**ДИПЛОМНА РОБОТА**

**ДР.122.041.004.ПЗ**

**НАУМЧУК ДІАНА БОГДАНІВНА**

**2025**

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЖИТОМИРСЬКИЙ АГРОТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ

ВІДДІЛЕННЯ «Інженерна інфраструктура та комп'ютерні науки»

ЦИКЛОВА КОМІСІЯ «Інженерна інфраструктура та комп'ютерні науки»

**ПОЯСНЮВАЛЬНА ЗАПИСКА**

до дипломної роботи освітнього ступеня «фаховий молодший бакалавр»

за спеціальністю 122 Комп’ютерні науки

(освітня програма 122 «Комп’ютерні науки»)

на тему:

**«Розробка платформи для обміну рецептами»**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Студентки | IV | | курсу | | | П-41 | групи | | | | |
| Наумчук Діани Богданівни | | | | | | | | | | |
| (ПІБ) | | | | | | | | | | |
| Керівник | | Габрійчук Наталя Ігорівна | | | | | | | | |
| Рецензент | | Можаровський Сергій Володимирович | | | | | | | | |
|  | | | | | | | | | | |
| Національна шкала | | | | |  | | | | | |
| Кількість балів | | | |  | | | | Оцінка: ECTS | |  |
|  | | | |  | | | |  | |  |
| Члени комісії | | | |  | | | | | Ісаєв А.М. | |
|  | | | | (підпис) | | | | | (прізвище та ініціали) | |
|  | | | |  | | | | | Габрійчук Н.І. | |
|  | | | | (підпис) | | | | | (прізвище та ініціали) | |
|  | | | |  | | | | | Устименко Л.М. | |
|  | | | | (підпис) | | | | | (прізвище та ініціали)  Устименко Я.І. | |
|  | | | | (підпис) | | | | | (прізвище та ініціали) | |
|  | | | |

Житомир – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЖИТОМИРСЬКИЙ АГРОТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ

ВІДДІЛЕННЯ «Інженерна інфраструктура та комп'ютерні науки»

ЦИКЛОВА КОМІСІЯ «Інженерна інфраструктура та комп'ютерні науки»

**«ЗАТВЕРДЖУЮ»**

Голова циклової комісії «Інженерна

інфраструктура та комп'ютерні науки»

Діана ПАЛІЙ

«06» листопада 2024 р.

**ЗАВДАННЯ**

**на дипломну роботу**

Здобувач вищої освіти: **НАУМЧУК Діана Богданівна**

Керівник роботи: **гАБРІЙЧУК Наталя Ігорівна**

Тема роботи: **«Розробка платформи для обміну рецептами»**, затверджена наказом закладу вищої освіти від «03» грудня 2024 р., №496н.

Вихідні дані для роботи: предметом дослідження є дизайн і розробка платформи для обміну рецептами для надання онлайн послуг клієнтам, популяризації платформи та задоволення потреб клієнтів.

|  |  |  |  |
| --- | --- | --- | --- |
| Розділ | Консультант | Завдання  видав | Завдання прийняв |
| 1 | Наталя ГАБРІЙЧУК | 13.12.2024 | 21.02.2025 |
| 2 | Наталя ГАБРІЙЧУК | 24.02.2025 | 10.04.2025 |
| 3 | Наталя ГАБРІЙЧУК | 10.04.2025 | 30.05.2025 |

Консультанти з дипломної роботи із зазначенням розділів, що їх стосується:

**Календарний план**

|  |  |  |  |
| --- | --- | --- | --- |
| № з/п | Етап роботи | Термін виконання | Примітка |
| 1 | Формулювання мети та цілей роботи, визначення завдань розробки | 13 грудня 2024 | Виконано |
| 2 | Планування розробки | 24 січня 2025 | Виконано |
| 3 | Проєктування користувацької частини сайту | 24 лютого 2025 | Виконано |
| 4 | Проєктування дизайну проекту | 11 березня 2025 | Виконано |
| 5 | Розробка алгоритмів | 26 березня 2025 | Виконано |
| 6 | Перевірка роботи алгоритмів | 2 квітня 2025 | Виконано |
| 7 | Проектування функцій проекту | 10 квітня 2025 | Виконано |
| 8 | Налагодження взаємодії між елементами | 18 квітня 2025 | Виконано |
| 9 | Розробка дизайну для різних пристроїв | 2 травня 2025 | Виконано |
| 10 | Тестування всіх функцій сайту | 21 травня 2025 | Виконано |
| 11 | Систематизація результатів розробки, написання та оформлення ПЗ | 30 травня 2025 | Виконано |
| 12 | Попередній захист роботи | 13 червня 2025 | Виконано |

Здобувач фахової передвищої освіти\_\_\_\_\_\_\_\_\_\_ Діана НАУМЧУК

Керівник \_\_\_\_\_\_\_\_\_\_ Наталя ГАБРІЙЧУК

РЕФЕРАТ

Записка: 74 стoр., 12 рис., 10 джерел.

Ключові слова: МОБІЛЬНИЙ ЗАСТОСУНОК, ОБМІН РЕЦЕПТАМИ, КУЛІНАРІЯ, NODE.JS, EXPRESS.JS, REACT NATIVE, MONGODB.

Об'єкт дослідження — процеси розробки та функціонування мобільних застосунків для обміну кулінарними рецептами.

Мета роботи — створення повнофункціонального мобільного застосунку для обміну кулінарними рецептами, який забезпечує користувачам зручний інтерфейс для публікації, перегляду, організації та взаємодії з рецептами.

Методи дослідження — методи аналізу сучасних технологій мобільної розробки, дослідження підходів до проектування клієнт-серверних архітектур, методи забезпечення безпеки даних та аутентифікації користувачів.

Результати — програмну реалізацію мобільного застосунку з функціоналом публікації та перегляду рецептів, системою реєстрації та авторизації, а також функціоналом для збереження улюблених страв.

ABSTRACT

The bachelor work consists of the source code of a mobile application for exchanging culinary recipes, along with documentation. The documentation has 74 pages, contains 12 figures and 10 sources.

The object of research is the processes of developing and functioning of mobile applications for exchanging culinary recipes.

The goal is to create a full-featured mobile application for exchanging culinary recipes, which provides users with a convenient interface for publishing, viewing, organizing, and interacting with recipes.

The research methods include analysis of modern mobile development technologies (in particular, the JavaScript stack), research of approaches to designing client-server architectures, methods for ensuring data security and user authentication.

The results include the mobile application has been programmatically implemented with functionality for publishing and viewing recipes.

Key words: WEB PLATFORM, ONLINE STORE, MOBILE PHONES, NODE.JS, EXPRESS.JS, REACT, REDUX, MONGODB.

**ЗМІСТ**

[ВСТУП 5](#_Toc200058473)

[РОЗДІЛ 1. АНАЛІЗ ТЕХНЛОГІЙ ДЛЯ РОЗРОБКИ ПЛАТФОРМИ ДЛЯ ОБМІНУ РЕЦЕПТАМИ 6](#_Toc200058474)

[1.1. Постановка основної задачі проекту 6](#_Toc200058475)

[1.2. Вибір технологій для розробки платформи 8](#_Toc200058476)

[Висновки до розділу 1 10](#_Toc200058477)

[РОЗДІЛ 2. ПРОЄКТУВАННЯ ПЛАТФОРМИ ДЛЯ ОБМІНУ РЕЦЕПТАМИ 11](#_Toc200058478)

[2.1. Аналіз вимог та потреб платформи 11](#_Toc200058479)

[2.2. Функціональні вимоги 12](#_Toc200058480)

[2.3. Нефункціональні вимоги 13](#_Toc200058481)

[2.4. Дослідження конкурентів 14](#_Toc200058482)

[2.5. Визначення вимог до користувацького інтерфейсу 15](#_Toc200058483)

[2.6. Дизайн інтерфейсу користувача 16](#_Toc200058484)

[Висновки до розділу 2 17](#_Toc200058485)

[РОЗДІЛ 3. РОЗРОБКА ПЛАТФОРМИ ДЛЯ ОБМІНУ РЕЦЕПТАМИ 18](#_Toc200058486)

[3.1. Структура проєкту 18](#_Toc200058487)

[3.2. Структура платформи 22](#_Toc200058488)

[3.3. Технічна реалізація платформи 29](#_Toc200058489)

[Висновки до розділу 3 72](#_Toc200058490)

[ВИСНОВКИ 73](#_Toc200058491)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 74](#_Toc200058492)

# ВСТУП

У сучасному цифровому світі обмін знаннями й досвідом набуває нових форм завдяки технологіям. Кулінарія — одна з тих сфер, де онлайн-спільноти відіграють важливу роль: люди активно діляться рецептами, обговорюють смаки, експериментують і надихають одне одного. Водночас, попри велику кількість сайтів та блогів, існує потреба у зручній, мобільній та інтерактивній платформі, орієнтованій саме на обмін рецептами.

Мій дипломний проєкт присвячений розробці платформи для обміну кулінарними рецептами. Це мобільний застосунок, який дозволяє користувачам створювати, переглядати, шукати, зберігати рецепти, а також взаємодіяти з іншими користувачами через оцінки, коментарі та обране.

Основна мета — створити сучасну мобільну платформу, яка забезпечить: авторизацію користувачів і захист особистих даних, додавання рецептів з описом, інгредієнтами, фотографіями, зручний пошук рецептів за ключовими словами чи категоріями, збереження улюблених рецептів, перегляд та взаємодію з рецептами інших користувачів.

Для реалізації проєкту використано сучасні інструменти розробки як на клієнтській, так і на серверній стороні:

Клієнт: створено за допомогою React Native та Expo, що дозволило реалізувати кросплатформений мобільний застосунок. Для HTTP-запитів використано Axios.

Сервер: побудований на основі Node.js з використанням TypeScript та Express. Для обробки авторизації реалізовано JWT (jsonwebtoken), збереження зображень реалізовано через Multer, а логування — за допомогою Winston.

База даних: використано MongoDB разом із TypeORM для зручної роботи з документами та структурованими запитами.

Розробка цього застосунку спрямована на створення функціонального та зручного сервісу, який об'єднує кулінарну спільноту, сприяє поширенню кулінарного досвіду та підвищує інтерес до домашнього приготування страв.

# РОЗДІЛ 1. АНАЛІЗ ТЕХНЛОГІЙ ДЛЯ РОЗРОБКИ ПЛАТФОРМИ ДЛЯ ОБМІНУ РЕЦЕПТАМИ

**1.1. Постановка основної задачі проекту**

Завданням даного проєкту є створення зручного, інтуїтивно зрозумілого та сучасного мобільного застосунку для обміну кулінарними рецептами, який дозволить користувачам публікувати власні рецепти, переглядати рецепти інших користувачів, взаємодіяти між собою, а також зберігати улюблені страви у власній кулінарній колекції.

Для досягнення цієї мети необхідно реалізувати такі завдання:

1. Аналіз потреб користувачів: Дослідити потреби цільової аудиторії, включаючи початківців і досвідчених кулінарів. Визначити основні функціональні вимоги до застосунку: реєстрація, додавання рецептів, фільтрація, пошук, оцінки, коментарі тощо.

2. Розробка архітектури застосунку: Спроєктувати архітектуру клієнтської та серверної частини застосунку. Визначити основні компоненти: авторизація, завантаження зображень, збереження рецептів, система оцінювання, структура бази даних.

3. Проєктування інтерфейсу користувача: Створити адаптивний, мінімалістичний та приємний дизайн інтерфейсу. Забезпечити просту навігацію між основними екранами: список рецептів, детальна інформація, створення/редагування рецепту, профіль користувача тощо.

