використанню контейнерів Docker і оркестрації Kubernetes вона здатна ефективно справлятися з ростом навантаження та забезпечує безперебійну роботу навіть під час пікових періодів. Висока доступність системи забезпечується завдяки реплікації баз даних, резервному копіюванню, механізмам відновлення, балансуванню навантаження через API Gateway і кешуванню на рівні сервісів (Redis), що мінімізує ризики простоїв і гарантує стабільний доступ до даних. Мікросервісна архітектура сприяє гнучкості розробки, дозволяючи командам незалежно розробляти, тестувати й розгортати сервіси, вибираючи для цього найбільш підходящі технології, такі як MS SQL Server для транзакцій або Redis для кешування. Висока продуктивність системи досягається через швидкий доступ до часто запитуваних даних через Redis, а також завдяки оптимізації маршрутизації запитів через Nginx як API Gateway, що дозволяє системі ефективно працювати навіть при великій кількости користувачів. Для забезпечення безпеки використовуються SSL/TLS для шифрування трафіку та перевірка автентифікації й авторизації на рівні API Gateway, що запобігає несанкціонованому доступу. Окрім того, система інтегрована з Prometheus і Grafana для моніторину в реальному часі, а також використовує централізоване логування через Elastic Stack.

4.1.6 Інтеграція компонентів

У систему інтегровано кілька ключових компонентів, кожен з яких виконує важливу роль у забезпеченні функціональности та ефективности. API Gateway використовується для маршрутизації запитів і централізованого управління доступом, забезпечуючи контроль і координацію взаємодії між клієнтами та мікросервісами. RabbitMQ служить для асинхронної обробки повідомлень, що дозволяє мікросервісам обмінюватися даними безпосередньо, знижуючи залежності між ними. Redis забезпечує високошвидкісне кешування даних, підвищуючи швидкість доступу до часто використовуваної інформації. Зовнішні сервіси, такі як SMTP, Twilio і FCM, інтегровані для багатоканальної доставки повідомлень, що охоплює електронну пошту, SMS і push-сповіщення. Крім того, система включає захищені механізми автентифікації та передачі даних, які гарантують високий рівень безпеки та захисту конфіденційної інформації.

4.1.7 Безпека, моніторин і керування доступом

Система забезпечує високий рівень безпеки, моніторину та керування доступом завдяки впровадженню таких механізмів, як авторизація та автентифікація з використанням OAuth 2.0 та OpenID Connect для захисту доступу до ресурсів через JWT токени, що гарантує надійний контроль доступу. Для чіткого визначення прав доступу до функціональних частин системи здійснюється розподіл ролей користувачів, таких як адміністратор, кур’єр та користувач. Шифрування даних здійснюється як під час збереження, так і передачі, з використанням TLS для API та AES для даних у сховищі, що забезпечує захист конфіденційности інформації. Продуктивність системи контролюється за допомогою моніторину через Prometheus і Grafana, що дає змогу відстежувати стан системи та ключові показники в реальному часі. Логування організовано через Serilog для .NET Core мікросервісів

або вбудовані механізми логування в СУБД, з подальшою обробкою через Filebeat чи Logstash для фільтрації логів за рівнем, типом подій або іншими параметрами, а також передачею їх до Elasticsearch, що дозволяє швидко шукати й аналізувати журнали та діагностувати проблеми.

4.1.8 Оптимізація роботи системи

Для забезпечення високої продуктивности та ефективного використання ресурсів системи застосовуються різноманітні методи обробки даних, кешування та тестування. Одним із таких методів є кешування HTTP-запитів через Nginx, що дозволяє знизити час відповіді та навантаження на бекенд. Для зберігання часто змінюваних даних, таких як статуси замовлень і заявок, використовуються високопродуктивні кеші на основі Redis. Система також здатна автоматично адаптуватися до змін навантаження завдяки горизонтальному масштабуванню через оркестрацію Kubernetes, що дозволяє швидко збільшувати кількість інстансів за потребою. Щоб прискорити операції вибірки, застосовується індексація даних у базах даних, що дозволяє швидше обробляти запити навіть за великих обсягів інформації. Крім того, для зменшення навантаження бази даних та скорочення часу відповіді використовуються кешовані результати запитів у Redis. Для забезпечення стабільности роботи системи при пікових навантаженнях запити лімітуються через API Gateway, що обмежує кількість запитів від окремих клієнтів. Система включає також модулі автоматичного тестування, що забезпечують її надійність та стабільність. Серед основних типів тестування – юнит-тести для перевірки роботи окремих мікросервісів, інтеграційні тести для перевірки взаємодії між компонентами, навантажувальні тести для оцінки продуктивности під час роботи з великими обсягами запитів та тести стійкости для перевірки здатности системи витримувати збої в окремих компонентах або відмову сервісів.

Ці заходи дозволяють забезпечити стабільність і високу продуктивність системи, а також знизити ризики, пов’язані з високим навантаженням, зберігаючи високу доступність і ефективність обробки даних.

4.1.9 Документація

Кожен компонент системи супроводжується документацією, створеною за допомогою Swagger для API та Graphviz для мікросервісів.

4.2 Розробка елементів алгоритмічної системи забезпечення

У процесі розробки алгоритмічної системи забезпечення було використане математичне моделювання. Математична модель була виділена та упорядкована до алгоритму [9]. У поточному розділі зазначений детальний огляд функціональности та алгоритмів, які забезпечують роботу системи за рахунок представленої загальної схеми алгоритму роботи системи на прикладі вебзастосунку, а також схеми алгоритмів окремих модулів. Загальна схема алгоритму роботи вебзастосунку направлена на розуміння логіки та взаємозв’язків між різними компонентами вебзастосунку. Крім загальної схеми алгоритму роботи вебзастосунку, в розділі наведені схеми алгоритмів окремих модулів системи, кожен з яких виконує конкретні функції. Схеми алгоритмів модулів дозволять детальніше розібрати принципи їх роботи, взаємодію з іншими компонентами системи.

На рисунку 4.2 зображена схема алгоритму роботи вебзастосунку, на рисунках 4.3, 4.4, 4.5, 4.6, 4.7, 4.8, 4.9, 4.10, 4.11, 4.12, 4.13, 4.14, 4.15, 4.16 зображені схеми алгоритмів модулів.

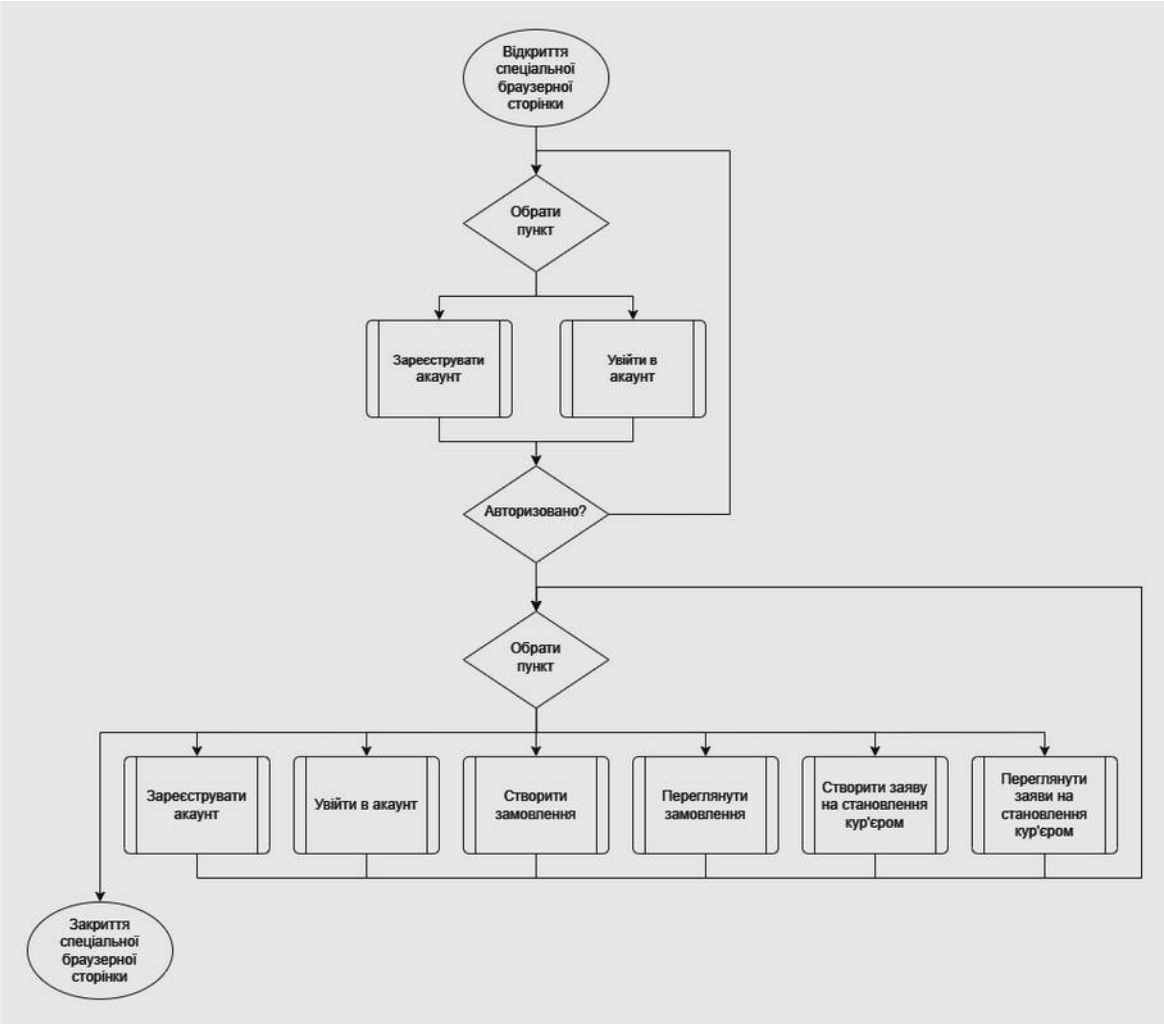

Рисунок 4.2 – Схема алгоритму роботи вебсайту

Схема алгоритму роботи вебсайту передбачає взаємодію користувача із інтерфейсом, який надсилає запити до API. API обробляє ці запити, звертаючись до відповідних мікросервісів, які виконують конкретні завдання, такі як обробка даних, автентифікація тощо. Результати опрацьовуються і повертаються у вигляді відповіді, яку відображає вебсайт для користувача.

