МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра комп’ютерної інженерії

Твердун Ярослав Олегович

Програмний модуль розподіленої системи обміну даними в реальному часі / Software module of a distributed real-time data exchange system

спеціальність: 123 – Комп’ютерна інженерія
освітньо-професійна програма – Комп’ютерна інженерія

Випускна кваліфікаційна робота

Виконав: студент групи КІ-42
Твердун Ярослав Олегович

Науковий керівник
Н. Я. Савка

2025

АНОТАЦІЯ

Твердун Я.О. Програмний модуль розподіленої системи обміну даними в реальному часі. – Рукопис.

Кваліфікаційна робота на здобуття освітнього ступеня «бакалавр» за спеціальністю 123 «Комп'ютерна інженерія», освітньо-професійна програма. Західноукраїнський національний університет, Тернопіль, 2025.

Робота написана на 89 сторінках і містить 21 рисунок, 6 таблиць, 5 додатків і 36 джерел за переліком посилань. Обсяг графічного матеріалу — 2 аркуші формату А3.

Метою кваліфікаційної роботи є розробка програмного модуля розподіленої системи обміну даними в реальному часі у вигляді мобільного клієнт-серверного застосунку для замовлення та доставки їжі.

Методи дослідження включають аналіз і синтез, проектування архітектури системи, розробку бази даних, моделювання HTTP-взаємодії та тестування.

У роботі описано архітектуру клієнт-серверної системи, розроблено інтерфейс користувача, реалізовано передачу даних між клієнтом і сервером за протоколом HTTP. Серверну частину створено на основі фреймворку Flask з використанням SQLAlchemy та бази даних SQLite. Клієнтську частину реалізовано за допомогою Flutter.

Практичне значення роботи полягає у створенні кросплатформного застосунку для автоматизації процесу замовлення та доставки їжі з підтримкою передачі даних у реальному часі.

Ключові слова: МОБІЛЬНИЙ ЗАСТОСУНОК, FLUTTER, PYTHON, FLASK, SQLALCHEMY, КЛІЄНТ-СЕРВЕР, ДОСТАВКА ЇЖІ.

ANNOTATION

Tverdun Y.O. Software Module of a Distributed Real-Time Data Exchange System. – Manuscript.

Qualification work for the Bachelor's degree in specialty 123 “Computer Engineering”, educational and professional program. Western Ukrainian National University, Ternopil, 2025.

The work is written on 89 pages and contains 21 figures, 6 tables, 5 appendices, and 36 references. The volume of graphic material is 2 sheets of A3 format.

The purpose of the qualification work is to develop a software module of a distributed real-time data exchange system implemented as a mobile client-server application for food ordering and delivery.

The research methods include analysis and synthesis, system architecture design, database development, HTTP communication modeling, and testing.

The work presents the architecture of the client-server system, development of the user interface, implementation of data transmission via HTTP protocol. The server part was built using the Flask framework with SQLAlchemy and an SQLite database, and the client part was developed using Flutter.

The practical value lies in the development of a cross-platform application that automates the process of food ordering and delivery with real-time data exchange support.

Keywords: MOBILE APPLICATION, FLUTTER, PYTHON, FLASK, SQLALCHEMY, CLIENT-SERVER, FOOD DELIVERY.

ЗМІСТ

ВСТУП

У сучасному світі, де мобільні технології охопили практично всі сфери діяльності людини, значну увагу приділяють автоматизації процесів обслуговування та взаємодії між користувачем і бізнесом. Особливої популярності набули мобільні застосунки, що дозволяють спростити виконання повсякденних задач, зокрема – замовлення товарів і послуг. Доставка їжі є одним із найбільш актуальних напрямів, адже вона надає користувачам швидкий та зручний доступ до широкого вибору страв і дозволяє оформити замовлення без необхідності відвідувати ресторан чи кафе.

Значний попит на такі сервіси пояснюється зростаючими вимогами до комфорту, швидкості та персоналізації обслуговування. Мобільний застосунок виступає в цьому контексті ефективним інструментом, що поєднує гнучкий інтерфейс користувача, можливість інтерактивної взаємодії із сервером, зберігання історії замовлень та інші функції, які сприяють покращенню користувацького досвіду. Водночас, для розробника постають задачі щодо вибору відповідного середовища розробки, забезпечення стабільності роботи застосунку, реалізації логіки обробки даних та побудови надійної архітектури клієнт-серверної взаємодії.

Таким чином, розробка мобільного додатку для замовлення їжі з інтеграцією серверної логіки, використанням бази даних та реалізацією сучасного інтерфейсу є актуальною задачею, що поєднує прикладне програмування, бази даних та мобільну розробку.

Метою кваліфікаційної роботи є розробка програмного модуля розподіленої системи обміну даними у реальному часі, як мобільного клієнт-серверного застосунку для замовлення та доставки їжі.

Застосунок повинен забезпечити можливість перегляду асортименту страв, вибору їх кількості, формування замовлення, вибору типу оплати, передачі даних на сервер та збереження їх у базі даних.

Для досягнення поставленої мети необхідно вирішити такі задачі:

– провести аналіз сучасних підходів до розробки мобільних застосунків;

– здійснити обґрунтування вибору програмних засобів і технологій

– розробити архітектуру клієнт-серверної взаємодії;

– реалізувати серверну частину;

– розробити базу даних для зберігання інформації про страви та замовлення;

– розробити інтерфейс користувача з підтримкою категорій, сортування, кошика, історії замовлень;

– реалізувати обробку замовлення з боку сервера;

– протестувати систему та оцінити її функціональні можливості.

Об’єкт дослідження – розподілені системи.

Предмет дослідження – алгоритми та технології обміну даними в реальному часі в розподілених системах.

Практична цінність полягає у розробці мобільного застосунку замовлення їжі, який забезпечує зручне та зрозуміле середовище для системи доставки їжі.

Кваліфікаційну роботу виконано відповідно до поставленої задачі з урахуванням вимог, викладених у методичних рекомендаціях [14-16].

Основні результати дослідження опубліковано у тезах доповіді на II Всеукраїнській науково-практичній конференції студентів, аспірантів та молодих вчених «Інтелектуальні комп’ютерні системи та мережі-2025» [17]. Копії публікації можна переглянути у додатку А.

1. РОЗПОДІЛЕНІ СИСТЕМИ ОБМІНУ ДАНИМИ В РЕАЛЬНОМУ ЧАСІ

1.1 Сучасні підходи до розробки розподілених систем

Мобільні технології пройшли довгий шлях розвитку – від простих пристроїв з функцією голосового дзвінка до сучасних багатофункціональних смартфонів, які виступають повноцінними обчислювальними платформами. Упродовж останніх десятиліть зміни торкнулися не лише апаратної частини мобільних пристроїв, а й програмного забезпечення, архітектурних рішень, методів розробки та взаємодії з користувачем.

Перші покоління мобільних телефонів обмежувалися елементарними функціями – дзвінки, SMS, базова телефонна книга. Це були пристрої, що не потребували складного програмного забезпечення, а всі функції реалізовувалися вбудованими системами на мікропрограмному рівні. Проте з появою телефонів, які підтримували JAVA-додатки, MMS та мобільний інтернет, з’явилася потреба у створенні програмного забезпечення нового рівня – більш складного, графічно орієнтованого, гнучкого та портованого.

Значним етапом у розвитку мобільних рішень стало впровадження смартфонів – пристроїв з операційними системами, такими як Symbian, Windows Mobile, а пізніше Android та iOS. Смартфони відкрили нові горизонти для розробників: можливість створювати складні багатофункціональні застосунки, працювати з сенсорними екранами, геолокацією, інтернет-з’єднанням, базами даних, обміном повідомленнями в реальному часі тощо.

У візуальній формі розвиток мобільних пристроїв ілюструється на рисунку 1.1. Тут показано послідовний перехід від простих моделей кнопкових телефонів до сучасних смартфонів із сенсорним екраном, широким функціональним набором і високою продуктивністю.

Рисунок 1.1 –Еволюція мобільних телефонів як апаратної платформи для програмних рішень

Такий розвиток апаратного забезпечення став основою для формування нових вимог до програмних засобів: зросли очікування щодо інтерактивності, швидкодії, доступності сервісів та зручності використання. Зі зростанням популярності мобільних додатків розробники почали використовувати спеціалізовані фреймворки для розроблення універсальних застосунків, які працюють на різних операційних системах з єдиною кодовою базою.

Таким чином, еволюція мобільних пристроїв безпосередньо вплинула на парадигму розробки програмного забезпечення: від вузькоспеціалізованих рішень до складних клієнт-серверних систем із повноцінною підтримкою зберігання, обробки та передачі даних у реальному часі.

У світі мобільної розробки програміст стикається з вибором між нативною та кросплатформною архітектурою. Цей вибір визначається не лише технічними параметрами, а й бюджетом, термінами реалізації, доступністю команди, а також вимогами до охоплення аудиторії та швидкості виведення продукту на ринок.

Нативна розробка передбачає розробку мобільного додатку окремо для кожної операційної системи, використовуючи мови програмування та офіційні SDK тієї чи іншої платформи. Наприклад, Swift або Objective-C для iOS, Java або Kotlin для Android. Такий підхід забезпечує максимальну продуктивність, нативний інтерфейс і повний доступ до функціональності пристрою, проте потребує значно більших витрат на розробку і подальше обслуговування.

Натомість кросплатформна розробка дозволяє створювати додатки з єдиною кодовою базою, які працюють як на Android, так і на iOS. Цей підхід передбачає

використання спеціалізованих фреймворків – зокрема Flutter, React Native, Xamarin та ін. Розробка стає швидшою, дешевшою та простішою з точки зору підтримки, однак може обмежити доступ до апаратних можливостей пристрою, а також дещо знижувати продуктивність у складних інтерфейсах.

У таблиці 1.1 узагальнено основні переваги та недоліки кожного з підходів.

Таблиця 1.1 – Порівняння нативної та кросплатформної розробки

Ознака	Нативна розробка	Кросплатформна розробка
Продуктивність	Висока	Нижча через шар абстракції
Інтерфейс користувача	Повністю нативний	Частково нативний, залежить від фреймворку
Доступ до функцій пристрою	Повний доступ	Обмежений або складніший в реалізації
Час розробки	Довший (2 окремі додатки)	Швидший (одна кодова база)
Вартість	Вища	Нижча
Підтримка	Окремо для кожної платформи	Уніфікована
Приклади технологій	Swift, Kotlin, Android SDK	Flutter, React Native, Xamarin

Найпопулярнішими фреймворками на сьогодні є Flutter, розроблений Google, який використовує мову Dart, React Native, створений Facebook і заснований на JavaScript, а також Xamarin від Microsoft, який дозволяє програмувати мовою C# [1].

Вибір між кросплатформною та нативною розробкою залежить від пріоритетів проєкту: якщо важлива максимальна продуктивність, нативна взаємодія з платформою і складна графіка – доцільно використовувати нативні технології. Якщо ж головним критерієм є швидкість, універсальність і обмежений бюджет – кросплатформні фреймворки надають оптимальне рішення.

Клієнт-серверна архітектура є однією з основних моделей побудови сучасних інформаційних систем, зокрема мобільних застосунків. Цей підхід передбачає чіткий розподіл функцій між клієнтською частиною (мобільним або вебдодатком), яка виконує роль інтерфейсу користувача, та серверною частиною, яка обробляє запити, виконує бізнес-логіку та зберігає дані.

Клієнт взаємодіє з сервером за допомогою інтернет-з'єднання, надсилаючи запити на виконання певних операцій (отримання даних, передача замовлення, авторизація користувача тощо). Сервер приймає запит, обробляє його відповідно до внутрішньої логіки, взаємодіє з базою даних або іншими модулями, формує відповідь (зазвичай у форматі JSON) і повертає її клієнту.

Модель клієнт-сервер дозволяє досягти централізованості, розділення відповідальностей, масштабованості та зручності в адмініструванні системи. У мобільній розробці такий підхід забезпечує надійну взаємодію між користувачем і хмарною частиною сервісу, незалежно від типу пристрою або операційної системи. Завдяки цьому можна централізовано оновлювати логіку бізнес-процесів, здійснювати моніторинг запитів, зберігати історію дій користувачів тощо.

На рисунку 1.2 представлено спрощену схему клієнт-серверної архітектури, яка ілюструє основні компоненти взаємодії між пристроєм користувача (мобільним телефоном, планшетом або ПК) та віддаленим сервером через інтернет [2].

До переваг клієнт-серверної моделі у мобільних застосунках слід віднести:

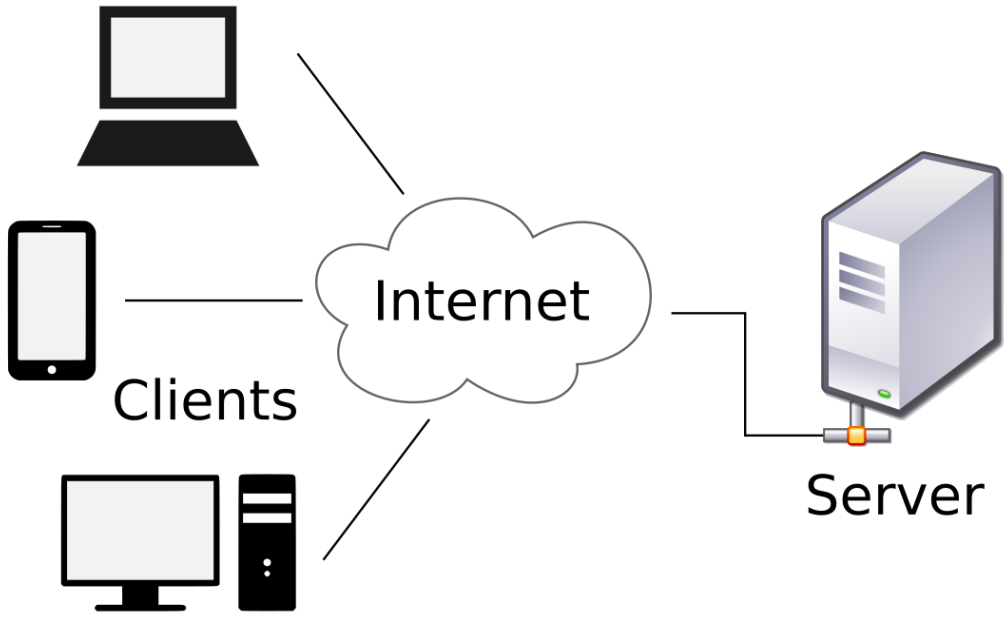

Рисунок 1.2 – Схема клієнт-серверної архітектури мобільного застосунку

До переваг клієнт-серверної моделі у мобільних застосунках слід віднести:

– централізоване управління даними;

– легкість масштабування;
– незалежність клієнта від реалізації серверної логіки;
– підтримку захищених протоколів обміну даними (HTTPS, WebSocket).