4. Реалізація функціоналу застосунку: Розробити серверну частину з використанням Node.js, Express, TypeScript, MongoDB, TypeORM, з підтримкою авторизації через JWT, збереженням зображень через Multer та логуванням з Winston. Реалізувати клієнтську частину на React Native з використанням Expo і Axios для взаємодії з сервером.

5. Тестування та вдосконалення: Провести тестування застосунку на Android та iOS-пристроях. Перевірити коректність роботи основних функцій, зручність користування, відповідність дизайну. Виявити й усунути баги та неточності.

Очікуваним результатом виконання цих завдань стане повнофункціональна мобільна платформа для обміну рецептами, яка не тільки задовольнятиме базові потреби користувачів, але й сприятиме формуванню активної кулінарної онлайн-спільноти. Створений застосунок надасть користувачам зручний інструмент для організації власної кулінарної творчості, пошуку натхнення та обміну досвідом. Його модульна архітектура та вибір сучасних технологій забезпечать легкість подальшого масштабування та інтеграції нових функцій, таких як персоналізовані рекомендації на основі вподобань, планування меню, або інтеграція з магазинами продуктів.

Важливим аспектом успіху проєкту є також забезпечення високої продуктивності та надійності. Це означає оптимізацію завантаження зображень, швидку обробку запитів до бази даних, а також забезпечення стабільної роботи застосунку навіть при великій кількості одночасних користувачів. Безпека персональних даних та інтеграція надійних механізмів авторизації є пріоритетом, щоб користувачі могли з упевненістю ділитися своїми рецептами та взаємодіяти на платформі. Таким чином, проєкт поєднує технічну досконалість з орієнтацією на користувача, створюючи цінний ресурс для кулінарного світу.

## 1.2. Вибір технологій для розробки платформи

Вибір технологій для розробки мобільного застосунку є критично важливим етапом, від якого залежить зручність користування, швидкодія, масштабованість і підтримка в майбутньому. При виборі інструментів було враховано специфіку задачі — розробку сучасної платформи для обміну кулінарними рецептами з підтримкою авторизації, завантаженням зображень та збереженням даних у хмарній базі.

**Клієнтська частина:**

1. React Native. Кросплатформений фреймворк, який дозволяє створювати мобільні застосунки для Android та iOS з єдиною кодовою базою. Забезпечує високу продуктивність і зручну інтеграцію з нативними можливостями пристроїв.

2. Expo. Інструмент, який спрощує розробку в React Native. Дозволяє швидко тестувати застосунок без складної конфігурації середовища. Підтримує автоматичне оновлення, роботу з камерами, мультимедіа, push-нотифікаціями тощо.

3. Axios. Бібліотека для виконання HTTP-запитів до серверу. Забезпечує обмін даними між клієнтом і сервером, наприклад, надсилання рецептів, авторизацію, отримання списку рецептів тощо.

**Серверна частина:**

1. Node.js. JavaScript-рушій для виконання коду на сервері. Забезпечує високу продуктивність і асинхронну обробку запитів. Ідеально підходить для REST API.

2. TypeScript. Надмножина JavaScript, яка додає типізацію та покращує читабельність і підтримку коду. Дозволяє уникнути багатьох помилок ще на етапі розробки.

3. Express. Легкий і гнучкий фреймворк для Node.js, який використовується для створення REST API. Забезпечує обробку маршрутів, запитів і відповідей між клієнтом і сервером.

4. jsonwebtoken (JWT). Бібліотека для реалізації захищеної авторизації за допомогою токенів. Дозволяє ідентифікувати користувачів і забезпечити доступ до персональних функцій, наприклад, додавання рецептів.

5. Multer. Middleware для Express, який обробляє завантаження файлів (зображень страв) від користувачів.

6. Winston. Бібліотека для логування подій на сервері. Дозволяє відслідковувати помилки, збої, події та забезпечити стабільну роботу системи.

7. MongoDB. Документоорієнтована база даних, що ідеально підходить для зберігання даних без жорсткої схеми — як-от рецепти, користувачі, зображення. Підтримує масштабування та високу швидкодію.

8. TypeORM. Інструмент для роботи з базою даних у стилі ORM (Object-Relational Mapping), який підтримує MongoDB і дозволяє зручно працювати з об'єктами у коді.

Комбінація зазначених технологій дозволяє створити продуктивний, зручний і надійний застосунок, який відповідає сучасним вимогам до мобільних платформ та забезпечує повноцінну взаємодію користувачів із кулінарною спільнотою.

## Висновки до розділу 1

Створення мобільного застосунку для обміну рецептами є ефективним інструментом для розвитку кулінарної спільноти, оскільки він розширює можливості обміну знаннями, пошуку нових ідей для приготування їжі та спілкування між користувачами. Під час виконання дипломного проєкту були реалізовані всі поставлені задачі, що дозволило створити швидкий, функціональний і зручний у використанні застосунок.

Спершу було проведено аналіз потреб потенційних користувачів, що дало змогу визначити основний набір функцій платформи. Було розроблено структуру застосунку, яка включає такі розділи, як список рецептів, детальний перегляд рецепту, можливість додавання та редагування власних рецептів, профіль користувача та система збереження улюблених страв.

Особливу увагу було приділено дизайну інтерфейсу, який виконаний у мінімалістичному стилі, що забезпечує інтуїтивну навігацію і приємний візуальний досвід. Дизайн сприяє швидкому пошуку потрібної інформації і простому взаємодії з функціями застосунку.

Для розробки використані сучасні технології: у серверній частині — Node.js, TypeScript, Express, MongoDB, TypeORM, JWT для авторизації, Multer для роботи з файлами та Winston для логування. Клієнтська частина реалізована на React Native з Expo, що дозволяє запускати застосунок на різних мобільних платформах. Використання Axios забезпечує ефективний обмін даними між клієнтом і сервером.

Проведене тестування на різних пристроях допомогло виявити та усунути помилки, забезпечивши стабільну та швидку роботу застосунку. Це дозволяє користувачам комфортно обмінюватися рецептами, знаходити нові ідеї та спілкуватися в межах платформи.

Розроблений застосунок повністю відповідає поставленим цілям і є зручним, сучасним інструментом для кулінарної спільноти, що сприяє розвитку кулінарних навичок і обміну досвідом серед користувачів.

# РОЗДІЛ 2. ПРОЄКТУВАННЯ ПЛАТФОРМИ ДЛЯ ОБМІНУ РЕЦЕПТАМИ

## 2.1. Аналіз вимог та потреб платформи

Аналіз вимог та потреб платформи для обміну рецептами є першим важливим етапом розробки, що допомагає визначити основні функції та можливості застосунку. Для цього потрібно вивчити сферу кулінарії, цільову аудиторію та завдання, які платформа має виконувати. Основні вимоги включають:

* Надання зручного доступу до різноманітних рецептів;
* Можливість додавати, редагувати та зберігати власні рецепти;
* Функціонал коментування та оцінювання рецептів;
* Інтуїтивний та привабливий дизайн;
* Адаптивність інтерфейсу для різних пристроїв.

Для точнішої реалізації важливо розуміти цільову аудиторію платформи. Основні користувачі:

* Любителі кулінарії: люди, які шукають нові ідеї для приготування страв, експериментують з рецептами;
* Професійні кухарі та блогери: користувачі, які діляться власними унікальними рецептами та шукають відгуки;
* Новачки у кулінарії: користувачі, які потребують простих інструкцій і порад.

## 2.2. Функціональні вимоги

Для забезпечення зручного користування платформою були визначені такі основні задачі:

1. Головна сторінка: показує популярні та нові рецепти, а також пошук за назвами чи інгредієнтами.

2. Сторінка рецепту: містить детальний опис рецепту, список інгредієнтів, покрокові інструкції приготування, фото/відео.

3. Додавання та редагування рецепту: користувачі можуть створювати нові рецепти, додавати фото, змінювати інформацію та зберігати їх у власному профілі.

4. Підписки користувача: зберігає улюблені рецепти та історію доданих рецептів.

5. Коментування та оцінювання: можливість залишати відгуки, ставити оцінки та обговорювати рецепти.

6. База даних: зберігає інформацію про рецепти, користувачів, коментарі та оцінки для подальшого аналізу та вдосконалення платформи.

## 2.3. Нефункціональні вимоги

Окрім функціональності, необхідно врахувати й такі аспекти, що забезпечать якісну роботу платформи:

1. Адаптивність: коректне відображення та функціонування на різних пристроях і роздільних здатностях екранів.

2. Швидкодія: оптимізація часу завантаження сторінок і обробки запитів для комфортного користування.

3. Надійність: стабільна робота платформи без збоїв та втрат даних.

4. Безпека: захист персональних даних користувачів, особливо в частині реєстрації, авторизації та обробки контенту.

5. Масштабованість: можливість подальшого розширення функціоналу і додавання нових сервісів.

6. Зручність використання: інтуїтивно зрозумілий інтерфейс з простою навігацією, що дозволяє швидко знаходити потрібну інформацію і функції.

## 2.4. Дослідження конкурентів

Аналіз платформ для обміну рецептами допоміг виділити найкращі практики та уникнути поширених помилок. Основні висновки, зроблені під час перегляду конкурентних ресурсів:

1. Структура: Розділи платформи логічні, прості для розуміння та зручні для користувача.

2. Дизайн: Мінімалістичний і сучасний стиль з приємною для ока кольоровою гамою.

3. Функціональність: Можливість легко додавати, редагувати та шукати рецепти.

4. Адаптивність і швидкість: Платформа коректно працює на різних пристроях і швидко завантажується.

5. Соціальна взаємодія: Наявність коментарів, лайків, оцінок і можливості ділитися рецептами в соцмережах.

6. Пошук і фільтраці: Зручні інструменти для пошуку рецептів за назвами, категоріями.

## 2.5. Визначення вимог до користувацького інтерфейсу

На основі аналізу конкурентів і потреб цільової аудиторії сформовані ключові вимоги до інтерфейсу платформи:

1. Простота і мінімалізм: Інтерфейс має бути зрозумілим і доступним навіть для новачків.

2. Зручна навігація: Користувачі повинні швидко знаходити потрібні рецепти і розділи платформи.

3. Стильний дизайн: Візуальне оформлення повинно відображати кулінарну тематику, надихати на приготування їжі.

4. Швидка реакція: Платформа має миттєво реагувати на дії користувача без затримок і помилок.

5. Інформативність: Чітке подання інформації про рецепт, автора, інгредієнти та відгуки.

Ці вимоги допоможуть сформувати чітке бачення проекту і визначити подальші кроки у розробці.

## 2.6. Дизайн інтерфейсу користувача

Візуальна складова платформи є важливим фактором залучення і утримання користувачів, а також підвищення їхньої активності. Основні принципи дизайну:

1. Ефективність користування: Інтерфейс має бути простим і зрозумілим, щоб користувачі могли легко додавати, шукати і переглядати рецепти.

2. Єдність стилю: Всі сторінки платформи повинні мати послідовний дизайн і структуру для комфортної навігації.

3. Візуальна привабливість: Використання яскравих, але ненав’язливих кольорів, які асоціюються з їжею і кулінарією, а також якісних зображень страв.

4. Відступи та простір: Раціональне використання білого простору для полегшення сприйняття тексту і елементів.

5. Типографіка: Вибір шрифтів, які легко читаються та добре поєднуються із загальним стилем сайту.