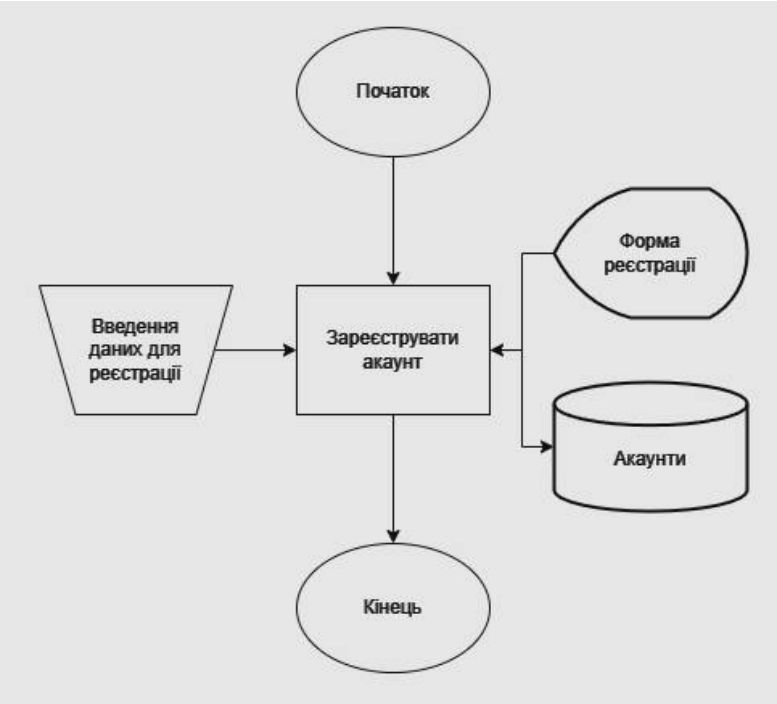

Рисунок 4.3 – Схема алгоритму модулю «Зареєструвати акавнт»

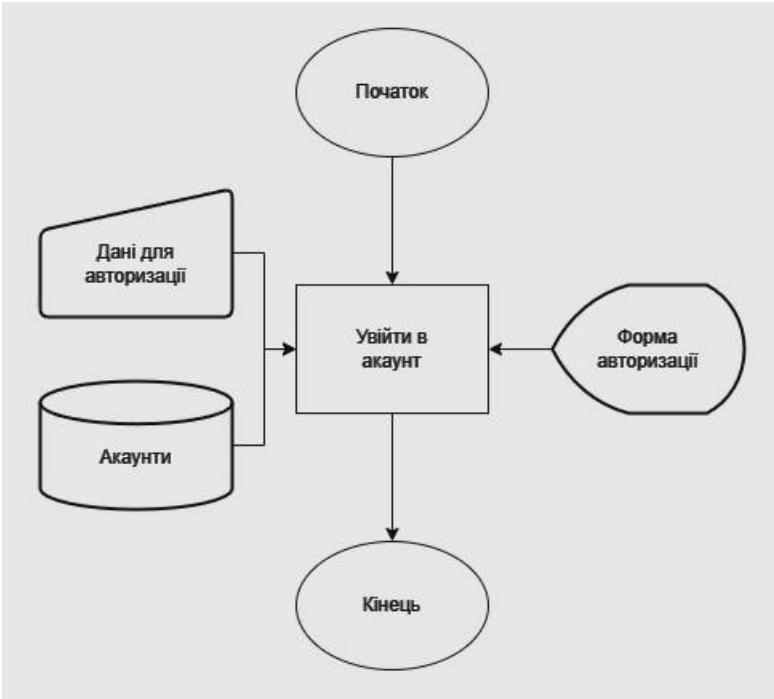

Рисунок 4.4 – Схема алгоритму модулю «Увійти в акавнт»

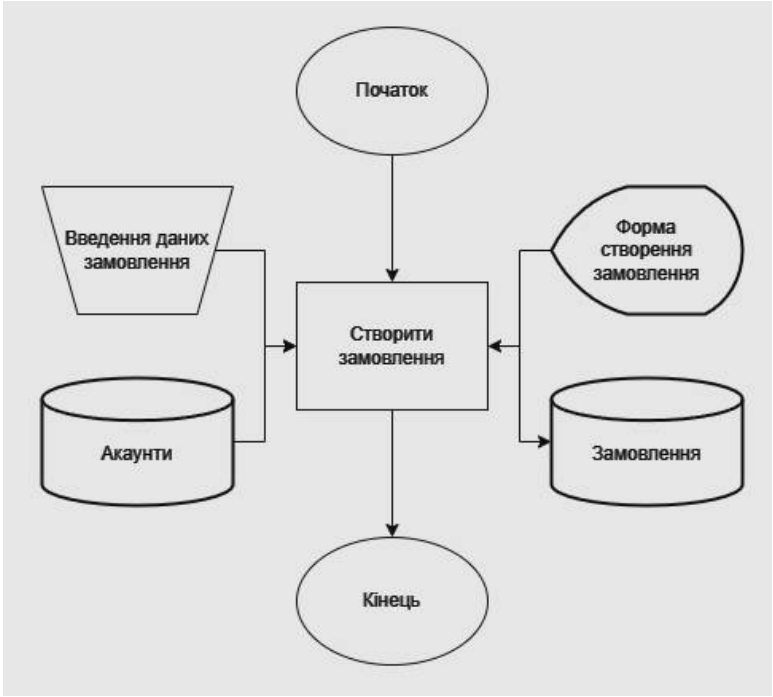

Рисунок 4.5 – Схема алгоритму модулю «Створити замовлення»

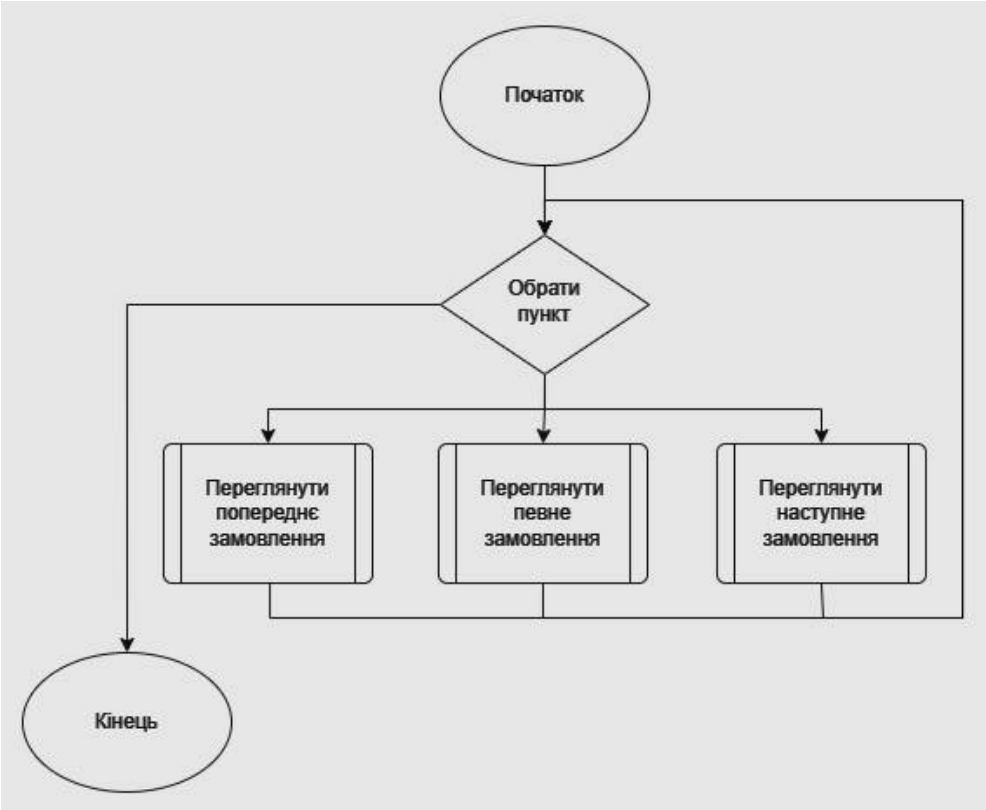

Рисунок 4.6 – Схема алгоритму модулю «Переглянути замовлення»