Таким чином, використання клієнт-серверної архітектури є виправданим рішенням для розробки мобільного додатку доставки їжі, оскільки вона дозволяє ефективно взаємодіяти з базою даних, керувати замовленнями та забезпечити багатокористувацький доступ до системи.

1.2 Аналіз мобільних платформ для обміну даними

Операційна система Android є найбільш поширеною платформою для мобільних пристроїв у світі. Вона була створена компанією Android Inc., а згодом придбана Google у 2005 році. Перша публічна версія Android з'явилася у 2008 році й з того часу зазнала численних оновлень, що супроводжувалися суттєвими технічними й функціональними змінами.

Однією з ключових переваг Android є її відкритий вихідний код, що дає змогу як великим виробникам, так і незалежним розробникам створювати власні модифікації, оболонки та оптимізації під різні пристрої. Ця відкритість забезпечує Android гнучкість у використанні – платформа адаптована для смартфонів, планшетів, телевізорів, автонавігаційних систем та пристроїв IoT.

Однак така гнучкість породжує проблему фрагментації. На ринку одночасно присутні сотні моделей пристроїв з різними версіями Android, різною роздільною здатністю екранів, архітектурою процесорів, обсягом оперативної пам'яті тощо. Це створює виклик для розробників, яким необхідно забезпечити стабільну роботу додатка на широкому спектрі пристроїв та враховувати сумісність із кількома версіями операційної системи.

Починаючи з Android 1.5 Cupcake, кожна стабільна версія операційної системи отримувала кодову назву у вигляді десерту, що стало фірмовою традицією Google. Зображення на рисунку 1.3 демонструє етапи розвитку Android – від ранніх версій

(Donut, Gingerbread) до сучасніших (Pie, Android 10, Android 11). Цей візуальний ряд підкреслює постійну еволюцію платформи як у функціональному, так і в інтерфейсному аспектах. [3].

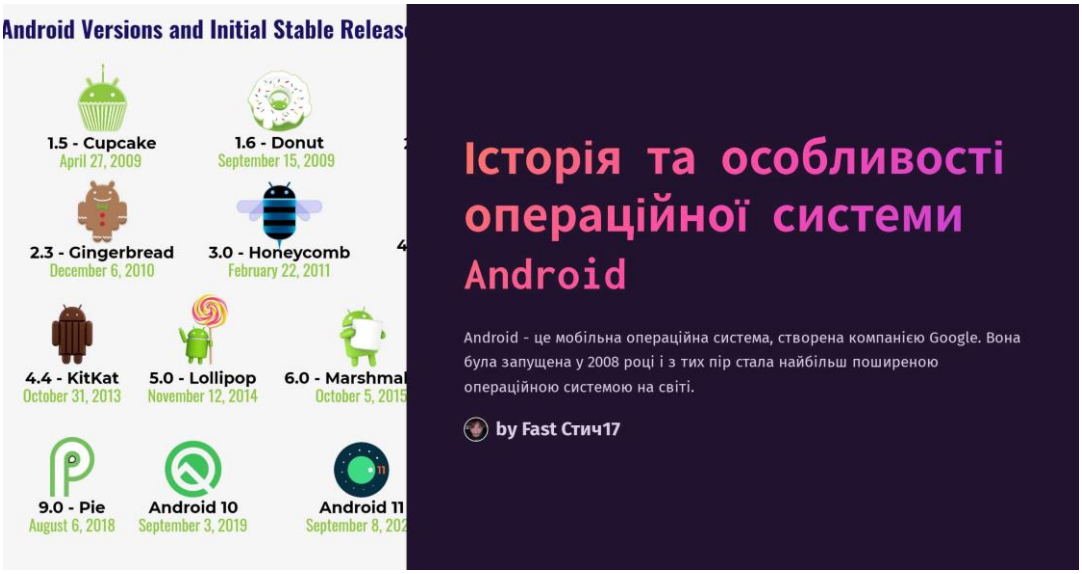

Рисунок 1.3 – Історія версій операційної системи Android

Сучасні версії Android надають потужні можливості для розробників мобільних додатків, включаючи підтримку асинхронної взаємодії з сервером, розгалужену систему дозволів, інтеграцію з апаратними сенсорами, підтримку Material Design та нові API для машинного навчання, AR і безпеки.

Операційна система iOS, розроблена корпорацією Apple, є однією з двох найпоширеніших мобільних платформ у світі. Вперше представлена у 2007 році разом з виходом першого iPhone, iOS від самого початку вирізнялася високим рівнем інтеграції з апаратним забезпеченням, що дозволило забезпечити стабільність, високу продуктивність і якісний досвід взаємодії користувача з інтерфейсом.

Однією з основних переваг iOS є її замкнена екосистема, яка дозволяє Apple контролювати не лише програмне середовище, а й апаратну частину, що дає змогу досягти виняткової оптимізації. Розробники створюють застосунки під обмежену кількість пристроїв, які мають схожі характеристики, роздільну здатність екрана та поведінку API, що значно зменшує ризики фрагментації порівняно з Android.

Крім того, iOS надає особливу увагу безпеці. Усі програми проходять ретельну перевірку перед публікацією в App Store, а сама система підтримує багаторівневий захист – шифрування даних, ізоляцію додатків (sandbox), захист від вторгнень на рівні ядра системи. Регулярні оновлення, які централізовано надходять на всі підтримувані пристрої, також посилюють цю перевагу.

З іншого боку, суворий контроль Apple над своєю платформою має і зворотний бік – обмежений доступ для розробників до багатьох системних функцій. Це може ускладнювати реалізацію деяких функціональностей або вимагати альтернативних рішень. До того ж, публікація додатку в App Store вимагає суворого дотримання технічних, етичних та безпекових стандартів Apple.

На рисунку 1.4 подано фірмовий логотип iOS, який асоціюється із закритістю, стабільністю та фокусом на бездоганному дизайні. Цей візуальний образ став символом не лише платформи, а й усього підходу Apple до розробки мобільних рішень.

Таким чином, iOS є платформою, орієнтованою на якість, безпеку та контроль, що робить її привабливою для розробників комерційних, корпоративних та медичних застосунків, де критичними є питання конфіденційності та стабільної роботи [4].

Рисунок 1.4 – Логотип iOS як символ закритої, оптимізованої та безпечної платформи

Android та iOS – це дві провідні операційні системи для мобільних пристроїв, кожна з яких має свої унікальні особливості, інструменти розробки, архітектуру та підхід до взаємодії з користувачем. Розробка мобільних застосунків для цих платформ суттєво відрізняється як з технічної, так і з організаційної точки зору.

Однією з ключових відмінностей є технологічний стек. Для Android використовується мова програмування Kotlin або Java, у той час як для iOS – Swift або Objective-C. Android-розробники використовують середовище Android Studio, яке забезпечує потужний набір інструментів для роботи з емулюванням, інтерфейсами та API. Розробка під iOS виконується в середовищі Xcode, що має глибоку інтеграцію з усіма сервісами Apple.

Ще однією важливою відмінністю є фрагментація. У випадку Android існує велика кількість пристроїв з різними характеристиками: розмірами екранів, процесорами, версіями ОС. Це потребує додаткового тестування, адаптації інтерфейсу та оптимізації продуктивності під широкий спектр конфігурацій. iOS, натомість, має обмежену кількість підтримуваних моделей пристроїв, що полегшує розробку та тестування.

Публікація додатків також має різну специфіку. У Google Play Store публікація відбувається швидко – перевірка автоматизована та зазвичай займає кілька годин. App Store від Apple вимагає проходження ретельної модерації, що може тривати декілька днів. Водночас політика Apple суворо регламентує вимоги до безпеки, дизайну та контенту, чим забезпечує високий рівень якості додатків.

Також відмінною рисою є монетизація. Користувачі iOS, як правило, частіше оплачують покупки або підписки у додатках, тоді як Android-аудиторія є більш чисельною, але менш схильною до витрат. Вибір платформи для запуску застосунку часто залежить від бізнес-моделі розробника або замовника [5]. Узагальнене порівняння особливостей обох платформ наведено в таблиці 1.2. Порівняння здійснено за певними критеріями, які є важливими для при розробці застосунків.

Таблиця 1.2 – Порівняння Android та iOS для розробників

Критерій	Android	iOS
Мови програмування	Kotlin, Java	Swift, Objective-C
IDE	Android Studio	Xcode
Фрагментація пристроїв	Висока	Низька
Вимоги до інтерфейсу	Гнучкі (Material Design)	Стандартизовані (Human Interface)
Складність публікації	Мінімальна перевірка	Ретельна модерація
Час публікації	Від кількох годин	1–3 дні
Монетизація	Більший ринок, нижча платоспроможність	Менший ринок, вища платоспроможність
Доступ до системних API	Високий	Обмежений
Підписка на SDK	Безкоштовна	Платна (99 $/рік)

У результаті можна зробити висновок, що кожна платформа має свої переваги й обмеження. Android краще підходить для швидкого виведення продукту на широкий ринок, у той час як iOS – для проєктів, орієнтованих на преміальну аудиторію та високі стандарти якості.

1.3 Аналіз фреймворків для кросплатформної розробки

Фреймворки – це програмні платформи, які забезпечують готову архітектуру та набір інструментів для спрощення процесу розробки програмного забезпечення. Вони дозволяють розробникам зосередитися на логіці застосунку, не витрачаючи час на реалізацію базових функцій, таких як маршрутизація, обробка запитів, підключення до бази даних, авторизація тощо.

Фреймворки часто включають набір бібліотек, шаблонів проєктування та стандартів, що сприяє підтримці коду, підвищенню продуктивності розробки й зменшенню кількості помилок. Їх застосування значно прискорює процес створення вебсайтів, мобільних застосунків, десктопних програм та інших програмних рішень.

Існує багато різних фреймворків, орієнтованих на різні мови програмування та задачі, наприклад:

- React, Angular та Vue.js – для веброзробки (JavaScript);
- Django (Python), Laravel (PHP), Spring (Java) – для серверної частини;
- Flutter (Dart), React Native (JavaScript) – для мобільної розробки.

Flutter – це відкритий фреймворк для кросплатформної розробки, створений компанією Google. Він дозволяє розробникам створювати високоякісні мобільні, веб- та десктопні додатки з єдиною базою коду. Завдяки використанню мови програмування Dart та власного графічного рушія, Flutter забезпечує швидку розробку та привабливий інтерфейс користувача [6].

Однією з ключових переваг Flutter є можливість створення нативних інтерфейсів з високою продуктивністю. Фреймворк надає широкий набір віджетів, які дозволяють легко реалізовувати складні UI-рішення. Крім того, функція "гарячого перезавантаження" (hot reload) значно прискорює процес розробки, дозволяючи миттєво бачити зміни в коді без необхідності повного перезапуску додатку.

Flutter активно використовується в розробці додатків для доставки їжі, електронної комерції, фінансових сервісів та інших галузей, де важливі швидкість розробки та якість інтерфейсу. Його популярність зростає завдяки активній спільноті розробників та постійній підтримці з боку Google.

На рисунку 1.5 зображено офіційний логотип Flutter, який символізує сучасний підхід до розробки кросплатформних додатків.

Рисунок 1.5 – Офіційний логотип Flutter

React Native – це популярний фреймворк для кросплатформної розробки мобільних застосунків, створений компанією Meta (колишня Facebook) у 2015 році. Він дозволяє використовувати мову JavaScript разом із бібліотекою React для розробки нативних додатків для платформ Android та iOS. Основна ідея полягає у написанні більшої частини коду один раз і подальшому його використанні на обох платформах, що суттєво економить час і ресурси розробників [7].

На відміну від гібридних рішень, React Native дозволяє створювати застосунки, що використовують нативні компоненти інтерфейсу, а не вбудовані веб-вью. Це забезпечує вищу продуктивність, гнучкість у розробці та покращений користувацький досвід. Додатки React Native виглядають і поводяться майже як нативні, що робить його ідеальним вибором для компаній, які прагнуть скоротити витрати, не втрачаючи якості.

Однією з найсильніших сторін React Native є його тісний зв’язок із екосистемою JavaScript. Розробники, які мають досвід роботи з веб-технологіями, такими як React.js, можуть легко адаптуватися до мобільного середовища, що знижує поріг входу та прискорює навчання. Крім того, велика спільнота забезпечує постійну підтримку, розширення функціоналу через npm-бібліотеки та оновлення.

Суттєвою перевагою є наявність величезної кількості готових рішень і плагінів, які дозволяють розширювати можливості застосунків: від доступу до апаратних можливостей смартфона до інтеграції з хмарними сервісами. Також React

Native підтримує використання нативного коду, що дозволяє реалізовувати функції, які не підтримуються напряму у JavaScript.