## Висновки до розділу 2

Проєктування платформи для обміну рецептами — це комплексне завдання, що включає аналіз вимог, розробку функціоналу та створення привабливого дизайну. Проаналізувавши потреби майбутніх користувачів та особливості ринку кулінарних сервісів, ми визначили основні завдання платформи та сформували ключові вимоги до неї. Проведене дослідження конкурентів допомогло обрати оптимальні підходи до реалізації проекту.

Завдяки продуманій стратегії розробки, ми отримали чітке уявлення про кінцевий вигляд платформи та найкращі шляхи досягнення поставлених цілей. У результаті був створений зрозумілий та привабливий дизайн, що враховує сучасні тенденції та потреби користувачів. Основні функціональні елементи включають каталог рецептів, можливість додавання нових рецептів, коментування, пошук та профілі користувачів.

Важливою складовою є швидкість роботи платформи та її адаптивність під різні пристрої — від десктопів до мобільних телефонів. Ці показники були ретельно перевірені під час тестування, щоб забезпечити комфортний користувацький досвід.

Застосування сучасних технологій дозволяє платформі ефективно працювати з великим обсягом даних, забезпечувати надійність та масштабованість. Це сприяє залученню нових користувачів, підтримці активної спільноти та створенню якісного сервісу для обміну кулінарними ідеями.

Загалом, нам вдалося створити платформу з простим і функціональним дизайном, швидкою продуктивністю та зручними інструментами, що допомагають розвивати кулінарне спільноту і вдосконалювати обмін рецептами.

# РОЗДІЛ 3. РОЗРОБКА ПЛАТФОРМИ ДЛЯ ОБМІНУ РЕЦЕПТАМИ

## 3.1. Структура проєкту

Структура проєкту відіграє ключову роль у підтримці зрозумілості, масштабованості та ефективної співпраці під час розробки платформи для обміну рецептами. У даному випадку проєкт поділений на дві основні частини: клієнтську частину та серверну частину. Такий поділ дозволяє розмежувати відповідальності між інтерфейсом користувача та логікою обробки запитів.

Клієнтська частина:

Клієнт реалізовано на React Native з використанням Expo, що дозволяє створювати кросплатформений мобільний застосунок. Структура клієнтської частини містить такі основні папки (див. рис. 3.1.1):

* components/UI/ — набір багаторазових візуальних компонентів, таких як CreatePost, Header, Footer, ProfileMenu, Category, Comment, Post тощо.
* hooks/ — містить користувацькі хуки, які інкапсулюють логіку роботи з API (коментарі, рецепти, профіль).
* screens/ — папка з основними екранами застосунку, зокрема HomeScreen, LoginScreen, CreatePostScreen, RegisterScreen, FullPostScreen, MyRecipesScreen, PostCommentScreen тощо.
* assets/ — зберігає статичні ресурси (іконки, зображення).
* .expo/ — службова папка, що використовується Expo для зберігання конфігурації.

Серверна частина (Backend)

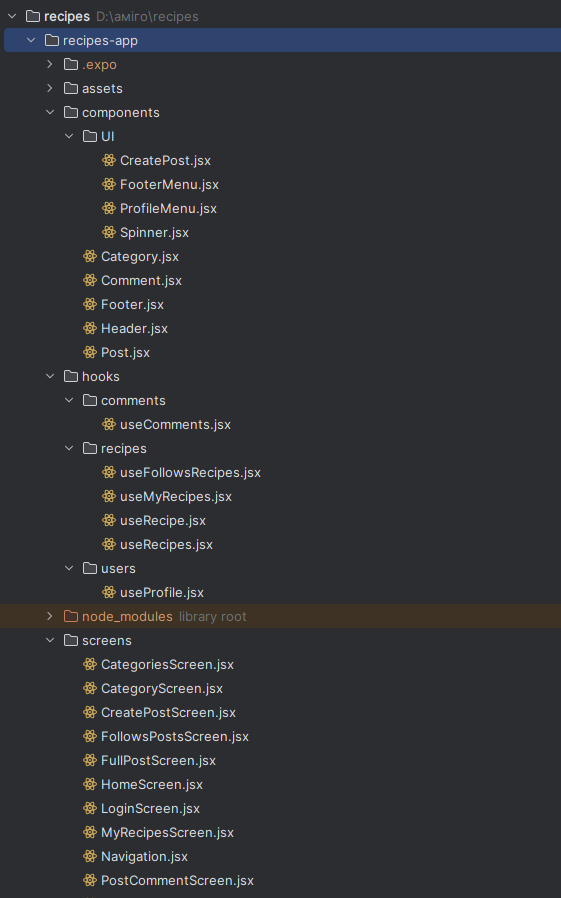


Рисунок 3.1.1

Сервер розроблено з використанням Node.js, Express і TypeScript, з використанням ORM бібліотеки TypeORM для роботи з базою даних MongoDB.

Структура включає (див. рис. 3.1.2):

* controllers/ — містить контролери auth.controller.ts, comments.controller.ts, recipes.controller.ts, user.controller.ts, які обробляють HTTP-запити.
* entity/ — опис сутностей бази даних (recipes.entity.ts, comments.entity.ts, user.entity.ts), які відповідають за структуру таблиць/колекцій.
* services/ — бізнес-логіка проєкту, що реалізує роботу з рецептами, користувачами та коментарями.
* types/ — опис інтерфейсів для типізації даних (recipes.interface.ts, user.interface.ts).
* utils/ — допоміжні утиліти, зокрема AuthCheck.ts (перевірка JWT), multer.ts (обробка файлів), log.ts (логування).
* uploads/ — директорія для зберігання завантажених зображень рецептів.
* db/ — налаштування підключення до бази даних (app-data-source.ts).
* index.ts — головна точка входу сервера.

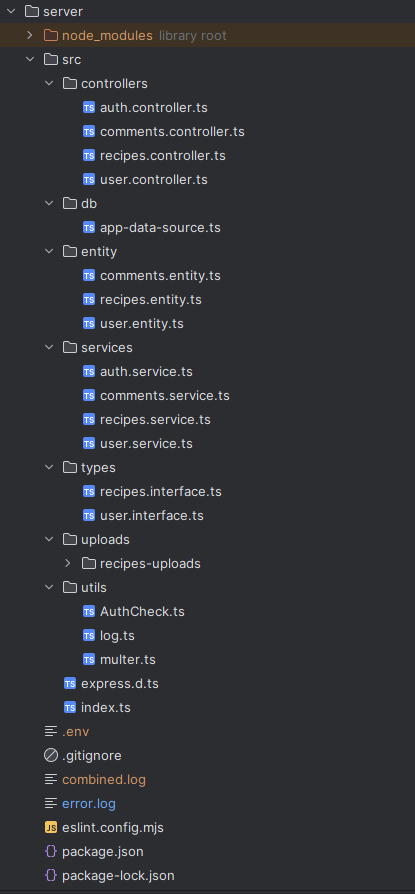


Рисунок 3.1.2

Завдяки такій структурі проєкт є легко масштабованим і підтримуваним. Всі компоненти логічно відокремлені, що дозволяє швидко знаходити потрібний функціонал, полегшує командну роботу та додає гнучкість при внесенні змін у майбутньому.

## 3.2. Структура платформи

Мобільний застосунок створено за допомогою React Native з використанням бібліотеки Expo, що дозволяє забезпечити кросплатформену доступність. Інтерфейс поділяється на кілька ключових елементів:

• Сторінка авторизації (рис. 3.2.1)

Користувач потрапляє на сторінку авторизації при першому відкритті застосунку або після виходу з акаунту. Сторінка містить: поля для введення електронної пошти та пароля, кнопку "Увійти", посилання для переходу на сторінку реєстрації, повідомлення про помилки (наприклад, "невірний логін або пароль").

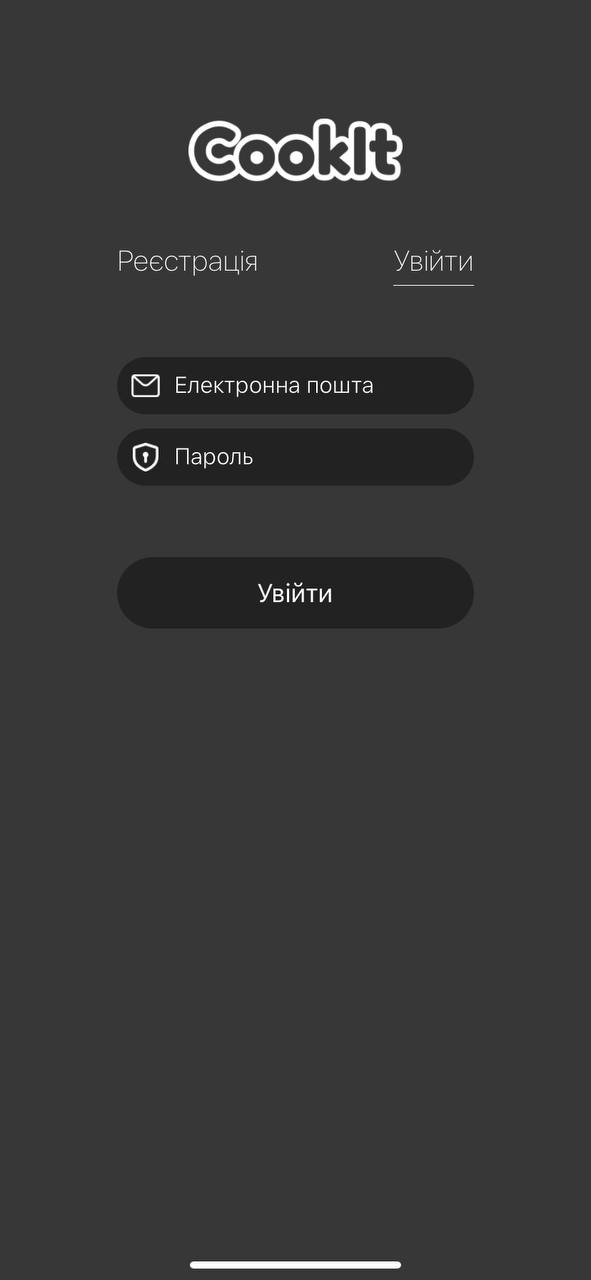


Рисунок 3.2.1

• Сторінка реєстрації (рис. 3.2.2)

Після натискання на посилання "Реєстрація" відкривається сторінка реєстрації, яка включає: поля для введення електронної пошти, пароля та підтвердження пароля, кнопку "Зареєструватися", повідомлення про помилки (наприклад, "паролі не співпадають", "некоректна електронна пошта").

