ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «**Веб-додаток з розпізнаванням продуктів та підбором рецептів на основі штучного інтелекту**»

на здобуття освітнього ступеня бакалавра
зі спеціальності 126 Інформаційні системи та технології
(код,найменування спеціальності)
освітньо-професійної програми Інформаційні системи та технології
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Денис УГРІМОВ
(підпис) *(ім'я, ПРІЗВИЩЕ здобувача)*

Виконав: здобувач вищої освіти група ІСД-42
Денис УГРІМОВ
(ім'я, ПРІЗВИЩЕ)

Керівник Оксана ТКАЛЕНКО
к.т.н.,доцент *(ім'я, ПРІЗВИЩЕ)*

Рецензент: _____
к.т.н.,доцент *(ім'я, ПРІЗВИЩЕ)*

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інформаційних систем та технологій
Ступінь вищої освіти Бакалавр
Спеціальність 126 Інформаційні системи та технології
Освітньо-професійна програма «Інформаційні системи та технології»

ЗАТВЕРДЖУЮ
Завідувач кафедрою ІСТ
Каміла СТОРЧАК

“____” _____2026 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Угрімову Денису Володимировичу
(прізвище, ім’я, по батькові здобувача)

1.Тема кваліфікаційної роботи: «Веб-додаток з розпізнаванням продуктів та підбором рецептів на основі штучного інтелекту »
керівник кваліфікаційної роботи: Оксана ТКАЛЕНКО к.т.н., доцент
(ім’я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від “20” лютого 2026 р. № 51

2. Строк подання кваліфікаційної роботи «1» червня 2026 р.

3. Вихідні дані кваліфікаційної роботи:
1. Принципи розробки веб-застосунків на основі клієнт-серверної архітектури
2. Архітектура та можливості мультимодальних великих мовних моделей (LLM).
3. Науково-технічна література з тематики Computer Vision та Prompt Engineering.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):
1. Дослідження теоретичних засад комп'ютерного зору, архітектури Transformer та мультимодальних великих мовних моделей, аналіз існуючих рішень у предметній галузі.

2. Проєктування архітектури веб-додатку, обґрунтування вибору технологічного стеку, розробка схеми бази даних та специфікації REST API.
3. Реалізація серверної та клієнтської частин веб-додатку, забезпечення безпеки системи, розгортання у хмарному середовищі та тестування функціональності

5.Перелік ілюстраційного матеріалу: *презентація*

6. Дата видачі завдання «20» лютого 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Підбір технічної літератури	20.02-05.03.26	Виконано
2.	Дослідження теоретичних засад комп'ютерного зору, LLM та промпт-інжинірингу п	04.03-19.03.25	Виконано
3.	Проєктування архітектури системи, бази даних та REST API	20.03-1.04.25	Виконано
4.	Розробка серверної та клієнтської частин, інтеграція Groq API	01.04-29.04.25	Виконано
5.	Тестування, оцінка результатів та формування висновків	30.03-02.05.25	Виконано
6.	Розробка демонстраційних матеріалів, доповідь	03.05-05.05.25	Виконано
7.	Оформлення кваліфікаційної роботи	05.05-30.05.25	Виконано

Здобувач вищої освіти _____ Денис УГРІМОВ
(підпис) *(ім'я, ПРІЗВИЩЕ)*

Керівник кваліфікаційної роботи _____ Оксана ТКАЛЕНКО
(підпис) *(ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 50 стор., 16 рис., 6 табл., 23 джерела.

Мета роботи - покращення процесу розпізнаванням продуктів та підбором рецептів на основі штучного інтелекту.

Об'єкт дослідження - процес автоматизованого розпізнавання харчових продуктів та генерації персоналізованих рецептів на основі комп'ютерного зору і великих мовних моделей.

Предмет дослідження - методи та засоби веб-додатку з інтеграцією AI-моделей для розпізнавання інгредієнтів на зображеннях та інтелектуального підбору рецептів.

