Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти перший (бакалаврський)

Кросплатформний застосунок для обміну рецептами з інтегрованим AI-помічником

(тема)

Виконав:
здобувач 4 року навчання,
групи КІУКІ-21-4
Олександр ЛЕОНОВ
(власне ім'я, прізвище)

Спеціальність
123 «Комп'ютерна інженерія»
(код і повна назва спеціальності)

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

Освітня програма
Комп'ютерна інженерія
(повна назва освітньої програми)

Керівник: ас. Єгор КОРНІЄНКО
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри ЕОМ _____ Андрій КОВАЛЕНКО
(підпис) (власне ім'я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти перший (бакалаврський)

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

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

Освітня програма Комп’ютерна інженерія
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
“____” _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Леонову Олександру Максимовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Кросплатформний застосунок для обміну рецептами з інтегрованим AI-помічником

затверджена наказом по університету від “ 26 ” травня 2025 р. № 424Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 17 червня 2025 р.

3. Вхідні дані до роботи 1) Service Layer Architecture; 2) IDE: Visual Studio Code; 3) мова програмування: JavaScript.

4. Перелік питань, що потрібно опрацювати у роботі
1) аналіз проблеми та огляд існуючих рішень;
2) вибір технології розробки;
3) розробка програмних модулів;
4) відлагодження і тестування програмних модулів;
5) висновки.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій

Слайд-презентація – 21 слайдів

6. Консультанти розділів роботи (заповнюється 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	Строк / терміни виконання етапів роботи	Примітка
1	Аналіз проблеми та огляд існуючих рішень	27.05.25-30.05.25	
2	Вибір технології розробки	31.05.25-01.06.25	
3	Розробка програмних модулів	02.06.25 – 07.06.25	
4	Відлагодження і тестування програмних	08.06.25 – 10.06.25	
	модулів		
5	Оформлення матеріалів кваліфікаційної роботи	11.06.25-12.06.25	
6	Подання кваліфікаційної роботи керівникові та	13.06.25-14.06.25	
	її попередній захист		
7	Подання кваліфікаційної роботи на рецензу-	15.06.25-16.06.25	
	вання		

Дата видачі завдання " 26 " травня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ ас. Єгор КОРНІЄНКО
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 73 с., 19 рис., 1 табл., 1 дод., 21 джерел.

МОБІЛЬНИЙ ЗАСТОСУНОК, ІНТЕРФЕЙС КОРИСТУВАЧА, АВТОРИЗАЦІЯ, ПУБЛІКАЦІЯ, БАЗА ДАНИХ, API, AI.

Метою кваліфікаційної роботи є розробка мобільного застосунку, що дозволяє користувачам знаходити рецепти, а також публікувати власні.

У ході виконання курсового кваліфікаційної роботи було проведено аналіз існуючих рішень для пошуку та публікації кулінарних рецептів. Розроблений застосунок включає функціонал реєстрації та авторизації користувачів, можливість додавання нових рецептів із текстовими описами та зображеннями, а також механізм пошуку за ключовими словами та категоріями.

Для реалізації проєкту було створено серверну частину, що забезпечує обмін даними між базою даних та клієнтською частиною. Інтерфейс застосунку розроблено з акцентом на зручність використання та сучасний дизайн. Особливу увагу приділено забезпеченню безпеки даних користувачів та швидкодії застосунку.

Результатом роботи став повноцінний мобільний застосунок, який може бути використаний як інструмент для пошуку кулінарних ідей та спільного обміну рецептами серед користувачів.

ABSTRACT

Bachelor’s thesis: 73 pages, 19 figures, 1 tables, 1 appendices, 21 sources.

MOBILE APPLICATION, USER INTERFACE, AUTHORIZATION, PUBLISHING, DATABASE, API, AI.

The purpose of the qualification work is to develop a mobile application that allows users to find recipes and publish their own.

During the course of the qualification work, we analyzed existing solutions for searching and publishing recipes. The developed application includes user registration and authorization functionality, the ability to add new recipes with text descriptions and images, and a search mechanism for keywords and categories.

To implement the project, we created a server side that provides data exchange between the database and the client side. The application interface was developed with a focus on usability and modern design. Particular attention was paid to ensuring the security of user data and application performance.

The result is a full-fledged mobile application that can be used as a tool for finding culinary ideas and sharing recipes among users.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

ВММ – велика мовна модель

ІШІ – штучний інтелект

AI – Artificial Intelligence

API – Application Programming Interface

CRUD – Create, Read, Update, Delete

CSRF – Cross-Site Request Forgery

DTO – Data Transfer Object

JWT – Json Web Token

LLaMA – Large Language Model Architecture

MVP – Minimum Viable Product

ORM – Object-Relational Mapping

SQL – Structured Query Language

TOTP – Time-based One-Time Password

UI – User Interface

XSS – Cross-Site Scripting

ВСТУП

У сучасному світі мобільні застосунки відіграють важливу роль у спрощенні повсякденного життя людей. Однією з популярних сфер використання є кулінарія, де цифрові рішення сприяють пошуку рецептів, обміну ідеями та збереженню улюблених страв. На сьогодні існує велика кількість платформ для публікації кулінарного контенту, однак багато з них не враховують потреби в інтуїтивному інтерфейсі, високій продуктивності та гнучкості в користуванні.

Метою роботи є створення мобільного застосунку для пошуку та публікації рецептів із використанням сучасних технологій. Сферами застосування розробленого проекту можуть бути кулінарні блоги, освітні проєкти, платформи для обміну досвідом та індивідуальні потреби користувачів.

Актуальність даної роботи полягає в потребі користувачів у зручних платформах для швидкого отримання інформації. Застосунок, розроблений у рамках цього проєкту, дозволяє вирішити низку практичних задач, зокрема полегшити доступ до кулінарного контенту та надати можливість ділитися власними рецептами в інтерактивному середовищі.

Розробка застосунку пов’язана з іншими дослідженнями в галузі мобільних технологій та програмного забезпечення, орієнтованого на роботу з базами даних і управління користувацькими сесіями [1].

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Огляд технологій реалізації

Розробка сучасного мобільного застосунку може здійснюватися за кількома основними підходами, кожен з яких має свої переваги та недоліки. Нативний підхід передбачає створення окремих застосунків для кожної платформи з використанням специфічних для них мов програмування та інструментів: Swift/Objective-C для iOS та Kotlin/Java для Android. Цей підхід забезпечує найвищу продуктивність, повний доступ до можливостей операційної системи та найкращий користувацький досвід, проте вимагає окремих команд розробників та значних витрат часу на підтримку двох кодових баз [2].

Гібридний підхід представлений такими технологіями, як Apache Cordova (PhoneGap) або Ionic, де застосунок розробляється за допомогою веб-технологій (HTML, CSS, JavaScript) та запускається у веб-контейнері всередині нативної оболонки. Хоча цей підхід дозволяє швидко портувати веб-застосунки на мобільні платформи, він часто страждає від проблем з продуктивністю та обмеженим доступом до нативних API [3].

Кросплатформний підхід займає проміжне положення між нативним та гібридним, дозволяючи писати код один раз та компілювати його для різних платформ. Основні представники цього підходу включають React Native, Flutter, Xamarin та Kotlin Multiplatform Mobile. Цей підхід дозволяє зберегти значну частину спільного коду, забезпечуючи при цьому близьку до нативної продуктивність [4].

Progressive Web Apps (PWA) представляють собою веб-застосунки з можливостями, які наближають їх до нативних мобільних застосунків. Вони можуть працювати офлайн, отримувати push-повідомлення та встановлюватися на домашній екран, проте все ще мають обмеження

порівняно з повноцінними мобільними застосунками [5].

Розробка сучасного мобільного застосунку передбачає також ретельний вибір інструментів і технологій, які здатні забезпечити стабільну роботу, ефективну розробку та можливість масштабування в майбутньому. У цьому контексті особливої уваги заслуговують засоби створення кросплатформних інтерфейсів, бібліотеки управління станом, серверні технології, системи зберігання даних і сервіси штучного інтелекту.

Одним з ключових аспектів є вибір фреймворку для клієнтської логіки застосунку. На сьогодні поширеними підходами до кросплатформної розробки є використання таких технологій, як React Native, Flutter та Kotlin Multiplatform. React Native, що розроблений компанією Meta, дозволяє створювати застосунки одночасно для Android та iOS із використанням JavaScript. Завдяки цьому значно скорочується час розробки, а активна спільнота забезпечує наявність великої кількості готових рішень. Альтернативою є Flutter, фреймворк від Google, який базується на мові Dart і характеризується високою продуктивністю та розвиненою системою рендерингу інтерфейсів. Ще одним варіантом є Kotlin Multiplatform, що дозволяє писати спільну бізнес-логіку, однак потребує окремої реалізації інтерфейсу для кожної платформи, що збільшує витрати часу та ресурсів. Вибір серед цих технологій залежить від пріоритетів проєкту: швидкість розробки, глибина нативної інтеграції або вимоги до UI.

У процесі розробки застосунку важливим є також питання управління станом. Для вирішення цього завдання можна застосовувати як класичні рішення на кшталт Redux, так і більш легковажні бібліотеки, зокрема Zustand. Redux забезпечує чітку централізацію стану, але потребує значної кількості шаблонного коду, що ускладнює підтримку. У той час як Zustand орієнтований на простоту і швидкість реалізації — він дозволяє створювати компактні і гнучкі сховища з мінімальними зусиллями. Іншим варіантом є MobX, який базується на реактивному програмуванні, однак його складність не завжди виправдана в невеликих застосунках.

З боку серверної логіки застосунку можливими варіантами є Node.js із фреймворком Express, Django (Python) або Spring Boot (Java). Node.js є асинхронним середовищем виконання JavaScript, що дозволяє ефективно обробляти велику кількість запитів і має широку підтримку через npm. Express, як легкий фреймворк для створення REST API, дозволяє швидко розгорнути серверну частину та інтегрувати її з базами даних. Django, у свою чергу, є повноцінним фреймворком з вбудованою ORM, адміністративною панеллю та строгим дотриманням патерну Model–View–Template. Він більше підходить для проєктів зі складною бізнес-логікою. Spring Boot, незважаючи на високу надійність і масштабованість, зазвичай застосовується у великих корпоративних рішеннях і вимагає більше часу на освоєння та налаштування [6].

Ще одним важливим елементом є вибір системи зберігання даних. Документоорієнтовані бази, зокрема MongoDB, пропонують гнучкий підхід до структурування інформації, що особливо зручно при роботі з даними, які не мають фіксованої схеми. MongoDB добре інтегрується з Node.js та підтримує горизонтальне масштабування, реплікацію і зберігання JSON-подібних документів. У той же час, реляційні бази, як-от PostgreSQL або MySQL, забезпечують високий рівень консистентності, складні запити, транзакції та надійність. Вибір між NoSQL та SQL-рішеннями визначається, передусім, структурою даних, потребою в гнучкості або суворій цілісності, а також очікуваним навантаженням.

Безпека мобільних застосунків є критично важливим аспектом, що потребує комплексного підходу до реалізації. Одним з найпоширеніших механізмів авторизації є JSON Web Token (JWT) — компактний спосіб безпечної передачі інформації між сторонами у вигляді JSON-об'єкта. JWT складається з трьох частин: заголовка, корисного навантаження та підпису, що дозволяє верифікувати автентичність та цілісність токену. Основними перевагами JWT є відсутність необхідності зберігати сесії на сервері та можливість включення інформації про користувача безпосередньо в токен.

Альтернативними рішеннями для авторизації є OAuth 2.0 та OpenID Connect, які особливо корисні при інтеграції з зовнішніми провайдерами (Google, Facebook, Apple). OAuth 2.0 забезпечує делеговану авторизацію, дозволяючи застосункам отримувати обмежений доступ до ресурсів користувача без розкриття паролів. OpenID Connect розширює OAuth 2.0, додаючи рівень автентифікації та стандартизовані способи отримання інформації про користувача.

Для підвищення безпеки також застосовуються технології двофакторної автентифікації (2FA), включаючи Time-based One-Time Password (TOTP) та SMS-коди. TOTP, реалізований в таких застосунках як Google Authenticator або Authy, генерує тимчасові коди на основі поточного часу та секретного ключа, що забезпечує додатковий рівень захисту [7].

Важливим аспектом є також шифрування даних як в транзиті, так і в спокої. Для захищеної передачі даних використовується протокол HTTPS з TLS/SSL сертифікатами, а для зберігання чутливої інформації на пристрої — механізми на кшталт Android Keystore або iOS Keychain. Додатково, для захисту API від зловживань застосовуються техніки rate limiting, API keys та whitelist IP-адрес.