Разом з тим, React Native має й свої недоліки. Наприклад, у складних або дуже продуктивних додатках можуть виникати труднощі з оптимізацією, особливо коли потрібно взаємодіяти з анімаціями чи обробляти велику кількість даних у реальному часі. У таких випадках часто доводиться писати власні нативні модулі на Swift, Kotlin або Java, що зменшує переваги повторного використання коду.

Також важливою особливістю React Native є «гаряче оновлення» (Fast Refresh), яке дозволяє розробникам швидко бачити зміни без повного перезапуску застосунку, що значно прискорює цикл розробки. Цей інструмент став ключовим для зручного UI-розроблення.

На рисунку 1.6 представлено офіційний логотип React Native. Він символізує ядро фреймворку – ядро React із можливістю масштабування на різні платформи. Його стилізована форма атома візуалізує головну концепцію повторного використання елементів (компонентів) і реактивного підходу до створення інтерфейсів.

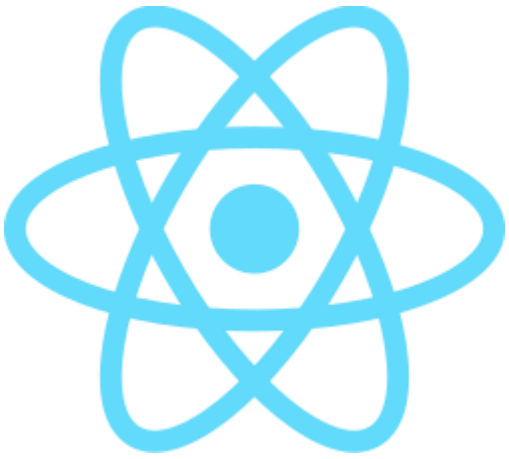

Рисунок 1.6 – Логотип фреймворку React

Вибір фреймворку для кросплатформної розробки мобільних застосунків є стратегічним рішенням, яке впливає на швидкість, якість та довготривалу підтримку проєкту. Найбільш поширеними рішеннями на сьогодні є Flutter та React Native, які

активно конкурують між собою. Щоб обрати оптимальну технологію, необхідно порівняти ключові характеристики: продуктивність, підтримку інструментів, легкість навчання, спільноту та доступ до нативних API.

Flutter, завдяки власному рушію рендерингу (Skia), забезпечує високу продуктивність та повну контрольованість інтерфейсу без залежності від нативних компонентів. React Native, натомість, використовує нативні елементи інтерфейсу, але іноді втрачає у швидкодії через міст між JavaScript та платформою.

Також важливо враховувати рівень документації, стабільність оновлень і легкість інтеграції з платформеними сервісами. Flutter має кращу документацію та підтримку від Google, тоді як React Native виграє завдяки великій екосистемі npm та досвідченій спільноті розробників JavaScript.

Узагальнене порівняння продуктивності фреймворків подано в таблиці 1.3.

Таблиця 1.3 – Порівняння Flutter і React Native за критерієм продуктивності

Критерій	Flutter	React Native
UI продуктивність	Висока завдяки власному рушію Skia	Середня, залежить від містка JavaScript-Native
FPS (кадри за секунду)	Стабільні 60–120 fps	Може знижуватись на складних анімаціях
Час завантаження додатку	Швидкий (особливо на Android)	Залежить від обсягу bridge-комунікації
Доступ до нативного функціоналу	Через плагіни або platform channels	Прямий доступ, але іноді потребує нативного коду
Підтримка апаратного прискорення	Повна	Часткова
Розмір APK/IPA	Трохи більший через вбудовану графіку	Менший, бо використовує нативні компоненти

Загалом, обидва фреймворки показують хороші результати продуктивності. Flutter більше підходить для проектів, де критично важлива плавність інтерфейсу та однаковий вигляд на всіх пристроях. React Native краще інтегрується з уже існуючими нативними застосунками та дозволяє швидко стартувати проєкт за участі веб-команди.

1.4 Постановка задачі кваліфікаційної роботи

Сфера громадського харчування та доставки їжі стрімко розвивається в умовах сучасного цифрового суспільства. Зі зростанням темпу життя, особливо в містах, користувачі дедалі частіше обирають онлайн-замовлення їжі замість відвідування фізичних закладів. При цьому якість сервісу, швидкість обробки замовлення, зручність інтерфейсу та можливість гнучкої оплати – це ключові чинники, що формують лояльність клієнтів і конкурентоспроможність компаній.

Багато малих та середніх закладів харчування поки що використовують ручні методи обробки замовлень або застарілі системи, що не відповідають очікуванням сучасного користувача. Це призводить до втрати клієнтів, помилок під час обробки заявок, неможливості швидко адаптувати меню або ціни, а також ускладнює аналіз і ведення статистики.

Додатковою проблемою є відсутність єдиної мобільної платформи, яка б поєднувала просту взаємодію для клієнта, адміністрування з боку ресторану та гнучкість у роботі з базою даних. В умовах обмеженого бюджету та ресурсу оптимальним рішенням є реалізація кросплатформного мобільного додатку з простим серверним компонентом для обміну даними, керування замовленнями та зберігання історії транзакцій.

Таким чином, виникає об’єктивна потреба у розробці сучасного інструменту, який забезпечить повний цикл взаємодії між клієнтом і службою доставки – від перегляду меню до підтвердження замовлення й отримання зворотного зв’язку. Розробка такого програмного забезпечення дозволить автоматизувати критично важливі процеси та підвищити ефективність закладу харчування.

Метою даного кваліфікаційної роботи є розробка кросплатформного мобільного застосунку для автоматизації процесу замовлення та доставки їжі, який включає зручний інтерфейс для клієнта, а також серверну частину з базою даних для обробки та зберігання замовлень. Застосунок повинен забезпечити швидку взаємодію користувача з меню, можливість формування кошика, введення персональних даних, вибір способу оплати та надсилання замовлення на сервер.

Для досягнення поставленої мети в межах кваліфікаційної роботи вирішуються наступні задачі:

– створити архітектуру клієнт-серверної взаємодії для обміну даними між мобільним додатком та сервером;

– реалізувати серверну частину з використанням Python та фреймворку Flask для обробки замовлень і роботи з базою даних;

– розробити графічний інтерфейс користувача мобільного додатку у середовищі Flutter;

– передбачити можливість збереження історії замовлень та гнучкого керування даними в БД;

– протестувати програмний продукт у локальному середовищі та оцінити його зручність, продуктивність і надійність.

Реалізація всіх зазначених етапів дозволить створити повноцінне рішення, яке можна використовувати в умовах невеликого ресторану або кафе, що прагне автоматизувати взаємодію зі своїми клієнтами без значних фінансових витрат.

2. ТЕХНОЛОГІЇ ПРОЄКТУВАННЯ КЛІЄНТ-СЕРВВЕРНОГО МОБІЛЬНОГО ЗАСТОСУНКУ

2.1 Середовище та засоби розробки мобільного додатку

Для реалізації серверної частини мобільного застосунку було обрано мову програмування Python та фреймворк Flask, який дозволяє швидко створювати REST API. Основним середовищем розробки став PyCharm – потужне IDE, розроблене компанією JetBrains. Його інтеграція з системами контролю версій, можливості автоматичного завершення коду, дебагінг та підтримка віртуального середовища значно полегшили процес розробки (рис 2.1) [8].

Рисунок 2.1 – Середовище розробки PyCharm

Однією з переваг використання PyCharm є можливість зручно працювати з локальною базою даних, зокрема SQLite, яка була обрана для зберігання інформації про страви та замовлення. Інтеграція з ORM-бібліотекою SQLAlchemy забезпечила просту та зрозумілу побудову запитів без необхідності писати сирий SQL-код [27].

Також у проєкті використовувалися додаткові бібліотеки:

– flask_cors – для обробки CORS-запитів між мобільним застосунком і сервером;

– json – для обробки форматованих даних при обміні інформацією;

– datetime – для збереження часу створення замовлення;

– PyInstaller – для пакування серверної частини в .exe-файл без потреби встановлення Python на цільовому ПК.

Flask – це сучасний веб-фреймворк, створений мовою програмування Python. Його автором є австрійський розробник Армін Ронахер, який розпочав проєкт у 2010 році. Flask є мікрофреймворком, тобто каркасом, який надає лише базові засоби для розробки вебзастосунків, залишаючи розробнику свободу у виборі архітектурних рішень. Незважаючи на свою легкість, Flask має велику спільноту користувачів та широкий спектр сторонніх розширень, що робить його одним із найпопулярніших рішень для створення REST API та серверної логіки вебсистем.

Фреймворк базується на інструментарії Werkzeug та використовує шаблонізатор Jinja2. Він повністю сумісний із WSGI 1.0 і підтримує Unicode, що дозволяє розробляти мультимовні застосунки. Завдяки вбудованому серверу розробки та модулю відлагодження, Flask особливо зручний у початкових етапах реалізації проєкту. Підтримується автоматична маршрутизація запитів, сесії зберігаються в cookie, а також доступна інтеграція з модульними тестами. Однією з ключових особливостей Flask є можливість легко додавати додаткову функціональність через офіційні та неофіційні пакети з префіксом flask-, серед яких flask-login, flask-sqlalchemy, flask-cors тощо.

Завдяки гнучкості та простоті, Flask ідеально підходить для реалізації серверної частини мобільного застосунку доставки їжі. Саме ця технологія використовується у дипломному проєкті для розробки API, що забезпечує взаємодію між клієнтом і сервером через мережу [9].

PyInstaller – це утиліта для мови програмування Python, яка дозволяє "згортати" Python-застосунок в самостійний виконуваний файл (.exe для Windows, .app для macOS, ELF для Linux). Вона аналізує програму, визначає її залежності – бібліотеки, модулі, ресурси – і автоматично включає їх у фінальний пакет. Таким чином, користувач може запускати застосунок без встановлення Python або окремих бібліотек. Це дуже зручно для розгортання програм серед користувачів, які не мають технічного досвіду.

PyInstaller працює з усіма основними бібліотеками, включно з Flask, SQLAlchemy, pandas, matplotlib тощо. У дипломному проєкті цей інструмент

використовується для реалізації зручного виконуваного файлу серверної частини, який можна запустити навіть на іншому комп’ютері без потреби встановлювати середовище розробки. Такий підхід значно полегшує тестування та розгортання системи в реальних умовах [10].

Розробка клієнтської частини мобільного застосунку здійснювалася у середовищі Visual Studio Code – зручному і потужному інструменті, який надає розробникам широкий спектр функціональних можливостей (рис 2.2). Завдяки підтримці великої кількості розширень, середовище дозволяє інтегрувати засоби налагодження, автодоповнення коду, перевірку помилок у реальному часі та симуляцію мобільного інтерфейсу без необхідності залишати редактор. Visual Studio Code був обраний завдяки своїй легкості, гнучкості налаштувань і активному розвитку спільноти, що забезпечує постійну підтримку нових технологій та платформ [22].

Основою клієнтської частини став фреймворк Flutter, який розроблений компанією Google і побудований на мові програмування Dart [23]. Flutter надає можливість розробки кросплатформних застосунків – тобто таких, що працюють на Android, iOS, а за потреби – й на Web чи Windows, використовуючи спільну кодову базу. Це дозволяє істотно скоротити час розробки, забезпечуючи при цьому високу продуктивність та нативну взаємодію з платформою. Особливу увагу було приділено підтримці адаптивного дизайну, що досягається через використання сучасного стилю оформлення Material Design, інтегрованого безпосередньо у Flutter SDK [11].

Рисунок 2.2 – Логотип середовища розробки VS Code

Flutter забезпечує високу швидкість розробки завдяки функції гарячого перезапуску (Hot Reload), яка дозволяє розробнику миттєво бачити зміни в інтерфейсі без перезапуску програми. Це значно пришвидшує процес налагодження, тестування та підбору дизайну. Крім того, Flutter надає величезну кількість готових віджетів, які покривають як стандартні елементи керування, так і складні інтерактивні компоненти. Усі елементи інтерфейсу повністю керовані кодом, що забезпечує гнучкість та точний контроль над логікою програми.

У дипломному проєкті було використано дизайн-систему Material 3, яка підтримується нативно у Flutter і дозволяє створювати сучасний, естетичний інтерфейс, що відповідає рекомендаціям Google. Компоненти оформлення, а також система навігації були адаптовані під мобільне середовище, що дозволило створити інтуїтивно зрозумілий інтерфейс для користувача. Завдяки цьому користувацький досвід став більш привабливим та зручним, що є важливою перевагою для мобільного застосунку доставки їжі.

Для реалізації обміну даними між клієнтською та серверною частиною застосунку було використано бібліотеку http – один із найпоширеніших пакетів у середовищі Flutter для взаємодії з RESTful API [25]. Вона дозволяє надсилати HTTP-запити будь-якого типу, включаючи GET, POST, PUT та DELETE, і зручно працювати з JSON-даними. У контексті проєкту пакет http був використаний для завантаження списку страв із бази даних, оформлення замовлень, а також перегляду останньої покупки.

Передача даних між клієнтом і сервером реалізується у форматі JSON, що є універсальним і добре підтримується у більшості мов програмування. Це дозволяє не лише передавати текстову інформацію (наприклад, ім’я, адресу, тип оплати), але й складні об’єкти, як-от перелік товарів з кількістю, цінами та ідентифікаторами. Завдяки цій структурі можна легко реалізувати серіалізацію/десеріалізацію об'єктів у Dart та відповідно обробляти їх на стороні Flask-сервера за допомогою бібліотеки json.

Така архітектура клієнт-серверної взаємодії є гнучкою та масштабованою. При зміні IP-адреси або перенесенні серверної частини на інший хост достатньо лише оновити відповідне поле в налаштуваннях застосунку. Це робить систему

придатною для віддаленого використання, тестування на локальних машинах або розгортання у хмарному середовищі. Такий підхід також дозволяє реалізувати майбутні розширення, наприклад, авторизацію, історію замовлень або аналітику без радикальних змін у клієнтському коді [12].