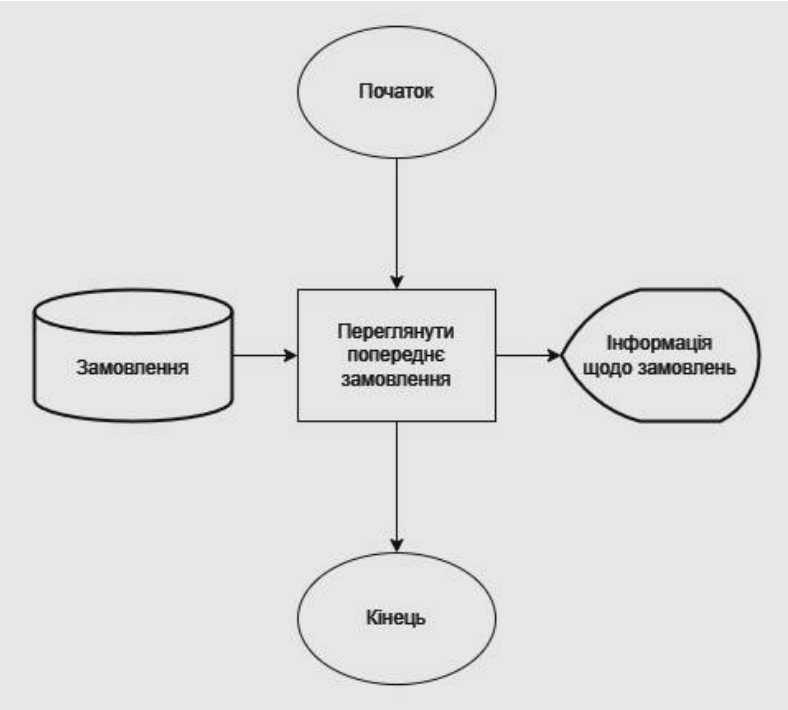

Рисунок 4.7 – Схема алгоритму модулю «Переглянути попереднє замовлення»

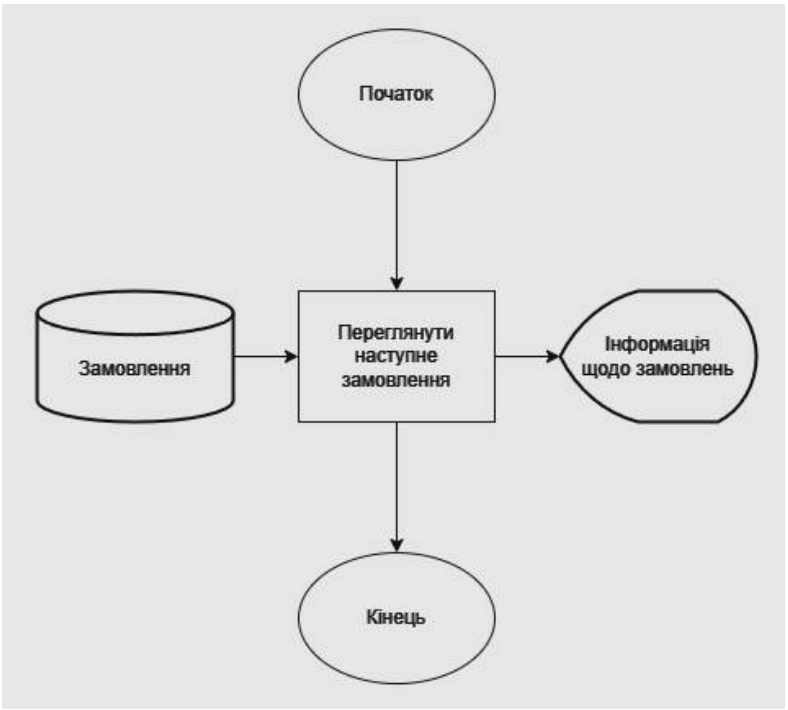

Рисунок 4.8 – Схема алгоритму модулю «Переглянути наступне замовлення»

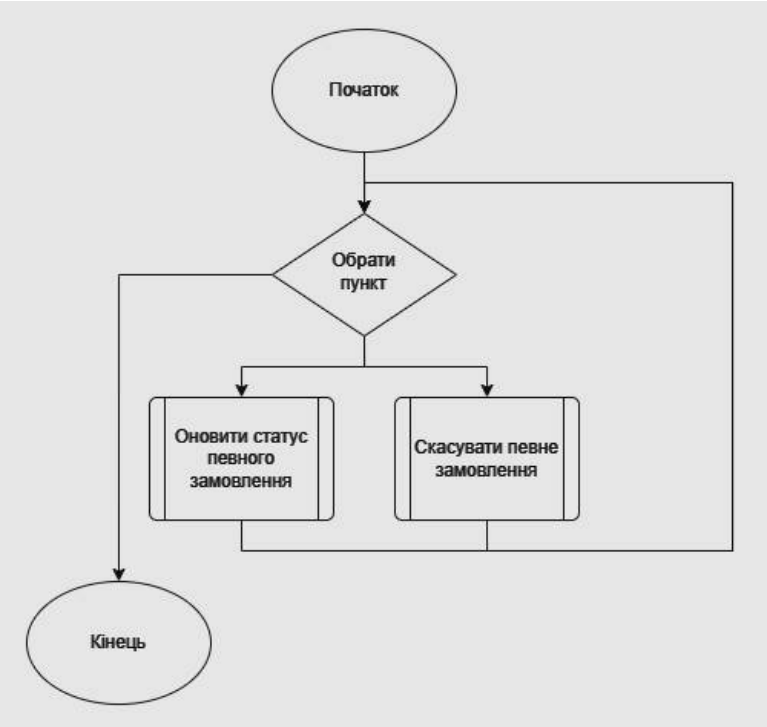

Рисунок 4.9 – Схема алгоритму модулю «Переглянути певне замовлення»

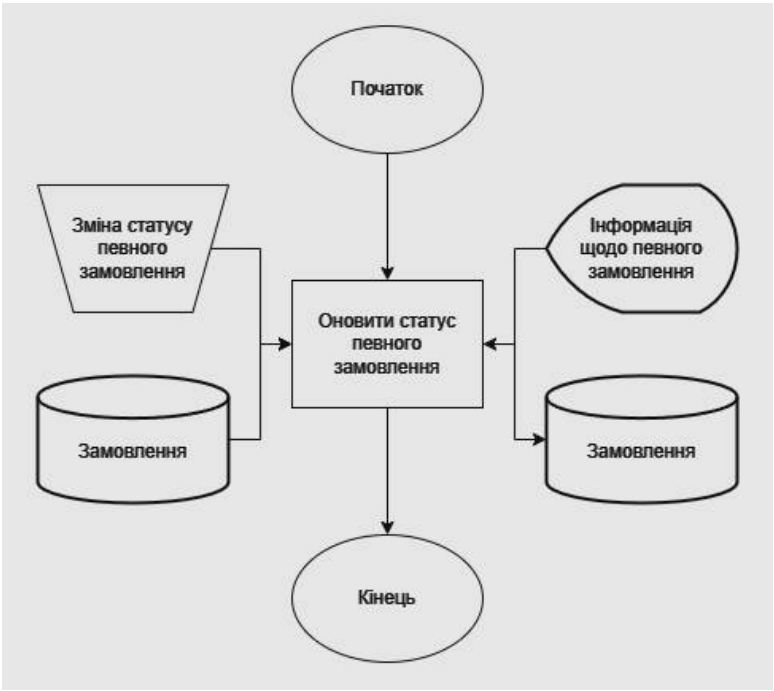

Рисунок 4.10 – Схема алгоритму модулю «Оновити статус певного замовлення»

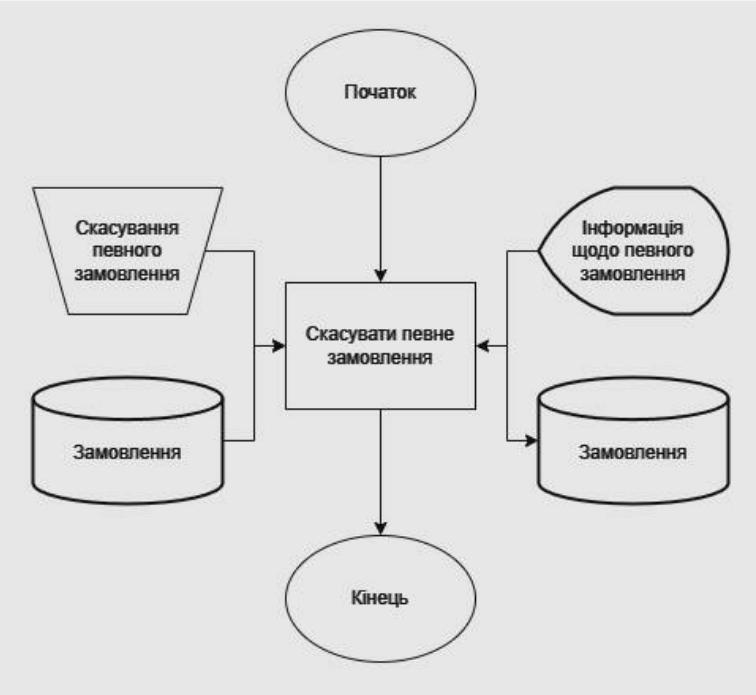

Рисунок 4.11 – Схема алгоритму модулю «Скасувати певне замовлення»