Один із важливих аспектів дослідження стосується вибору технологій для реалізації модуля штучного інтелекту. В останні роки найбільш популярними є мовні моделі, що надаються компанією OpenAI: GPT-3.5, GPT-4, GPT-4-turbo та GPT-4o. Вони характеризуються високою якістю генерації тексту, широкими можливостями інтеграції через API та активною підтримкою. Особливе місце займає модель GPT-4o та її полегшена версія GPT-4o-mini, що забезпечує баланс між швидкістю, точністю та споживанням ресурсів. Альтернативно розглядаються моделі Claude від Anthropic, які акцентують увагу на етичності та безпечності відповідей, а також open-source рішення, зокрема LLaMA від компанії Meta. Останні дозволяють локальне розгортання, однак вимагають значних ресурсів для обслуговування [8].

1.2 Аналіз існуючих рішень

Мобільні застосунки, що орієнтовані на пошук та публікацію кулінарних рецептів, є одним із популярних напрямів у сфері цифрових технологій. На ринку вже існує значна кількість рішень, які забезпечують користувачів базовим функціоналом для доступу до рецептів. Серед таких рішень можна виділити платформи, як-от Cookpad Recipes, Easy Recipes та Tasty, які пропонують великий вибір рецептів із можливістю фільтрації за категоріями, інгредієнтами чи дієтичними обмеженнями (рисунок 1.1).

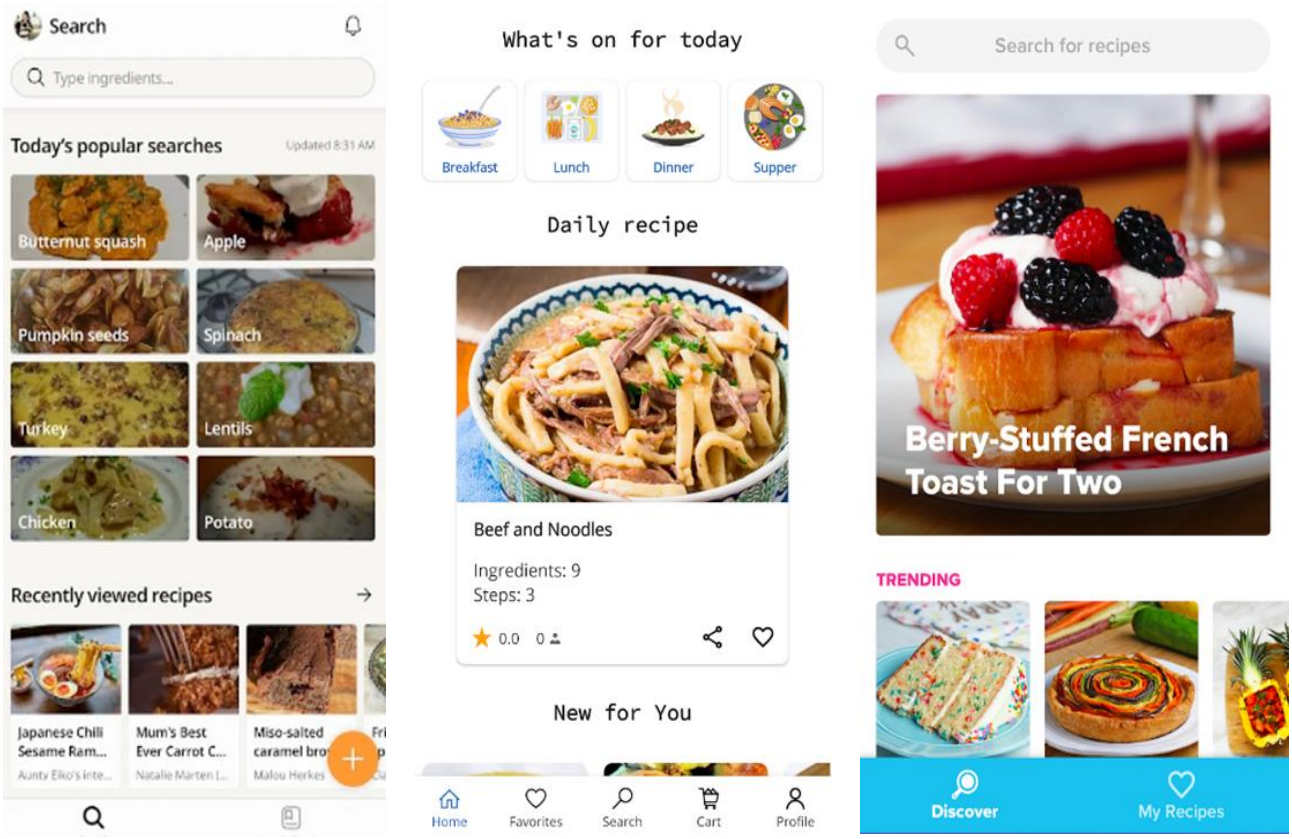

Рисунок 1.1 – Інтерфейси застосунків Cookpad Recipes, Easy Recipes та Tasty

Cookpad Recipes — це глобальна платформа з понад 8 мільйонами рецептів, що пропонує:

- елегантний та інтуїтивно зрозумілий інтерфейс з персоналізованою стрічкою рекомендацій;

- потужну соціальну складову з можливістю публікації авторських рецептів та створення власної аудиторії підписників;
- інтерактивну систему відгуків та оцінок, що дозволяє користувачам обмінюватися досвідом та удосконалювати рецепти.

Easy Recipes вирізняється своєю практичністю та орієнтацією на початківців:

- мінімалістичний дизайн з акцентом на швидкий доступ до збережених і часто використовуваних рецептів;
- покрокові інструкції з високоякісними фотографіями кожного етапу приготування;
- функція планування меню на тиждень із автоматичним формуванням списку необхідних продуктів та можливістю налаштування під бюджет користувача.

Tasty завоював популярність завдяки мультимедійному підходу до кулінарії:

- унікальна бібліотека рецептів різних рівнів складності від команди професійних кулінарів;
- динамічні відеоінструкції, що роблять процес приготування наочним і захоплюючим;
- складний алгоритм рекомендацій, що аналізує вподобання користувача та пропонує персоналізований контент.

1.3 Вимоги до мобільного застосунку

Багато існуючих продуктів мають обмеження, такі як відсутність можливості публікації власних рецептів, складність інтерфейсу або надмірна кількість реклами. Тому виникає потреба у створенні рішення, яке поєднуватиме зручний доступ до існуючих рецептів із можливістю інтерактивного обміну кулінарними ідеями.

Також, сьогодні революційним трендом у розвитку мобільних

застосунків є впровадження технологій штучного інтелекту, що дозволяє значно розширити функціональні можливості та якісно змінити взаємодію з користувачем. Дослідження демонструють, що використання алгоритмів штучного інтелекту дозволяє суттєво підвищити ключові показники ефективності застосунків. Наприклад, середній час сеансу може зрости на 25%, а частота використання — на 30%. Крім того, загальний рівень утримання користувачів після впровадження таких технологій збільшується в середньому на 20% [9].

Аналіз предметної області показав, що ключовими вимогами до таких застосунків є:

- простий і зрозумілий інтерфейс користувача, що забезпечує швидкий доступ до функціоналу застосунку;
- швидкий пошук рецептів. Використання фільтрів, категорій та ключових слів дозволяє зменшити час пошуку;
- можливість публікації власного контенту. Це мотивує користувачів до створення та поширення кулінарних ідей;
- захист даних користувачів;
- гнучкість у роботі з даними;
- використання ШІ-асистента для генерації рецептів та рекомендацій.

1.4 Постановка задачі

На основі проведеного аналізу можна визначити такі основні задачі, які необхідно вирішити в рамках проєкту:

- розробити клієнтський інтерфейс, яка забезпечить зручний доступ до функціоналу;
- серверну інфраструктуру для управління обміном даними між клієнтський інтерфейсом та базою даних;
- розробити базу даних, яка дозволить зберігати інформацію про рецепти, користувачів та їх взаємодію із застосунком;

- забезпечити пошук за ключовими словами, категоріями та іншими параметрами;
- інтегрувати модуль штучного інтелекту, який виконуватиме функції інтелектуального асистента;
- перевірити стабільність і продуктивність роботи застосунку.

Результатом виконання зазначених задач стане мобільний застосунок, що поєднує функціональність, простоту використання та безпеку, спрямований на задоволення потреб користувачів.

2 АНАЛІЗ ТЕХНОЛОГІЙ

2.1 Загальні критерії вибору технологій

Під час вибору технологічного стеку для розробки мобільного застосунку ключовим завданням є забезпечення балансу між якістю продукту, ефективністю розробки та можливістю масштабування у майбутньому. Одним із головних пріоритетів стала кросплатформність — здатність запускати застосунок на різних операційних системах, таких як iOS та Android, з використанням спільної кодової бази. Це дозволяє значно знизити витрати на розробку, пришвидшити оновлення та полегшити підтримку.

Оскільки проєкт орієнтований на мобільні пристрої, велике значення має продуктивність — застосунок має швидко запускатися, працювати плавно навіть на менш потужних пристроях і не перевантажувати ресурси смартфону [10]. Важливим також є те, наскільки гнучко обрана технологія дозволяє інтегрувати зовнішні сервіси та розширювати функціональність без значних затрат часу, що актуально на ранніх етапах розвитку стартапу.

Безпека даних — ще один критичний фактор, оскільки додаток працює з персональною інформацією користувачів. Технології мають відповідати сучасним стандартам захисту, передбачати надійні механізми авторизації, шифрування та протидії несанкціонованому доступу.

Крім того, перевагу було надано рішенням з активною спільнотою, широкою підтримкою, хорошою документацією та наявністю перевірених бібліотек, що дозволяють уникати повторного «винаходу колеса» і зосередитись на основному функціоналі.

Останнім, але не менш важливим критерієм, є масштабованість. Обрана архітектура має підтримувати зростання: як у плані кількості користувачів, так і в обсягах оброблюваних даних, не втрачаючи стабільності й швидкодії.

2.2 Вибір засобів для клієнтської логіки застосунку

React Native став одним із найпопулярніших виборів розробників завдяки низці важливих переваг (рисунок 2.1). Це платформа, створена компанією Facebook, яка зарекомендувала себе у масштабних проєктах, забезпечуючи надійність та продуктивність. Використання JavaScript та компонентної структури React значно прискорює процес створення застосунків, а генерація нативних елементів дозволяє досягти плавної та швидкої роботи [11].

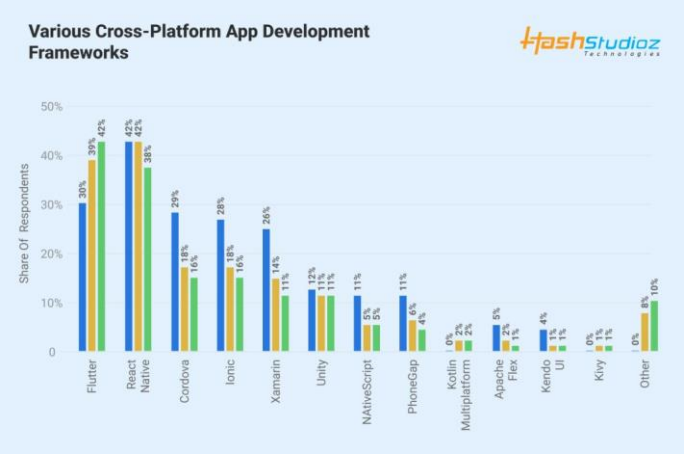

Рисунок 2.1 - Популярність фреймворків для кросплатформної розробки

Однією з ключових переваг React Native є його кросплатформність. Завдяки спільній кодовій базі для iOS та Android розробники можуть підтримувати обидві платформи одночасно, що сприяє оперативному внесенню змін. Крім того, велика та активна спільнота розробників, а також широкий вибір готових бібліотек, допомагають швидко знаходити рішення для різних завдань.

Втім, React Native має і певні обмеження. Наприклад, реалізація складних анімацій або специфічних нативних функцій може вимагати додаткової розробки на Java або Swift, що ускладнює підтримку застосунку, якщо команда не має достатнього досвіду з цими мовами. Однак для багатьох проєктів цей компроміс прийнятний, оскільки скорочує терміни розробки.

Для прискорення розробки і спрощення робочого процесу можна використати фреймворк Expo [12]. Expo надає розробникам готовий набір інструментів, що усуває необхідність глибокого налаштування нативних середовищ для Android і iOS.