Етап інтеграції та тестування клієнт-серверного мобільного застосунку здійснювався у межах локальної мережі, до якої були підключені як комп'ютер із серверною частиною, так і мобільний пристрій з Flutter-додатком. Такий підхід дозволяє оперативно перевірити функціональність усіх компонентів без необхідності використання зовнішнього хостингу чи публічної IP-адреси.

Для забезпечення коректного з'єднання між пристроями було реалізовано механізм ручного введення IP-адреси сервера в інтерфейсі програми. Це дало змогу враховувати змінні мережеві умови та гарантувати успішне надсилання запитів на потрібний вузол. Під час тестування було перевірено коректність відображення списку страв, обробку оформлення замовлення, збереження даних у базу даних та відображення останнього замовлення.

Окрему увагу приділено обробці CORS-запитів та безпечному налаштуванню Flask-сервера на боці комп'ютера. Було протестовано обробку помилок, введення некоректних даних і стабільність програми у випадку розриву з'єднання. В результаті успішного тестування підтверджено працездатність застосунку в умовах локального середовища та закладено основу для подальшого масштабування або перенесення на віддалений сервер.

2.2 Серверна частина: використання Flask та SQLAlchemy

У процесі розробки серверної частини мобільного застосунку для доставки їжі було обрано архітектуру клієнт-сервер з використанням веб-фреймворку Flask, який є легким мікрофреймворком на мові програмування Python. Основною причиною вибору саме Flask стала його мінімалістичність, гнучкість та швидка інтеграція з іншими бібліотеками, зокрема SQLAlchemy для роботи з базою даних. Завдяки

підтримці RESTful-архітектури, Flask ідеально підходить для побудови API, через яке мобільний клієнт може взаємодіяти з сервером [24].

Вся логіка роботи із клієнтом реалізується через окремі маршрути (routes), кожен з яких обробляє запити певного типу (GET, POST). На рівні коду це виглядає просто та структуровано. Наприклад, наступний фрагмент коду відповідає за обробку запиту на отримання списку страв:

```
@app.route("/foods", methods=["GET"])
def get_foods():
    session = Session()
    foods = session.query(Food).all()
    result = []
    for f in foods:
        result.append({
            "id": f.id,
            "name": f.name,
            "price": f.price,
            "category": f.category,
            "image": f"http://{SERVER_IP}/images/{f.image}"
        })
    session.close()
    return jsonify(result)
```

У цьому прикладі реалізовано обробник GET-запиту до шляху /foods. Він виконує зчитування даних із бази, формує відповідь у форматі JSON і повертає її клієнту [29]. Це дозволяє мобільному застосунку динамічно формувати інтерфейс меню в залежності від даних, отриманих із сервера. Така архітектура забезпечує чітке розділення клієнтської та серверної логіки, дозволяючи масштабувати систему, змінювати логіку серверної частини без необхідності оновлення мобільного додатку.

На рисунку 2.3 зображено структуру проєкту серверної частини мобільного застосунку, реалізованого за допомогою середовища розробки PyCharm [31]. У папці RestoranMenu міститься основна логіка сервера, який працює на Flask. Весь код розташований у відповідних Python-файлах, а також присутні файли бази даних і конфігураційні файли.

У папці static знаходяться ключові компоненти – app.py, який відповідає за запуск сервера та обробку HTTP-запитів, і db.py, що містить опис моделей бази

даних через ORM-бібліотеку SQLAlchemy. Два файли SQLite – restaurant.db та database.db – вказують на наявність структури для зберігання інформації про замовлення та страви. Окремо виділено файл foods.json, який, імовірно, використовується для збереження первинного набору страв у форматі JSON. Усе це організовано логічно й демонструє чітке відокремлення бізнес-логіки, бази даних та ресурсів, що є прикладом добре спроектованої архітектури серверного застосунку.

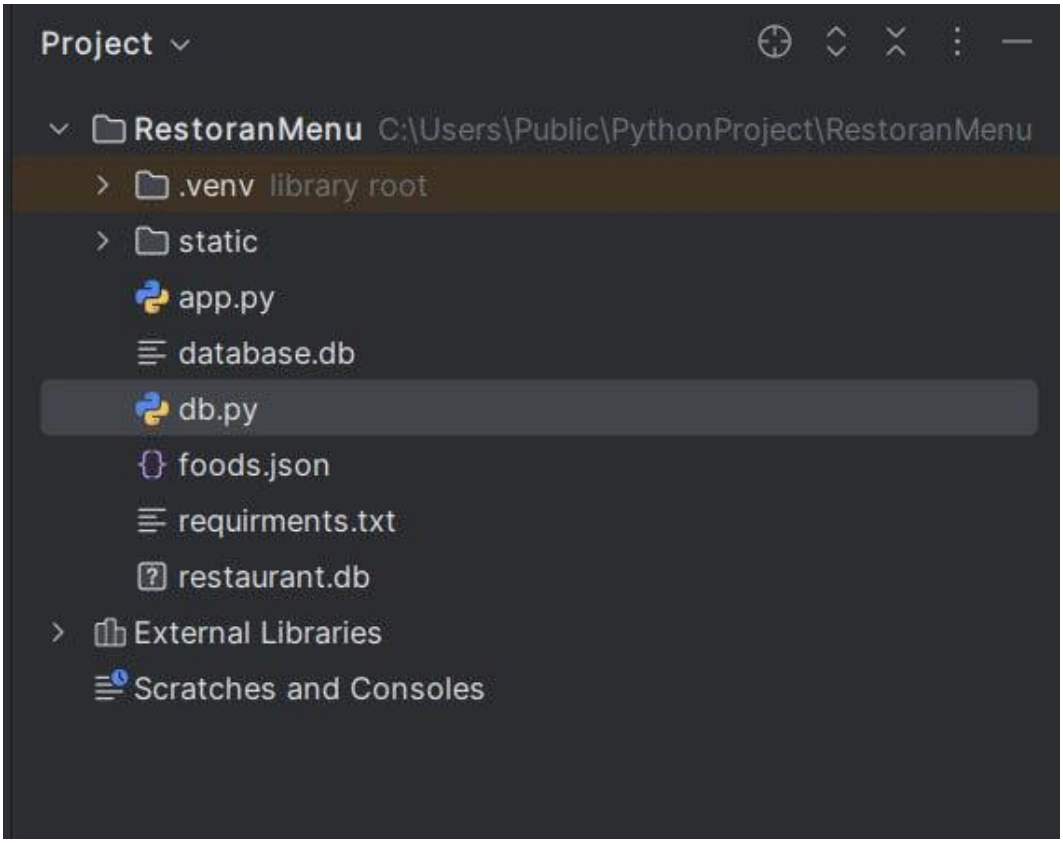

Рисунок 2.3 – Структура серверної частини мобільного застосунку у середовищі PyCharm

У процесі реалізації серверної частини було розроблено систему обробки HTTP-запитів за допомогою фреймворку Flask. Основне завдання цієї частини полягає у прийомі запитів від клієнта, обробці отриманих даних, взаємодії з базою даних та поверненні відповідей у форматі JSON. Роутинг у Flask дозволяє гнучко визначати URL-шляхи та методи доступу (GET, POST тощо), що забезпечує чітку структуру сервісу.

На прикладі коду нижче демонструється реалізація маршруту `/foods`, який обробляє GET-запит і повертає список усіх доступних страв. Серверна логіка

використовує SQLAlchemy для звернення до бази даних та формування списку об'єктів:

```
@app.route("/foods", methods=["GET"])
def get_foods():
    session = Session()
    foods = session.query(Food).all()
    result = []
    for f in foods:
        result.append({
            "id": f.id,
            "name": f.name,
            "price": f.price,
            "category": f.category,
            "image": f"http://192.168.0.101:8080/images/{f.image}"
        })
    session.close()
    return jsonify(result)
```

Крім того, для прийому нових замовлень клієнта реалізовано POST-маршрут /orders. Він отримує JSON-дані з інформацією про клієнта та перелік страв, зберігає їх у базі даних, використовуючи модель Order, і повертає повідомлення про успішне збереження:

```
@app.route("/orders", methods=["POST"])
def place_order():
    session = Session()
    data = request.get_json()

    customer = data.get("customer", {})
    items = data.get("order", [])

    order = Order(
        customer_name=customer.get("name", ""),
        customer_address=customer.get("address", ""),
        payment_method=customer.get("payment", ""),
        items=json.dumps(items, ensure_ascii=False),
        created_at=datetime.utcnow()
    )

    session.add(order)
    session.commit()
    session.close()

    return jsonify({"message": "Замовлення збережено"}), 200
```

Алгоритмічну схему обробки замовлення представлено на КР.КІ.10660504.00.00.000 А1.

Такий підхід забезпечує ефективну взаємодію між клієнтом та сервером у межах локальної мережі або при віддаленому розміщенні серверної частини, а формат JSON дозволяє гнучко передавати структуровану інформацію.

2.3 Проєктування архітектури та бази даних

Логічна архітектура клієнт-серверного застосунку базується на моделі «тонкий клієнт – потужний сервер». Клієнтська частина, реалізована за допомогою Flutter, відповідає за візуалізацію інтерфейсу, обробку дій користувача та формування HTTP-запитів. Серверна частина, написана з використанням Flask, приймає ці запити, виконує відповідні дії – зчитування чи запис у базу даних – і повертає відповідь у форматі JSON. Структурну схему архітектури програмної системи обміну даними в реальному часі представлено на КР.КІ.10660504.00.00.000 С1.

Ключовим компонентом серверної частини є обробка запитів до бази даних, у якій зберігається перелік страв, інформація про замовлення, а також інші структуровані дані. Передача даних між мобільним застосунком і сервером здійснюється через локальну мережу з використанням REST-інтерфейсу. Такий підхід дозволяє гнучко розширювати застосунок, масштабувати функціонал і легко адаптувати сервер для хмарного розгортання, змінивши лише IP-адресу в налаштуваннях клієнта.

Для реалізації зберігання даних використано ORM-бібліотеку SQLAlchemy, яка дозволяє описати структуру таблиць у вигляді Python-класів. Нижче наведено фрагмент ініціалізації бази даних, який створює таблиці foods і orders, а також заповнює таблицю foods початковими даними:

```
class Food(Base):
```

```
    __tablename__ = "foods"
    id = Column(Integer, primary_key=True)
    name = Column(String(100), nullable=False)
    price = Column(Integer, nullable=False)
    image = Column(String(100), nullable=False)
    category = Column(String(50), nullable=False)
class Order(Base):
    __tablename__ = "orders"
    id = Column(Integer, primary_key=True, autoincrement=True)
    customer_name = Column(String(100), nullable=False)
    customer_address = Column(String(255), nullable=False)
    payment_method = Column(String(50), nullable=False)
    items = Column(Text, nullable=False)
    created_at = Column(DateTime, default=datetime.utcnow)
```

У процесі реалізації серверної частини мобільного застосунку була створена повноцінна реляційна база даних, яка забезпечує збереження інформації про наявні страви та оброблені замовлення. Для цього було використано SQLite у поєднанні з ORM-бібліотекою SQLAlchemy, що дозволяє працювати з таблицями на рівні об’єктів Python.

Базу даних умовно поділено на дві основні таблиці: foods та orders. Таблиця foods містить інформацію про всі доступні позиції меню, включаючи назву страви, ціну, шлях до зображення та категорію. Кожен запис має унікальний ідентифікатор (id), який використовується для подальших зв’язків. Таблиця orders, у свою чергу, зберігає відомості про замовлення клієнтів, зокрема їхні контактні дані, спосіб оплати, обрані страви у вигляді JSON-рядка та дату створення замовлення.

На рисунку 2.4 представлена загальна схема структури бази даних із вказанням SQL-запитів, які були використані для створення таблиць. Візуально видно, що типи полів підібрані відповідно до очікуваних даних – текстові значення мають тип VARCHAR, а числові – INTEGER.

Рисунок 2.5 ілюструє вміст таблиці foods, що демонструє вже заповнені позиції меню: піци, бургери, роли, кожна з яких має відповідний файл зображення. Це забезпечує коректне відображення інтерфейсу мобільного застосунку.

Имя	Тип	Схема
Таблицы (2)		
foods		CREATE TABLE foods (id INTEGER NOT NULL, name VARCHAR(100) NOT NULL, price INTEGER NOT NULL, image VARCHAR(100) NOT NULL, category VAI
id	INTEGER	"id" INTEGER NOT NULL
name	VARCHAR(100)	"name" VARCHAR(100) NOT NULL
price	INTEGER	"price" INTEGER NOT NULL
image	VARCHAR(100)	"image" VARCHAR(100) NOT NULL
category	VARCHAR(50)	"category" VARCHAR(50) NOT NULL
orders		CREATE TABLE orders (id INTEGER NOT NULL, customer_name VARCHAR(100) NOT NULL, customer_address VARCHAR(255) NOT NULL, payment_metho
id	INTEGER	"id" INTEGER NOT NULL
customer_name	VARCHAR(100)	"customer_name" VARCHAR(100) NOT NULL
customer_address	VARCHAR(255)	"customer_address" VARCHAR(255) NOT NULL
payment_method	VARCHAR(50)	"payment_method" VARCHAR(50) NOT NULL
items	TEXT	"items" TEXT NOT NULL
created_at	DATETIME	"created_at" DATETIME
Индексы (0)		
Представления (0)		
Триггеры (0)		

Рисунок 2.4 – Структура таблиць бази даних foods і orders у SQLite

Таблиця: foods

Filter in any column

	id	name	price	image	category
	Ф...	Фильтр	Фил...	Фильтр	Фильтр
1	1	Маргарита	120	1.png	Піца
2	2	Бургер Чіз	95	2.png	Бургери
3	3	Каліфорнія	150	3.png	Суші
4	4	Чотири сири	135	4.png	Піца
5	5	Бургер BBQ	105	5.png	Бургери

Рисунок 2.5 – Заповнена таблиця foods зі стравами меню

Рисунок 2.6 показує таблицю orders, у якій фіксуються замовлення. Поле items зберігає JSON-структуру з інформацією про обрані страви, що дозволяє передавати склад замовлення без додаткових зв’язків між таблицями, зберігаючи простоту реалізації.