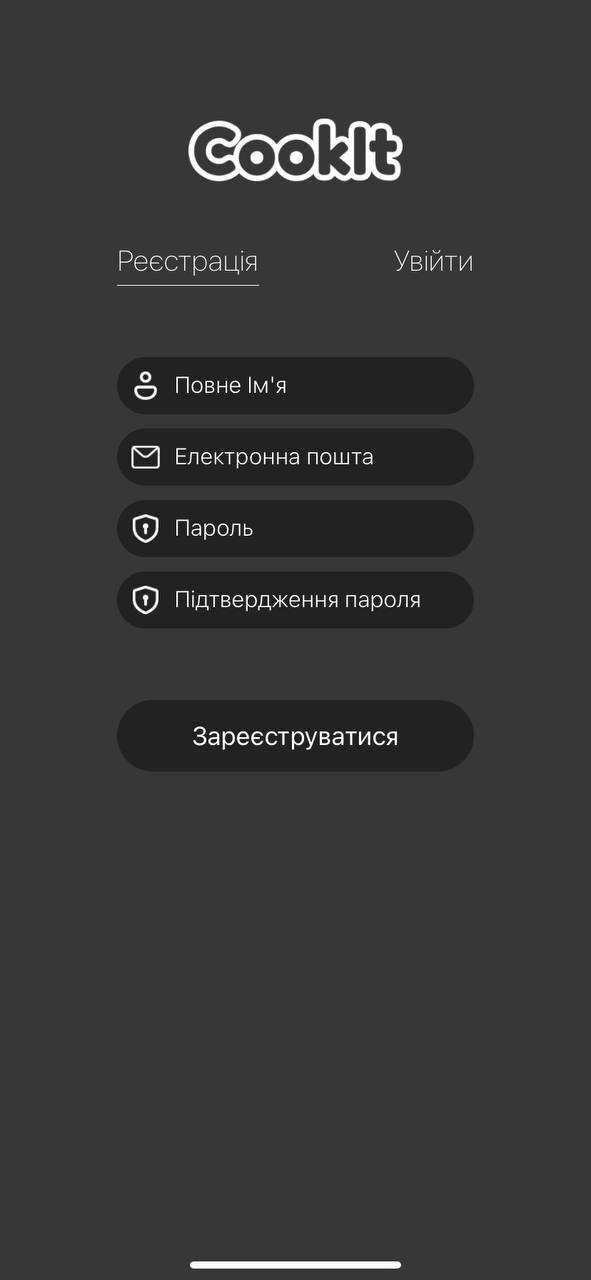


Рисунок 3.2.1

• Хедер з пошуком і створенням посту (рис. 3.2.3)

У верхній частині знаходиться аватар користувача, поле для пошуку рецептів і кнопка створення нового посту.

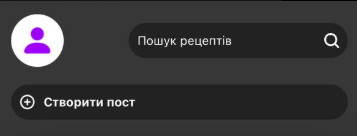


Рисунок 3.2.3

• Головна стрічка постів (рис. 3.2.4)

У стрічці відображаються рецепти інших користувачів, з фотографією, описом, лайками та коментарями. Кожен блок посту включає назву, дату публікації, зображення рецепту та статистику взаємодії.

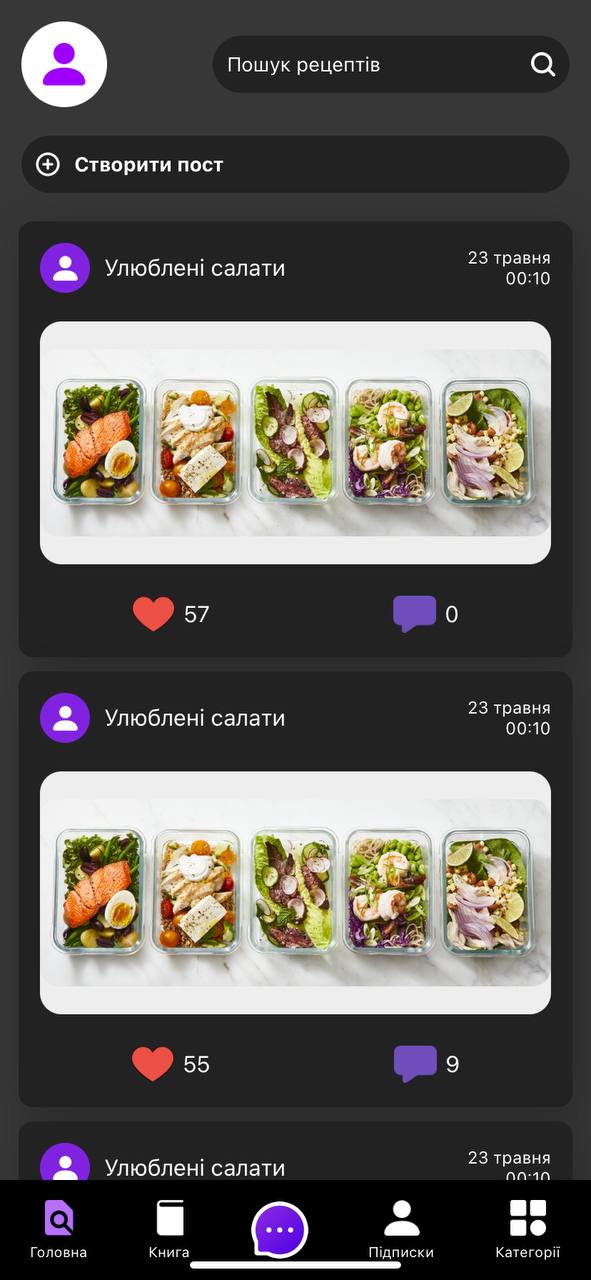


Рисунок 3.2.4

• Навігаційне меню внизу (рис. 3.2.5)

Меню включає вкладки: Головна, Книга рецептів, Налаштування, Підписки та Категорії. Іконки інтуїтивно зрозумілі, активна вкладка виділяється кольором.

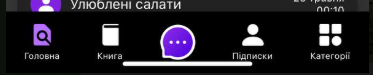


Рисунок 3.2.5

• Список коментарів до рецепту (рис. 3.2.6)

При натисканні на іконку коментарів можна переглянути повний список коментарів до рецепту, з можливістю додавання свого.

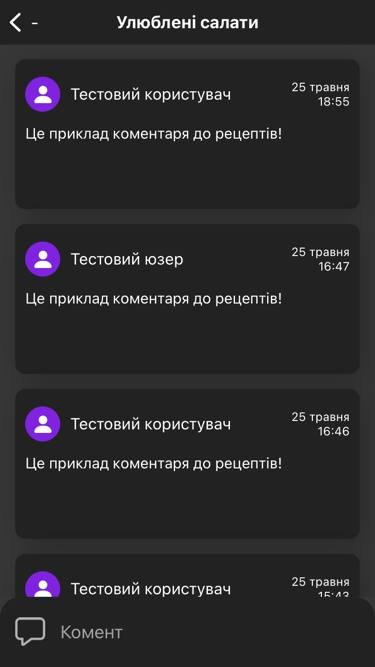


Рисунок 3.2.6

• Сторінка створення рецепту (рис. 3.2.7)

Доступна через хедер, користувач може додати фото, назву, опис рецепту та вибрати категорію.

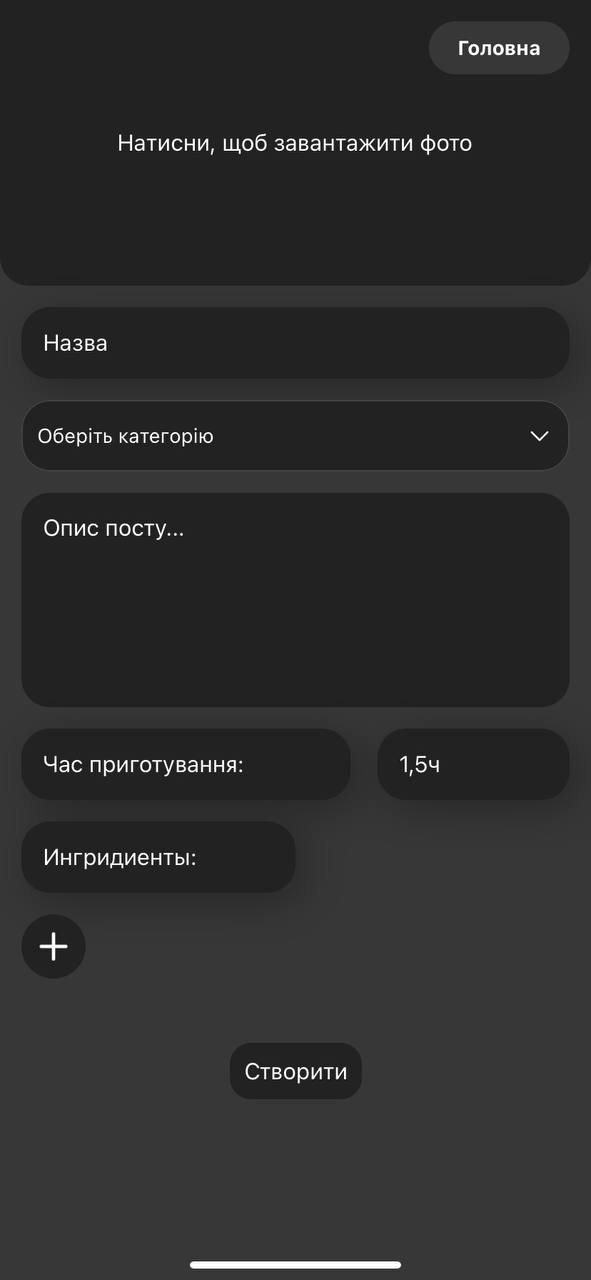


Рисунок 3.2.7

• Сторінка рецептів користувачів на які підписаний користувач (рис. 3.2.8)

Містить пости користувачів з підписок.

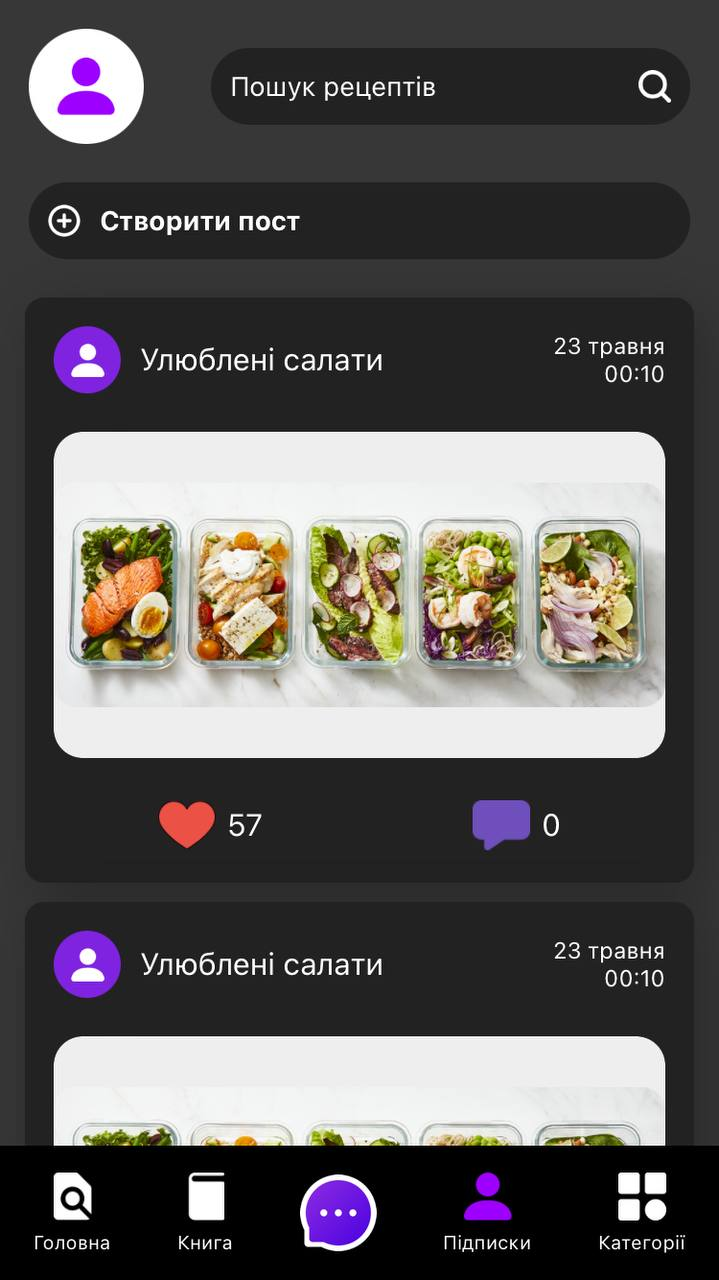


Рисунок 3.2.8

• Сторінка з категоріями рецептів (рис. 3.2.9)

Сторінка доступна через крайню праву вкладку нижнього меню "Категорії". Тут представлені всі наявні категорії рецептів у вигляді плиток або списку. Після натискання на певну категорію відкривається сторінка зі стрічкою постів, що відповідають цій категорії. Кожен пост містить зображення страви, назву.