Короткий зміст роботи: досліджено методи комп'ютерного зору та можливості мультимодальних LLM для розпізнавання зображень; проаналізовано існуючі рішення у предметній галузі; спроєктовано архітектуру системи та схему бази даних; розроблено серверну частину на базі FastAPI з інтеграцією Groq API; розроблено клієнтську частину на базі React; систему розгорнуто у хмарному середовищі та проведено тестування.

КЛЮЧОВІ СЛОВА: ВЕБ-ДОДАТОК, ШТУЧНИЙ ІНТЕЛЕКТ, РОЗПІЗНАВАННЯ ЗОБРАЖЕНЬ, ВЕЛИКІ МОВНІ МОДЕЛІ, КОМП'ЮТЕРНИЙ ЗІР, РЕЦЕПТИ, REACT, FASTAPI, GROQ API, SUPABASE.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

AI - Artificial Intelligence (штучний інтелект)

API - Application Programming Interface (інтерфейс програмування застосунків)

CDN - Content Delivery Network (мережа доставки контенту)

CI/CD - Continuous Integration / Continuous Deployment (безперервна інтеграція та розгортання)

CNN - Convolutional Neural Network (згорткова нейронна мережа)

CORS - Cross-Origin Resource Sharing (спільне використання ресурсів між різними джерелами)

CSS - Cascading Style Sheets (каскадні таблиці стилів)

GPU - Graphics Processing Unit (графічний процесор)

HTML - HyperText Markup Language (мова гіпертекстової розмітки)

HTTP - HyperText Transfer Protocol (протокол передачі гіпертексту)

JSON - JavaScript Object Notation (формат обміну даними)

JSX - JavaScript XML (розширення синтаксису JavaScript для React)

JWT - JSON Web Token (токен автентифікації)

LLM - Large Language Model (велика мовна модель)

LPU - Language Processing Unit (процесор обробки мови)

MoE - Mixture of Experts (суміш експертів, архітектура нейронних мереж)

REST - Representational State Transfer (архітектурний стиль веб-сервісів)

RLS - Row Level Security (безпека на рівні рядків)

SPA - Single Page Application (односторінковий застосунок)

SQL - Structured Query Language (мова структурованих запитів)

СУБД - система управління базами даних

URL - Uniform Resource Locator (уніфікований локатор ресурсів)

UUID - Universally Unique Identifier (універсальний унікальний ідентифікатор)

ViT - Vision Transformer (трансформер для обробки зображень)

LSTM - Long Short-Term Memory (довга короткочасна пам'ять)

ВСТУП

Актуальність теми. В умовах стрімкого розвитку штучного інтелекту та комп'ютерного зору все більшої популярності набувають застосунки, що автоматизують повсякденні задачі людини. Одним із актуальних напрямів є інтелектуальна обробка харчових зображень - визначення складу продуктів на фото та генерація рецептів на їх основі. За даними звіту UNEP [22], щорічно у світі утворюється близько 931 мільйона тонн харчових відходів, значна частина яких виникає через невміння раціонально використовувати наявні інгредієнти. Сучасні великі мовні моделі (LLM) з підтримкою мультимодального введення відкривають принципово нові можливості для вирішення цієї проблеми: одна модель здатна одночасно аналізувати зображення та генерувати структурований текстовий результат без необхідності навчання спеціалізованих класифікаторів. Незважаючи на зростаючий інтерес до цієї тематики, більшість існуючих рішень є або вузькоспеціалізованими, або комерційними закритими продуктами без можливості адаптації. Тому розробка доступного веб-додатку, який поєднує розпізнавання продуктів і підбір рецептів на основі відкритих AI-моделей, є актуальною науково-практичною задачею.