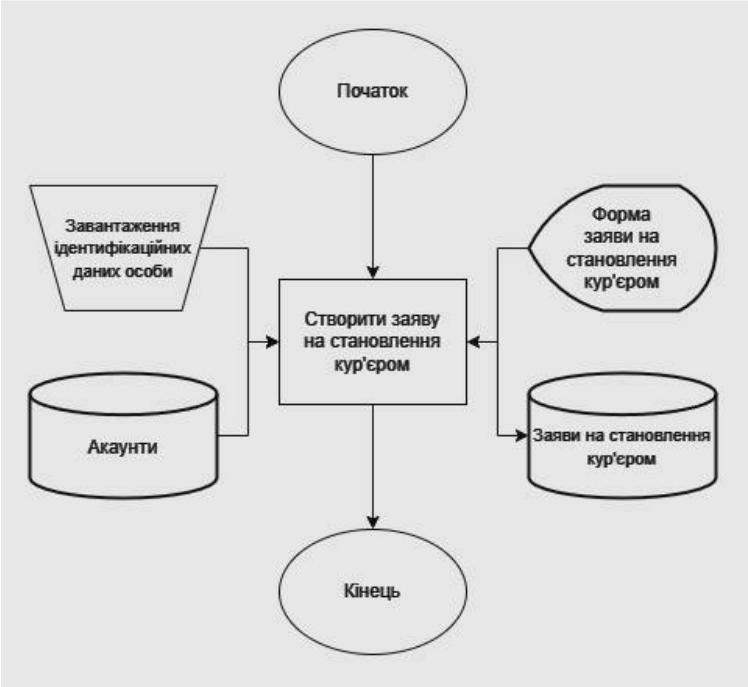

Рисунок 4.12 – Схема алгоритму модулю «Створити заяву на становлення кур’єром»

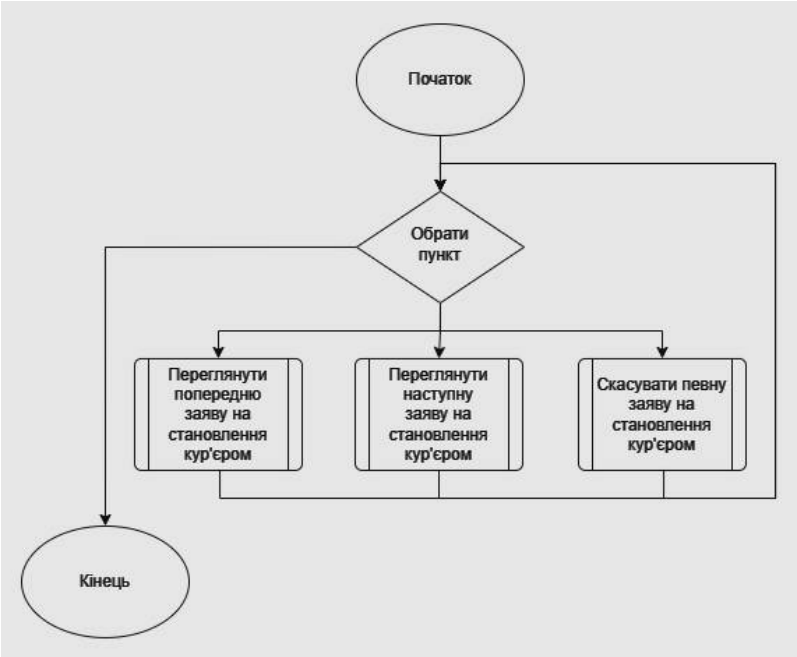

Рисунок 4.13 – Схема алгоритму модулю «Переглянути заяви на становлення кур’єром»

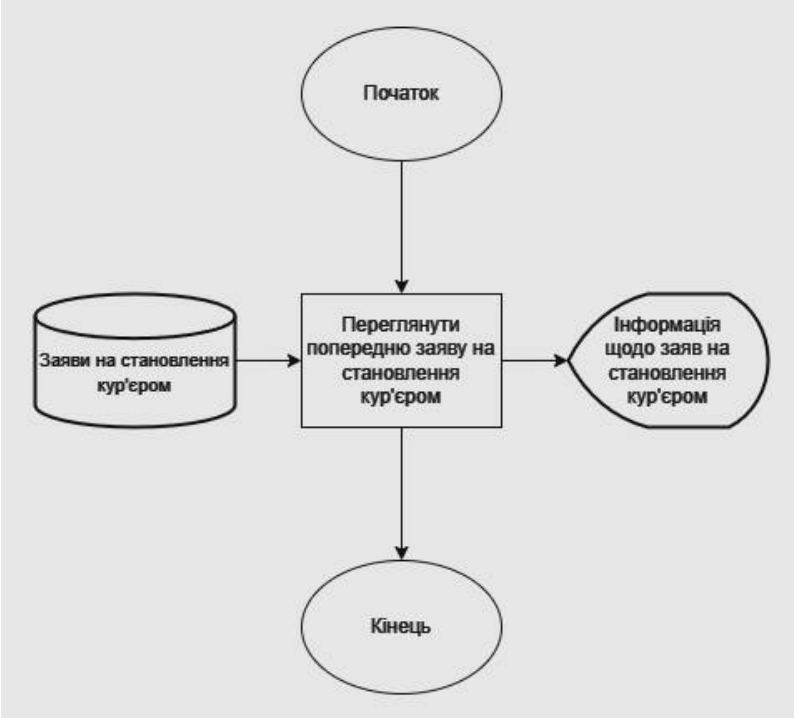

Рисунок 4.14 – Схема алгоритму модулю «Переглянути попередню заяву на становлення кур’єром»

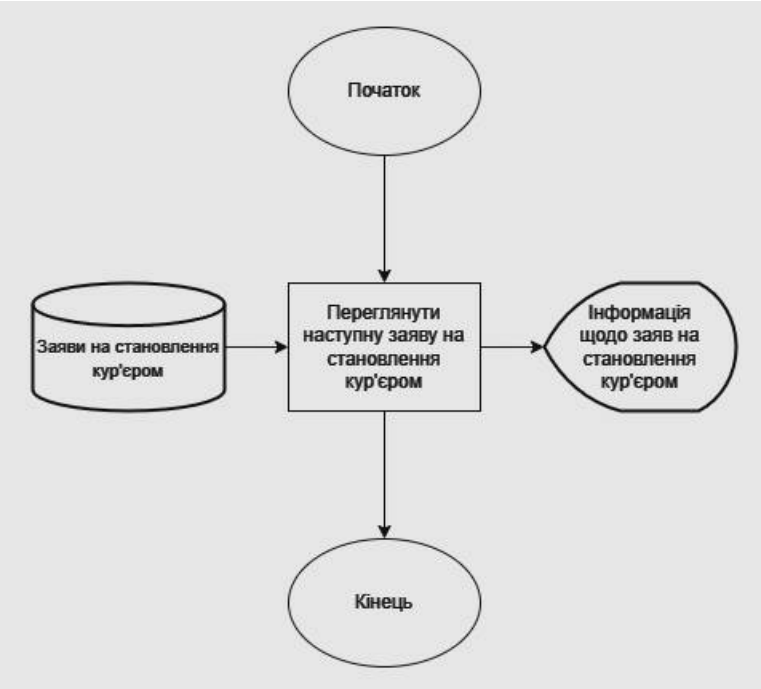

Рисунок 4.15 – Схема алгоритму модулю «Переглянути наступну заяву на становлення кур’єром»

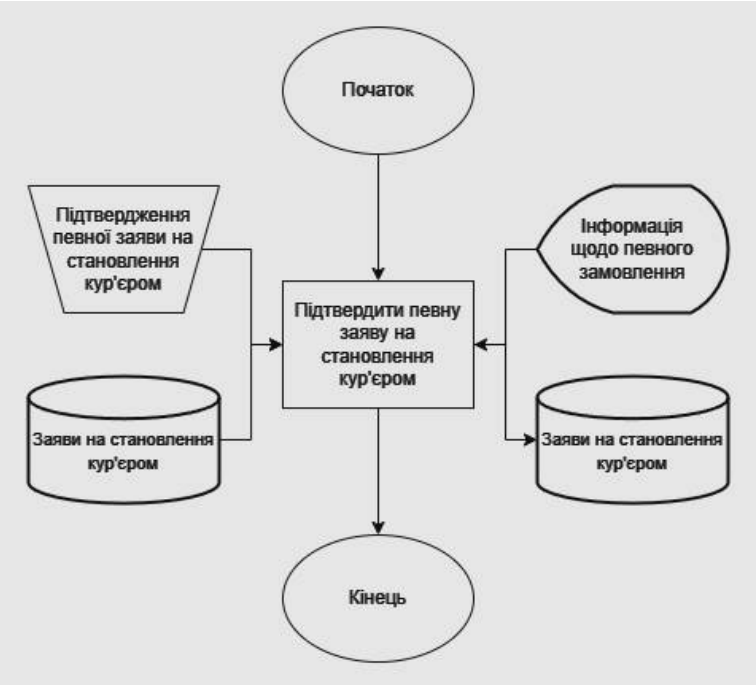

Рисунок 4.16 – Схема алгоритму модулю «Підтвердити певну заяву на становлення кур’єром»

4.3 Розробка елементів дизайнової системи забезпечення

Для розробки елементів дизайнової системи забезпечення було використано популярний інструмент для розробки дизайну і прототипування вебинтерфейсів Figma [10]. Він надає можливості для створення дизайну, співпраці розробників та дизайнерів, а також експорту готових дизайнів для подальшої розробки.