Завдяки Expo, можна миттєво тестувати застосунок на реальних пристроях, отримувати доступ до нативних функцій, таких як камера та геолокація, без написання специфічного нативного коду. Окрім того, технологія підтримує гаряче оновлення, що значно пришвидшує цикл розробки, дозволяючи змінам у коді негайно відображатися в додатку.

Ефективне управління станом у додатках на React Native критично важливе для забезпечення логічної послідовності роботи та швидкої реакції інтерфейсу. Для цього проєкту було обрано Zustand як основну бібліотеку для збереження стану.

На відміну від популярного Redux, Zustand пропонує простіший та інтуїтивніший API, усуваючи необхідність написання великої кількості шаблонного коду (лістинг 2.1). Це значно спрощує розробку та підтримку механізмів управління станом. Крім того, бібліотека використовує функціональний підхід, що мінімізує непотрібні перерендери компонентів і позитивно впливає на продуктивність застосунку [13].

Лістинг 2.1 – Приклад використання Zustand

```
const useStore = create((set) => ({
  count: 1,
  inc: () => set((state) => ({ count: state.count + 1 })),
}))

function Counter() {
  const { count, inc } = useStore()
  return (
    <div>
      <span>{count}</span>
      <button onClick={inc}>one up</button>
    </div>
  )}
```

Один із практичних прикладів використання Zustand — створення глобального сховища, яке зберігає дані користувача та список категорій.

Структура проєкту організована за модульним принципом і забезпечує логічне розділення відповідальностей між окремими частинами коду, що позитивно впливає на масштабованість та підтримку проєкту.

Основна логіка застосунку розділена між декількома каталогами. Каталог app відповідає за навігацію та точку входу в застосунок. У components розміщено багаторазові UI-компоненти, що використовуються на різних екранах (наприклад, кнопки, картки, поля введення), що сприяє перевикористанню коду та стилів. Каталог constants містить незмінні значення, такі як кольори, ключі, налаштування або строки інтерфейсу. У entities зосереджена логіка предметних сутностей — це типи, інтерфейси, моделі даних, які описують структуру об'єктів, що використовуються у проєкті. У providers реалізовано контексти або сервіси, які відповідають за глобальний стан (наприклад, авторизацію або тему оформлення). Каталог shared включає допоміжні функції, утиліти, хуки, які не прив’язані до конкретної частини застосунку, але потрібні в багатьох місцях. Каталог store використовується для реалізації керування станом, зокрема, у випадку використання Zustand чи аналогічної бібліотеки — тут розміщуються редьюсери, слайси та middleware. Каталог assets зберігає статичні ресурси, зображення, шрифти, аудіофайли. Основною перевагою цього підходу є модульність, ізольованість та легка масштабованість при зростанні функціональності.

Серед альтернативних підходів можна виділити ще feature-based структуру, де код групується не за типом (компоненти, утиліти тощо), а за функціональними модулями. Наприклад, окрема директорія auth/ може містити свої компоненти, хуки, сторінки, стилі. Це зручно для командної роботи та модульного рефакторингу. У невеликих проєктах або MVP часто використовується плоска структура, де всі компоненти і логіка розміщуються разом, але такий підхід швидко стає незручним при збільшенні обсягів коду.

2.3 Вибір засобів для серверної логіки застосунку

Серверна логіка проєкту відіграє ключову роль у забезпеченні надійного та ефективного обміну даними між клієнтським застосунком і базою даних. Вибір технологій має враховувати швидкість розробки, продуктивність, масштабованість і простоту підтримки. Після аналізу різних варіантів для реалізації серверної логіки було прийнято рішення використовувати Node.js у поєднанні з фреймворком Express.

Node.js — це середовище виконання JavaScript, яке працює поза браузером і побудоване на рушії V8 від Google. Основною перевагою Node.js є асинхронна модель обробки подій, що дозволяє ефективно масштабувати додатки під високі навантаження. Завдяки неблокуючому вводу-виводу, сервер на Node.js здатний одночасно обробляти тисячі підключень, що робить його ідеальним вибором для мобільних застосунків із великою кількістю користувачів.

Ще одним важливим фактором є велика екосистема пакетів і бібліотек npm, що спрощує інтеграцію з різними сервісами, базами даних і іншими компонентами. Крім того, завдяки використанню JavaScript як на клієнті, так і на сервері, розробка стає більш узгодженою і менш фрагментованою, що полегшує підтримку коду.

Для організації роботи серверної логіки застосунку обрано Express — популярний фреймворк для Node.js, який забезпечує простий і гнучкий спосіб створення REST API (лістинг 2.2). Express має структуру, що дозволяє швидко налаштовувати маршрути, обробляти запити та інтегруватися з проміжними компонентами. Це дозволяє швидко будувати та розширювати функціонал сервера, що важливо для адаптивного розвитку проєкту [16].

Лістинг 2.2 – Приклад простого маршруту у Express

```
const express = require('express');
const app = express();
```

```
app.get('/recipes', (req, res) => {
  // Логіка отримання рецептів з бази даних
  res.json([{ id: 1, name: 'Борщ' }, { id: 2, name: 'Вареники'
}]);
});

app.listen(3000, () => {
  console.log('Сервер запущено на порту 3000');
});
```

Структура серверної логіки проєкту побудована за модульним принципом, що базується на концепції розділення відповідальностей. Такий підхід дозволяє зменшити зв’язаність компонентів системи, спрощує розробку, тестування та підтримку коду в довгостроковій перспективі. Усі основні елементи логіки розділені на окремі шари, що мають чітко визначені функції.

Основні компоненти серверної структури.

Маршрути (routes) — відповідають за прийом HTTP-запитів від клієнта. На цьому рівні визначаються кінцеві точки API (наприклад, /api/login, /api/recipes, /api/user/preferences) та типи запитів (GET, POST, PUT, DELETE). Основна задача маршрутизаторів — делегувати обробку запиту відповідному контролеру.

Контролери (controllers) — обробляють вхідні дані, викликають відповідні сервіси або моделі, формують і повертають відповіді.

Моделі (models) — описують структуру даних, що зберігаються в базі. У випадку використання ORM (наприклад, Mongoose у зв’язці з MongoDB) моделі надають методи для взаємодії з базою — створення, читання, оновлення та видалення записів. Вони виступають своєрідним шаром абстракції між логікою додатку та фізичним зберіганням даних.

Проміжні шари (middleware) — це функції, які виконуються перед або після обробки запиту. Їхнє призначення — реалізація додаткових функцій, таких як перевірка автентифікації (JWT), валідація вхідних даних, логування запитів, обробка помилок.

Сервіси (services) — шар, який містить основну бізнес-логіку. Сервіси

дозволяють розвантажити контролери і зробити логіку більш універсальною.

Така архітектура дозволяє чітко розділити відповідальність кожного блоку, підвищує читабельність коду.

2.4 Засоби авторизації та безпеки

У сучасних веб та мобільних застосунках питання авторизації користувачів і захисту даних є надзвичайно важливим. Для автентифікації та контролю доступу у цьому проєкті було обрано JWT (JSON Web Token).

JWT — це відкритий стандарт, що дозволяє безпечно передавати дані у вигляді JSON-об'єкта [17]. Токен складається з трьох частин (рисунок 2.2):

- header – містить тип токена та алгоритм підпису (наприклад, HS256);
- payload – основний вміст (claims), включаючи ідентифікатор користувача, роль, час створення тощо;
- signature – криптографічний підпис для перевірки автентичності та цілісності токена.

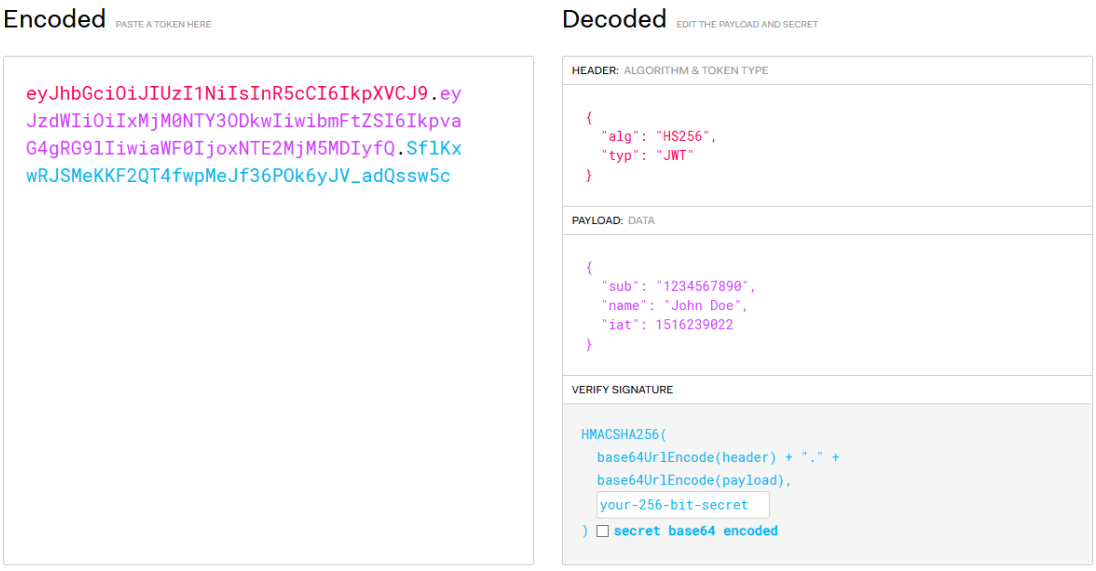

Рисунок 2.2 - Структура JWT-токена та його декодування

Після успішного входу користувача сервер створює JWT-токен, який містить закодовану інформацію про користувача (наприклад, ID, email, роль)

і підписується секретним ключем. Цей підпис гарантує цілісність токена та захист від підробки. Отриманий токен передається клієнту, який зберігає його та надсилає разом із кожним запитом у заголовку (Authorization: Bearer <token>). На стороні сервера токен перевіряється на справжність, і, якщо він дійсний, запит обробляється відповідно до прав користувача.

Секретний ключ (JWT_SECRET) є критично важливим для безпеки, оскільки він використовується для створення та перевірки підпису токенів. Цей ключ повинен зберігатися виключно на сервері, бути достатньо довгим і випадковим, і не потрапляти у відкриті репозиторії або клієнтський код. Його слід захищати у середовищі виконання, наприклад, через змінні середовища.

Із міркувань безпеки найкращою практикою є використання двох типів токенів: access і refresh. Access-токен має короткий термін дії (зазвичай 5–15 хвилин) і використовується для автентифікації кожного запиту. Якщо його буде викрадено, потенційний зловмисник зможе використовувати його лише обмежений час. Refresh-токен має значно довший термін життя (наприклад, 7–30 днів) і дозволяє отримувати нові access-токени без необхідності повторного входу. Він зазвичай зберігається в HttpOnly cookie, що робить його недоступним для JavaScript та захищає від атак типу XSS.

Типова схема роботи передбачає, що після логіну сервер повертає access-токен і refresh-токен. Access-токен зберігається в пам’яті клієнта або у cookie (якщо це безпечно), а refresh-токен — у захищеному HttpOnly cookie. Кожен запит до API супроводжується access-токеном. Коли термін дії access-токена спливає, клієнт надсилає запит на оновлення, використовуючи refresh-токен, і сервер повертає новий access-токен, а за потреби — і новий refresh-токен. Це дозволяє реалізувати "rotate refresh token" — стратегію, за якої кожен новий refresh-токен замінює попередній, підвищуючи безпеку.

Для підвищення захисту також застосовують додаткові заходи, як-от прив’язка токенів до fingerprint браузера або IP-адреси, контроль часу останньої активності користувача, захист від CSRF, використання TLS для захищеної передачі даних тощо.

Такий підхід забезпечує баланс між безпекою та зручністю, дозволяючи користувачам залишатися авторизованими протягом тривалого часу без потреби в повторному введенні облікових даних, але з контрольованим доступом і можливістю управління сесією з боку сервера.

2.5 База даних

Для збереження даних у цьому проєкті використовується MongoDB, що забезпечує високу гнучкість, масштабованість та природну підтримку JSON-структур.

MongoDB чудово підходить для веб і мобільних застосунків, які потребують швидкого доступу до даних та можливості обробки великих обсягів інформації. Експериментальні дані підтверджують переваги MongoDB у плані швидкості опрацювання операцій порівняно з традиційними реляційними СУБД (рисунок 2.3). Як показують дослідження, час завантаження і вставки даних у MongoDB істотно нижчий за аналогічні показники для MySQL [18].