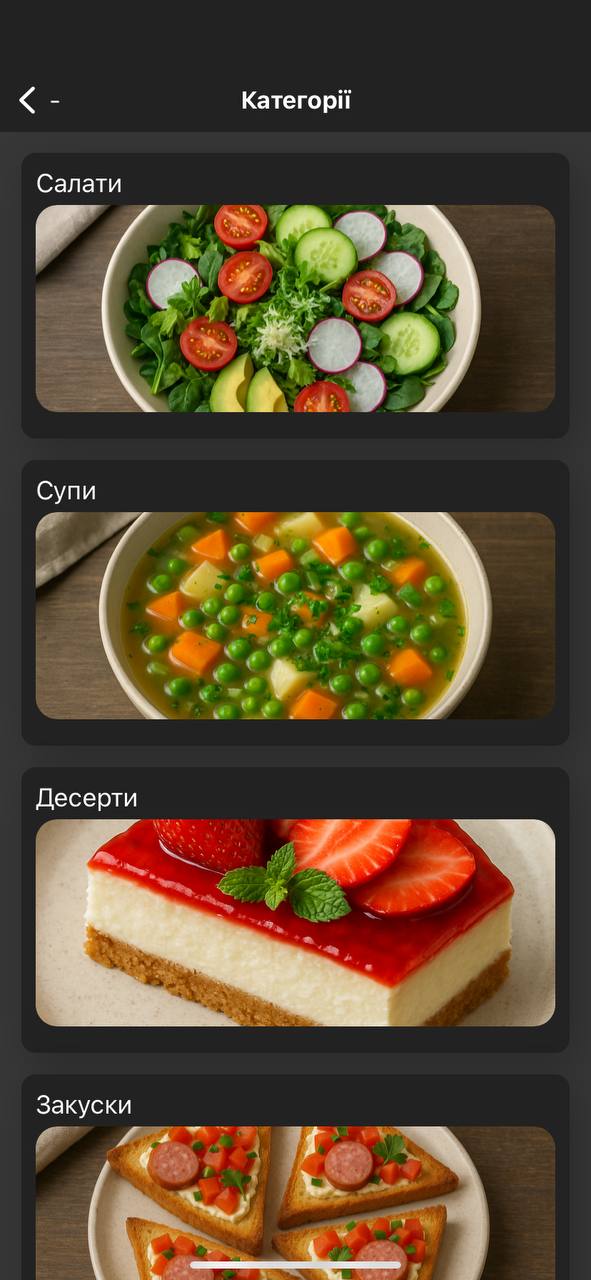


Рисунок 3.2.9

Адаптивність і прокрутка

Інтерфейс підтримує плавну вертикальну прокрутку, а також адаптивний вигляд — застосунок однаково добре виглядає як на Android, так і на iOS пристроях.

## 3.3. Технічна реалізація платформи

Платформа для обміну рецептами реалізована за допомогою сучасного стеку веб- та мобільних технологій, які забезпечують швидку розробку, масштабованість та зручність використання як для користувачів, так і для розробників.

**Серверна частина:**

Для реалізації бекенду обрано стек: Node.js + TypeScript + Express + MongoDB.

Директорія Services:

Сервісний шар у дипломному проєкті відповідає за реалізацію бізнес-логіки додатку. Він взаємодіє з репозиторіями бази даних, обробляє запити від контролерів та повертає відповідні результати. Розглянемо основні сервіси, які реалізовані у системі.

**Сервіс автентифікації**

Сервіс auth.service.ts відповідає за реєстрацію та логін користувачів.

Розбір коду:

1. register(dto: User): Promise<User & { token: string }> — перевіряє, чи існує користувач з такою електронною поштою, хешує пароль за допомогою бібліотеки argon2, створює нового користувача і генерує JWT-токен.

2. login(email: string, password: string): Promise<User & { token: string }> — перевіряє наявність користувача, валідує пароль, і в разі успіху повертає користувача з JWT-токеном.

3. Для створення токена використовується бібліотека jsonwebtoken, а ключ береться з .env.

Лістинг коду (auth.service.ts):

import { hash, verify } from "argon2";

import "dotenv/config";

import jwt from "jsonwebtoken";

import { User } from "../entity/user.entity";

import { createUser, findUserByEmail } from "./user.service";

const JWT\_SECRET = *process*.env.JWT\_SECRET;

export const register = async (

dto: User

): Promise<User & { token: string }> => {

const existsUser = await findUserByEmail(dto.email!);

if (existsUser) {

throw new Error("User with email exists");

}

dto.password = await hash(dto.password!);

const user = await createUser(dto);

const token = jwt.sign(

{

id: user.id,

},

JWT\_SECRET!,

{

expiresIn: "30d",

}

);

return { ...user, token };

};

export const login = async (

email: string,

password: string

): Promise<User & { token: string }> => {

const existsUser = await findUserByEmail(email);

if (!existsUser) {

throw new Error("Wrong data");

}

const validatePassword = await verify(existsUser.password, password);

if (!validatePassword) {

throw new Error("Wrong data");

}

const token = jwt.sign(

{

id: existsUser.id,

},

JWT\_SECRET!,

{

expiresIn: "30d",

}

);

return { ...existsUser, token };

};

**Сервіс рецептів**

Сервіс recipe.service.ts відповідає за створення, отримання та взаємодію з рецептами.

Розбір коду:

1. createRecipe(dto, userId, pictureUrl) — створює новий рецепт, зберігає в базу даних разом з інформацією про автора, тегами, категоріями тощо.

2. getRecipe(postId) — повертає рецепт разом з автором.

3. getUserRecipes(userId) — повертає всі рецепти певного користувача.

4. getFollowRecipes(userId) — повертає рецепти користувачів, на яких підписаний поточний користувач.

5. getRecipes(category) — повертає всі рецепти або рецепти, що належать до певної категорії.

6. likeRecipe(userId, postId) — реалізує функціональність лайків: додає або видаляє лайк від користувача, оновлює лічильники.

Лістинг коду (recipes.service.ts):

import { ObjectId } from "mongodb";

// import { MongoRepository } from "typeorm";

import { *myDataSource* } from "../db/app-data-source";

import { Recipes } from "../entity/recipes.entity";

import { User } from "../entity/user.entity";

import { TRecipeWithAuthor } from "../types/recipes.interface";

export const createRecipe = async (

dto: Recipes,

id: ObjectId | string,

pictureUrl: string

): Promise<Recipes> => {

return await *myDataSource*.getRepository(Recipes).save({

title: dto.title,

description: dto.description,

userId: id,

picture: pictureUrl,

tags: dto.tags,

categories: dto.categories,

cookingTime: dto.cookingTime,

createdAt: new *Date*(),

countLikes: 0,

countComments: 0,

countShares: 0,

});

};

export const getRecipe = async (

postId: string | ObjectId

): Promise<TRecipeWithAuthor | null> => {

const objectRecipeId = new ObjectId(postId);

const recipeRepo = *myDataSource*.getMongoRepository(Recipes);

const userRepo = *myDataSource*.getMongoRepository(User);

const recipe = await recipeRepo.findOne({ where: { \_id: objectRecipeId } });

const objectUserId = new ObjectId(recipe?.userId);

const user = await userRepo.findOne({ where: { \_id: objectUserId } });

return {

...(recipe as Recipes),

...(user as User),

};

};

export const getUserRecipes = async (

userId: ObjectId

): Promise<Recipes[] | null> => {

const objectId = new ObjectId(userId);

return await *myDataSource*

.getMongoRepository(Recipes)

.find({ where: { userId: objectId } });

};

export const getFollowRecipes = async (

userId: ObjectId

): Promise<Recipes[] | null> => {

const objectId = new ObjectId(userId);

const userRepo = *myDataSource*.getMongoRepository(User);

const recipeRepo = *myDataSource*.getMongoRepository(Recipes);

const user = await userRepo.findOne({ where: { \_id: objectId } });

if (!user || !user.follows?.length) {

return [];

}

const followsObjectIds = user.follows.map(id => new ObjectId(id));

const recipes = await recipeRepo.find({

where: {

userId: { $in: followsObjectIds },

},

order: {

createdAt: "DESC",

},

});

return recipes;

};

export const getRecipes = async (

category?: string

): Promise<Recipes[] | null> => {

if (category) {

return await *myDataSource*.getMongoRepository(Recipes).find({

where: {

categories: category,

},

order: {

createdAt: "DESC",

},

});

} else {

return await *myDataSource*.getMongoRepository(Recipes).find({

order: {

createdAt: "DESC",

},

});

}

};

export const likeRecipe = async (

userId: ObjectId,

postId: ObjectId | string

): Promise<void> => {

const objectUserId = new ObjectId(userId);

const objectRecipeId = new ObjectId(postId);

const recipeRepo = *myDataSource*.getMongoRepository(Recipes);

const userRepo = *myDataSource*.getMongoRepository(User);

const recipe = await recipeRepo.findOne({ where: { \_id: objectRecipeId } });

const user = await userRepo.findOne({ where: { \_id: objectUserId } });

if (!recipe || !user) throw new Error("Recipe or user not found");

const recipeIdStr = recipe.id.toString();

const hasLiked = user.likedPosts.includes(recipeIdStr);

if (hasLiked) {

recipe.countLikes = *Math*.max(0, recipe.countLikes - 1);

user.likedPosts = user.likedPosts.filter(id => id !== recipeIdStr);

} else {

recipe.countLikes += 1;

user.likedPosts.push(recipeIdStr);

}

await recipeRepo.save(recipe);

await userRepo.save(user);

};

**Cервіс коментарів**

Сервіс comment.service.ts реалізує функціональність коментування.

Розбір коду:

1. createComment(postId, userId, text) — додає коментар до рецепта та збільшує лічильник коментарів у рецепті.

2. getCommentsByPostId(postId) — повертає список коментарів до рецепта разом із користувачами, що їх залишили. Коментарі сортуються за датою створення у зворотному порядку.