Аналіз останніх досліджень і публікацій. Питання автоматичного розпізнавання їжі на зображеннях активно досліджується з середини 2010-х років. Класичні підходи базувались на згорткових нейронних мережах (CNN) - зокрема, моделі ResNet та InceptionV3 застосовувались для класифікації страв у наборах даних Food-101 [4] та UECFOOD256 [21]. Однак такі підходи вимагають великих розмічених датасетів і не здатні розпізнавати довільні сирі інгредієнти в умовах реального побуту. З появою великих мультимодальних мовних моделей - GPT-4V, Google Gemini Vision, Llama 4 Scout - з'явилась можливість вирішувати задачу без спеціалізованого навчання: достатньо правильно сформулювати запит до моделі. Роботи таких дослідників, як Bossard et al. (Food-101) [4], а також розробки Meta AI у напрямку мультимодальних LLM [6] підтверджують ефективність цього підходу. Водночас питання інтеграції таких моделей у повноцінний веб-сервіс з

автентифікацією, збереженням даних та зручним інтерфейсом залишається недостатньо висвітленим у відкритих джерелах.

Мета і завдання дослідження. Метою кваліфікаційної роботи є покращення процесу розпізнаванням продуктів та підбором рецептів на основі штучного інтелекту.

Для досягнення поставленої мети вирішувались такі завдання:

1. Проаналізувати існуючі рішення у сфері розпізнавання харчових продуктів та підбору рецептів.
2. Дослідити можливості мультимодальних великих мовних моделей для аналізу зображень та генерації структурованого тексту.
3. Спроєктувати архітектуру веб-додатку та схему бази даних.
4. Розробити серверну частину на базі FastAPI з інтеграцією Groq API (модель Llama 4 Scout).
5. Розробити клієнтську частину на базі React із підтримкою автентифікації та збереження рецептів.
6. Розгорнути систему у хмарному середовищі та провести тестування функціональності.

Об'єкт дослідження - процес автоматизованого розпізнавання харчових продуктів та генерації персоналізованих рецептів на основі комп'ютерного зору і великих мовних моделей.

Предмет дослідження - методи та засоби розробки веб-додатку з інтеграцією AI-моделей для розпізнавання інгредієнтів на зображеннях та інтелектуального підбору рецептів.

Методи дослідження. У роботі використано такі методи: аналіз і порівняння існуючих рішень - для обґрунтування вибору технологічного стеку; метод прототипування - для поетапної розробки та тестування компонентів системи; методи проєктування REST API - для визначення структури серверної частини; мультимодальний промпт-інжиніринг - для налаштування взаємодії з LLM-моделлю та отримання структурованих JSON-відповідей; методи тестування програмного забезпечення - для перевірки коректності роботи системи.

Наукова новизна та практична значущість отриманих результатів. Наукова новизна роботи полягає в розробці підходу до інтеграції мультимодальної LLM-моделі у веб-додаток, що реалізує двоетапний конвеєр обробки: розпізнавання інгредієнтів з довільного побутового зображення та подальша генерація рецептів з урахуванням базових продуктів кухні. На відміну від існуючих рішень, запропонована система не потребує спеціалізованих датасетів та дозволяє обробляти одночасно кілька зображень. Практична значущість роботи полягає в тому, що розроблений веб-додаток розгорнутий у хмарному середовищі, має реєстрацію та автентифікацію користувачів, можливість збереження рецептів і є готовим до використання.

Структура роботи. Робота складається зі вступу, трьох розділів, висновків, переліку посилань та додатків. У першому розділі розглянуто теоретичні основи комп'ютерного зору, архітектуру Transformer та механізм самоуваги, можливості мультимодальних LLM, методи промпт-інжинірингу та існуючі рішення у предметній галузі. У другому розділі проведено аналіз вимог до системи, спроєктовано архітектуру, обґрунтовано вибір технологічного стеку, розроблено схему бази даних та специфікацію REST API. У третьому розділі описано реалізацію серверної та клієнтської частин, забезпечення безпеки системи, розгортання у хмарному середовищі та результати тестування. Загальний обсяг текстової частини роботи складає 50 сторінок.