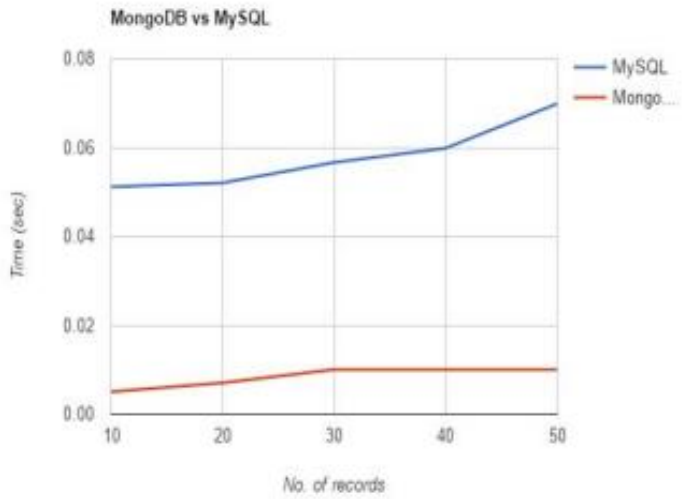

Рисунок 2.3 – Час завантаження/вставки даних в порівнянні з MySql

У межах цього проєкту використовується MongoDB Atlas — хмарний сервіс від MongoDB, який забезпечує автоматизоване розгортання,

масштабування та моніторинг бази даних. Завдяки Atlas стало можливим швидко налаштувати надійне та захищене підключення до бази без необхідності локального хостингу. Платформа також пропонує функції автоматичного резервного копіювання, аналітики продуктивності та управління доступом, що підвищує загальну безпеку й надійність інфраструктури [19].

MongoDB відрізняється від реляційних баз даних (РСУБД) своєю документно-орієнтованою моделлю, яка дозволяє зберігати дані у форматі JSON-подібних документів. У той час як PostgreSQL, MySQL та Oracle використовують табличну модель з фіксованими схемами, чіткою структурою та реляційними зв'язками між таблицями.

Основна проблема, яка може виникнути при використанні MongoDB це денормалізації даних [20]. MongoDB схильна до дублювання інформації, що може призвести до проблем консистентності. Іншою складнощою є організація транзакцій для операцій, що потребують атомарності. Хоча MongoDB підтримує транзакції на рівні бази даних, їхня продуктивність значно нижча порівняно з реляційними системами.

2.6 Інтеграція модуля штучного інтелекту

У цьому проєкті ШІ-асистент виконує дві основні задачі — генерує зміст рецептів на основі вхідних даних, а також формує кулінарні рекомендації. Таке застосування дозволяє зробити взаємодію з додатком зручніше, що прямо впливає на залученість і рівень задоволеності користувачів.

Під час вибору моделі для інтеграції було проаналізовано широкий спектр мовних моделей (рисунок 2.4), що розрізняються за точністю, обсягом знань, швидкістю відповіді, вимогами до ресурсів та зручністю інтеграції.

Модель	Провайдер	Довжина контексту	Швидкість	Відкритий код	Ціна / мільйон токенів	Ціна	Якість	Швидкість
GPT-4o	OpenAI	128k	83	Ні	4,38	★★★★☆	★★★★★	★★★★☆
GPT-4o Mini	OpenAI	128k	113	Ні	0,26	★★★★★	★★★★☆	★★★★★
o1	OpenAI	200k	144	Ні	27,56	★☆☆☆☆	★★★★★	★★★★☆
o1 Mini	OpenAI	128k	208	Ні	5,25	★★☆☆☆	★★★★☆	★★★★★
Gemini 2.0 Flash	Google	1м	108	Ні	x	★★★☆☆	★★★★★	★★★★☆
Gemini 1.5 Flash	Google	1м	123	Ні	2,19	★★★★☆	★★★★☆	★★★★☆
Gemini 1.5 Pro	Google	2м	160	Ні	0,13	★★★★★	★★★★★	★★★★★
Claude 3.5 Sonnet	Anthropic	200k	72	Ні	6,00	★★☆☆☆	★★★★★	★★★☆☆
Claude 3.5 Haiku	Anthropic	200k	85	Ні	1,60	★★★☆☆	★★★★☆	★★★★☆
Claude 3 Opus	Anthropic	200k	26	Ні	30,00	★☆☆☆☆	★★★★★	★☆☆☆☆
Command R+	Cohere	128k	49	Ні	6	★★★★★	★★★★☆	★★★★☆
Command R	Cohere	128k	108	Ні	0,51	★★★★★	★★★☆☆	★★★★☆
Llama 3.3 70b	Meta AI	128k	72	Так	0,69	★★★★☆	★★★★★	★★★☆☆
Llama 3.1 405b	Meta AI	128k	30	Так	3,50	★★★★☆	★★★★☆	★★☆☆☆
Llama 3.2 90b	Meta AI	128k	49	Так	0,81	★★★★★	★★★☆☆	★★☆☆☆
Llama 3.2 11b	Meta AI	128k	131	Так	0,18	★★★★☆	★★★☆☆	★★☆☆☆
Pixtral Large	Mistral AI	128k	36	Ні	3	★★★★★	★★★★★	★★★★☆
Ministral 8x7b	Mistral AI	128k	168	Ні	0,04	★★★★★	★★★★★	★★★★★
Ministral 7b	Mistral AI	128k	136	Ні	0,1	★★★★★	★★★☆☆	★★★★★
Mistral Small	Mistral AI	32k	60	Ні	0,3	★★★★★	★★★☆☆	★★★★☆
Codestral	Mistral AI	256k	X	Ні	0,6	★★★★★	★★★☆☆	★★★☆☆
Nova Pro	AWS	300k	91	Ні	1,40	★★☆☆☆	★★★★☆	★★★★★
Nova Lite	AWS	300k	148	Ні	0,10	★★★★★	★★★☆☆	★★★★★
Nova Micro	AWS	300k	195	Ні	0,06	★★★★★	★★☆☆☆	★★★★★
DeepSeek V3	DeepSeek	128k	20	Так	0,9	★★★★☆	★★★★☆	★☆☆☆☆
DeepSeek-Coder V2	DeepSeek	128k	61	Так	0,17	★★★★★	★★★☆☆	★★★☆☆

Рисунок 2.4 – Порівняння мовних моделей ШІ

Моделі OpenAI — GPT-3.5, GPT-4, GPT-4-turbo, GPT-4o та GPT-4o-mini — є одними з найпопулярніших завдяки високій якості генерації, широкому контексту, стабільності роботи та зручному API. GPT-4o та її компактна версія GPT-4o-mini демонструють значне покращення у швидкості відповіді при збереженні глибини контекстного розуміння. GPT-4o підтримує мультимодальність (текст, зображення, аудіо), хоча в даному проєкті використовується лише текстова частина.

У процесі тестування на прикладах типових запитів користувача модель продемонструвала не лише високу швидкість генерації (рисунок 2.5), а також успішно обробляла як прямі, так і непрямі інструкції, демонструючи адаптивність до стилю користувача, граматичної структури речень, що підвищує загальний рівень персоналізації.

Name	Status	Type	Initiator	Size	Time
t	(blocked:...	fetch	k7mp8km6f6uslm8w.j	0 B	6 ms
rgstr?k=client-nb0qtYl...	(blocked:...	fetch	k7mp8km6f6uslm8w.j	0 B	7 ms
rgstr?k=client-nb0qtYl...	(blocked:...	fetch	k7mp8km6f6uslm8w.j	0 B	22 ms
conversation	200	fetch	k7mp8km6f6uslm8w.j	3.7 kB	5.32 s

Рисунок 2.5 – Тест на швидкість генерації

Конкурентом у процесі вибору була також модель Claude від Anthropic. Вона вирізняється посиленою орієнтацією на безпеку та етичність відповідей, добре працює з великим контекстом і дає виважені, логічно послідовні відповіді. Проте її інтеграція виявилась менш гнучкою: API доступний в обмеженому режимі, а документація не забезпечує тієї прозорості й зручності, яку пропонує OpenAI. Це створює додаткові бар'єри для швидкої інтеграції у мобільне середовище.

Також були проаналізовані відкриті моделі серії LLaMA від Meta. Незважаючи на те, що вони надають більший контроль над середовищем запуску, їх використання пов'язане з рядом технічних викликів. Зокрема, моделі LLaMA 2-7B і LLaMA 3-8B вимагають значної обчислювальної потужності, а для запуску навіть на середньому рівні навантаження потрібне серверне середовище з GPU. Це значно ускладнює процес розгортання, масштабування й підтримки. Крім того, якість відповіді, зокрема на українській мові, не досягала рівня моделей OpenAI, що також стало критичним фактором.

З огляду на всі зазначені аспекти, остаточний вибір зупинився на GPT-4o-mini. Найважливішими аргументами стали добре задокументований API, який дозволив швидко інтегрувати модель у backend-зв'язку з мобільним клієнтом, здатність давати доречні, логічні відповіді. Поточна архітектура передбачає, що запити від клієнта надходять до серверної частини, де вони попередньо обробляються (наприклад, форматується prompt або додається службова інструкція), після чого сервер надсилає запит до OpenAI API. Відповідь повертається в межах кількох сотень мілісекунд, забезпечуючи користувачу природну, без затримок взаємодію з віртуальним асистентом [8].

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ЗАСТОСУНКУ

3.1 Опис схеми роботи проєкту

Основними компонентами системи є клієнтська частина (мобільний додаток), серверна частина та база даних (рисунок 3.1). Користувач взаємодіє з мобільним застосунком через інтерфейс, який надсилає HTTP-запити до серверу через REST API. Сервер приймає запити від мобільного додатку, обробляє їх, виконує необхідні операції (наприклад, пошук, додавання нових рецептів) і взаємодіє з базою даних. Також сервер відповідає за верифікацію користувача через JWT. Після успішної авторизації користувач отримує токен доступу, який використовуватиметься для подальших запитів до серверу.

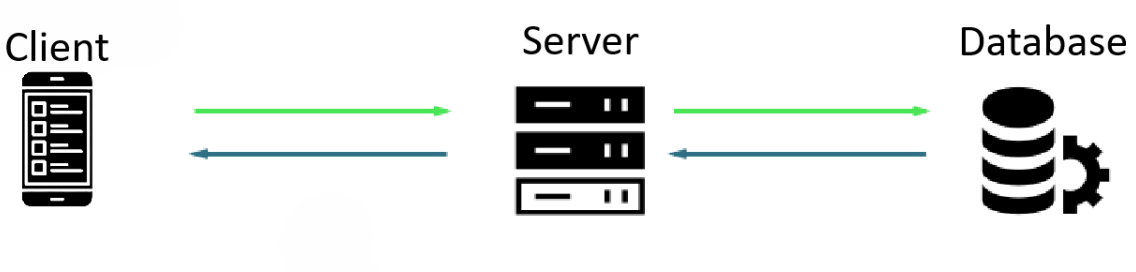

Рисунок 3.1 – Взаємодія між компонентами системи

3.2 Проектування бази даних

Проєктування бази даних є ключовим етапом розробки застосунку, що забезпечує правильне зберігання та доступ до даних, необхідних для функціонування системи (рисунок 3.2).

На основі вимог до застосунку визначено такі основні сутності:

- Session – відповідає за зберігання інформації про авторизацію користувачів, включаючи їхній refreshToken, ідентифікатор користувача (userId) і час дії токена (expiresIn);

- User – містить дані про користувачів, такі як електронна пошта (email), ім'я (name), хешований пароль (password), статус активації (isActivated), посилання для активації акаунта (activationLink), список уподобаних рецептів (likes) та часові мітки створення/оновлення;

- Recipes – ключова сутність для зберігання даних про рецепти. Вона включає назву (title), список інгредієнтів (ingredients), інструкції (instructions), опис (description), автора (authorId), категорію (category), статус публічності (isPublic), час приготування (cookTime), складність (difficulty), зображення (image), кількість вподобань (likes), переглядів (views) та часові мітки створення/оновлення;

- Categories – забезпечує організацію рецептів за категоріями, містить назву (name) та опис (description).

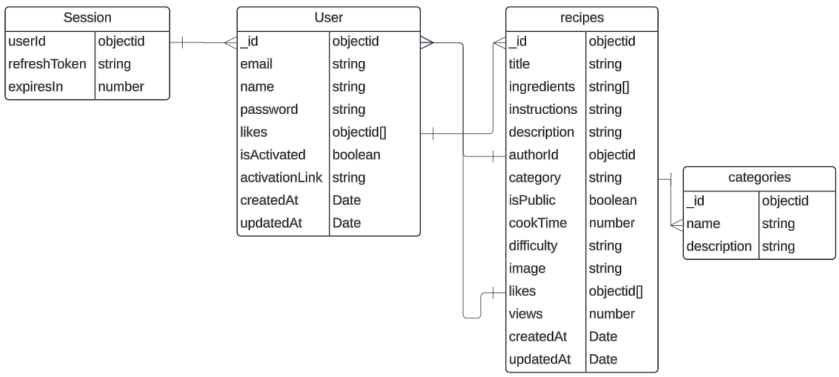

Рисунок 3.2 – Діаграма бази даних

Взаємозв’язки між сутностями:

- колекція Session пов’язана з користувачами за допомогою поля userId;

- колекція User є автором рецептів та може мати список уподобаних рецептів через посилання на їхні objectId;

- колекція Recipes містить вбудовану структуру інгредієнтів та пов’язана з категоріями через поле category.