Таблиця: orders

Filter in any column

	id	customer_name	customer_address	payment_method	items	created_at
	Ф...	Фильтр	Фильтр	Фильтр	Фильтр	Фильтр
1	1	1	1	Готівка	[{"item": {"id": 5, "name": "Бургер …	2025-04-29 22:48:50.908958
2	2	Test	Test	Карта	[{"item": {"id": 2, "name": "Бургер …	2025-04-29 23:01:48.352294

Рисунок 2.6 – Таблиця orders зі збереженими клієнтськими замовленнями

Вибір поєднання SQLite та SQLAlchemy для реалізації серверної частини мобільного застосунку ґрунтується на їхній практичності, зручності використання та відповідності вимогам проєкту. SQLite – це легка вбудована система управління

базами даних, яка не потребує окремого серверного процесу. Вона дозволяє зберігати всі дані в одному файлі, що спрощує розгортання та обслуговування застосунку, особливо на етапах розробки, локального тестування та використання у невеликих проєктах. Завдяки підтримці стандарту SQL розробникам доступні знайомі запити для взаємодії з даними, а продуктивність SQLite повністю покриває потреби мобільного або середньонавантаженого клієнт-серверного рішення.

Доповненням до цієї системи виступає бібліотека SQLAlchemy – одна з найпопулярніших ORM-бібліотек для Python. Вона забезпечує гнучкий підхід до роботи з базами даних, дозволяючи описувати структури таблиць і запити у вигляді об’єктів і класів Python. SQLAlchemy дозволяє розробнику уникати написання сирого SQL-коду, забезпечує підтримку транзакцій, зв’язків між таблицями, управління сесіями та масштабування проєкту за потреби. Бібліотека підтримує як високорівневе об'єктне моделювання, так і низькорівневий контроль над SQL-запитами, що є суттєвою перевагою в ситуаціях, де потрібна оптимізація або нестандартна логіка [13].

У поєднанні ці інструменти створюють ефективне середовище для побудови серверної логіки застосунку. SQLite забезпечує простоту й автономність, а SQLAlchemy – гнучкість і зручність у взаємодії з даними. Такий підхід дозволяє розробникам зосередитися на функціональності програми та бізнес-логіці, мінімізуючи витрати на конфігурацію та інфраструктурне обслуговування. Усе це робить комбінацію SQLite і SQLAlchemy ідеальним вибором для швидкої, зручної та стабільної реалізації серверної частини мобільного клієнт-серверного проєкту.

2.4 Інтерфейс користувача мобільного додатку

Під час реалізації клієнтської частини мобільного застосунку було обрано концепцію дизайну, яка базується на сучасних принципах UX/UI та максимально адаптована до мобільного середовища. Основний акцент було зроблено на зручності взаємодії, швидкому доступі до функцій та візуальній простоті. Застосунок

орієнтований на користувача, що бажає зробити замовлення за кілька торкань, тому головна сторінка одразу відкриває меню з категоріями та стравами, які можна відфільтрувати або відсортувати. Всі кнопки, поля введення, списки та повідомлення адаптовані до мобільного екрану, мають великий розмір натискання, а також використовують віджети Material 3, які автоматично враховують тему оформлення та розмір екрана пристрою.

Flutter дозволяє реалізувати такий інтерфейс, використовуючи гнучку систему віджетів. Наприклад, навігація здійснюється за допомогою AppBar з дією переходу до кошика, а елементи меню будуються як списки ListView.builder із картками Card, що зберігають візуальну цілісність. Ось приклад фрагменту коду, який демонструє відображення окремої страви:

```
return Card(
  margin: const EdgeInsets.all(8),
  shape: RoundedRectangleBorder(
    borderRadius: BorderRadius.circular(16),
  ),
  elevation: 4,
  child: Column(
    children: [
      ListTile(
        leading: Image.network(
          item.image,
          width: 50,
          height: 50,
          fit: BoxFit.cover,
        ),
        title: Text(
          item.name,
          maxLines: 1,
          overflow: TextOverflow.ellipsis,
        ),
        subtitle: Text('${item.price} ₴'),
        trailing: IconButton(
          icon: const Icon(Icons.add_shopping_cart),
          onPressed: () => _addToCart(item),
        ),
      ),     ],   ), );
```

Цей код демонструє використання естетично привабливої картки з тінями, заокругленими кутами та інтеграцією мережевого зображення. Він ілюструє загальну концепцію: проста форма, лаконічне розміщення елементів та швидкий

доступ до основної дії – додавання товару до кошика. Подібний підхід дозволив створити зручний, логічний та візуально сучасний застосунок.

Головний інтерфейс мобільного застосунку є центральним елементом взаємодії користувача з системою. На рисунку 2.7 зображено головний екран, який представляє собою адаптивне меню доставки з відображенням доступних страв у вигляді окремих карток. Кожна картка містить зображення страви, її назву, вартість, а також інструменти для зміни кількості обраних одиниць та додавання товару до кошика. Дизайн виконано згідно з концепцією Material 3, що забезпечує чистий, сучасний і зручний візуальний стиль. Компоненти оформлення мають закруглені краї, тіні для виокремлення елементів і достатню кількість вільного простору між ними, що полегшує сприйняття на сенсорних екранах.

Важливою складовою функціонального інтерфейсу є також фільтри: зверху реалізовано спадні списки для вибору категорії страв і сортування – за назвою або ціною. Таке компонування дозволяє користувачеві швидко знаходити необхідні товари, адаптуючи вивід відповідно до особистих уподобань. З правого боку у верхній панелі розташовані кнопки для переходу до історії замовлень та кошика, які є постійно доступними незалежно від поточного вмісту меню.

На рисунку 2.8 продемонстровано інтерфейс кошика, у якому реалізовано повноцінну форму оформлення замовлення. Вона включає поля для введення імені, адреси та вибору способу оплати – готівкою або банківською картою. У разі вибору картки з’являються додаткові поля для введення номера, терміну дії та CVV. Уся інформація подається послідовно та логічно згруповано, що забезпечує зручність заповнення. Нижче знаходиться кнопка «Оформити замовлення», яка відображає суму поточного кошика, динамічно оновлюючи її при зміні кількості товарів.

Рисунок 2.7 – Головний екран мобільного застосунку з меню доставки функціоналом фільтрації

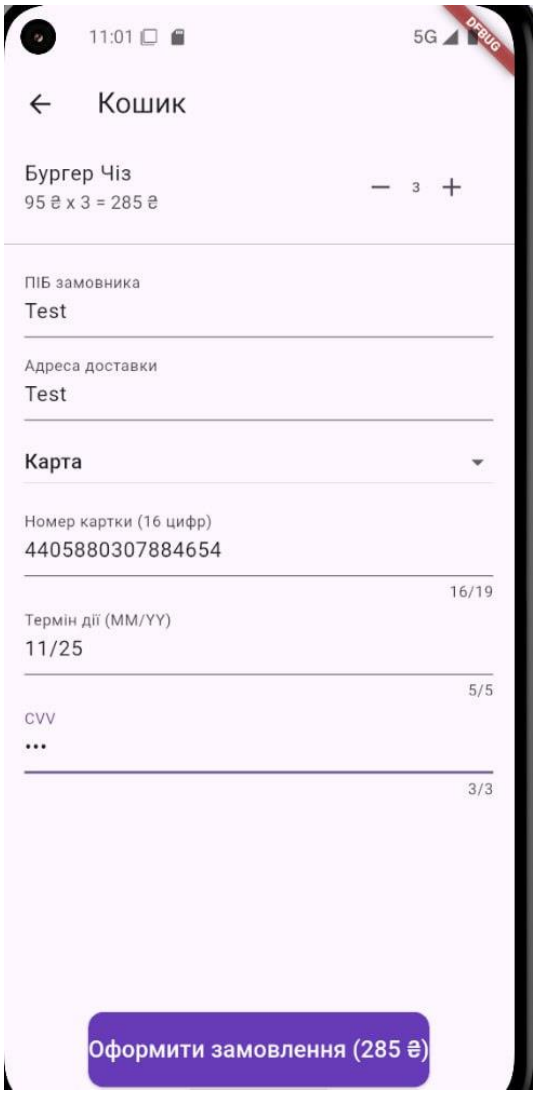

Рисунок 2.8 – Екран кошика із формою замовлення та вибором способу оплати

3. РЕАЛІЗАЦІЯ ПРОГРАМНОГО МОДУЛЯ

3.1 Структура та логіка взаємодії клієнта з сервером

Основу взаємодії становить клієнт-серверна архітектура, де мобільний застосунок, створений за допомогою Flutter, виступає як клієнт, що надсилає HTTP-запити до сервера. У відповідь сервер, реалізований із використанням Flask, обробляє запити, звертається до бази даних SQLite, формує відповідь у форматі JSON та надсилає її назад до клієнта [18].

На рисунку 3.1 представлено схематичну модель цієї взаємодії. Користувач через Flutter-застосунок надсилає HTTP-запит на сервер, який обробляється Flask-частиною. Сервер взаємодіє з базою даних SQLite, зберігає або витягує необхідні дані та формує JSON-відповідь. Після цього клієнт отримує результат, який може бути відображений в інтерфейсі мобільного додатку. Така модель дозволяє чітко розділити обов'язки між клієнтом і сервером, спрощуючи розробку, обслуговування та розгортання системи.

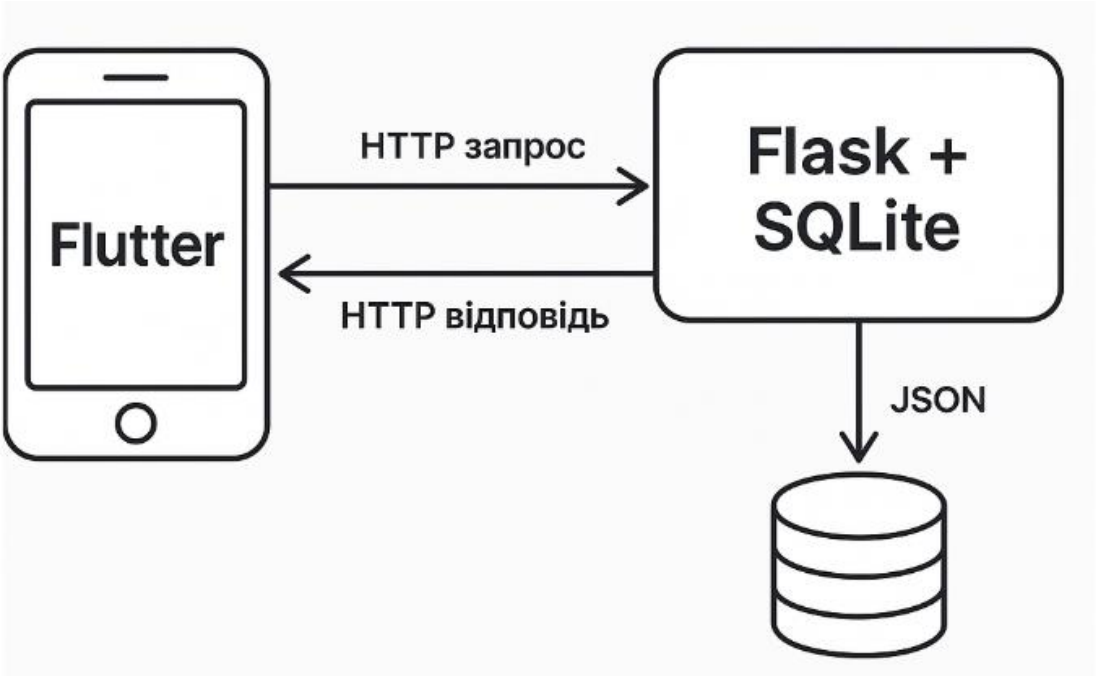

Рисунок 3.1 – Схема архітектури взаємодії між мобільним застосунком на Flutter та сервером на Flask з базою SQLite

Обмін даними відбувається за допомогою простого протоколу HTTP, що забезпечує ефективну та зручну передачу інформації через локальну мережу. У структурі проєкту це дозволяє клієнтському додатку динамічно оновлювати інформацію про меню, надсилати замовлення, а також отримувати список попередніх покупок. Серверна логіка ізольована та може бути масштабована або перенесена в хмарне середовище без зміни архітектури клієнта.

У клієнт-серверному мобільному застосунку, реалізованому в межах кваліфікаційної роботи, логіка взаємодії між клієнтом і сервером базується на обміні HTTP-запитами та використанні формату JSON для передачі даних. Серверна частина створена з використанням фреймворку Flask, який надає зручний інтерфейс для створення RESTful маршрутів, а також взаємодіє з базою даних SQLite за допомогою ORM-бібліотеки SQLAlchemy. На стороні клієнта, написаного за допомогою Flutter, для надсилання запитів використовується пакет http, який дозволяє формувати як GET-, так і POST-запити, обробляти отримані відповіді та серіалізувати/десеріалізувати дані у форматі JSON.