Структура сторінки, зображеної на рисунку 8.1, передбачає чіткий розподіл на кілька основних частин. У верхній частині сторінки, гедері, розташований логотип. В нижній частині, футері, користувач може знайти розділ зворотного зв’язку. Змістовна частина сторінки, мейну, поділяється на ліву та праву частини. У правій частині знаходиться промо-секція з новинами, оновленнями чи спеціальними пропозиціями. Ліва частина мейну відведена для форми реєстрації.

На рисунку 4.17 зображений дизайн вебсторінки з формою реєстрації.

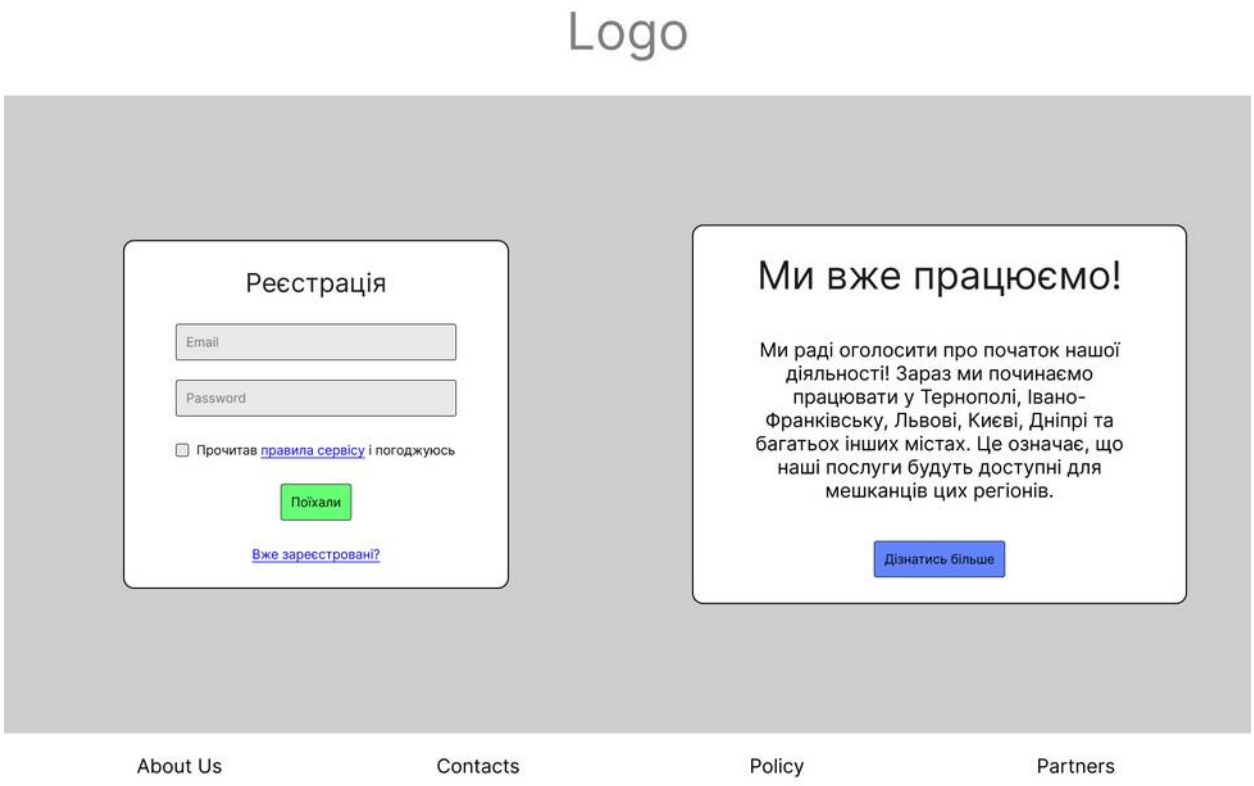

Рисунок 4.17 – Дизайн вебсторінки з формою реєстрації

Структура сторінки, що зображена на рисунку 4.17 залишається такою ж, як і у попередній, за винятком лівої частини мейну: замість форми реєстрації розташована форма авторизації, де зареєстровані користувачі можуть ввести свої облікові дані, такі як ім’я користувача та пароль, для входу в систему.

На рисунку 4.18 зображений дизайн вебсторінки з формою авторизації.

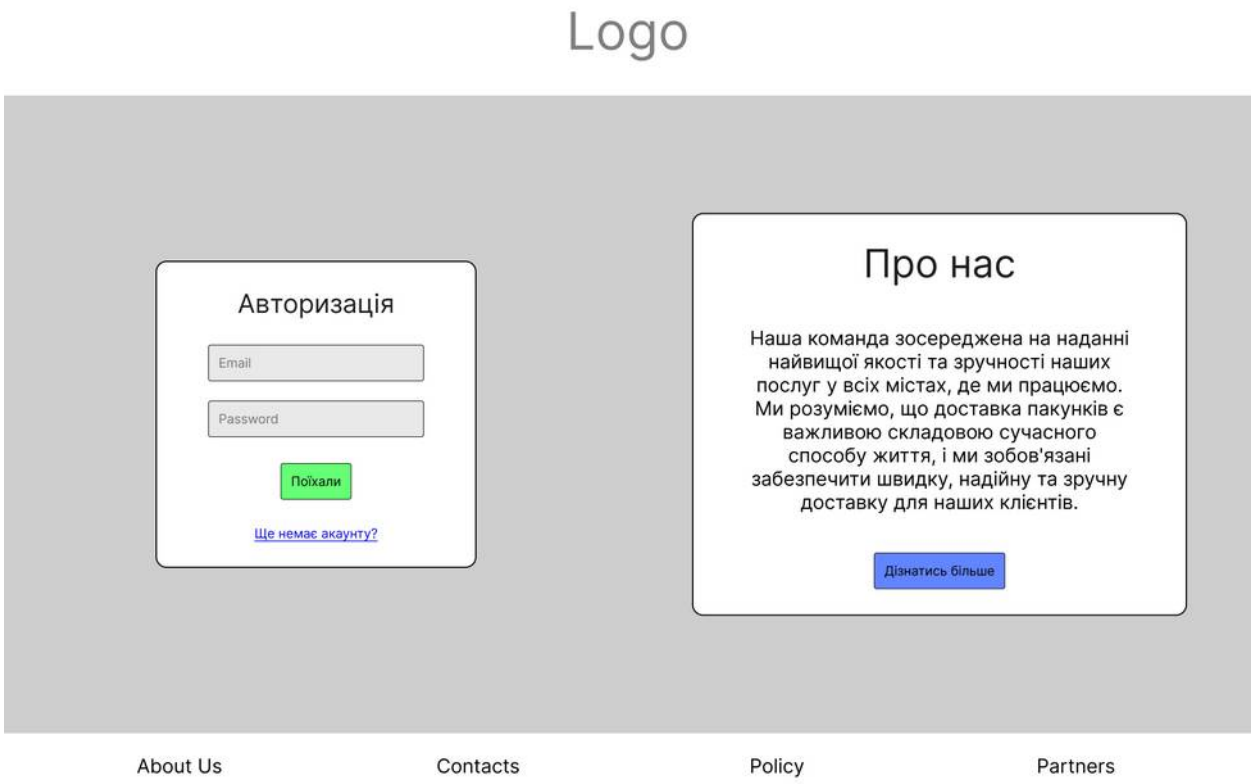

Рисунок 4.18 – Дизайн вебсторінки з формою авторизації

Структура вебсторінки, зображеної на рисунку 4.18, зазнає змін, що стосуються як розташування елементів на сторінці, так і їхнього змісту. Гедер сторінки тепер поділяється на ліву та праву частини. У лівій частині розміщується логотип, який представляє назву сервісу, а в правій частині знаходяться кнопки для переходу до певних вебсторінок, таких як «Увійти в акавнт», «Стати кур’єром» та інші. Ліва частина мейну тепер відведена для форми створення замовлення. Ця форма включає поля для вводу даних замовлення, таких як адреса пакунка, адреса доставки,

вартість замовлення, опис замовлення та контакти для зв’язку, а також містить поля для введення даних картки для оплати.

На рисунку 4.19 зображений дизайн вебсторінки з формою створення замовлення.