ТЕОРЕТИЧНІ ОСНОВИ ТА АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Комп'ютерний зір та розпізнавання харчових продуктів

Комп'ютерний зір (Computer Vision) - це галузь штучного інтелекту, що вивчає методи автоматичного аналізу та інтерпретації візуальної інформації. Основним завданням комп'ютерного зору є отримання змістовного опису зображення або відеопотоку без участі людини. Протягом останніх десятиліть ця галузь зазнала революційних змін завдяки розвитку глибокого навчання (Deep Learning).

До появи нейронних мереж задача класифікації зображень розв'язувалась із застосуванням методів ручного виділення ознак. Алгоритм SIFT (Scale-Invariant Feature Transform) забезпечував виявлення ключових точок зображення інваріантних до масштабу та повороту. Алгоритм HOG (Histogram of Oriented Gradients) описував локальну форму об'єкта через розподіл градієнтів інтенсивності пікселів. Отримані ознаки передавались до класифікатора на основі метода опорних векторів (SVM), який визначав приналежність зображення до певного класу шляхом побудови оптимальної розділяючої гіперплощини у просторі ознак. Такі підходи вимагали значних зусиль від дослідника та мали обмежену точність на складних реальних зображеннях.

Переломним моментом стала поява згорткових нейронних мереж (Convolutional Neural Networks, CNN). Архітектура AlexNet [2], представлена у 2012 році на змаганні ImageNet, продемонструвала якісний стрибок точності класифікації - з ~74% до ~84% accuracy. Подальший розвиток архітектур VGGNet (2014), GoogLeNet/InceptionV1 (2014) та ResNet (2015) [3] закріпив домінування CNN у задачах розпізнавання зображень. Динаміку зростання точності CNN-архітектур на змаганні ImageNet наведено на рис. 1.1.