Серед ключових маршрутів серверної частини слід виділити /foods, /orders та /images/<filename>. Перший маршрут (/foods) використовується для отримання переліку доступних страв, які витягуються з бази даних та надсилаються клієнту у вигляді списку об'єктів JSON. У коді цей маршрут виглядає так:

```
@app.route("/foods", methods=["GET"])
def get_foods():
    session = Session()
    foods = session.query(Food).all()
    result = []
    for f in foods:
        result.append({
            "id": f.id,
            "name": f.name,
            "price": f.price,
            "category": f.category,
            "image": f"http://<IP>:8080/images/{f.image}"
        })
    session.close()
    return jsonify(result)
```

Другий маршрут (/orders) підтримує як GET-, так і POST-запити. При GET-запиті сервер повертає всі збережені замовлення у зворотному хронологічному порядку, а POST-запит використовується для прийому нових замовлень від клієнта. Кожне замовлення містить дані про клієнта, спосіб оплати та список вибраних страв. Третій маршрут (/images/<filename>) забезпечує віддачу статичних зображень страв, збережених у папці static/images.

На стороні клієнта, у Flutter, для отримання переліку страв використовується метод _fetchFoods, який виконує GET-запит до маршруту /foods. Відповідь обробляється шляхом декодування JSON-масиву та збереження отриманих даних у локальний список моделей FoodItem:

```
Future<void> _fetchFoods() async {
  final response = await http.get(Uri.parse(getBaseUrl('/foods')));
  if (response.statusCode == 200) {
    final List<dynamic> data = json.decode(response.body);
    setState(() {
      _foods = data.map((item) => FoodItem.fromJson(item)).toList();
    });
  }
}
```

Ще одним важливим прикладом є метод _submitOrder, який формується при оформленні замовлення. Він складає JSON-структуру із даними клієнта та списком товарів, після чого відправляє POST-запит на маршрут /orders:

```
Future<void> _submitOrder() async {
  final order = widget.cartItems.entries.map(
    (entry) => {'item': entry.key.toJson(), 'quantity': entry.value},
  ).toList();
  final customer = {
    'name': _nameController.text.trim(),
    'address': _addressController.text.trim(),
    'payment': _paymentMethod,
  };
  final response = await http.post(
    Uri.parse(getBaseUrl('/orders')),
    headers: {'Content-Type': 'application/json'},
    body: jsonEncode({'order': order, 'customer': customer}),
```

```
  );
  if (response.statusCode == 200) {
    // Підтвердження
  } else {
    // Обробка помилки
  }
}
```

Архітектура дозволяє швидко масштабувати застосунок або змінити структуру обробки запитів без суттєвих змін у коді. Сервер відповідає лише за логіку даних і збереження, тоді як клієнт відповідає за представлення, що повністю відповідає сучасним принципам розробки багаторівневих вебсистем.

У розробленому застосунку дані передаються між мобільним клієнтом (Flutter) та сервером (Flask) у форматі JSON – легкій для читання людині та зручній для обробки машинно-орієнтованій структурі. JSON (JavaScript Object Notation) використовується як для надсилання запитів від клієнта, так і для формування відповідей сервером. Це дозволяє реалізувати єдину, уніфіковану схему обміну, яку легко масштабувати, дебажити та тестувати.

Коли користувач переглядає меню у мобільному застосунку, виконується HTTP GET-запит на маршрут /foods. У відповідь сервер надсилає масив об'єктів JSON, де кожен елемент містить інформацію про страву: ідентифікатор, назву, ціну, категорію та посилання на зображення. Наприклад, відповідь виглядає так:

```
[
  {
    "id": 1,
    "name": "Маргарита",
    "price": 120,
    "category": "Піца",
    "image": "http://192.168.0.101:8080/images/1.png"
  },
  {
    "id": 2,
    "name": "Бургер Чіз",
    "price": 95,
    "category": "Бургери",
    "image": "http://192.168.0.101:8080/images/2.png"
  }
]
```

У випадку оформлення замовлення Flutter-застосунок формує POST-запит на маршрут /orders, який включає у собі JSON-об’єкт із детальною інформацією про клієнта та вибрані страви. Приклад JSON-запиту виглядає наступним чином:

```
{
  "customer": {
    "name": "Іван Іванов",
    "address": "вул. Центральна, 10",
    "payment": "Готівка"
  },
  "order": [
    {
      "item": {
        "id": 2,
        "name": "Бургер Чіз",
        "price": 95,
        "category": "Бургери",
        "image": "http://192.168.0.101:8080/images/2.png"
      },
      "quantity": 3
    }
  ]
}
```

На сервері ці дані обробляються відповідним маршрутом /orders. Сервер розпаковує JSON-тіло запиту, витягує дані клієнта, обчислює замовлення і зберігає його у базі даних SQLite. Кожен запис у таблиці orders містить дані клієнта, спосіб оплати, дату створення та JSON-рядок із замовленими стравами. Наприклад, реальний запис у базі буде у вигляді таблиці 3.1.

Таблиця 3.1 – Запис у базі даних

id	customer_name	customer_address	payment_method	items	created_at
1	Іван Іванов	вул. Центральна, 10	Готівка	[{"item": {...}, "quantity": 3}]	2025-04-29 22:48:50

Перевірити доступність сервера перед запуском застосунку можна кількома способами. Найпростішим є перехід у браузері або Postman за адресою, наприклад: http://192.168.0.101:8080/foods. Якщо сервер працює коректно, у відповідь має бути

повернуто JSON-список. Аналогічно, у Postman можна створити POST-запит до /orders і вручну передати структуру JSON, перевіряючи, як сервер обробляє дані і що записується у базу.

3.2 Функціональність додатку

У мобільному додатку, реалізованому за допомогою Flutter, користувач може переглядати список доступних страв, розбитих за категоріями. Кожна страва представлена карткою, що включає назву, зображення, ціну, а також кнопки для додавання її до кошика та вибору кількості порцій.

При кожному натисканні кнопки «+» або «−» біля певної страви, додаток змінює відповідне значення у мапі _selectedQuantities, яка прив’язана до унікального ідентифікатора кожної страви (item.id). За замовчуванням для кожного продукту встановлюється кількість 1. Якщо користувач збільшує чи зменшує порцію, нове значення зберігається у стані компонента, і повторне відкриття сторінки зберігає останній вибір.

Додавання товару до кошика відбувається через метод _addToCart(). Він перевіряє, чи вже присутній даний елемент у мапі _cart, що виступає в ролі кошика. Якщо страва вже є в кошику, її кількість збільшується на обране значення, інакше додається новий запис з відповідною кількістю. Після кожного додавання виводиться сповіщення про успішну дію, яке інформує користувача про назву страви та кількість, що додана.

Під час оформлення замовлення мобільний застосунок переходить на спеціальний екран, де користувачеві пропонується ввести необхідні персональні дані для доставки. Цей етап реалізовано через декілька текстових полів, що дозволяють зібрати базову інформацію: повне ім’я замовника та адресу доставки. Обидва поля є обов’язковими для заповнення, і в разі, якщо користувач залишає їх порожніми, інтерфейс виводить повідомлення з попередженням про необхідність завершення введення.

Крім того, користувач повинен обрати спосіб оплати. Для цього реалізовано випадаючий список, який містить два варіанти: «Готівка» та «Карта». У разі вибору готівкового розрахунку система не вимагає додаткової інформації. Якщо ж обрано оплату карткою, на екрані динамічно з'являється додатковий блок із полями для введення реквізитів: номера банківської картки (16 цифр), терміну її дії (формат MM/YY) та CVV-коду (3 цифри) . Всі ці поля також мають базову валідацію: перевіряється довжина рядків та наявність значення.

Дані з цих полів збираються у структуру об'єкта customer, який, у поєднанні з кошиком, відправляється на сервер. Важливо зазначити, що навіть якщо дані картки передаються на сервер, вони не зберігаються в базі даних – передача таких даних слугує лише для демонстраційної моделі симуляції безпечного платежу. У разі невведення повної інформації про картку при вибраному способі «Карта», додаток блокує підтвердження та виводить відповідне повідомлення, що сприяє запобіганню помилкам та неповним замовленням.

Процес передачі замовлення на сервер починається з формування об'єкта у форматі JSON, який містить усю необхідну інформацію про користувача та перелік замовлених страв. У Flutter це реалізується у методі _submitOrder, який створює два ключові компоненти: customer і order. Перший включає дані, введені користувачем (ПІБ, адреса, спосіб оплати, реквізити картки – якщо обрано відповідний метод), а другий – це список страв, що були додані до кошика, разом із кількістю кожної з них.

Зібраний JSON передається через HTTP POST-запит на серверний маршрут http://<IP>:8080/orders за допомогою пакета http. Заголовок Content-Type встановлено як application/json, що вказує серверу на формат переданих даних. Запит надсилається асинхронно, після чого застосунок очікує на відповідь від сервера. Якщо статус відповіді дорівнює 200 OK, це означає, що замовлення успішно прийнято, і клієнту виводиться повідомлення про успішну обробку.

На стороні Flask-сервера цей запит приймається маршрутом /orders, де функція place_order() витягує дані з тіла запиту, використовуючи request.get_json(). Далі з нього зчитуються блок customer і масив order. За допомогою SQLAlchemy ці дані

зберігаються в базі даних SQLite. Поле items записується як JSON-рядок у таблицю orders, а додатково фіксується час створення замовлення.

Після завершення транзакції сервер повертає клієнту підтвердження у вигляді JSON-відповіді { "message": "Замовлення збережено" }. У мобільному додатку цю відповідь обробляє інтерфейс, очищуючи кошик і повертаючи користувача на головний екран. Таким чином, реалізується повний цикл – від вибору страв до збереження замовлення у БД – із гарантією надійної передачі та обробки інформації.

3.3 Інтеграція з базою даних й обробка замовлень

У сучасних вебзастосунках важливо забезпечити надійну інтеграцію із системою зберігання даних, що дає змогу ефективно керувати запитами, транзакціями та зберіганням структурованої інформації. У підсистемі обробки замовлень, реалізованій з використанням мікрофреймворку Flask, для взаємодії з базою даних застосовується ORM-бібліотека SQLAlchemy, яка забезпечує гнучкий і зручний механізм роботи з реляційними структурами без необхідності писати сирі SQL-запити.

Обробка замовлення починається з надсилання клієнтського запиту на визначений маршрут сервера, зокрема /orders, методом POST. Дані передаються у форматі JSON і включають повну інформацію про користувача: ім’я, адресу доставки, спосіб оплати, а також список вибраних страв із відповідними характеристиками (назва, кількість, ціна тощо). У Flask ці дані обробляються за допомогою функції request.get_json(), яка розпаковує JSON-об’єкт у зручний для подальшої обробки формат – словник Python.

Після отримання інформації формується об’єкт моделі Order, що є відображенням таблиці orders у базі даних.

Модель містить такі основні поля:

- id (унікальний ідентифікатор замовлення),
- customer_name (ім’я клієнта),
- address (адреса доставки),

- payment_method (спосіб оплати),
- items (список замовлених страв у вигляді JSON-рядка),
- timestamp (мітка часу створення замовлення).

Завдяки можливостям SQLAlchemy, визначені у Python класи моделей автоматично транслюються у відповідні SQL-запити. Після створення нового екземпляра моделі Order, його додають до сесії за допомогою db.session.add(order), а далі виконується збереження змін через db.session.commit(). Такий підхід гарантує, що дані будуть надійно записані у базу з дотриманням транзакційної цілісності. У випадку помилок з'єднання або порушення обмежень цілісності, SQLAlchemy дозволяє коректно обробити виключення та повернути користувачу відповідне повідомлення про помилку.

Окрім створення замовлень, важливу роль відіграє реалізація механізму їх зчитування та відображення. Це реалізовано через маршрут /orders, але вже з методом GET, що дозволяє отримати список усіх наявних замовлень. Запити до цього маршруту формують відповідь у вигляді серіалізованого JSON-масиву, де кожен елемент є окремим замовленням. Для кожного з них надається базова інформація: ім’я клієнта, адреса, тип оплати, а також декодований список страв, який формується через обробку JSON-рядка методом json.loads().

Додатково до маршруту можуть бути реалізовані фільтрація, сортування або пошук за певними параметрами – наприклад, відображення замовлень лише за останній день або лише тих, що мають певний статус (очікує, готується, доставлено). У поточній реалізації замовлення сортуються за датою створення – від найновіших до найстаріших, що дозволяє зручно переглядати останні дії користувача.

На боці клієнтського інтерфейсу (мобільного застосунку) отримані дані використовуються для побудови сторінки історії замовлень, перегляду статусу або повторного замовлення. Такий підхід дозволяє швидко взаємодіяти з даними в режимі реального часу, що суттєво покращує зручність користування сервісом.

Інтеграція з базою даних через ORM дозволяє зменшити ймовірність помилок, пов’язаних із некоректними SQL-запитами, підвищує рівень абстракції та підтримує модульність коду. У випадку зміни структури таблиці достатньо оновити лише модель у Python, після чого SQLAlchemy автоматично синхронізує зміни при створенні

міграцій. Це значно спрощує обслуговування та масштабування застосунку в майбутньому.

3.4 Тестування та аналіз отриманих результатів

Після завершення розробки як серверної частини на Flask з використанням SQLite, так і клієнтського інтерфейсу на Flutter, відбувся етап інтеграційного тестування, під час якого перевірялася стабільність зв'язку, обробка запитів і загальна відповідність очікуваній логіці роботи.

У процесі перевірки основну увагу було приділено коректності завантаження меню, фільтрації за категоріями, обробці замовлень, взаємодії з базою даних та відображенню результатів. Було змодельовано кілька типових сценаріїв користувача: перегляд списку страв, формування кошика, вибір способу оплати, надсилання замовлення на сервер та його обробка. Кожен з етапів тестувався як окремо, так і в комбінації з іншими функціями. Оцінювались як технічні аспекти роботи, так і зручність інтерфейсу з позиції кінцевого користувача.

На наведеному зображенні (рисунок 3.2) продемонстровано результат тестування функціоналу фільтрації за категоріями у мобільному додатку. У цьому конкретному випадку користувач обрав категорію «Піца», що призвело до відображення лише відповідних позицій з бази даних – «Маргарита» та «Чотири сири». Це підтверджує коректну роботу механізму фільтрації, реалізованого у Flutter, який взаємодіє із загальним списком страв, завантаженим із сервера.