Лістинг коду (coments.service.ts):

import { ObjectId } from "mongodb";

import { *myDataSource* } from "../db/app-data-source";

import { Comments } from "../entity/comments.entity";

import { Recipes } from "../entity/recipes.entity";

import { User } from "../entity/user.entity";  
export const getCommentsByPostId = async (

postId: string

): Promise<

{  
 comment: Comments;

user: User;

}[]  
> => {

const objectId = new ObjectId(postId);

const commentsRepo = *myDataSource*.getMongoRepository(Comments);

const userRepo = *myDataSource*.getMongoRepository(User);  
 const comments = await commentsRepo.find({

where: { recipesId: objectId },

});  
 const sortedComments = comments.sort(

(a, b) =>

new *Date*(b.createdAt).getTime() - new *Date*(a.createdAt).getTime()

);

const commentsWithUsers = await *Promise*.all(

sortedComments.map(async comment => {

const user = await userRepo.findOneBy({

\_id: new ObjectId(comment.userId),

});

if (!user) {

throw new Error(`User not found for comment ${comment.id}`);

}

return { comment, user };

})

);

return commentsWithUsers;

};

export const createComment = async (

postId: string,

userId: string | ObjectId,

text: string

) => {

const objectUserId = new ObjectId(userId);

const objectPostId = new ObjectId(postId);

const recipesRepo = *myDataSource*.getRepository(Recipes);

const commentsRepo = *myDataSource*.getRepository(Comments);

const recipe = await recipesRepo.findOne({  
 where: {  
 // @ts-ignore  
 \_id: objectPostId,  
 },  
 });  
 if (!recipe) {  
 throw new Error("Post not found");  
 }  
 recipe.countComments += 1;  
 await recipesRepo.save(recipe);  
 return await commentsRepo.save({  
 text,  
 recipesId: objectPostId,  
 userId: objectUserId,  
 createdAt: new *Date*(),  
 });  
};

**Сервіс користувачів**

Сервіс user.service.ts відповідає за основні CRUD-операції з користувачами, а також за підписки

Розбір коду:

1. getUsers() — повертає список усіх користувачів.

2. findUserById(id) — повертає користувача за його ідентифікатором.

3. findUserByEmail(email) — повертає користувача за email.

4. createUser(dto) — створює нового користувача.

5. findUserByPostId(postId) — повертає автора рецепта.

6. toggleSubscribe(targetUserId, userId) — реалізує функціональність підписок і відписок на користувачів (додає/видаляє з масивів follows та followers).

Лістинг коду (user.service.ts):

import { ObjectId } from "mongodb";

import { *myDataSource* } from "../db/app-data-source";

import { Recipes } from "../entity/recipes.entity";

import { User } from "../entity/user.entity";

export const getUsers = async (): Promise<User[]> => {

return await *myDataSource*.getRepository(User).find();

};

export const findUserById = async (id: ObjectId): Promise<User | null> => {

const objectId = new ObjectId(id);

return await *myDataSource*

.getMongoRepository(User)

.findOne({ where: { \_id: objectId } });

};

export const findUserByEmail = async (email: string): Promise<User | null> => {

return await *myDataSource*.getRepository(User).findOne({ where: { email } });

};

export const createUser = async (dto: User): Promise<User> => {

const userToSave: User = {

...dto,

likedPosts: dto.likedPosts ?? [],

followers: dto.followers ?? [],

follows: dto.follows ?? [],

};

return await *myDataSource*.getMongoRepository(User).save(userToSave);

};

export const findUserByPostId = async (

postId: string

): Promise<User | null> => {

const objectId = new ObjectId(postId);

const post = await *myDataSource*

.getMongoRepository(Recipes)

.findOne({ where: { \_id: objectId } });

if (!post) return null;

const user = await *myDataSource*.getMongoRepository(User).findOne({

where: { \_id: new ObjectId(post.userId) },

});

return user;

};

export const toggleSubscribe = async (

targetUserId: string,

userId: string | ObjectId

) => {

const targetUserIdObject = new ObjectId(targetUserId);

const userIdObject = new ObjectId(userId);

const repo = *myDataSource*.getMongoRepository(User);

const userTarget = await repo.findOne({

where: { \_id: targetUserIdObject },

});

const user = await repo.findOne({ where: { \_id: userIdObject } });

if (!user || !userTarget) {

throw new Error("User or target user not found");

}

const userIdStr = userIdObject.toHexString();

const targetIdStr = targetUserIdObject.toHexString();

const isFollowing = user.follows.includes(targetIdStr);

if (isFollowing) {

user.follows = user.follows.filter(id => id !== targetIdStr);

userTarget.followers = userTarget.followers.filter(

id => id !== userIdStr

);

} else {

user.follows.push(targetIdStr);

userTarget.followers.push(userIdStr);

}

await repo.save(user);

await repo.save(userTarget);

};

Сервісний шар системи чітко розмежовує відповідальність між контролерами та репозиторіями, інкапсулюючи логіку бізнес-процесів. Це спрощує тестування, підтримку та розвиток платформи в майбутньому.

Директорія controllers:

Директорія controllers у проєкті реалізує маршрутизацію та обробку HTTP-запитів на стороні сервера. Контролери є посередниками між вхідними запитами клієнта і відповідними сервісами, які містять бізнес-логіку.

Усього реалізовано чотири контролери, кожен з яких відповідає за окрему частину функціоналу.

**Контролер auth:**

Відповідає за реєстрацію та вхід користувачів. Використовує register() і login() із auth.service.ts.

У випадку помилки логування відбувається через утиліту logger.error(...).

Лістинг коду (auth.controller.ts):

import { Request, Response, Router } from "express";

import { login, register } from "../services/auth.service";

import { *logger* } from "../utils/log";

const router = Router();

router.post("/register", async (req: Request, res: Response) => {

try {

const user = await register(req.body);

res.status(200).json(user);

} catch (error) {

*logger*.error("Error register user: ", error);

res.status(500).json({ message: "Error register user", error: error });

}

});

router.post("/login", async (req: Request, res: Response) => {

try {

const { email, password } = req.body;

const user = await login(email, password);

res.status(200).json(user);

} catch (error) {

*logger*.error("Error login user: ", error);

res.status(500).json({ message: "Error login user", error: error });

}

});

export const *AuthRouter* = router;

**Контролер recipes:**

Найбільший контролер, що реалізує:

* створення рецепту з завантаженням зображення через Multer;
* отримання всіх або фільтрованих рецептів;
* отримання конкретного рецепту за ID;
* лайк рецепта;
* рецепти від підписок.

Лістинг коду (recipes.controller.ts):

import { Request, Response, Router } from "express";

import {

createRecipe,

getFollowRecipes,

getRecipe,

getRecipes,

getUserRecipes,

likeRecipe,

} from "../services/recipes.service";

import AuthCheck from "../utils/AuthCheck";

import { *logger* } from "../utils/log";

import { *uploadPictures* } from "../utils/multer";

const router = Router();

router.post(

"/create",

AuthCheck,

*uploadPictures*.single("picture"),

async (req: Request, res: Response) => {

try {

if (!req.file) {

res.status(400).json({ message: "No file uploaded" });

return;

}

const userId = req.userId;

const pictureUrl = `../uploads/recipes-uploads/${req.file.filename}`;

const recipe = await createRecipe(req.body, userId!, pictureUrl);

res.status(200).json(recipe);

} catch (error) {

*logger*.error(error);

res.status(500).json({ message: "Error create recipe" });

}

}

);

router.get("/", async (req: Request, res: Response) => {

try {

const rawCategory = req.query.category;

const category =

typeof rawCategory === "string" ? rawCategory.trim() : undefined;

const recipes = await getRecipes(category);

res.status(200).json(recipes);

} catch (error) {

*logger*.error(error);

res.status(500).json({ message: "Error get recipes" });

}

});

router.get(`/:id`, async (req: Request, res: Response) => {

try {

const id = String(req.params.id);

const recipe = await getRecipe(id);

res.status(200).json(recipe);

} catch (error) {

*logger*.error(error);

res.status(500).json({ message: "Error get recipe" });

}

});

router.get("/my/recipes", AuthCheck, async (req: Request, res: Response) => {

try {

const recipes = await getUserRecipes(req.userId!);

res.status(200).json(recipes);

} catch (error) {

res.status(500).json({ message: "Error get recipes", error: error });

}

});

router.post("/like/:id", AuthCheck, async (req: Request, res: Response) => {

try {

const userId = req.userId;

const postId = String(req.params.id);

await likeRecipe(userId!, postId);

res.status(200).json({ success: true });

} catch (error) {

*logger*.error(error);

res.status(500).json({ message: "Error like recipe" });

}

});

router.get(

"/follow/recipes",

AuthCheck,

async (req: Request, res: Response) => {

try {

const recipes = await getFollowRecipes(req.userId!);

*logger*.info(recipes);

res.status(200).json(recipes);

} catch (error) {

res.status(500).json({

message: "Error get recipes",

error: error,

});

}

}

);

export const *RecipesRouter* = router;

**Контролер comments:**

Дозволяє отримати коментарі до конкретного рецепта та додати новий коментар. Додавання можливе тільки для авторизованих користувачів.

Лістинг коду (comments.controller.ts):

import { Request, Response, Router } from "express";

import {

createComment,

getCommentsByPostId,

} from "../services/comments.service";

import AuthCheck from "../utils/AuthCheck";

import { *logger* } from "../utils/log";

const router = Router();

router.get("/:postId", async (req: Request, res: Response) => {

try {

const postId = req.params.postId;

const comments = await getCommentsByPostId(postId);

res.status(200).json(comments);

} catch (error) {

*logger*.error("Error getting comments:", error);

res.status(500).json({

message: "Error getting comments",

error: error instanceof Error ? error.message : String(error),

});

}

});

router.post("/:postId", AuthCheck, async (req: Request, res: Response) => {

try {

const postId = req.params.postId;

const userId = req.userId;

const { text } = req.body;

await createComment(postId, userId!, text);

res.status(200).json({ success: true });

} catch (error) {

*logger*.error(error);

res.status(500).json({ message: "Error create comment" });

}

});

export const *CommentsRouter* = router;

**Контролер user:**

Контролер, який реалізує функції: отримання інформації про користувача, пошук автора рецепта за ID посту, підписка/відписка на користувача.

Використовується сервіс toggleSubscribe() для реалізації логіки підписок**:**

Лістинг коду (user.controller.ts):

import { Request, Response, Router } from "express";

import {

findUserById,

findUserByPostId,

getUsers,

toggleSubscribe,

} from "../services/user.service";

import AuthCheck from "../utils/AuthCheck";

import { *logger* } from "../utils/log";

const router = Router();

router.get("/users", async (req: Request, res: Response) => {

try {

const users = await getUsers();

res.status(200).json(users);

} catch (error) {

*logger*.error(error);

res.status(500).json({ message: "Error get users" });

}

});

router.get("/", AuthCheck, async (req: Request, res: Response) => {

try {

const userId = req.userId;

const user = await findUserById(userId!);

res.status(200).json(user);

} catch (error) {

*logger*.error(error);

res.status(500).json({ message: "Error get user" });

}

});

router.get("/profile/:postId", async (req: Request, res: Response) => {

try {

const postId = req.params.postId;

const user = await findUserByPostId(postId);

res.status(200).json(user);

} catch (error) {

*logger*.error(error);

res.status(500).json({ message: "Error get user by post id" });

}

});

router.post(

"/subscribe/:userId",

AuthCheck,

async (req: Request, res: Response) => {

try {

const targetUserId = req.params.userId;

const userId = req.userId;

await toggleSubscribe(targetUserId, userId!);

res.status(200).json({ success: true });

} catch (error) {

*logger*.error(error);

res.status(500).json({ message: "Error subscribe" });

}

}

);

export const *UserRouter* = router;

База даних (рис. 3.3.1):