У реалізації застосунку для взаємодії з базою даних використовується бібліотека Mongoose — обгортка над MongoDB для Node.js.

Кожна сутність у базі даних описується через схему (Schema), де зазначаються поля, їхні типи, обов’язковість, унікальність, значення за замовчуванням, посилання на інші колекції тощо. Приклад визначення схеми користувача представлено у лістингу 3.1.

Лістинг 3.1 – Визначення моделі User

```
const User = new mongoose.Schema({
    email: {
        type: String,
        required: true,
        unique: true
    },
    name: {
        type: String,
        required: true
    },
    password: {
        type: String,
        required: true
    },
    likes: [
        {
            type: mongoose.Schema.Types.ObjectId,
            ref: "Recipe"
        }
],
    dislikes: [
        {
            type: mongoose.Schema.Types.ObjectId,
            ref: "Recipe"
        }
    ],
    isActivated: {
        type: Boolean,
        default: false
    },
    activationLink: {
        type: String
    }
}, { timestamps: true });

export default mongoose.model("User", User);
```

Створена схема перетворюється у модель через функцію mongoose.model, після чого її можна використовувати для взаємодії з

відповідною колекцією бази даних у вигляді повноцінного об’єкта — створювати, оновлювати, видаляти та шукати документи.

3.3 Програмна реалізація серверної логіки застосунку

3.3.1 Структура

Серверна логіка застосунку побудована за архітектурним підходом Service Layer Architecture, який забезпечує чітке розділення обов’язків між компонентами програми (рисунок 3.3).

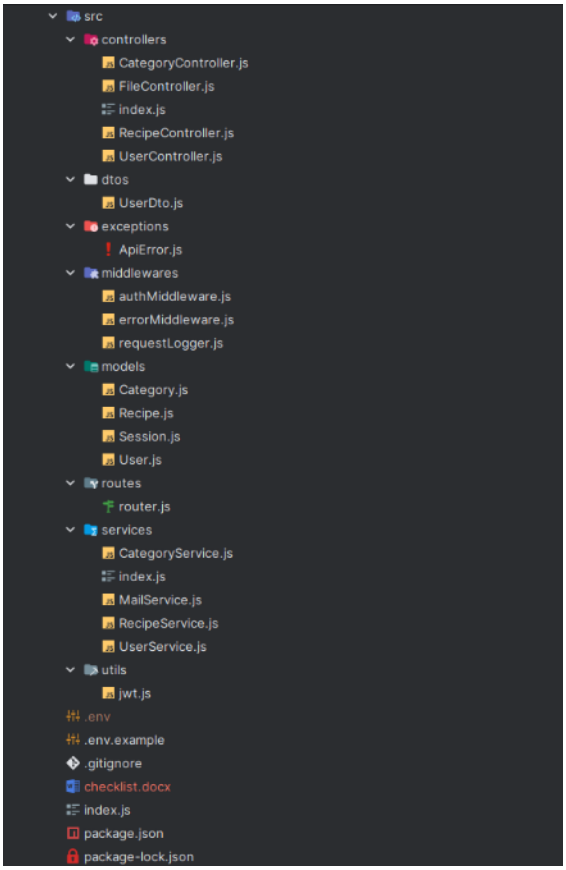

Рисунок 3.3 – Структура проекту серверної логіки застосунку

Структура файлів у проєкті організована так, щоб забезпечити зручність роботи з різними компонентами додатку, їх інтеграцію та масштабування.

3.3.2 Опис основних директорій

Controllers містить модулі, які обробляють HTTP-запити та відповідають за взаємодію між сервісами та моделями. Контролери координують запити, викликають відповідні сервіси для виконання бізнес-логіки та формують відповіді для клієнтів (лістинг 3.2).

Основні файли:

- CategoryController.js – керує запитами, пов'язаними з категоріями;
- FileController.js – відповідає за завантаження/завантаження файлів;
- RecipeController.js – керує рецептами;
- UserController.js – керує користувачами.

Лістинг 3.2 – Приклад функції контролеру

```
async activate(req, res, next) {
        try {
            const activationLink = req.params.link;
            await userService.activate(activationLink);
            return res.redirect(process.env.CLIENT_URL);
        } catch (e) {
            next(e);
        }
    }
```

Dtos містить об'єкти, які призначені для стандартизації структури даних, які передаються між різними шарами застосунку, зокрема між сервісами та контролерами або у відповідях до клієнта (лістинг 3.3). DTO дозволяють ізолювати внутрішню структуру моделей від зовнішніх запитів і відповідей.

Основні файли:

- UserDto.js – описує формат даних користувачів.

Лістинг 3.3 - Код файлу UserDto.js

```
export class UserDto {
    id;
    email;
```

```
    name;
    likes;
    isActivated;

    constructor(model) {
        this.id = model._id;
        this.email = model.email;
        this.name = model.name;
        this.likes = model.likes;
        this.isActivated = model.isActivated;
        return this;
    }
}
```

Exceptions містить клас для обробки виключних ситуацій у додатку. Метою є централізоване управління помилками, що дозволяє стандартизувати формат повідомлень про помилки та забезпечити більш зручну обробку помилок як на стороні сервера, так і на стороні клієнта. Це особливо корисно при побудові REST API, де важливо повертати відповідні HTTP-коди та зрозумілі повідомлення про помилки (лістинг 3.4).

Основні файли:

- ApiError.js – визначає власні помилки.

Лістинг 3.4 - Код файлу ApiError.js

```
export class ApiError extends Error {
    status;
    errors;

    constructor(status, message, errors = []) {
        super(message);
        this.status = status;
        this.errors = errors;
    }

    static UnauthorizedError() {
        return new ApiError(401, 'Користувач не авторизований')
    }

    static BadRequest(message, errors = []) {
        return new ApiError(400, message, errors);
    }

    static ForbiddenError() {
        return new ApiError(403, 'Недостатньо прав');
    }}
```

Middlewares містить проміжні шари, які обробляють запити до того, як вони досягнуть контролерів. У Node.js-застосунках проміжні шари є важливою частиною архітектури серверного коду. Вони дозволяють обробляти HTTP-запити на етапах між моментом їх надходження на сервер і передачею в контролери. Це зручно для реалізації таких функцій, як автентифікація, обробка помилок, логування, перевірка прав доступу тощо (лістинг 3.5).

Основні файли:

- authMiddleware.js – перевіряє аутентифікацію користувача;
- errorMiddleware.js – глобальний обробник помилок;
- requestLogger.js – логує HTTP-запити.

Лістинг 3.5 - Код файлу requestLogger.js

```
export function requestLogger(req, res, next) {
    const logData=`${new Date().toISOString()}-
${req.method}${req.url}`;
    fs.appendFile("./logs/request.txt", logData, (err) => {
        if (err) {
            console.error('Помилка запису в журнал:', err);
        }
    });
    next();}
```

Models містить моделі даних, які описують структуру об'єктів у базі даних. Кожен файл відповідає за окрему колекцію (лістинг 3.6). Моделі даних виконують роль схеми для зберігання й обробки об'єктів у колекціях MongoDB. Вони забезпечують валідацію, встановлюють типи полів, обов’язковість значень, а також дозволяють створювати зв’язки між об’єктами.

Основні файли:

- Category.js – модель категорій;
- Recipe.js – модель рецептів;
- Session.js – модель сесій користувачів;
- User.js – модель користувачів.

Лістинг 3.6 - Код файлу Category.js

```
const Category = new mongoose.Schema({
    name: {
        type: String,
        required: true,
        unique: true
    },
    description: String
});

export default mongoose.model("Category", Category);
```

Routes містить маршрути програми, які визначають, які контролери викликати для кожного HTTP-запиту (лістинг 3.7). Маршрути формують структуру API — набір доступних кінцевих точок, через які клієнтська частина може взаємодіяти з сервером. Це дозволяє реалізувати CRUD-функціонал для сутностей, таких як рецепти, категорії, користувачі тощо.

Основні файли:

- router.js – визначає всі маршрути API.

Лістинг 3.7 - Приклад маршрутизації

```
router.get("/category", categoryController.getAllCategories);
router.get("/category/:name",
categoryController.getCategoryByName);
```

Services містить сервіси, які відокремлюють бізнес-логіку від контролерів, щоб забезпечити модульність та спрощення коду (лістинг 3.8).

Основні файли:

- CategoryService.js – логіка роботи з категоріями;
- MailService.js – надсилання електронних листів;
- RecipeService.js – логіка роботи з рецептами;
- UserService.js – логіка роботи з користувачами.

Лістинг 3.8 – Приклад функції сервісу

```
async getAll(query) {
        let page = parseInt(query.page) || 0;
        let limit = parseInt(query.limit) || 10;
```

```
        if (page < 0) page = 0;
        if (limit < 1) limit = 1;
        if (limit > 100) limit = 100;

        const categories = await Category.find({}).skip(page *
limit).limit(limit);
        const totalCount = await Category.countDocuments();

        return {
            page,
            limit,
            totalCount,
            totalPages: Math.ceil(totalCount / limit),
            data: categories
        }
    }
```

Utils містить допоміжні функції загального призначення, які не прив'язані до конкретного функціонального модуля, але часто використовуються в різних частинах застосунку (лістинг 3.9). Їх винесення в окремий модуль дозволяє уникнути дублювання коду, покращує повторне використання та спрощує тестування.

Основні файли:

- jwt.js – робота з JWT.

Лістинг 3.9 – Код файлу jwt.js

```
export const hashPassword = (password) => {
    return bcrypt.hashSync(password, 10);
}

export const comparePasswords = (password, hashedPassword) => {
    return bcrypt.compareSync(password, hashedPassword);
}
export const generateAccessToken = (payload) => {
    return jwt.sign(
        payload,
        process.env.JWT_SECRET,
        {
            expiresIn: '30m',
        }
    );
};

export const generateRefreshToken = (payload) => {
    return jwt.sign(
```

```
        payload,
        process.env.JWT_SECRET,
        {
            expiresIn: '7d',
        }
    );
};

export const verifyToken = (token) => {
    return jwt.verify(token, process.env.JWT_SECRET);
}
```

3.3.3 Реалізація AI-функціональності

Однією з ключових функцій застосунку є можливість автоматичного генерування кулінарного контенту за допомогою штучного інтелекту. Для цього реалізовано окремий сервіс AiAssistantService, який взаємодіє з API моделі GPT від OpenAI. Завдяки цьому сервісу користувачі можуть отримувати опис рецепта, покрокові інструкції приготування, приблизний час готування та базову інформацію про поживну цінність страви. Крім того, на основі введених даних застосунок формує короткі текстові рекомендації, які покликані підкреслити переваги рецепта або надати корисні поради.

Функціональність реалізовано через окремий клас, який інкапсулює всю логіку взаємодії з OpenAI (лістинг 3.10). Основні два методи — generateRecipeDetails та generateRecipeRecommendation. Перший метод формує JSON-відповідь з описом страви, інструкціями та поживною цінністю на основі назви, категорії та списку інгредієнтів. Другий створює текстову рекомендацію з оцінкою складності рецепта та порадами. Лістинг нижче демонструє основну структуру сервісу.

Лістинг 3.10 – Структура сервісу AiAssistantService

```
import { OpenAI } from 'openai';

class AiAssistantService {
    constructor() {
        this.openai = new OpenAI({
            apiKey: process.env.OPENAI_API_KEY,
        });
```

```
        this.model = 'gpt-4o-mini';
    }

    async generateRecipeDetails({ title, category, ingredients
}) {
        // логіка
    }

    async generateRecipeRecommendation({ title, category,
ingredients, instructions, description }) {
        // логіка
    }
}

export const aiAssistantService = new AiAssistantService();
```

Для здійснення запиту до OpenAI API у застосунку використовується модель 'gpt-4o-mini'. Запит формується у вигляді масиву повідомлень messages, де кожен елемент має формат { role: 'user', content: '...' }, а вміст content включає сформований запит на основі даних рецепта. Додатково задається параметр temperature, який визначає рівень варіативності відповіді. Для методу generateRecipeDetails також вказується параметр response_format зі значенням 'json', що дозволяє отримати структуровану відповідь.