Під час тестування перевірялося, чи оновлюється список без потреби повторного запиту до сервера, і чи не з'являються товари інших категорій при виборі конкретної. Функція фільтрації працює на клієнтській стороні, застосовуючи відповідний критерій до вже отриманого списку страв. Кнопка сортування праворуч також працює без конфліктів із фільтрацією, дозволяючи додатково впорядковувати результати за назвою чи ціною. Тестовий сценарій підтвердив, що інтерфейс реагує швидко й стабільно, відображаючи лише ті позиції, які відповідають обраній категорії.

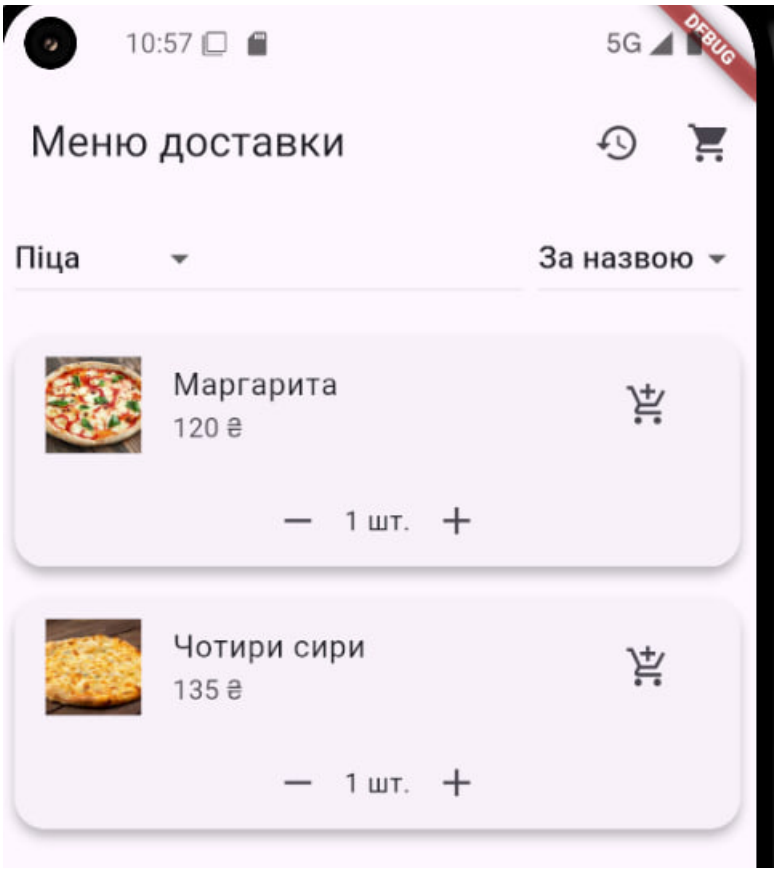

Рисунок 3.2 – Результат фільтрації меню за категорією "Піца" під час тестування мобільного застосунку

На зображенні 3.3 продемонстровано процес сортування елементів меню в мобільному застосунку «Меню доставки». Інтерфейс дозволяє користувачу обрати критерій впорядкування страв за допомогою випадаючого списку, який містить два основні параметри: «За назвою» та «За ціною». Це надає гнучкість у навігації по асортименту та полегшує пошук бажаної позиції, що особливо важливо при великій кількості товарів. Таке рішення відповідає принципам UX-дизайну та реалізує зручний користувацький сценарій.

На представленому прикладі видно, що при виборі сортування «За назвою», список страв впорядкований в алфавітному порядку: спочатку відображається «Бургер Чіз», далі «Бургер BBQ», «Маргарита», «Чотири сири» і «Каліфорнія». Це підтверджує правильну роботу відповідної логіки у додатку, реалізованої за допомогою Dart. Аналогічно, при виборі опції «За ціною», список буде

перебудовано за зростанням вартості. Такий підхід не лише спрощує взаємодію користувача із системою, а й демонструє тестування інтерактивних UI-елементів.

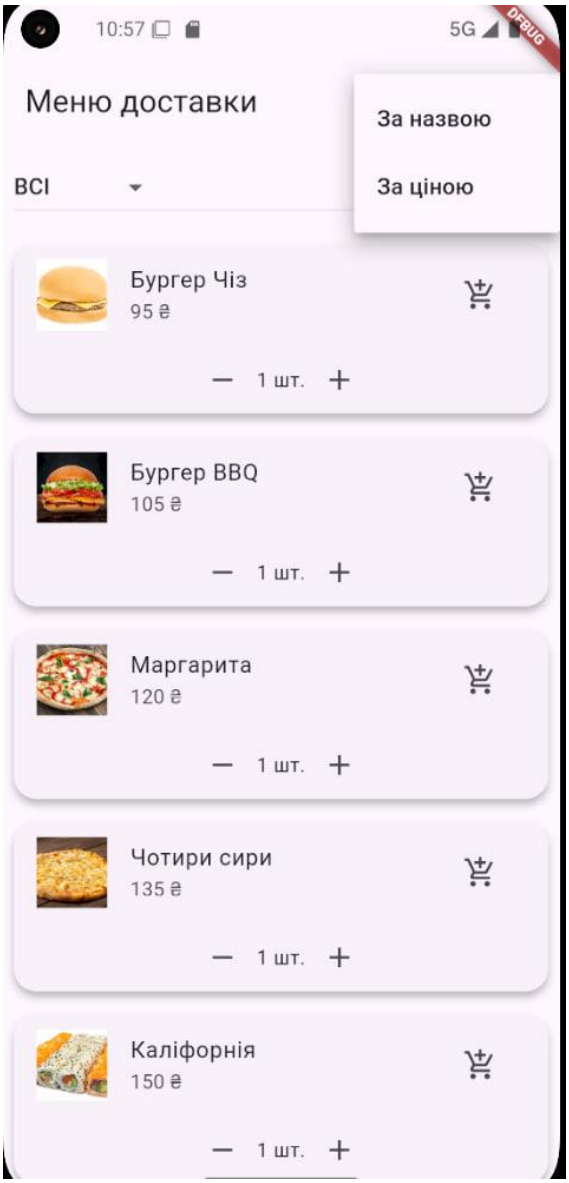

Рисунок 3.3 – Візуалізація функції сортування страв за назвою та ціною в мобільному інтерфейсі

На зображенні 3.4 зображено один із ключових етапів процесу оформлення замовлення в мобільному застосунку – екран «Кошик». Центральною частиною інтерфейсу є можливість змінювати кількість обраної страви за допомогою кнопок «–» та «+», які дозволяють зменшити або збільшити кількість одиниць замовлення. Праворуч відображається елемент інтерфейсу з червоним фоном та іконкою кошика,

що активується при свайпі вбік – ця дія надає змогу швидко видалити товар із кошика, що реалізовано через компонент Dismissible у Flutter.

Нижче розташовано форму для введення особистих даних клієнта. Користувач заповнює поля для ПІБ та адреси доставки, що необхідні для обробки та виконання замовлення. Окремий елемент інтерфейсу відповідає за вибір способу оплати – у цьому випадку вказано «Готівка». Завдяки випадаючому списку можна змінити метод на інший, наприклад «Карта», що активує додаткові поля для введення банківських реквізитів. Таким чином, тестування цієї частини дозволило перевірити, чи коректно обробляються дії користувача, відображаються дані, та чи відбувається збереження у відповідні змінні стану додатку.

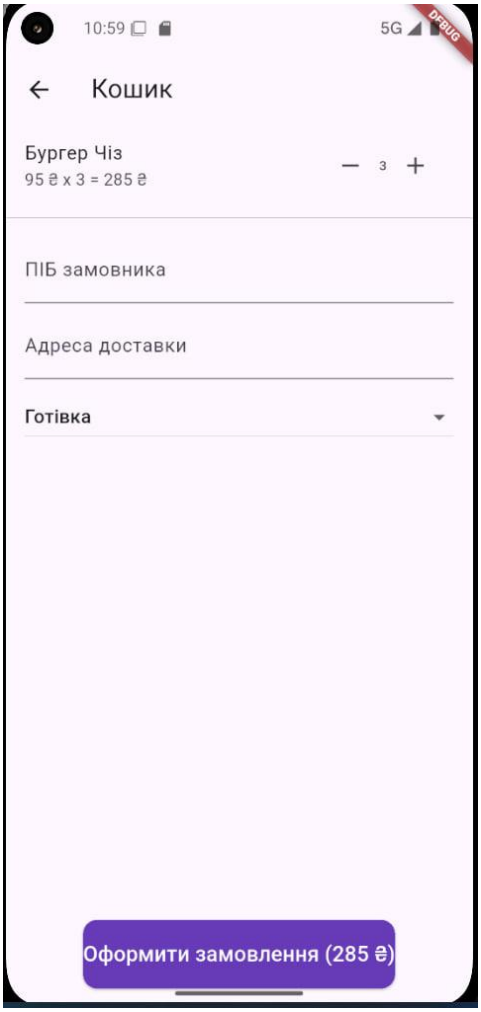

Рисунок 3.4 – Екран підтвердження замовлення з розрахованою сумою і кнопкою надсилання даних на сервер

На зображенні 3.5 представлено екран «Кошик», де реалізована функціональність видалення замовлення за допомогою свайпу. Користувач може провести пальцем ліворуч по елементу списку, щоб активувати червоне тло з іконкою смітника – візуальне підтвердження готовності до видалення. Цей інтерфейсний жест інтуїтивно зрозумілий та відповідає стандартам мобільних UI, забезпечуючи швидке та зручне редагування вмісту кошика.

У контексті тестування цей функціонал дозволяє перевірити, чи правильно оновлюється стан кошика після видалення товару, чи коректно перераховується загальна вартість та чи не виникають помилки при подальшому оформленні замовлення. Особливо важливо, що зміни застосовуються локально у стані додатку, що засвідчує ефективність керування внутрішньою логікою через Stateful-архітектуру Flutter.

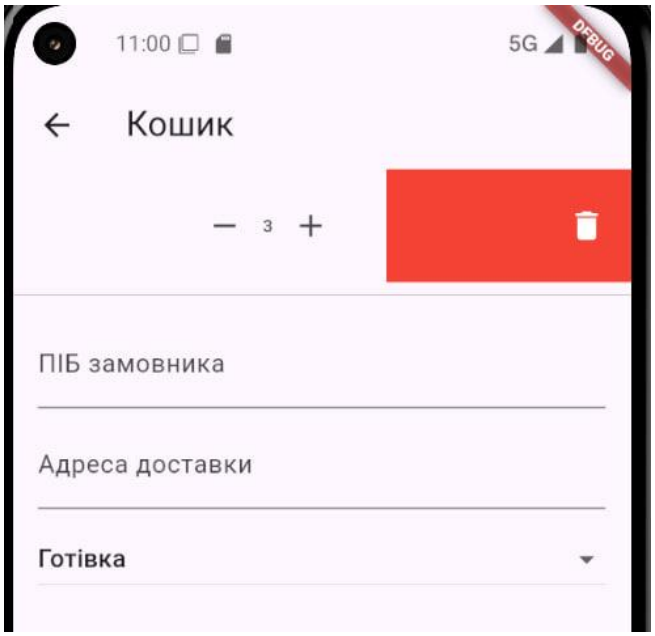

Рисунок 3.5 – Реалізація можливості видалення замовлення зі списку шляхом свайпу ліворуч

На зображенні представлено діалогове вікно підтвердження дії, яке з’являється перед остаточним оформленням замовлення. Це рішення є важливою частиною UX-дизайну, оскільки воно запобігає випадковому надсиланню незаповнених або помилкових даних [30]. Вікно містить зрозуміле запитання та два чіткі варіанти дій – скасування або підтвердження.

Така реалізація дозволяє користувачеві зосередитися, ще раз перевірити введені дані та усвідомлено завершити процес оформлення. З технічної точки зору, це діалог AlertDialog з обробниками подій на кнопках, які або закривають вікно, або запускають функцію надсилання запиту на сервер. Цей етап був протестований в процесі розробки для перевірки, чи дійсно кнопка підтвердження активує функцію _submitOrder() лише після усіх валідацій.

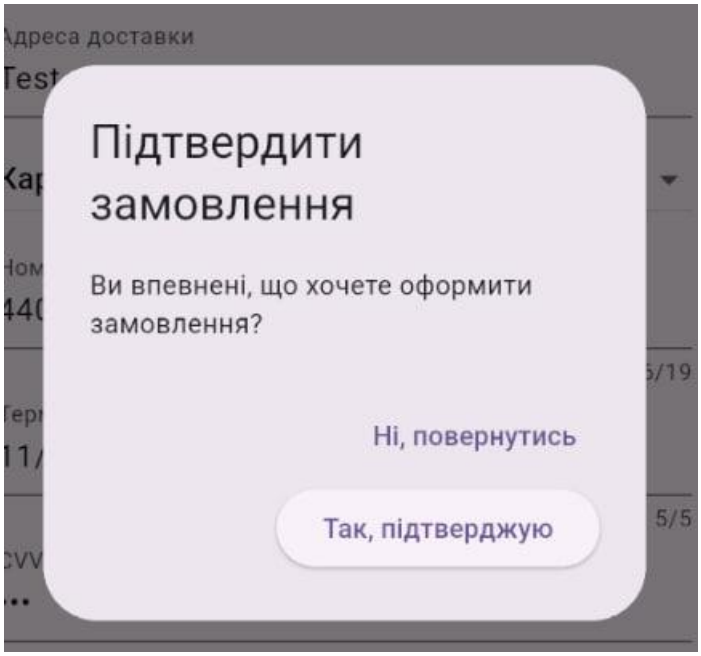

Рисунок 3.6 – Вікно підтвердження оформлення замовлення

На зображенні 3.7 представлено інтерфейс перегляду останнього замовлення, який реалізовано у вкладці з іконкою годинника у верхньому правому куті головного екрану. Користувач може побачити основну інформацію: ім’я та адресу отримувача, спосіб оплати, точну дату й час оформлення, а також деталі кожної позиції в замовленні (назва, кількість, підсумкова вартість).