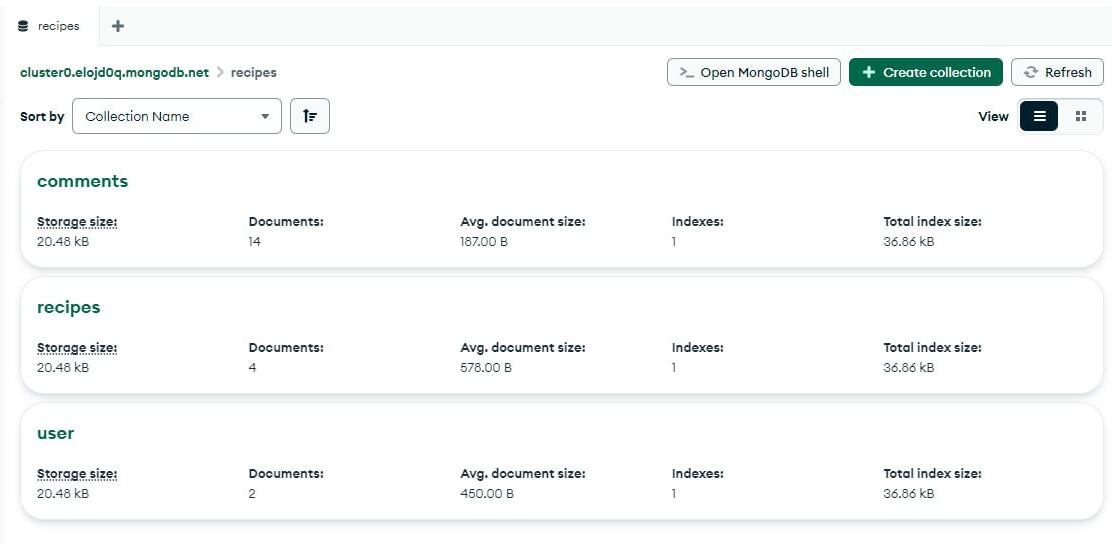


Рисунок 3.3.1

База даних підключається за допомогою файлу app-data-source

Лістинг коду (файлу app-data-source.ts):

import "dotenv/config";

import { DataSource } from "typeorm";

import { Comments } from "../entity/comments.entity";

import { Recipes } from "../entity/recipes.entity";

import { User } from "../entity/user.entity";

export const *myDataSource* = new DataSource({

type: "mongodb",

url: *process*.env.DATABASE\_CONNECTION,

entities: [User, Recipes, Comments],

logging: true,

synchronize: true,

});

**Клієнтська частина:**

Загальні особливості інтерфейсу

Темна кольорова палітра (#232222, #000, #fff) з акцентами, що створює сучасний вигляд.

Компонентний підхід: окремі функціональні частини (категорії, коментарі, пости, хедер, футер) винесені в окремі компоненти.

Реактивність: використовуються хуки (useState, useNavigation, useRoute) для контролю стану та навігації.

Респонсивний інтерфейс: підтримка стилів з elevation/shadow, зручна верстка.

Компонент **Category**

Відповідає за виведення окремої категорії рецептів.

Показує назву категорії (categoryName) та зображення (imgUri).

Оформлений як картка з тінню та скругленими кутами.

Використовується для відображення розділів (напр. «Супи», «Десерти» тощо).

Лістинг коду (Category.jsx):

import { Text } from "react-native";

import *styled* from "styled-components/native";

const Container = *styled*.*View*`

width: 100%;

min-height: 200px;

padding: 10px;

background-color: #232222;

border-radius: 10px;

margin-bottom: 15px;

elevation: 6;

shadow-color: #000;

shadow-offset: 6px 6px;

shadow-opacity: 0.25;

shadow-radius: 15px;

`;

const ContainerImageCategory = *styled*.*View*`

width: 100%;

height: 150px;

padding-top: 5px;

`;

const ImageCategory = *styled*.*Image*.attrs({

resizeMode: "cover",

})`

width: 100%;

height: 100%;

border-radius: 15px;

`;

export const Category = ({ categoryName, imgUri }) => {

return (

<Container>

<Text style={{ color: "#fff", fontSize: 18 }}>{categoryName}</Text>

<ContainerImageCategory>

<ImageCategory source={imgUri} />

</ContainerImageCategory>

</Container>

);

};

Компонент **Comment**

Відповідає за виведення коментарів до рецептів.

Відображає аватар користувача, ім’я (fullName), дату (createdAt) та сам текст коментаря (text).

Дата форматована локально (uk-UA) з виведенням дня, місяця та часу.

Має чітку структуру: заголовок з аватаром і іменем + основний текст.

Лістинг коду (Comment.jsx):

import { View } from "react-native";

import *styled* from "styled-components/native";

const Container = *styled*.*View*`

width: 100%;

min-height: 150px;

padding: 10px;

background-color: #232222;

border-radius: 10px;

margin-bottom: 15px;

elevation: 6;

shadow-color: #000;

shadow-offset: 6px 6px;

shadow-opacity: 0.25;

shadow-radius: 15px;

`;

const TopContent = *styled*.*View*`

width: 100%;

height: 50px;

flex-direction: row;

align-items: center;

margin-bottom: 5px;

`;

const CommentUserAvatar = *styled*.*View*`

width: 35px;

height: 35px;

background-color: #7f23e1;

border-radius: 100px;

display: flex;

align-items: center;

justify-content: center;

margin-right: 10px;

`;

const CommentAvatarImage = *styled*.*Image*.attrs({

resizeMode: "contain",

})`

width: 50%;

height: 50%;

position: relative;

`;

const FullNameText = *styled*.*Text*`

font-size: 16px;

color: #fff;

`;

const MainText = *styled*.*Text*`

font-size: 15px;

color: #fff;

`;

const CommentDateHeader = *styled*.*Text*`

color: #fff;

font-size: 12px;

text-align: right;

`;

export const Comment = ({ text, fullName, createdAt }) => {

const formatDate = dateString => {

const date = new *Date*(dateString);

const day = date.toLocaleDateString("uk-UA", {

day: "numeric",

month: "long",

});

const time = date.toLocaleTimeString("uk-UA", {

hour: "2-digit",

minute: "2-digit",

});

return `${day}\n${time}`;

};

return (

<Container>

<TopContent>

<CommentUserAvatar>

<CommentAvatarImage

source={*require*("../assets/avatar.png")}

/>

</CommentUserAvatar>

<FullNameText>{fullName}</FullNameText>

<View style={{ flex: 1 }} />

<CommentDateHeader>{formatDate(createdAt)}</CommentDateHeader>

</TopContent>

<MainText>{text}</MainText>

</Container>

);

};

Компонент **Footer**

Відповідає за нижню панель навігації застосунку:

Кнопки: «Головна», «Книга», «Підписки», «Категорії».

Центральна кнопка відкриває додаткове меню (FooterMenu).

Активна сторінка визначається через useRoute() — підсвічування відповідної іконки.

Лістинг коду (Footer.jsx):

import { useNavigation, useRoute } from "@react-navigation/native";

import { useState } from "react";

import { *TouchableOpacity* } from "react-native";

import *styled* from "styled-components/native";

import { FooterMenu } from "./UI/FooterMenu";

const FooterContainer = *styled*.*View*`

position: absolute;

bottom: 0;

left: 0;

right: 0;

height: 70px;

background-color: #000;

flex-direction: row;

align-items: center;

justify-content: space-around;

z-index: 10;

`;

const FooterBtnWrapper = *styled*.*View*`

flex-direction: column;

align-items: center;

justify-content: center;

`;

const FooterBtns = *styled*.*Image*.attrs({

resizeMode: "contain",

})`

width: 25px;

height: 25px;

`;

const FooterBtnLarge = *styled*.*Image*.attrs({

resizeMode: "contain",

})`

width: 40px;

height: 40px;

`;

const FooterLabel = *styled*.*Text*`

color: #fff;

font-size: 10px;

margin-top: 5px;

`;

export const Footer = () => {

const navigation = useNavigation();

const [isOpenMenu, setIsOpenMenu] = useState(false);

const route = useRoute();

const currentRoute = route.*name*;

return (

<>

<FooterContainer>

{isOpenMenu && <FooterMenu />}

<TouchableOpacity onPress={() => navigation.navigate("Home")}>

<FooterBtnWrapper>

<FooterBtns

source={

currentRoute === "Home"

? *require*("../assets/homeActiveIcon.png")

: *require*("../assets/home.png")

}

/>

<FooterLabel>Головна</FooterLabel>

</FooterBtnWrapper>

</TouchableOpacity>

<TouchableOpacity

onPress={() => navigation.navigate("MyRecipes")}

>

<FooterBtnWrapper>

<FooterBtns

source={

currentRoute === "MyRecipes"

? *require*("../assets/bookActiveIcon.png")

: *require*("../assets/book.png")

}

/>

<FooterLabel>Книга</FooterLabel>

</FooterBtnWrapper>

</TouchableOpacity>

<TouchableOpacity onPress={() => setIsOpenMenu(prev => !prev)}>

<FooterBtnWrapper>

<FooterBtnLarge

source={*require*("../assets/mainBtn.png")}

/>

</FooterBtnWrapper>

</TouchableOpacity>

<TouchableOpacity

onPress={() => navigation.navigate("FollowsPosts")}

>

<FooterBtnWrapper>

<FooterBtns

source={

currentRoute === "FollowsPosts"

? *require*("../assets/secondAvatar.png")

: *require*("../assets/avatar.png")

}

/>

<FooterLabel>Підписки</FooterLabel>

</FooterBtnWrapper>

</TouchableOpacity>

<TouchableOpacity

onPress={() => navigation.navigate("Categories")}

>

<FooterBtnWrapper>

<FooterBtns

source={

currentRoute === "Categories"

? *require*("../assets/categoryActiveIcon.png")

: *require*("../assets/category.png")

}

/>

<FooterLabel>Категорії</FooterLabel>

</FooterBtnWrapper>

</TouchableOpacity>

</FooterContainer>

</>

);

};

Компонент **Header**

Відповідає за верхню панель застосунку:

Включає: аватар користувача (відкриває профільне меню), пошуковий рядок (search) і кнопку «Створити пост».

Здійснює навігацію до екрану створення посту (CreatePost), передаючи токен авторизації.

Пошук працює через TextInput, зберігаючи значення у стані.

Лістинг коду (Header.jsx):

import { useNavigation } from "@react-navigation/native";

import *styled* from "styled-components/native";

const HeaderView = *styled*.*View*`

width: 100%;

height: 150px;

padding: 15px;

`;

const HeaderTop = *styled*.*View*`

width: 100%;

height: 60px;

flex-direction: row;

display: flex;

align-items: center;

justify-content: space-between;

`;

const HeaderAvatar = *styled*.*TouchableOpacity*`

width: 60px;

height: 60px;

background-color: #fff;

border-radius: 100px;

display: flex;

align-items: center;

justify-content: center;

`;

const HeaderAvatarImage = *styled*.*Image*.attrs({

resizeMode: "contain",

})`

width: 50%;

height: 50%;

`;

const SearchContainer = *styled*.*View*`

flex-direction: row;

align-items: center;

background-color: #232222;

border-radius: 25px;

padding: 0 10px;

width: 250px;

height: 40px;

`;

const HeaderSearch = *styled*.*TextInput*.attrs({

placeholderTextColor: "#fff",

})`

flex: 1;

color: #fff;

`;

const HeaderSearchImg = *styled*.*Image*.attrs({

resizeMode: "contain",

})`

width: 17px;

height: 17px;

margin-left: 5px;

tint-color: #fff;

`;

const HeaderBottom = *styled*.*View*`

height: 10px;

`;

const HeaderCreatePost = *styled*.*TouchableOpacity*`

width: 100%;

height: 40px;

background-color: #232222;

flex-direction: row;

align-items: center;

border-radius: 25px;

margin: 20px 0;

`;

const HeaderCreatePostText = *styled*.*Text*`

color: white;

font-weight: bold;

font-size: 14px;

`;

const HeaderCreatePostImage = *styled*.*Image*.attrs({

resizeMode: "contain",

})`

width: 17px;

height: 17px;

tint-color: #fff;

margin: 0 10px;

`;

export const Header = ({ setIsOpenProfileMenu, token, search, setSearch }) => {

const navigation = useNavigation();

return (

<HeaderView>

<HeaderTop>

<HeaderAvatar

onPress={() => setIsOpenProfileMenu(prev => !prev)}

>

<HeaderAvatarImage

source={*require*("../assets/secondAvatar.png")}

/>

</HeaderAvatar>

<SearchContainer>

<HeaderSearch

placeholder='Пошук рецептів'

value={search}

onChangeText={text => setSearch(text)}

/>

<HeaderSearchImg source={*require*("../assets/search.png")} />

</SearchContainer>

</HeaderTop>

<HeaderBottom>

<HeaderCreatePost

onPress={() => navigation.navigate("CreatePost", { token })}

>

<HeaderCreatePostImage

source={*require*("../assets/create.png")}

/>

<HeaderCreatePostText>Створити пост</HeaderCreatePostText>

</HeaderCreatePost>

</HeaderBottom>

</HeaderView>

);

};

Компонент **Post**

Відповідає за відображення посту з рецептом:

Містить: заголовок з ім’ям користувача та датою створення, зображення рецепту, панель взаємодії (лайки, коментарі, тощо).