У методі generateRecipeDetails передається назва рецепта, його категорія та список інгредієнтів. На основі цих даних GPT-модель повертає відповідь у вигляді JSON-об’єкта, що містить опис страви, покрокові інструкції приготування, приблизний час готування та базову інформацію про поживну цінність (лістинг 3.11).

Лістинг 3.11 – Метод generateRecipeDetails

```
async generateRecipeDetails({ title, category, ingredients }) {
        if (!title || !category || !ingredients) {
            return {};
        }

        const prompt = `
            Ти асистент у мобільному застосунку з рецептами.
            На основі наданої інформації про рецепт згенеруй:
          ...

            Відповідь повинна бути у форматі JSON наступного
```

```
вигляду:
            {
            "instructions": "Текст покрокових інструкцій",
            ...
            }

            Вхідні дані:
            - Назва рецепта: ${title}
            ...`;

        try {
            const response = await
this.openai.chat.completions.create({
                model: this.model,
                messages: [
                    { role: "system", content: "Ти професійний
кулінарний асистент. Відповідай тільки..." },
                    { role: "user", content: prompt }
                ],
                temperature: 0.4,
                response_format: { type: "json_object" }
            });

            const content =
response.choices[0].message.content.trim();
            let jsonStr = content;

            if (content.includes("```json")) {
                jsonStr = content.replace(/```json\s*|\s*```/g,
"");
            } else if (content.includes("```")) {
                jsonStr = content.replace(/```\s*|\s*```/g, "");
            }

            jsonStr = jsonStr.replace(/`/g, "");

            return JSON.parse(jsonStr);
        } catch (e) {
            console.error('Помилка при отриманні відповіді:',
e);
            return {};
        }
    }
```

У методі generateRecipeRecommendation використовується вже сформований контент рецепта — його назва, категорія, інгредієнти, інструкції та опис — для створення короткої текстової рекомендації. Ця рекомендація зазвичай включає оцінку складності, кулінарну пораду або перевагу рецепта, оформлену у вигляді 1–2 речень (лістинг 3.12).

Лістинг 3.12 – Метод generateRecipeRecommendation

```
async generateRecipeRecommendation(
    {
     title,
     category,
     ingredients,
     instructions,
     description
          }) {
        if (!title || !category || !ingredients || !instructions
|| !description) {
            return '';
        }

        const prompt = `
            Ти кулінарний експерт у мобільному застосунку з
рецептами.
            На основі наданої інформації напиши
               ...
            Вхідні дані:
            - Назва рецепта: ${title}
            - Категорія рецепта: ${category}
            ...
        `;

        try {
            const response = await
this.openai.chat.completions.create({
                model: this.model,
                messages: [
                    { role: "system", content: "Ти кулінарний
експерт. Відповідай..." },
                    { role: "user", content: prompt }
                ],
                temperature: 0.7,
            });

            return response.choices[0].message.content.trim();
        } catch (e) {
            console.error('Помилка при отриманні відповіді:',
e);
            return '';
        }
    }
```

3.4 Програмна реалізація мобільного застосунку

3.4.1 Структура

Структура мобільного застосунку представлена на рисунку 3.4.

Рисунок 3.4 – Структура мобільного застосунку

3.4.2 Опис основних директорій

App містить сторінки застосунку, кожна з яких відповідає за певну функціональність. Також розміщення файлів визначає навігацію у застосунку. Expo Router, який використовується в цьому проекті, базується на підході file-based routing (файлова навігація). Цей підхід забезпечує автоматичне створення маршрутів залежно від структури файлів у директорії.

Основні файли:

- login – екран входу;
- profile – екран профілю користувача (лістинг 3.13);
- recipe – сторінка детальної інформації про рецепт;
- register – екран реєстрації;
- search – екран пошуку;
- _layout.js – головний шаблон для розташування сторінок;
- index.js – вхідна точка.

Лістинг 3.13 – Код сторінки профілю

```
const Profile = () => {
    const router = useRouter();
    const user = useStore(state => state.user);
    const fetchMyUser = useStore(state => state.fetchMyUser);
    const fetchUserLogout = useStore(state =>
state.fetchUserLogout);

    const handleLogout = async () => {
        await fetchUserLogout();
        router.push("/login");
    }

    useEffect(() => {
        fetchMyUser();
    }, []);

    return (
        <SafeAreaView
style={{backgroundColor:COLORS.bgSecondary}}>
            <Stack.Screen
                options={{
                    statusBarTranslucent: false,
                    headerShown: true,
                    headerTitle: "",
                    headerStyle: {backgroundColor:
COLORS.bgSecondary},
                    headerShadowVisible: false,
                    headerLeft: () => (
                        <TouchableOpacity onPress={() => {
                            router.back();
                        }}>
                            <MaterialCommunityIcons name="arrow"
/>
                        </TouchableOpacity>
                    )
```

```
                }}
            />
            <View style={{paddingHorizontal: SIZES.xLarge}}>
                //JSX розмітка сторінки
                    .
    .
    .
            </View>
        </SafeAreaView>)};
```

Assets містить статичні ресурси, які використовуються в застосунку.

Components містить компоненти, які є повторюваними блоками інтерфейсу.

Основні файли:

- categoryCards – компонент для відображення карток категорій;
- popularRecipes – компонент для відображення популярних рецептів;
- search – компонент для пошуку (лістинг 3.14);
- shared – універсальні компоненти, що можуть використовуватися в різних частинах застосунку.

Лістинг 3.14 – Код компоненту пошуку

```
export const Search = () => {
    const router = useRouter();

    const [value, setValue] = useState("");

    return (
        <View>
            <Image source={mainBg} style={styles.image}/>
                <View>
                    <Text style={styles.title}>Оберіть свій
                        ідеальний рецепт</Text>
                </View>
                <View style={{marginTop: SIZES.small}}>
                    <Input placeholder="Пошук"
style={styles.input}/>
                </View>
        </View>
    );
};
```

Constants містить константи, які доступні в різних частинах програми.

Основні файли:

- theme.js – тема програми (кольори, стилі).

Entities відповідає за опис основних предметних сутностей застосунку та сервісів для роботи з ними (лістинг 3.15). Сутності відображають ключові об'єкти бізнес-логіки — категорії, рецепти, користувачі — і реалізують взаємодію з сервером через API.

Основні файли:

- category – сутність категорій;
- recipe – сутність рецептів;
- user – сутність користувачів.

Лістинг 3.15 – Приклад функції сервісу сутності

```
static async getRecipes(filters) {
        const response = await $api.get('/recipes', {
            params: filters
        });
        return response.data;
    }
```

Shared містить утиліти та налаштування, які застосовуються у всьому проєкті і є спільними для різних частин клієнтського застосунку. Він служить для централізації конфігурації і допоміжних функцій, що дозволяє уникнути дублювання коду і спрощує підтримку.

Основні файли:

- http – конфігурація модуля роботи з HTTP-запитами (лістинг 3.16);
- utils – допоміжні функції.

Лістинг 3.16 – Код конфігурація модуля роботи з HTTP-запитами

```
const $api = axios.create({
    withCredentials: true,
    baseURL: process.env.EXPO_PUBLIC_SERVER_URL + 'api/',
})

$api.interceptors.request.use(async (config) => {
```

```
    config.headers.Authorization = `Bearer ${await
AsyncStorage.getItem('accessToken')}`
    config.headers.ContentType = 'application/json'
    return config
})
```

Store відповідає за централізоване керування глобальним станом клієнтської частини застосунку. Він забезпечує реактивність, спрощує обмін даними між компонентами та підтримує синхронізацію стану протягом життєвого циклу застосунку.

Основні файли:

- store.js – конфігурація магазину стану (лістинг 3.17).

Лістинг 3.17 – Код файлу store.js

```
export const useStore = create()(immer(persist((set, get) => ({
    user: {},
    isAuth: false,
    isLoading: true,
    error: "",
    categories: [],

    fetchAllCategories: async () => {
        try {
            const result = await
CategoryService.getCategories();
            set(state => {
                state.categories = result.data
                state.error = ""
                state.isLoading = false
            })
        }catch (e) {
            console.log(e.response?.data?.message)
            set(state => {
                state.error = e.response?.data?.message
                state.isLoading = false
            })
        }
    })));
```

4 АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

4.1 Функціональні можливості застосунку

Застосунок реалізує надійну систему аутентифікації користувачів. Система включає процес реєстрації нових користувачів із збором основної інформації, включаючи ім'я, електронну адресу та пароль з обов'язковим підтвердженням для забезпечення безпеки. Форма реєстрації включає валідацію введених даних та зручний інтерфейс для швидкого заповнення необхідних полів (рисунок 4.1).

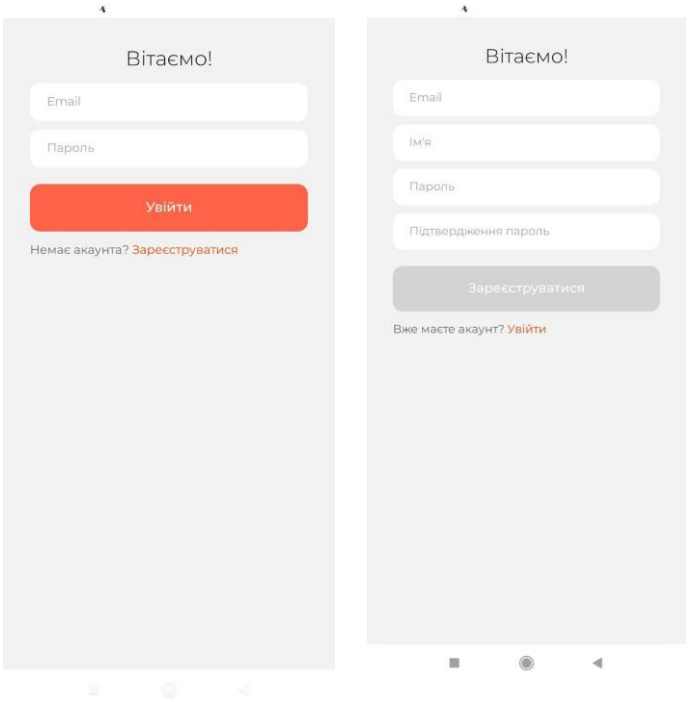

Рисунок 4.1 – Інтерфейс логіну та реєстрації

Після успішної авторизації користувачів зустрічає головна сторінка (рисунок 4.2) та вони отримують доступ до персонального профілю, який відображає їхню інформацію та надає можливості для редагування особистих даних (рисунок 4.3).

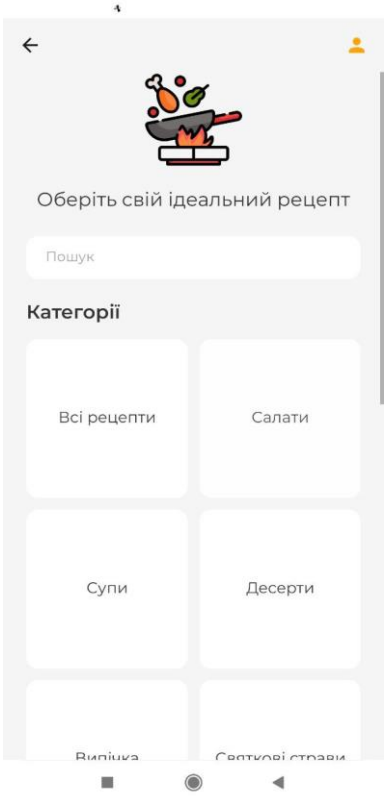

Рисунок 4.2 – Інтерфейс головної сторінки

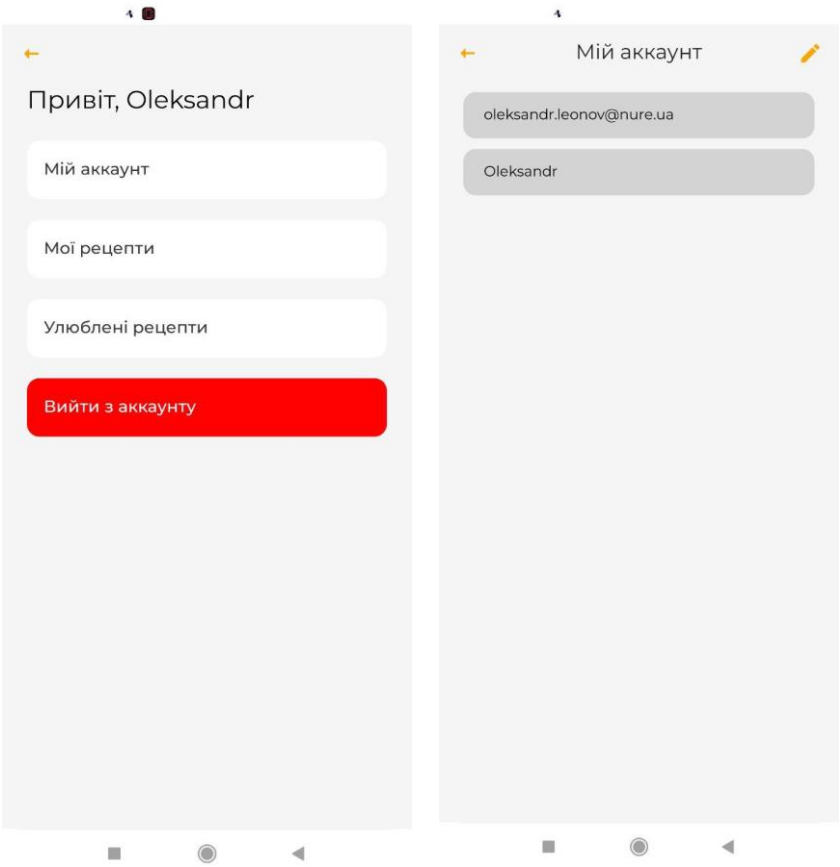

Рисунок 4.3 – Інтерфейс особистого профілю користувача

Головною функціональністю застосунку є система створення та управління рецептами (рисунок 4.4). Користувачі мають можливість додавати нові рецепти через детальну форму, яка включає введення назви рецепту, вибір категорії зі списку доступних варіантів, визначення рівня складності приготування та інгедієнтів. Для таких даних як інструкція, опис, час приготування, харчова цінність доступна генерація за допомогою інтелектуального асистенту ШІ.