Цей екран створений із використанням елементів ListTile, Text, Divider, що дозволяє компактно структурувати дані. Він відіграє важливу роль у тестуванні збереження замовлень, оскільки дає змогу переконатись у правильності передачі й обробки інформації. Формат дати та відображення страв повністю відповідають даним, збереженим на сервері у базі SQLite, що засвідчує цілісність і коректність обміну даними.

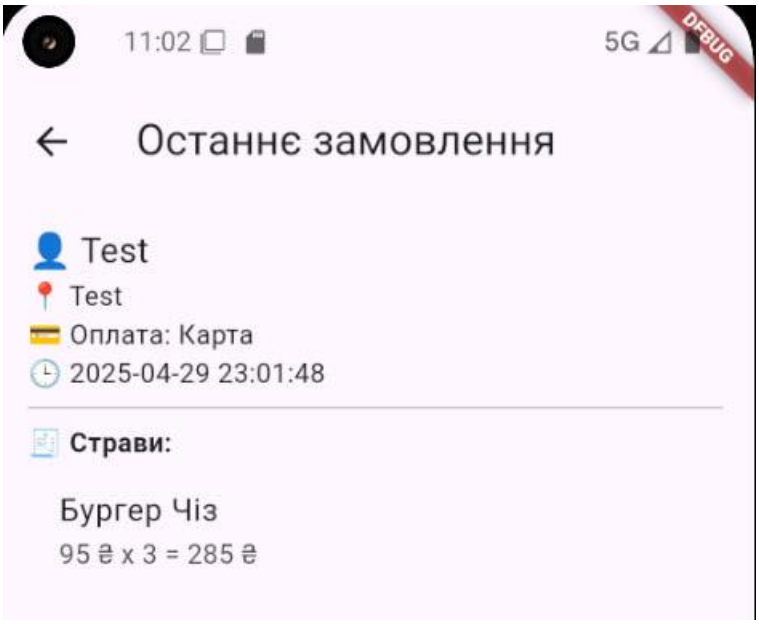

Рисунок 3.7 – Екран перегляду останнього замовлення

У цьому розділі було детально розглянуто всі етапи розробки клієнт-серверного мобільного застосунку для оформлення та обробки замовлень на доставку їжі. Було реалізовано архітектуру, що поєднує сервер на Flask із базою даних SQLite та клієнтську частину на Flutter, яка забезпечує зручну взаємодію користувача з системою [35]. Було розроблено модулі обробки запитів, маршрути для надсилання та отримання даних, а також зручний і інтуїтивний інтерфейс для кінцевого користувача.

Тестування підтвердило стабільну роботу всієї системи, правильну обробку даних та збереження замовлень у базі. Було протестовано всі функції – від фільтрації меню до фінального оформлення й перегляду історії замовлень. Застосунок продемонстрував коректну передачу JSON-запитів, обробку форм введення та підтримку різних методів оплати. Таким чином, поставлені задачі реалізації логіки взаємодії, створення бази даних і побудови UI було успішно виконано.

Тестування підтвердило стабільну роботу всієї системи, правильну обробку даних та збереження замовлень у базі. Було протестовано всі функції – від фільтрації меню до фінального оформлення й перегляду історії замовлень. Застосунок продемонстрував коректну передачу JSON-запитів, обробку форм вве-

дення та підтримку різних методів оплати. Таким чином, поставлені задачі реалізації логіки взаємодії, створення бази даних і побудови UI було успішно виконано.

Досягнутий результат відображено в таблиці 3.2, де наведено основні реалізовані функціональні модулі системи. Кожен з них відповідає окремому етапу користувацької взаємодії – від перегляду меню з можливістю сортування, до підтвердження, оформлення й збереження замовлення. Завдяки цій структурі забезпечено зручний і повний процес оформлення замовлень у мобільному застосунку.

Таблиця 3.2 – Основні реалізовані функціональні модулі мобільного застосунку

Функціональний модуль	Опис реалізації
Меню доставки	Виведення списку страв з фільтрацією та сортуванням
Кошик	Додавання та зменшення кількості товарів, видалення за свайпом
Форма оформлення замовлення	Введення ПІБ, адреси, вибір типу оплати, підтримка введення реквізитів картки
Підтвердження замовлення	Діалог підтвердження дії перед надсиланням
Відправка на сервер	Формування JSON-структури та надсилання POST-запиту
Збереження в базі даних	Обробка на Flask, запис у SQLite (таблиця orders)
Перегляд останнього замовлення	Вивід останньої покупки з деталями
Обробка зображень	Підвантаження файлів зі шляху /static/images/<назва> у Flutter через URL

У результаті реалізації функціональних модулів забезпечено повний цикл взаємодії користувача із системою – від вибору страв до збереження замовлення в базі даних. Модульна структура спрощує підтримку та розвиток застосунку, а також забезпечує гнучкість і зручність у використанні як для користувача, так і для розробника.

ВИСНОВКИ

При виконанні роботи розглянуто практичну задачу, зокрема, розробка програмного модуля розподіленої системи обміну даними в реальному часі вигляді вебзастосунку для замовлення та доставки їжі й отримано результати.

1. Проведено аналіз сучасних підходів до розробки мобільних застосунків, зокрема нативного та кросплатформного підходів, що дало змогу обґрунтовано обрати Flutter як оптимальну технологію для реалізації кросплатформного застосунку з високою продуктивністю та сучасним інтерфейсом.

2. Розроблено мобільний застосунок для замовлення страв, який реалізує клієнт-серверну взаємодію з передачею даних у форматі JSON. У клієнтській частині використано фреймворк Flutter, що забезпечив кросплатформенність, адаптивний інтерфейс і високу продуктивність.

3. Розроблено серверну частину додатку з використанням Python та Flask, яка відповідає за обробку HTTP-запитів, зберігання інформації в базі даних, а також логіку формування й обробки замовлень й забезпечує швидку реалізацію маршрутизації й обробку запитів.

4. Інтегровано локальну реляційну базу даних SQLite через ORM-бібліотеку SQLAlchemy, що дало змогу уникнути ручного написання SQL-запитів й забезпечило реалізацію стабільного збереження інформації про клієнтів, замовлення та позиції меню, з дотриманням цілісності даних та підтримкою транзакцій.

5. Реалізовано основні функціональні модулі мобільного за стосунку, які повністю охоплюють цикл користувацької взаємодії – від вибору товару до збереження покупки.

6. Розроблено гнучку структуру зчитування, збереження та обробки даних, що дозволяє легко масштабувати систему, переносити її на віддалені сервери та адаптувати під інші СУБД (наприклад, PostgreSQL), не змінюючи логіку взаємодії.

7. Здійснено тестування реалізованого застосунку на Android-пристроях, у результаті чого підтверджено стабільність роботи функціональних модулів, коректність взаємодії з сервером і базою даних, а також відповідність функціоналу поставленим вимогам.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. PNN Soft. Плюси і мінуси кросплатформенної і нативної розробки: веб-сайт. URL: https://pnn.com.ua/ua/blog/detail/pros-and-cons-of-cross-platform-and-native-development (дата звернення: 15.04.2025 р.).
2. Клієнт-серверна архітектура: веб-сайт. URL: https://uk.wikipedia.org/wiki/Клієнт-серверна_архітектура (дата звернення: 15.04.2025 р.).
3. Android. Презентація в Gamma: веб-сайт. URL: https://gamma.app/docs/Android-pxqzj4e2s6kn51d?mode=doc (дата звернення: 15.04.2025 р.).
4. Apple iOS. Опис ОС: веб-сайт. URL: https://ipkey.com.ua/uk/faq/935-apple-ios.html (дата звернення: 15.04.2025 р.).
5. Wezom. Відмінності в розробці додатків для Android та iOS: веб-сайт. URL: https://wezom.com.ua/ua/blog/vidminnosti-v-rozrobtsi-dodatkiv-dlya-android-ta-ios-9-skladovih-dlya-porivnyannya (дата звернення: 15.04.2025 р.).
6. React Native: веб-сайт. URL: https://en.wikipedia.org/wiki/React_Native (дата звернення: 15.04.2025 р.).
7. PyCharm IDE: веб-сайт. URL: https://en.wikipedia.org/wiki/PyCharm (дата звернення: 15.04.2025 р.).
8. PyInstaller. Документація проєкту: веб-сайт. URL: https://pyinstaller.org/en/stable/ (дата звернення: 15.04.2025 р.).
9. Aiohttp: веб-сайт. URL: https://pypi.org/project/aiohttp/ (дата звернення: 15.04.2025 р.).
10. SQLAlchemy ORM Tutorial for Python Developers: веб-сайт. URL: https://overiq.com/sqlalchemy-101 (дата звернення: 15.04.2025 р.).
11. SQLite Documentation. Features: веб-сайт. URL: https://www.sqlite.org/features.html (дата звернення: 15.04.2025 р.).
12. Google Developers. Material Design 3: веб-сайт. URL: https://m3.material.io (дата звернення: 15.04.2025 р.).

13. Flutter Documentation: веб-сайт. URL: https://docs.flutter.dev (дата звернення: 15.04.2025 р.).

14. Методичні вказівки до виконання практичних робіт з дисципліни «Техніко-економічне обґрунтування розробки комп'ютерних систем"/ Н.Я. Савка, І.Р. Паздрій / Під ред. О.М. Березького. Тернопіль: ТНЕУ, 2019. 40 с.

15. Методичні вказівки до оформлення курсових, звітів про проходження практики, випускних кваліфікаційних робіт для студентів спеціальності «Комп'ютерна інженерія» / І.В. Гураль, Л.О.Дубчак / під ред. О.М. Березького. Тернопіль: ТНЕУ, 2019. 34 с.

16. Методичні рекомендації до виконання кваліфікаційної роботи з освітнього ступеня "Бакалавр" спеціальності 123 «Комп'ютерна інженерія» галузі знань 12 Інформаційні технології / О.М. Березький, Л.О.Дубчак, Г.М. Мельник, Ю.М. Батько, О.Й. Піцун / Під ред. Л.О.Дубчак. Тернопіль: ЗУНУ, 2024. 54 с.

17. Чорний В.В., Твердун Я.О. Технології розробки систем моніторингу проєктів та обміну даними в реальному часі / Матеріали II Всеукраїнської науково-практичної конференції «Інтелектуальні комп'ютерні системи та мережі», 20 травня 2025 р. – Тернопіль: ЗУНУ, 2025. – С. 82-83 – веб-сайт. URL: https://ki.wunu.edu.ua/conference/archive.html (дата звернення: 29.05.2025 р.).

18. Таненбаум А. С., Стін М. Розподілені системи: принципи та парадигми. К. : Вільямс, 2010. 624 с.

19. Дьяконов А. М. Основи програмної інженерії: навч. посіб. Львів: ЛНУ імені Івана Франка, 2021. 268 с.

20. Буч Г. Об'єктно-орієнтований аналіз та проектування з прикладами застосування UML. К. : Діалектика, 2005. 450 с.

21. Sommerville I. Software Engineering. 10th ed. Pearson, 2016. 840 p.

22. Pressman R. S. Software Engineering*: A* Practitioner's Approach. McGraw-Hill, 2014. 896 p.

23. VS Code Documentation: веб-сайт. URL: https://code.visualstudio.com/docs (дата звернення: 29.05.2025 р.).

24. Dart Language Tour: веб-сайт. URL: https://dart.dev/guides/language/language-tour (дата звернення: 29.05.2025 р.).

25. Flask Documentation: веб-сайт. URL: https://flask.palletsprojects.com/en/2.3.x/ (дата звернення: 29.05.2025 р.).

26. REST API Design – Best Practices: веб-сайт. URL: https://restfulapi.net/rest-api-design-tutorial-with-example/ (дата звернення: 29.05.2025 р.).

27. WebSockets – Mozilla Developer Network: веб-сайт. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API (дата звернення: 29.05.2025 р.).

28. Ковальчук О. В. Застосування мови Python та фреймворку Flask для реалізації REST-сервісів // Інформаційні технології та комп'ютерна інженерія. – 2021. – №1. – С. 45–51.

29. Firebase Authentication Documentation: веб-сайт. URL: https://firebase.google.com/docs/auth (дата звернення: 29.05.2025 р.).

30. JSON формат – офіційна документація: веб-сайт. URL: https://www.json.org/json-en.html (дата звернення: 29.05.2025 р.).

31. UX Planet. Керівництво з проектування UI: веб-сайт. URL: https://uxplanet.org/ (дата звернення: 29.05.2025 р.).

32. Стадник Р. О. Локальні бази даних у мобільних додатках на прикладі SQLite // Системи обробки інформації. — 2022. — № 2(175). — С. 112–117.

33. Білик М. В. Застосування HTTP-запитів у мобільних застосунках з використанням Flutter // Наукові записки НУ «Львівська політехніка». — 2023. — №78. — С. 66–70.

34. Горбач Л. І. Аналіз продуктивності мобільних застосунків, розроблених на Flutter: дипломна робота. — Київ: КПІ ім. Ігоря Сікорського, 2023. — 65 с.

35. OWASP Mobile Security Testing Guide: веб-сайт. URL: https://owasp.org/www-project-mobile-security-testing-guide/ (дата звернення: 29.05.2025 р.).

36. SQLite Tutorial by Tutorialspoint: веб-сайт. URL: https://www.tutorialspoint.com/sqlite/ (дата звернення: 29.05.2025 р.).