Дані отримуються за допомогою react-query (приклад використання useMutation, useQuery).

Аватар, іконки, зображення — з assets.

Лістинг коду (Post.jsx):

import { useNavigation } from "@react-navigation/native";

import { useMutation, useQuery, useQueryClient } from "@tanstack/react-query";

import *axios* from "axios";

import *Constants* from "expo-constants";

import { useState } from "react";

import { *TouchableOpacity* } from "react-native";

import *styled* from "styled-components/native";

import { ProfileMenu } from "./UI/ProfileMenu";

import { Spinner } from "./UI/Spinner";

const PostView = *styled*.*View*`

padding: 15px;

margin: 5px 13px 5px 13px;

border-radius: 10px;

background-color: #232222;

elevation: 6;

shadow-color: #000;

shadow-offset: 6px 6px;

shadow-opacity: 0.25;

shadow-radius: 15px;

`;

const PostHeader = *styled*.*View*`

width: 100%;

margin-bottom: 20px;

flex-direction: row;

align-items: center;

justify-content: space-between;

`;

const UserInfo = *styled*.*View*`

flex-direction: row;

align-items: center;

flex: 1;

`;

const PostTitle = *styled*.*Text*`

color: #fff;

margin-left: 10px;

font-size: 16px;

`;

const PostDateHeader = *styled*.*Text*`

color: #fff;

font-size: 12px;

text-align: right;

`;

const PostImage = *styled*.*Image*.attrs({

resizeMode: "contain",

})`

width: 100%;

height: 170px;

background-color: #eee;

border-radius: 15px;

`;

const PostUserAvatar = *styled*.*View*`

width: 35px;

height: 35px;

background-color: #7f23e1;

border-radius: 100px;

display: flex;

align-items: center;

justify-content: center;

`;

const PostUserAvatarImage = *styled*.*Image*.attrs({

resizeMode: "contain",

})`

width: 50%;

height: 50%;

`;

const PostFooter = *styled*.*View*`

width: 100%;

margin-top: 20px;

flex-direction: row;

align-items: center;

justify-content: space-around;

`;

const FooterItem = *styled*.*View*`

flex-direction: row;

align-items: center;

`;

const FooterText = *styled*.*Text*`

color: #fff;

margin-left: 6px;

font-size: 16px;

`;

const PostFooterImages = *styled*.*Image*.attrs({

resizeMode: "contain",

})`

width: 30px;

height: 30px;

`;

export const Post = ({

id,

title,

imageUrl,

countLikes,

countComments,

createdAt,

isLike,

token,

follows,

profileId,

}) => {

const apiUrl = *Constants*.expoConfig.extra.apiUrl;

const {

data: authorData,

isLoading: authorLoading,

error,

} = useQuery({

queryKey: ["recipe-user-profile", id],

queryFn: async () => {

const res = await axios.get(`${apiUrl}/users/profile/${id}`);

return res.data;

},

enabled: !!id,

});

const [isLikeRecipe, setIsLikeRecipe] = useState(isLike);

const [isOpenProfileMenu, setIsOpenProfileMenu] = useState(false);

const [isFollow, setIsFollow] = useState(false);

const queryClient = useQueryClient();

const navigation = useNavigation();

const formatDate = dateString => {

const date = new *Date*(dateString);

const day = date.toLocaleDateString("uk-UA", {

day: "numeric",

month: "long",

});

const time = date.toLocaleTimeString("uk-UA", {

hour: "2-digit",

minute: "2-digit",

});

return `${day}\n${time}`;

};

const cleanUrl = url => {

return url ? url.replace(/\/\.\.\//g, "/") : null;

};

const imageUrlClean = cleanUrl(imageUrl);

const likeRecipe = useMutation({

mutationFn: async () => {

await axios.post(

`${apiUrl}/recipes/like/${id}`,

{},

{

headers: {

Authorization: `Bearer ${token}`,

},

}

);

},

onSuccess: () => {

queryClient.invalidateQueries(["recipes"]);

},

});

if (authorLoading || !authorData) {

return <Spinner />;

}

return (

<PostView>

{isOpenProfileMenu && (

<ProfileMenu

token={token}

authorPostId={authorData.id}

name={authorData.fullName}

email={authorData.email}

isVisible={isOpenProfileMenu}

isSubscribes={

follows.includes(authorData.id) ? true : false

}

isMyProfile={profileId === authorData.id ? true : false}

followers={authorData.followers}

follows={authorData.follows}

/>

)}

<PostHeader>

<UserInfo>

<TouchableOpacity

onPress={() => setIsOpenProfileMenu(prev => !prev)}

>

<PostUserAvatar>

<PostUserAvatarImage

source={*require*("../assets/avatar.png")}

/>

</PostUserAvatar>

</TouchableOpacity>

<PostTitle>{title}</PostTitle>

</UserInfo>

<PostDateHeader>{formatDate(createdAt)}</PostDateHeader>

</PostHeader>

<TouchableOpacity

onPress={() => navigation.navigate("FullPost", { id })}

>

{imageUrlClean ? (

<PostImage source={{ uri: imageUrlClean }} />

) : (

<PostImage source={*require*("../assets/avatar.png")} />

)}

</TouchableOpacity>

<PostFooter>

<FooterItem>

{isLike ? (

<TouchableOpacity

onPress={() => {

likeRecipe.mutate();

setIsLikeRecipe(prev => !prev);

}}

>

<PostFooterImages

source={*require*("../assets/activeLike.png")}

/>

</TouchableOpacity>

) : (

<TouchableOpacity

onPress={() => {

likeRecipe.mutate();

setIsLikeRecipe(prev => !prev);

}}

>

<PostFooterImages

source={*require*("../assets/like.png")}

/>

</TouchableOpacity>

)}

<FooterText>{countLikes}</FooterText>

</FooterItem>

<FooterItem>

<TouchableOpacity

onPress={() =>

navigation.navigate("PostComment", {

id,

title,

token,

})

}

>

<PostFooterImages

source={*require*("../assets/comments.png")}

/>

</TouchableOpacity>

<FooterText>{countComments}</FooterText>

</FooterItem>

</PostFooter>

</PostView>

);

};

**Інші особливості.**

Зовнішні бібліотеки:

@react-navigation/native — для маршрутизації.

@tanstack/react-query — для асинхронного завантаження та мутацій.

styled-components/native — зручна стилізація компонентів.

Асинхронна логіка:

Отримання постів з бекенду через axios + react-query.

Завантаження, обробка помилок та лоадери (Spinner).

Клієнтська частина застосунку реалізована на сучасному технологічному стеку з акцентом на зручність користування, розділення відповідальностей, ефективну маршрутизацію та взаємодію з сервером. Компоненти легко підтримувати, масштабувати і стилізувати.

## Висновки до розділу 3

Для надання послуг було розроблено функціональність платформи, яка забезпечує ефективну взаємодію з користувачем. Платформа має можливість створювати, переглядати, редагувати та зберігати рецепти, а також додавати їх у персональні колекції. Користувачі можуть заповнювати форми для додавання рецептів із детальним описом інгредієнтів, покроковими інструкціями та фото, що допомагає іншим учасникам спільноти легко відтворити страву.

Функціональність платформи спрямована на збільшення активності користувачів, які хочуть спробувати нові рецепти, а також на створення спільноти, що обмінюється кулінарними ідеями та досвідом. Розробка бази даних дозволяє зберігати інформацію про рецепти, уподобання користувачів та їхні колекції, що автоматизує процес пошуку та рекомендацій. Це дає змогу ефективно персоналізувати контент і покращити взаємодію користувачів із платформою.

Заповнення форм надає інформацію про рецепт і його автора, що сприяє подальшому розвитку спільноти та обміну корисною інформацією. Використавши всі набуті знання, я створив платформу, яка дозволяє легко ділитися рецептами, знаходити нові ідеї для приготування страв і безперешкодно отримувати потрібну інформацію. Це дає змогу залучати більшу кількість користувачів та підтримувати їх інтерес до кулінарії.

# ВИСНОВКИ

Розробка платформи для обміну рецептами є важливим кроком для розвитку кулінарної спільноти та популяризації домашньої та авторської кухні. Онлайн-платформа дозволяє користувачам ділитися власними рецептами, відкривати для себе нові страви та обмінюватися кулінарними ідеями, що створює новий потік активних користувачів, які прагнуть експериментувати з приготуванням їжі.

Під час роботи над дипломним проєктом було проведено аналіз ринку, потреб користувачів та основних функціональних вимог для створення зручної, інтуїтивної та привабливої платформи. Результатом проектування стала платформа яка відповідає сучасним вимогам користувачів і є легким для розуміння та навігації.

Платформа дозволяє конкурувати з іншими подібними сервісами, залучаючи нових користувачів завдяки простому інтерфейсу та якісному контенту. Ця дипломна робота показує, що правильний аналіз ринку та використання сучасних інструментів дозволяють створити ефективний та естетичний продукт, який сприяє розвитку кулінарної спільноти.

Зручний дизайн платформи допомагає легко орієнтуватися в поданні інформації і навігації, що підвищує залученість користувачів і збільшує час їхнього перебування на платформі, сприяючи активному обміну рецептами і досвідом.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Node.js Documentation — офіційна документація Node.js. URL: https://nodejs.org/en/docs/
2. Express.js Guide — офіційний сайт Express.js. URL: <https://expressjs.com/>
3. TypeScript Handbook — офіційна документація TypeScript. URL: https://www.typescriptlang.org/docs/
4. MongoDB Manual — офіційна документація MongoDB. URL: <https://docs.mongodb.com/>
5. TypeORM Documentation — офіційна документація TypeORM. URL: <https://typeorm.io/>
6. JSON Web Tokens (JWT) — офіційний сайт JWT. URL: <https://jwt.io/>
7. Multer Documentation — офіційна документація для Multer (middleware для завантаження файлів в Node.js). URL: <https://github.com/expressjs/multer>
8. Winston Logger — офіційна документація Winston (логування в Node.js). URL: <https://github.com/winstonjs/winston>
9. React Native Docs — офіційна документація React Native. URL: https://reactnative.dev/docs/getting-started
10. Expo Documentation — офіційна документація Expo. URL: https://docs.expo.dev/