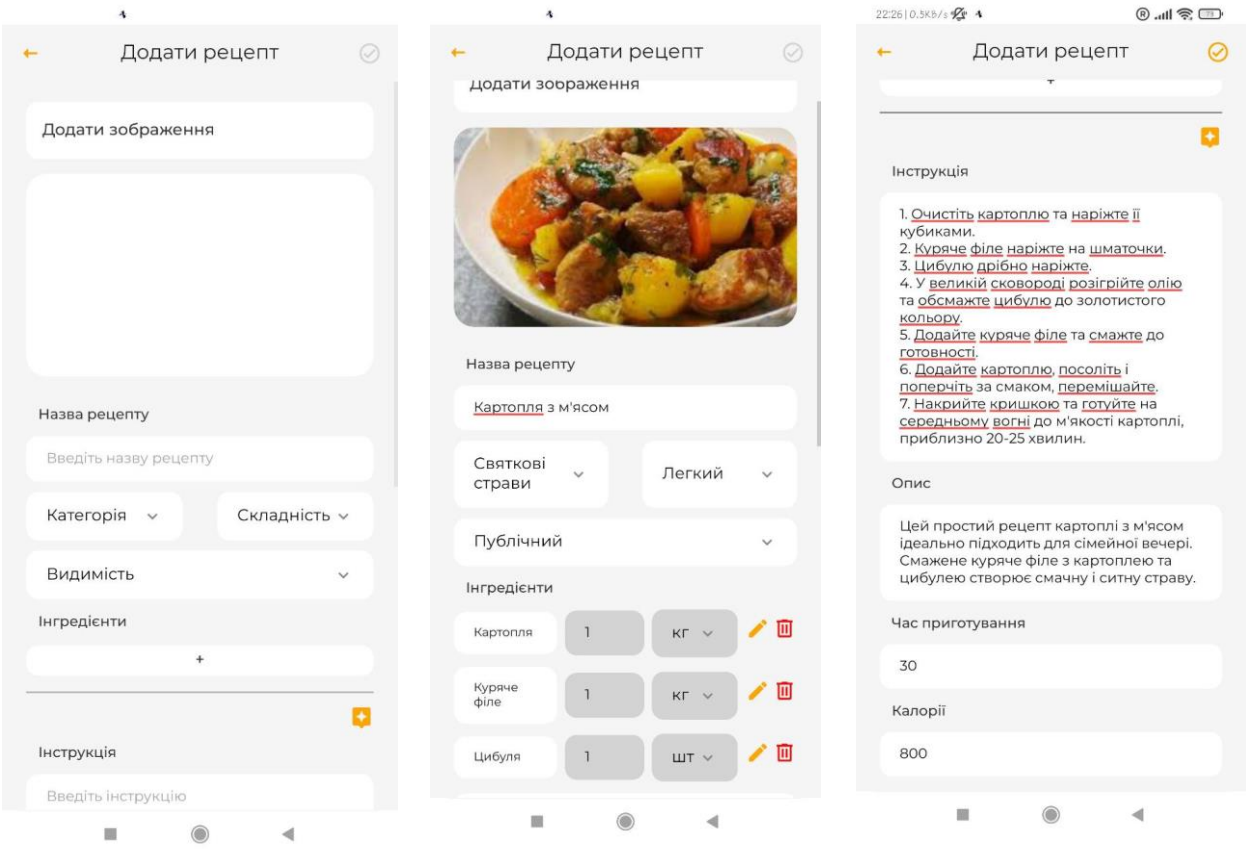

Рисунок 4.4 – Інтерфейс створення нового рецепту

Також після додавання рецепту ШІ автоматично генерує рекомендацію, яка буде відображатися на сторінці рецепта (рисунок 4.5).

Рисунок 4.5 – Детальна інформація про рецепт з рекомендацією від ШІ

Реалізована система пошуку рецептів, яка включає можливість знаходження страв за наявними інгредієнтами, категоріями і сортуванням. Користувачі можуть вводити декілька інгредієнтів одночасно та отримувати релевантні результати (рисунок 4.6).

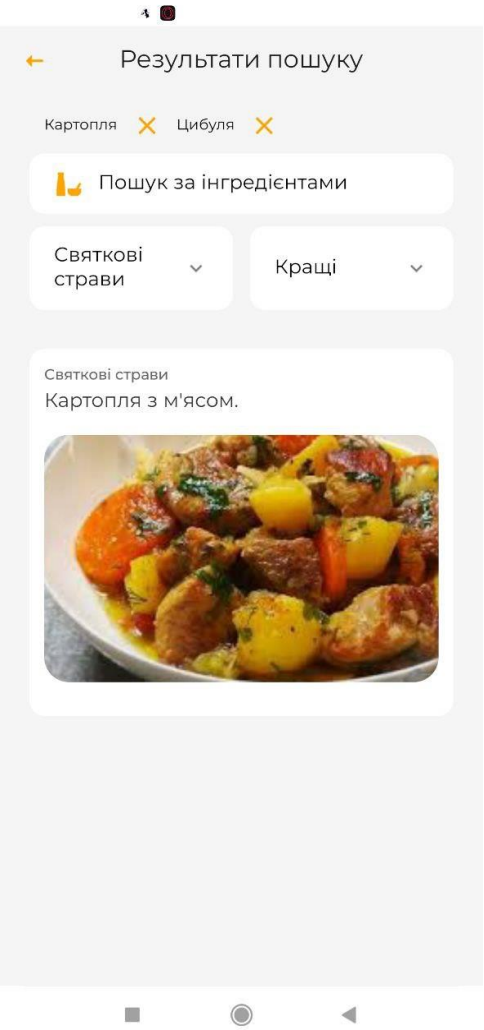

Рисунок 4.6 – Інтерфейс пошуку рецептів

Тестування кросплатформенного застосунку для обміну рецептами проводилося комплексно на всіх рівнях архітектури системи. Основними цілями тестування були перевірка коректності роботи бізнес-логіки, валідація взаємодії між компонентами системи, тестування інтеграції з зовнішніми сервісами та перевірка функціональності користувацького інтерфейсу. Окрема увага приділялася валідації безпеки та авторизації користувачів, оскільки застосунок працює з персональними даними та створеним контентом.

Для забезпечення якості системи використовувався багаторівневий підхід до тестування, що включав модульне тестування для перевірки

окремих функцій та методів, інтеграційне тестування для валідації взаємодії між компонентами, функціональне тестування через детальний аналіз користувацьких сценаріїв, тестування API за допомогою Postman. Такий комплексний підхід дозволив виявити та усунути потенційні проблеми на різних етапах розробки.

4.2 Модульне тестування

Для модульного тестування серверної частини використовувався фреймворк Jest.

Основні групи тестів:

- тестування сервісів (лістинг 4.1);
- тестування утилітарних функцій (лістинг 4.2);
- тестування AI-сервісу (лістинг 4.3).

Лістинг 4.1 - Приклад тестування UserService

```
describe('UserService', () => {
    test('should create user without returning password field',
async () => {
        const userData = {
            email: 'test@example.com',
            name: 'Test User',
            password: 'password123'
        };

        const user = await userService.registration(userData);
        expect(user.email).toBe(userData.email);
        expect(user.isActivated).toBe(false);
        expect(user).not.toHaveProperty('password');
    });
});
    test('should throw error for duplicate email', async () => {
        const userData = {
            email: 'existing@example.com',
            name: 'Test User',
            password: 'password123'
        };
        await expect(userService.registration(userData))
            .rejects.toThrow('Користувач з таким email вже
існує');
    });});
```

Лістинг 4.2 - Тестування JWT утиліт

```
describe('JWT Utils', () => {
    test('should generate valid access token', () => {
        const payload = { id: '123', email: 'test@test.com' };
        const token = generateAccessToken(payload);
        expect(token).toBeDefined();
        expect(typeof token).toBe('string');
        const decoded = verifyToken(token);
        expect(decoded.id).toBe(payload.id);
    });

    test('should hash password correctly', () => {
        const password = 'testpassword';
        const hashed = hashPassword(password);
        expect(hashed).not.toBe(password);
        expect(comparePasswords(password, hashed)).toBe(true);
    });
});
```

Лістинг 4.3 – Приклад тестування методу з AiAssistantService

```
describe('AiAssistantService', () => {
    test('should generate recipe details', async () => {
        const input = {
            title: 'Борщ',
            category: 'Перші страви',
            ingredients: ['буряк', 'капуста', 'морква']
        };

        const result = await
aiAssistantService.generateRecipeDetails(input);

        expect(result).toHaveProperty('instructions');
        expect(result).toHaveProperty('cookTime');
        expect(result).toHaveProperty('nutritionInfo');
    });

    test('should correctly handle empty input ', async () => {
        const result = await
aiAssistantService.generateRecipeDetails({});
        expect(result).toEqual({});
    });
});
```

Для тестування React Native компонентів використовувався React Native Testing Library (лістинг 4.4).

Лістинг 4.4 - Тестування компонента пошуку

```
describe('Search Component', () => {
    test('renders search input correctly', () => {
        const { getByPlaceholderText } = render(<Search />);
        const searchInput = getByPlaceholderText('Пошук');

        expect(searchInput).toBeTruthy();
    });

    test('handles search input changes', () => {
        const { getByPlaceholderText } = render(<Search />);
        const searchInput = getByPlaceholderText('Пошук');

        fireEvent.changeText(searchInput, 'борщ');
        expect(searchInput.props.value).toBe('борщ');
    });
});
```

4.3 Тестування API через Postman

Для комплексного тестування REST API було створено колекцію в Postman, яка охоплює всі доступні маршрути серверної логіки застосунку (рисунок 4.7). Колекція організована за функціональними групами, що дозволяє проводити як окремі тести специфічних функцій (рисунок 4.8), так і комплексні сценарії, що включають декілька взаємопов'язаних запитів. Тестування охоплює повний CRUD функціонал, включаючи створення, читання, оновлення та видалення.