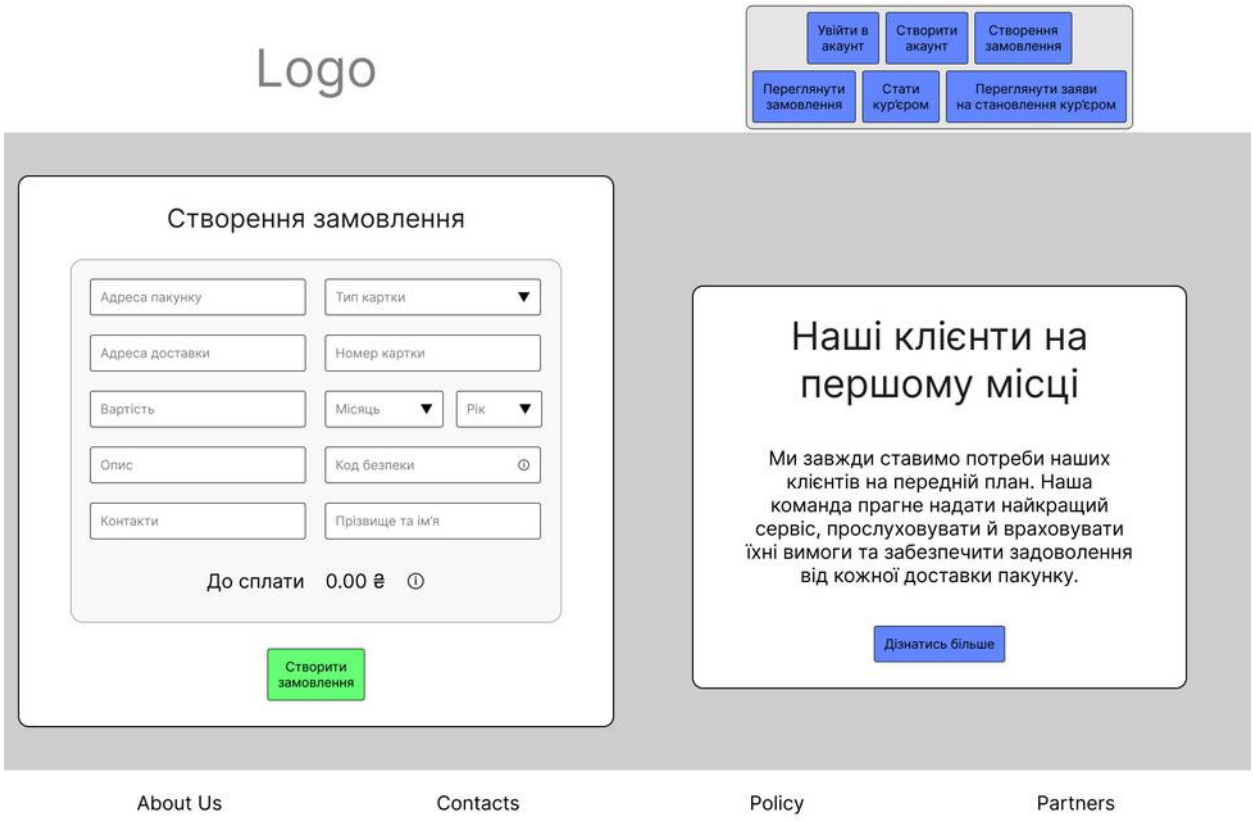

Рисунок 4.19 – Дизайн вебсторінки з формою створення замовлення

Структура вебсторінки, що зображена на рисунку 4.19, у лівій частині мейну містить інформацію щодо замовлення з можливістю створити замовлення.

На рисунку 4.20 зображений дизайн вебсторінки зі списком замовлень.

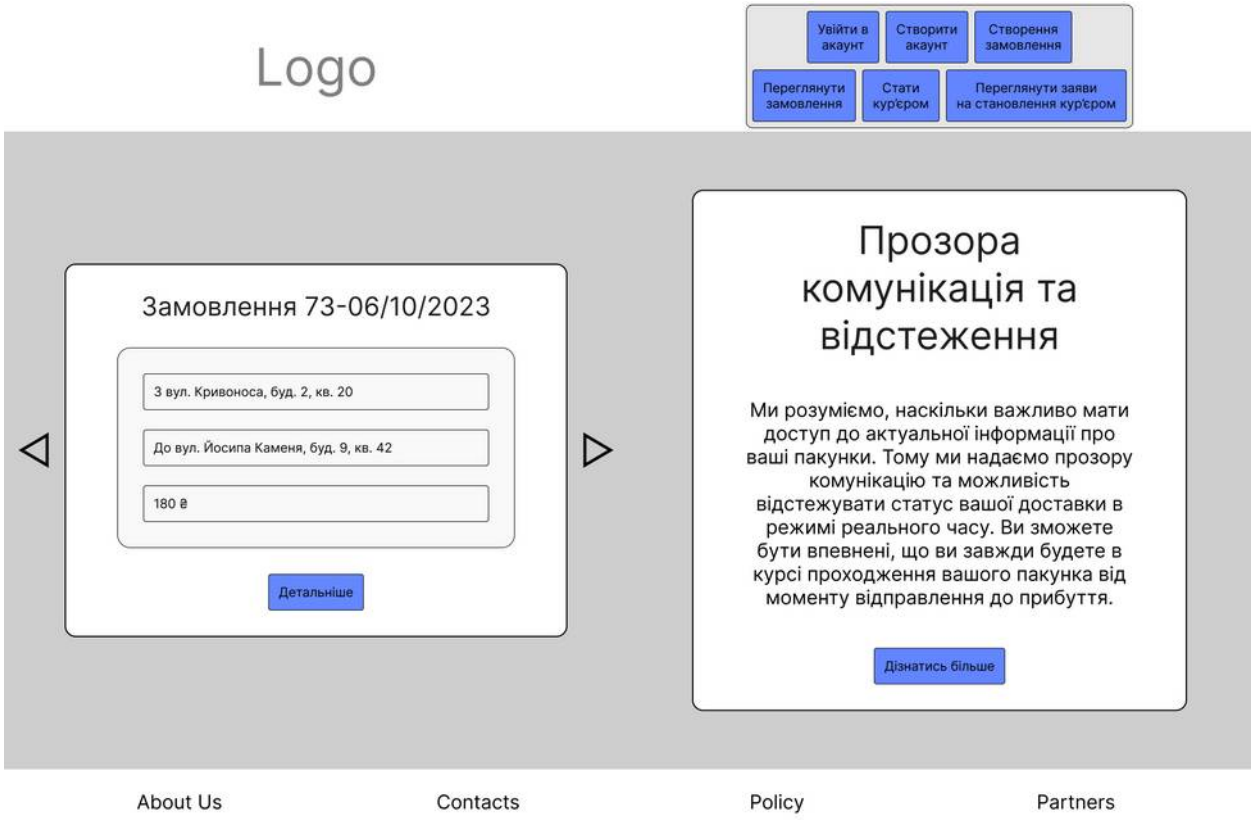

Рисунок 4.20 – Дизайн вебсторінки зі списком замовлень

Структура вебсторінки, що зображена на рисунку 4.20, у лівій частині мейну містить коротку інформацію щодо замовлення з можливістю детальніше ознайомитись із ним.

На рисунку 4.21 зображений дизайн вебсторінки оновленням статусу певного замовлення.

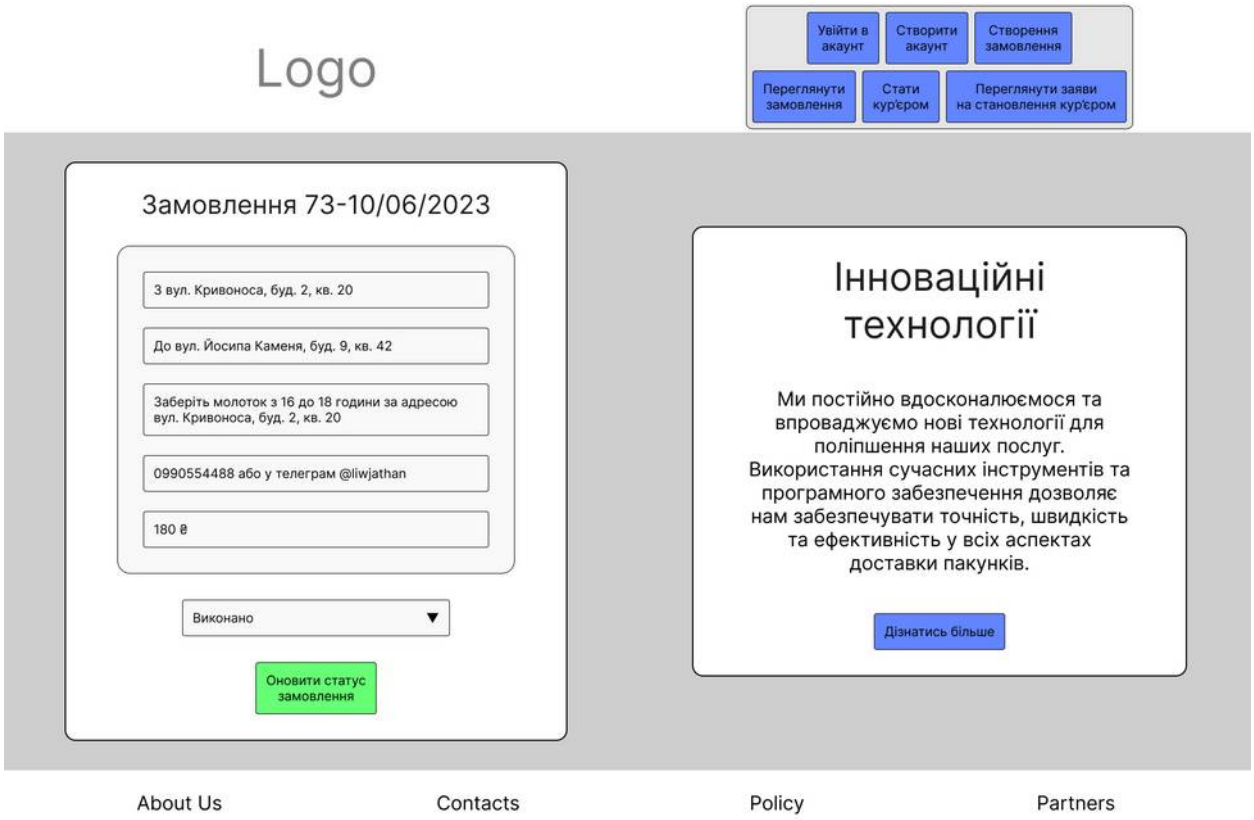

Рисунок 4.21 – Дизайн вебсторінки з оновленням статусу певного замовлення

Структура вебсторінки, що зображена на рисунку 4.21, у лівій частині мейну містить детальну інформацію щодо замовлення з можливістю оновити статус замовлення.