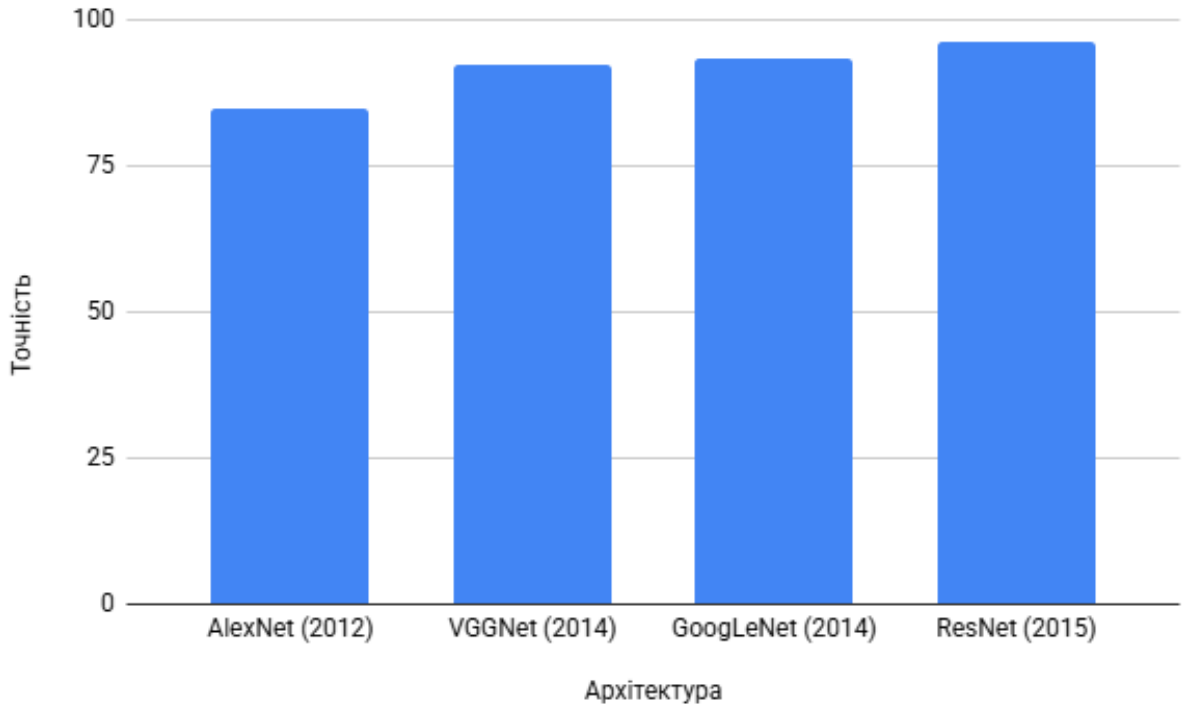


Рис. 1.1 Точність CNN-архітектур на змаганні ImageNet

Розпізнавання харчових продуктів є специфічним підзавданням комп'ютерного зору з низкою особливостей. По-перше, харчові продукти мають значну варіативність зовнішнього вигляду - один і той самий інгредієнт (наприклад, перець) може відрізнятись за кольором, формою та ступенем обробки. По-друге, продукти часто частково перекривають один одного на зображенні. По-третє, умови освітлення, кут зйомки та фон суттєво впливають на результат. Перший значущий публічний датасет для цієї задачі - Food-101 [4] був представлений Bossard et al. у 2014 році і містив 101 000 зображень 101 категорії страв. Пізніше з'явились датасети UECFOOD256 (256 категорій, орієнтований на японську кухню) та VireoFood-172 [23].

Класичні підходи до розпізнавання інгредієнтів на зображеннях вимагали великих розмічених датасетів та тривалого навчання. Крім того, вони були спроможні класифікувати лише ті категорії, що були представлені у навчальній вибірці, що суттєво обмежувало їх практичне застосування у побутових умовах, де множина можливих інгредієнтів є відкритою.

1.2 Архітектура Transformer та механізм самоуваги

Архітектура Transformer [1], запропонована Vaswani et al. у 2017 році, стала революційним проривом у галузі обробки природної мови та заклала фундамент для всіх сучасних великих мовних моделей. На відміну від попередніх архітектур на основі рекурентних нейронних мереж (RNN) та LSTM, Transformer відмовився від послідовної обробки даних на користь механізму уваги, що дозволяє опрацьовувати всі елементи послідовності паралельно.

Ключовим компонентом архітектури є механізм самоуваги. Для кожного елемента вхідної послідовності обчислюються три вектори: запит, ключ та значення. Це дозволяє моделі враховувати залежності між будь-якими елементами послідовності незалежно від відстані між ними - на відміну від RNN, де інформація про далекі елементи поступово "згасала" через кількість кроків обробки.

Повна архітектура Transformer складається з двох основних блоків - енкодера та декодера, кожен з яких містить шари самоуваги та повнозв'язні шари прямого поширення feed-forward. Для збереження інформації про порядок елементів у послідовності використовується позиційне кодування (positional encoding) - спеціальний вектор, що додається до представлення кожного токена і кодує його позицію.

Механізм багатоголової уваги є розширенням базового механізму самоуваги: замість одного обчислення уваги виконується кілька паралельних обчислень з різними навченими проекційними матрицями. Результати кожної "голови" конкатенуються та проєктуються у фінальний простір представлень. Це дозволяє моделі одночасно фокусуватись на різних аспектах залежностей у тексті - семантичних, синтаксичних, позиційних тощо.

Для навчання Transformer використовується техніка масштабованої передобробки на великих корпусах текстів - передтренування з наступним тонким налаштуванням на конкретних задачах. Саме здатність до ефективного масштабування - збільшення кількості параметрів та обсягу навчальних даних -

дозволила Transformer стати основою для GPT, BERT, LLaMA та всіх інших сучасних великих мовних моделей.

Архітектура Mixture of Experts (MoE), використана у Llama 4 Scout, є подальшим розвитком ідеї масштабування [6]. Замість активації всіх параметрів мережі для кожного токена, MoE-архітектура ділить нейронну мережу на кілька незалежних "експертів" - підмереж. Спеціальна маршрутизуюча мережа (router) для кожного токена обирає лише кількох найбільш релевантних експертів (як правило 2 з 16 або більше). Llama 4 Scout має 109 мільярдів загальних параметрів, але активує лише 17 мільярдів на кожен токен [6]. Це забезпечує якість моделі великого розміру при обчислювальній вартості значно меншої моделі.