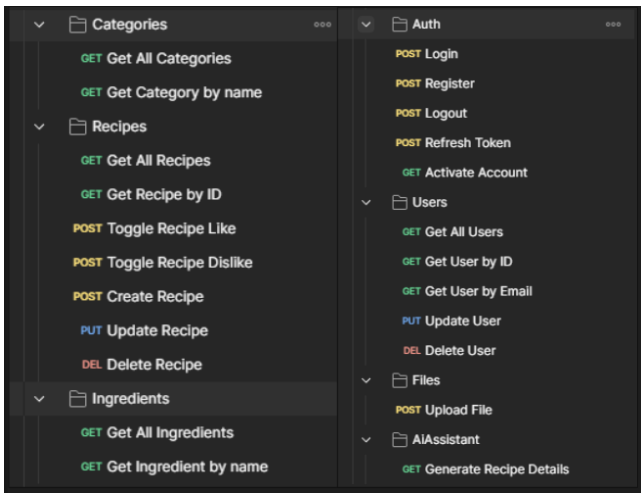

Рисунок 4.7 – Створена колекція у Postman

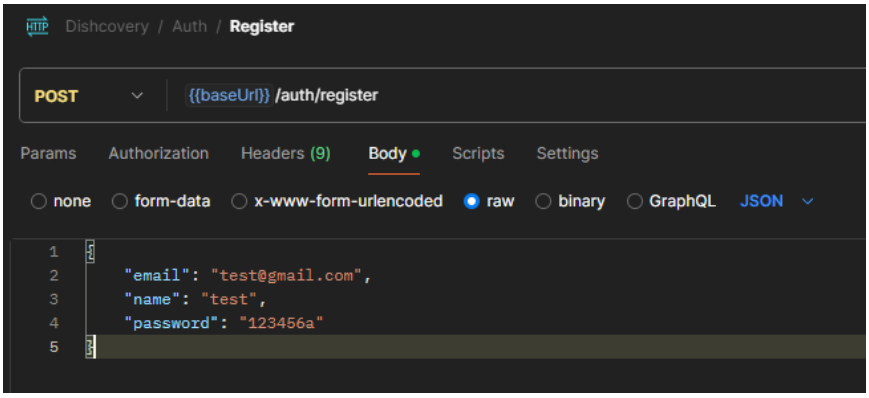

Рисунок 4.8 – Приклад запиту

У Postman існує можливість створювати автоматизовані тести, які виконуються одразу після відправлення запиту. Це дозволяє автоматично перевіряти коректність відповіді сервера, зменшувати кількість ручної перевірки та швидко виявляти помилки на етапі розробки.

Для кожного HTTP-запиту можна написати JavaScript-код у вкладці Scripts, який перевіряє статус відповіді, наявність або відсутність певних полів у JSON-об'єкті, типи даних, відповідність значень тощо (рисунок 4.9). Це особливо корисно при побудові автоматизованих сценаріїв у колекціях, які можна запускати послідовно, і перевіряти результат виконання кожного кроку.

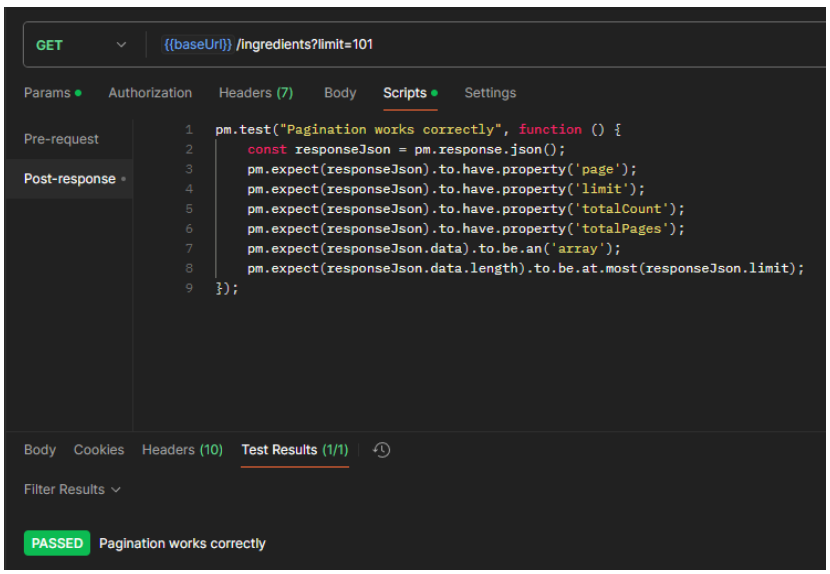

Рисунок 4.9 – Приклад автоматичного тесту у Postman

4.4 Тестування помилкових сценаріїв

Окрема увага приділялася тестуванню різноманітних помилкових сценаріїв для забезпечення стабільності системи в несприятливих умовах. Тести для неавторизованого доступу перевіряють правильність повернення 401 статус коду та відповідних повідомлень про помилки при спробах доступу до захищених маршрутів без валідного JWT токена. Валідація вхідних даних тестується через надсилання запитів з некоректними або неповними даними. Система повинна повертати 400 статус код з повідомленнями про конкретні помилки валідації, що допомагає клієнтським застосункам надавати користувачам зрозумілі інструкції для виправлення помилок. Тестування обробки неіснуючих ресурсів включає спроби доступу до рецептів або користувачів з несправжніми ідентифікаторами. Система повинна коректно повертати 404 статус код.

4.5 Чекліст тестування

Використання чеклісту дозволяє структурувати процес тестування, забезпечити повне покриття функціональності та систематично документувати результати перевірок. Кожен тест-кейс може бути виконаний незалежно, що спрощує процес регресійного тестування при внесенні змін до коду застосунку. Такий підхід значно підвищує якість продукту та зменшує ймовірність пропуску критичних помилок перед релізом.

Структура чеклісту дозволяє легко відстежувати прогрес тестування та документувати результати. Кожен тест-кейс містить опис компонента, що тестується, детальний опис сценарію, покрокові інструкції для відтворення та очікуваний результат (таблиця 4.1).

Таблиця 4.1 – Приклад чеклісту з test cases

ID	Компонент	Опис	Кроки для відтворення	Баги	Очікуваний результат
TC-001	Авторизація	Перевірка входу користувача з валідними даними	1. Відкрити екран входу. 2. Ввести коректний email і пароль. 3. Натиснути кнопку "Увійти".	-	Користувача успішно авторизовано, відкривається головний екран.
TC-002	Авторизація	Перевірка входу з невірними даними	1. Відкрити екран входу. 2. Ввести некоректний email або пароль. 3. Натиснути кнопку "Увійти".	-	Відображається повідомлення про помилку "Невірний email або пароль".
TC-003	Реєстрація	Перевірка реєстрації нового користувача	1. Відкрити екран реєстрації. 2. Заповнити всі обов'язкові поля (email, ім’я, пароль). 3. Натиснути кнопку "Зареєструватися".	-	Користувача зареєстровано, надіслано листа для активації акаунта.
TC-004	Пошук рецептів	Перевірка пошуку рецептів за ключовим словом	1. Відкрити екран пошуку. 2. Ввести ключове слово 3. Підтвердити.	-	Відображаються рецепти, що відповідають ключовому слову.
TC-005	Пошук рецептів	Перевірка пошуку з порожнім запитом	1. Відкрити екран пошуку. 2. Залишити поле пошуку порожнім. 3. Натиснути кнопку "Знайти".	-	Відображаються всі доступні рецепти або повідомлення "Введіть запит".
TC-006	Додавання рецепта	Перевірка додавання нового рецепта	1. Відкрити екран створення рецепта. 2. Заповнити всі обов'язкові поля (назва, інгредієнти, інструкції). 3. Зберегти.	-	Рецепт успішно збережено і відображається у списку.

Продовження таблиці 4.1

ID	Компонент	Опис	Кроки для відтворення	Баги	Очікуваний результат
ТС-007	Перегляд рецепта	Перевірка відображення деталей рецепта	1. Відкрити список рецептів. 2. Обрати будь-який рецепт.	-	Відображаються всі деталі рецепта: назва, інгредієнти, інструкції, автор.
ТС-008	AI-помічник	Перевірка генерації опису рецепта	1. Відкрити екран створення рецепта. 2. Ввести назву та інгредієнти. 3. Сгенерувати опис.	-	Відображається згенерований опис рецепта.
ТС-009	Профіль користувача	Перевірка редагування профілю	1. Відкрити екран профілю. 2. Натиснути кнопку "Редагувати". 3. Змінити ім’я. 4. Натиснути кнопку "Зберегти".	-	Дані профілю успішно оновлено.
ТС-010	Безпека	Захист від неавторизованого доступу	1. Спробувати відкрити екран профілю без авторизації.	-	Користувач перенаправляється на екран входу.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було проведено аналіз предметної області мобільних застосунків для пошуку та публікації кулінарних рецептів. Дослідження показало, що сучасний ринок кулінарних додатків характеризується високою конкуренцією та різноманітністю підходів до реалізації функціональності.

На основі проведеного аналізу було розроблено мобільний застосунок для пошуку та публікації кулінарних рецептів, що враховує основні вимоги користувачів, зокрема простоту інтерфейсу, швидкий доступ до рецептів, можливість публікації власних ідей та надійний захист даних користувачів. Архітектурне рішення базується на сучасних принципах розробки з чітким розділенням клієнтської та серверної логіки, що забезпечує масштабованість та підтримуваність системи.

Завдяки використанню таких сучасних технологій, як React Native для клієнтської логіки та Node.js, Express.js та MongoDB для серверної логіки, було забезпечено високу ефективність роботи застосунку та можливість підтримки кросплатформенності для користувачів iOS та Android. Вибір React Native як основної платформи розробки дозволив значно скоротити час розробки та забезпечити єдину кодову базу для обох мобільних платформ, зберігаючи при цьому нативну продуктивність та користувацький досвід.

Серверна архітектура, побудована на Node.js та Express.js, забезпечує високу швидкість обробки запитів та можливість горизонтального масштабування при зростанні навантаження. Використання NoSQL бази даних MongoDB дозволяє ефективно зберігати та обробляти різноманітні типи даних.

Особливу увагу було приділено забезпеченню безпеки та захисту особистих даних користувачів. Реалізована система аутентифікації та авторизації на основі JWT токенів забезпечує надійний контроль доступу до

персональних даних та функцій застосунку. Додатково впроваджено шифрування паролів за допомогою bcrypt та валідацію вхідних даних для запобігання потенційним атакам.

У ході розробки було здійснено багаторівневе тестування на різних етапах життєвого циклу проекту, включаючи модульне тестування окремих компонентів, інтеграційне тестування взаємодії між модулями та користувацьке тестування інтерфейсу. Це дозволило виявити та усунути потенційні проблеми на ранніх стадіях розробки, забезпечивши стабільну роботу застосунку та високу якість користувацького досвіду.

Результатом роботи є повнофункціональний мобільний застосунок, який успішно вирішує поставлені завдання та відповідає сучасним стандартам розробки мобільних додатків. Застосунок демонструє високу продуктивність та зручність використання.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Food Recipe Finder Mobile Applications Based On Similarity Of Materials – DOI: 10.1109/SIET.2018.8693218

2. Mobile application development: web vs. native – DOI: 10.1145/1941487.1941504

3. Developing hybrid mobile applications for learning – DOI: 10.1109/ISETC.2018.8584005

4. Cross-platform mobile development approaches – DOI: 10.1109/CIST.2014.7016616

5. Progressive web apps: An alternative to the native mobile Apps – DOI: 10.23919/CISTI.2018.8399228

6. WEB BACK-END DEVELOPMENT TECHNOLOGIES – DOI: 10.56726/IRJMETS73418

7. A Comparison of Authentication Protocols for Unified Client Applications – DOI: 10.1109/ISNCC58260.2023.10323861

8. Benchmarking Open-Source Large Language Models, GPT-4 and Claude 2 on Multiple-Choice Questions in Nephrology – DOI: 10.1056/AIdbp2300092

9. Enhancing user experience in mobile applications through AI-driven personalization and adaptive learning algorithms – DOI: 10.30574/wjaets.2021.3.2.0064.

10. Impact of mobile cross-platform development on CPU, memory and battery of mobile devices when using common mobile app features – DOI: 10.1016/j.procs.2020.07.029

11. Документація React Native [Електронний ресурс] – режим доступу: https://reactnative.dev/docs/getting-started

12. Документація Expo [Електронний ресурс] – режим доступу: https://docs.expo.dev/

13. Документація Zustand [Електронний ресурс] – режим доступу: https://zustand-demo.pmnd.rs

14. Eisenman, Bonnie. Learning React Native: Building Native Mobile Apps with JavaScript, 2nd Edition [Текст] / Bonnie Eisenman, 2017. - 242 с.

15. Node.js в дії. 2-е видання [Текст] / Майк Хартер, 2018. - 432 с.

16. Using Express.js to Create Node.js Web Apps – DOI: 10.1007/978-1-4842-3039-8_2

17. JSON Web Token (JWT) – DOI: 10.17487/RFC7519

18. A qualitative analysis of the performance of MongoDB vs MySQL database – DOI: 10.1109/I-SMAC.2017.8058365

19. Create a MongoDB Atlas cluster – DOI: 10.17504/protocols.io.rm7vzb1r5vx1/v1

20. Denormalization strategies for data retrieval from data warehouses – DOI: 10.1016/j.dss.2004.12.004

21. Using MongoDB with Node.js – DOI: 10.1007/978-1-4842-1598-2_5