На рисунку 4.22 зображений дизайн вебсторінки зі скасуванням певного замовлення.

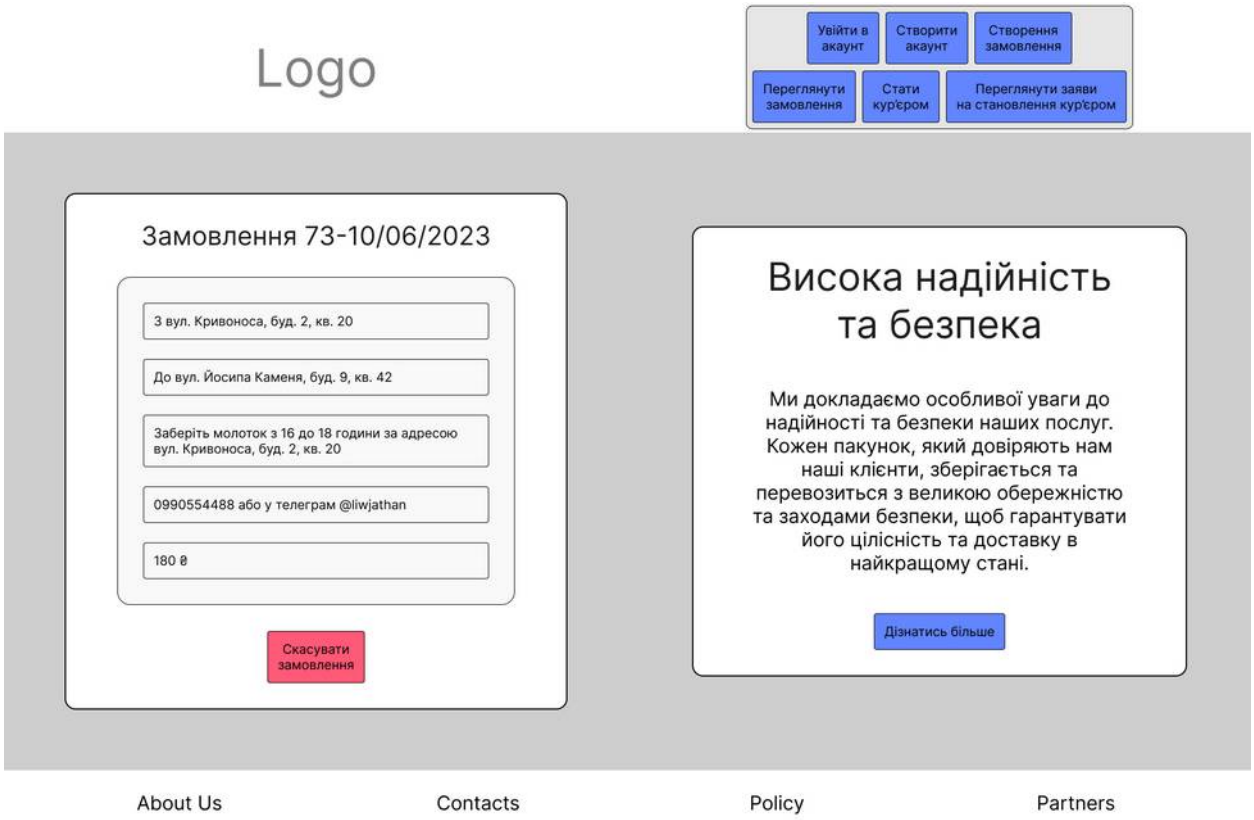

Рисунок 4.22 – Дизайн вебсторінки зі скасуванням певного замовлення

Структура вебсторінки, що зображена на рисунку 4.22, у лівій частині мейну містить детальну інформацію щодо замовлення з можливістю скасувати замовлення.

На рисунку 4.23 зображений дизайн вебсторінки з формою заяви на становлення кур’єром.

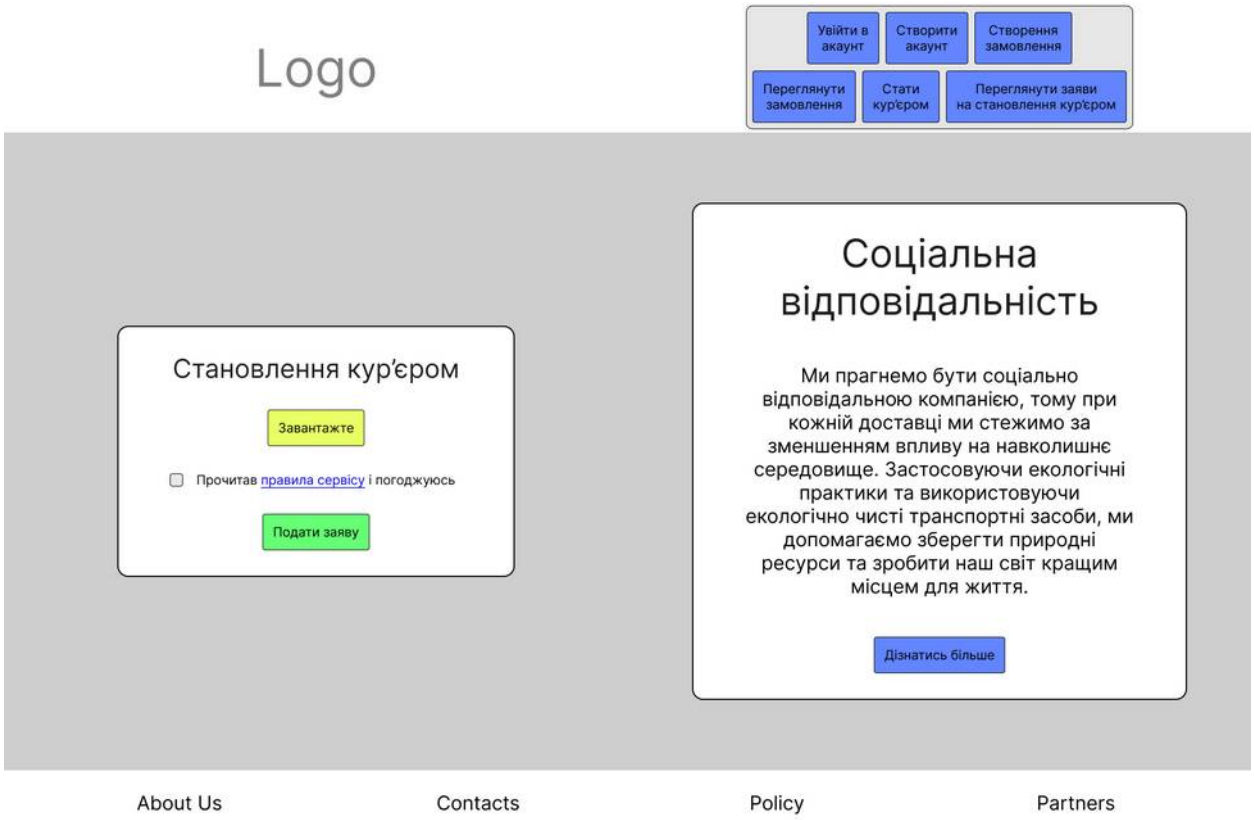

Рисунок 4.23 – Дизайн вебсторінки з формою заяви на становлення кур’єром

Структура вебсторінки, що зображена на рисунку 4.23, у лівій частині мейну містить форму для становлення кур’єром з можливістю подати заяву.

На рисунку 4.24 зображений дизайн вебсторінки зі списком заяв на становлення кур’єром.