1.3 Великі мовні моделі та мультимодальний підхід

Великі мовні моделі (LLM) [16] - це нейронні мережі з мільярдами параметрів, навчені на масштабних корпусах текстових даних. В основі сучасних LLM лежить архітектура Transformer, запропонована у статті «Attention Is All You Need» (2017) [1]. Ключовим механізмом є самоувага (self-attention), що дозволяє моделі враховувати залежності між будь-якими елементами послідовності незалежно від відстані між ними.

Мультимодальні LLM розширюють класичну архітектуру можливістю обробки не лише тексту, а й зображень, аудіо та інших типів даних. Типова архітектура мультимодальної моделі [11],[17] включає три компоненти: енкодер зображень (як правило, на основі Vision Transformer - ViT) [5], проекційний шар для узгодження просторів ознак зображення та тексту, і власне мовну модель-декодер.

Серед найбільш відомих мультимодальних моделей [17] слід виділити такі: GPT-4V від OpenAI - перша широкодоступна комерційна мультимодальна модель, що продемонструвала здатність описувати зображення, відповідати на запитання щодо їх вмісту та виконувати складні задачі на перетині зору та мовного розуміння. Google Gemini - сімейство моделей від Google DeepMind, спроєктованих як нативно

мультимодальні з першого дня. Модель Gemini 1.5 Flash вирізняється великим контекстним вікном та ефективністю. Llama 4 Scout від Meta AI - відкрита мультимодальна модель із 17 мільярдами активних параметрів (109B загалом, архітектура Mixture of Experts). Модель підтримує обробку до 48 зображень одночасно та має контекстне вікно у 10 мільйонів токенів [6]. Саме ця модель використовується у розробленому веб-додатку через API-сервіс Groq.

Groq - це компанія, що розробила спеціалізовані процесори LPU (Language Processing Unit) для прискореної інференції LLM. На відміну від GPU, LPU оптимізований для послідовної генерації токенів і забезпечує значно вищу швидкість відповіді. Groq API надає безкоштовний доступ до актуальних відкритих моделей [10], зокрема Llama 4 Scout, із затримкою значно нижчою, ніж у конкурентних хмарних сервісів. Це робить його оптимальним вибором для прототипування та навчальних проєктів.

Ключовою технікою роботи з LLM є промпт-інжиніринг (prompt engineering) - мистецтво формулювання запитів таким чином, щоб отримати від моделі детерміновану, структуровану та точну відповідь. У контексті даної роботи промпт-інжиніринг відіграє критичну роль: модель отримує зображення та текстовий запит, і має повернути відповідь виключно у форматі JSON без зайвого тексту. Для цього у запиті явно вказується очікуваний формат відповіді та забороняється генерація будь-якого вступного тексту.

Двоетапний підхід, реалізований у даній роботі, передбачає: на першому етапі модель аналізує зображення та повертає JSON зі списком виявлених інгредієнтів; на другому - отримує список інгредієнтів у текстовому вигляді та генерує JSON з трьома рецептами. Такий поділ дозволяє послідовно обробляти кілька зображень, об'єднувати результати розпізнавання та передавати єдиний консолідований список до етапу генерації рецептів.

1.4 Промпт-інжиніринг як метод керування LLM

Промпт-інжиніринг (prompt engineering) - це дисципліна, що вивчає методи формулювання вхідних запитів до великих мовних моделей з метою отримання бажаного результату без зміни параметрів самої моделі. На відміну від класичного машинного навчання, де для зміни поведінки моделі необхідне перенавчання, промпт-інжиніринг дозволяє керувати виводом моделі виключно через текстові інструкції.

Виділяють кілька ключових технік промпт-інжинірингу. Техніка Zero-shot prompting передбачає подачу задачі без прикладів її виконання - модель спирається виключно на знання, отримані під час передтренування. Техніка Few-shot prompting включає у запит кілька прикладів бажаного формату вводу та виводу, що суттєво підвищує точність і стабільність результатів. Chain-of-thought prompting спонукає модель розмірковувати покроково перед наданням відповіді, що підвищує якість на складних логічних задачах. Role prompting визначає роль або персонажа для моделі ("Ти є професійним шеф-кухарем"), що задає контекст і стиль відповіді.

У розробленому веб-додатку використовуються декілька з цих технік одночасно. Для ендпоінту розпізнавання інгредієнтів застосовується zero-shot prompting з явним визначенням формату відповіді та прикладом очікуваного JSON. Для генерації рецептів використовується role prompting ("You are a professional chef") у поєднанні з детальним переліком обмежень і правил, що забезпечує структуровану та кулінарно логічну відповідь.

Критично важливим аспектом промпт-інжинірингу для виробничих систем є забезпечення детермінізму відповіді - гарантії, що модель поверне відповідь у строго визначеному форматі. Для цього застосовуються такі прийоми: явна вказівка формату відповіді з прикладом, пряма заборона будь-якого тексту поза очікуваним форматом ("Reply ONLY in JSON"), визначення точної структури JSON-об'єкту та типів значень кожного поля. Додатково на рівні постобробки реалізовано очищення від markdown-обгорток, які модель інколи додає попри заборону - це підвищує стійкість системи до непередбаченої поведінки моделі.

Дослідження у галузі промпт-інжинірингу демонструють, що якість сформульованого запиту може суттєво впливати на точність та корисність відповіді LLM. Sahoo et al. (2024) [7] систематизували основні техніки та виявили що комбінування role prompting з явними обмеженнями формату забезпечує найстабільніші результати для задач структурованої генерації тексту. Agarwal та Choudhary (2024) [8] показали що явне визначення заборон ("Do NOT...") є ефективнішим ніж лише позитивні інструкції для задач з чіткими обмеженнями на вміст відповіді.

1.5 Аналіз існуючих рішень

Перед проєктуванням власної системи було проведено аналіз існуючих рішень у сфері розпізнавання їжі та підбору рецептів. Розглянуто як комерційні продукти, так і відкриті дослідницькі системи. Аналіз проводився за чотирма критеріями: наявність функції розпізнавання інгредієнтів з фотографії, спосіб генерації рецептів, наявність відкритого API та підтримка автентифікації користувачів.

Yummly - мобільний та веб-застосунок для пошуку рецептів з персоналізованими рекомендаціями. Користувач вручну вводить наявні продукти, після чого система підбирає страви з того, що є вдома. Передбачена фільтрація за алергіями, дієтами та часом приготування. Рекомендаційний алгоритм базується на аналізі вподобань користувача та машинному навчанні. Проте автоматичного розпізнавання інгредієнтів з фото не передбачено, а рецепти беруться з фіксованої бази даних, а не генеруються AI. Частковий доступ до API надається лише корпоративним партнерам.

Supercook - спеціалізований пошуковий рушій рецептів, орієнтований виключно на підбір страв з наявних продуктів. Користувач вводить інгредієнти вручну, після чого сервіс агрегує відповідні рецепти з тисяч кулінарних сайтів за допомогою власного індексу. Перевагою системи є широке охоплення рецептурних джерел - понад 1 мільйон рецептів. Відсутня функція аналізу зображень, генерація

нових рецептів не підтримується - система працює лише з наперед проіндексованим контентом. Публічний API відсутній, реєстрація користувачів не передбачена.

Google Lens - інструмент візуального пошуку від Google, що використовує глибоке навчання для розпізнавання об'єктів реального світу через камеру смартфона. Здатний ідентифікувати окремі страви та продукти харчування з високою точністю завдяки навчанню на масштабних датасетах Google. Однак результатом розпізнавання є лише перенаправлення на пошукову видачу Google - цілеспрямованого підбору рецептів на основі розпізнаних інгредієнтів немає. Крім того, відкритого публічного API для інтеграції у сторонні застосунки не надається, що унеможливлює використання технології у власних розробках.