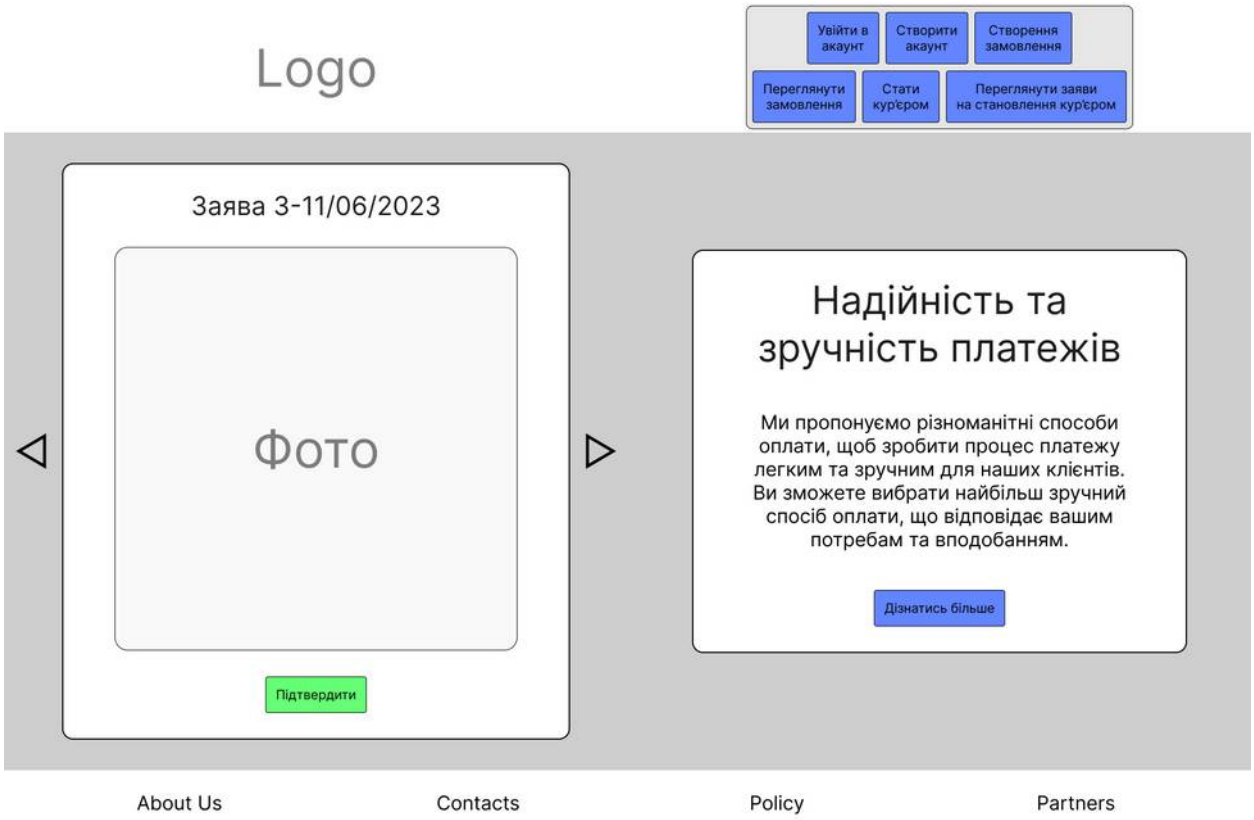

Рисунок 4.24 – Дизайн вебсторінки зі списком заяв на становлення кур’єром

Структура вебсторінки, що зображена на рисунку 4.24, у лівій частині мейну містить заяви на становлення кур’єром з можливістю їх підтвердження.

4.4 Рекомендації щодо елементів програмної системи забезпечення

Для розробки бекенду рекомендовано використовувати високопродуктивний, крос-платформенний фреймворк .NET Core, котрий дозволяє розробляти потужні та масштабовані вебзастосунки. Він надає багато можливостей для створення вебсерверів, API та бізнес-логіки, а також включає в себе широкий набір бібліотек і інструментів для ефективної розробки. Використання .NET Core дозволить розробити потужний та масштабований бекенд.

Для розробки фронтенду рекомендовано використовувати Vue.js, котрий є прогресивним фреймворком JavaScript для розробки користувацького інтерфейсу . Він дозволяє створювати динамічні та інтерактивні вебсторінки зі зручними компонентами, реактивними оновленнями та простою інтеграцією. Vue.js має легку вагу, добре документований та має велику спільноту розробників. Використання Vue.js дозволить створити сучасний та ефективний фронтенд.

Використання цих інструментів забезпечить потужний інструментарій для розробки програмної системи забезпечення, що включає надійну базу даних, ефективний бекенд, сучасний фронтенд та дизайн, що задовольняє потреби користувачів.

4.5 Рекомендації щодо захисту інформації системи

Враховуючи важливість захисту даних у сучасному цифровому середовищі, було розроблено комплекс заходів, спрямованих на забезпечення високого рівня безпеки інформації, яка обробляється та зберігається в системі. Захист інформації є важливим аспектом будь-якої інформаційної системи, зокрема для вебзастосунків, де дані передаються через мережу інтернет. Для цього, серед іншого, застосовуються різні методи захисту, що включають шифрування даних, автентифікацію користувачів та обмеження доступу на різних рівнях системи.

Одним із основних елементів захисту є приховування символів при введенні пароля. Це забезпечує збереження конфіденційности пароля під час введення його користувачем у полі для автентифікації. Крім того, для підвищення рівня безпеки введення пароля, система встановлює вимоги до його складності: пароль має включати літери верхнього та нижнього регистеру, цифри та спеціальні символи, а його мінімальна довжина повинна складати не менше 10 символів. Це забезпечує надійний захист від атак на основі підбору паролів (brute-force).

Ще одним важливим аспектом є контроль доступу до бази даних, що містить конфіденційну інформацію. Доступ до бази даних обмежений лише для адміністраторів системи. Всі з’єднання між графічним інтерфейсом користувача вебзастосунку та базою даних передбачають автентифікацію за допомогою унікальних логінів та паролів, що були встановлені під час реєстрації користувача. Лише після успішної автентифікації, адміністратор може отримати доступ до конфіденційної інформації в базі даних, що значно знижує ризик несанкціонованого втручання.

ВИСНОВКИ

У кваліфікаційній роботі було розроблено модель мікросервісної архітектури інформаційної системи та елементи забезпечення задля підвищення масштабованости ІТ-сервісу обслуговування замовлень.

На початковому етапі було виконано аналіз предметної області, що дозволило визначити основні виклики, зокрема низьку гнучкість монолітної архітектури, труднощі масштабування та інтеграції нових функцій, що ускладнювали реалізацію підтримки різноманітних платформ, включно зі смарт-пристроями. Ці проблеми стали основою для формулювання задачі роботи, яка полягала в аналізі предметної області, розгляді існуючих архітектурних підходів, обґрунтовуванні вибіру мікросервісної архітектури як оптимальної для підвищення масштабованости системи та розробці моделі мікросервісної архітектури інформаційної системи, елементів алгоритмічної та дизайнової систем забезпечення, а також рекомендацій щодо елементів програмної системи забезпечення.

В межах поглибленого аналізу існуючих архітектур інформаційних систем було досліджено монолітну, сервісно-орієнтовану та мікросервісну архітектури. Здійснено порівняння за ключовими параметрами, такими як: гнучкість, масштабованість, продуктивність і складність впровадження. Мікросервісна архітектура була визнана найефективнішим вибором завдяки її децентралізованости, незалежности компонентів та адаптивности до змін.

Детально описано елементи існуючих систем забезпечення об’єкта дослідження, а саме: схема організаційної структури, IDEF0 та ERD діаграми. Це дало змогу структурувати модель даних та підготувати основу для переходу до мікросервісної архітектури.

Завдяки описаним елементам існуючих систем забезпечення були розроблені модель мікросервісної архітектури інформаційної системи, елементи алгоритмічної

та дизайнової систем забезпечення, що включали: діаграму рівня Container за моделлю C4, опис компонентів системи, її взаємодій, характеристик, інтегрованих компонентів, елементів безпеки, моніторину і керування доступом, оптимізації роботи системи, документації, схеми алгоритмів роботи вебсайту, дизайн вебсайту, а також надані рекомендації щодо елементів програмної системи забезпечення та щодо захисту інформації системи.

Таким чином, результати роботи демонструють ефективність переходу на мікросервісну архітектуру для вирішення проблем масштабованости інформаційної системи. Розроблене рішення дозволило не лише забезпечити необхідну масштабованість системи, а й швидко адаптувати її до змін ринку та вимог користувачів. Ці результати можуть бути застосовані в інших проєктах, спрямованих на розробку сучасних інформаційних систем для обслуговування замовлень та інших бізнес-процесів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Evolve the monolith to microservices with java and node / S. D. Santis та ін. International Technical Support Organization, 2016. 134 с.

2. Sanctis V. D. ASP. NET Core Web API. Manning Publications Co. LLC, 2023.

3. Pairan R. The Road to Microservice Architecture. 2019.

4. Howell-Barber H., Lawler J. P. Service-Oriented Architecture. Taylor & Francis Group, 2019.

5. Типи організаційних структур управління підприємством. URL: https://buklib.net/books/32066/.

6. Методологія IDEF0. URL: https://stud.com.ua/87184/ekonomika/metodologiya_idef0.

7. Модель діаграми зв’язків сутностей (ER) із прикладом СУБД. URL: https://www.guru99.com/uk/er-diagram-tutorial-dbms.html.

8. The C4 model for visualising software architecture. URL: https://c4model.com/.

9. Основи алгоритмізації: Блок-схеми алгоритму. URL: https://yevshan.com.ua/info/006/content/content3.html.

10. Рай для дизайнера або що таке Figma. URL: https://kukurudza.com/blog/shho-take-figma/.

11. Методичні вказівки до організації виконання та захисту кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти спеціальності 123 Комп'ютерна інженерія за освітньою програмою «Комп'ютерні інтелектуальні технології» для студентів усіх форм навчання / Упоряд.: О.Г. Руденко, Н.М. Сердюк. Харків: ХНУРЕ, 2022. 55 с.

12. ДСТУ 3008:2015. Інформація та документація. Звіти у сфері науки і техніки. Структура та правила оформлювання, Чинний від 22.06.2015. Київ: ДП «УкрНДНЦ», 2016, 26 с.

13. ДСТУ 8302:2015. Бібліографічне посилання. Загальні положення та правила складання. / Видання офіційне. – К.: ДП «УкрНДНЦ», 2016 – 20 с.

14. Куренков Б. М. The main advantages of microservice architecture. *Здобутки та досягнення прикладних та фундаментальних наук XXI століття*: матеріали міжнар. наук. конф., м. Черкаси, 8 груд. 2023 р. Вінниця, 2023. С. 249–250.