PlantNet та аналоги - мобільні застосунки для розпізнавання рослин та грибів, побудовані на спеціалізованих згорткових нейронних мережах. База даних PlantNet налічує понад 380 000 видів рослин, а розпізнавання відбувається шляхом порівняння ознак завантаженого зображення з еталонними зразками. Демонструють можливість точного розпізнавання природних об'єктів у реальних умовах, однак є вузькоспеціалізованими і не є універсальними рішеннями для харчових продуктів. Архітектурно подібні системи підтверджують доцільність використання CNN для розпізнавання об'єктів з обмеженим набором категорій.

Im2Recipe (MIT) [23] - дослідницька система, навчена на датасеті з 1 мільйона пар «зображення страви - рецепт». Використовує крос-модальне вбудовування для узгодження візуальних ознак зображення з текстовим представленням рецепту. Здатна за фото готової страви передбачити рецепт її приготування включно з переліком інгредієнтів та інструкціями. Незважаючи на академічну цінність підходу, система є прототипом і недоступна як публічний сервіс, а її підхід обмежений розпізнаванням готових страв а не сирих інгредієнтів.

Порівняльний аналіз існуючих рішень зведено у таблиці 1.1.

Таблиця 1.1

Порівняльний аналіз існуючих рішень

Рішення	**Розпізнавання з фото**	**Генерація рецептів**	**Відкритий API**	**Автентифікація**
Yummly	Ні	Ні (база даних)	Частково	Так
Supercook	Ні	Ні (база даних)	Ні	Ні
Google Lens	Так	Ні	Ні	Так
Im2Recipe	Так	Так	Ні	Ні
Click&Cook	**Так**	**Так (AI)**	**Так**	**Так**

З таблиці видно, що жодне з розглянутих рішень не поєднує в собі всі чотири характеристики одночасно. Розроблений веб-додаток заповнює цю нішу, пропонуючи повний цикл від завантаження фотографії до отримання рецептів із можливістю їх збереження в особистому кабінеті.

У першому розділі розглянуто теоретичні основи та предметну галузь дослідження. Проаналізовано еволюцію методів комп'ютерного зору - від класичних алгоритмів ручного виділення ознак (SIFT, HOG) до згорткових нейронних мереж та сучасних мультимодальних трансформерів. Встановлено, що класичні підходи на основі CNN потребують великих розмічених датасетів і обмежені фіксованим набором категорій, що робить їх непридатними для розпізнавання довільних побутових інгредієнтів у відкритому середовищі.

Детально розглянуто архітектуру Transformer - механізм самоуваги, позиційне кодування та багатоголову увагу. Показано, що саме здатність Transformer до ефективного масштабування стала основою для всіх сучасних великих мовних моделей. Окремо розглянуто архітектуру Mixture of Experts,

реалізовану у моделі Llama 4 Scout, яка забезпечує якість моделі з 109 мільярдами параметрів при активації лише 17 мільярдів на кожен токен.

Досліджено можливості мультимодальних LLM для задачі розпізнавання зображень. Встановлено, що такі моделі дозволяють вирішити задачу розпізнавання інгредієнтів без спеціалізованого навчання завдяки здатності до розуміння зображень у поєднанні з текстовим запитом. Розглянуто методи промпт-інжинірингу - zero-shot prompting, role prompting та техніки забезпечення детермінізму відповіді - які є ключовими для отримання структурованих JSON-відповідей від моделі.

Проведений аналіз існуючих рішень показав, що жоден публічний сервіс - Yummly, Supercook, Google Lens, Im2Recipe - не реалізує повний цикл «фото, інгредієнти, рецепти» з можливістю персоналізації та збереження результатів. Це підтверджує доцільність розробки власного рішення на основі мультимодальних LLM з використанням Groq API як платформи для прискореної інференції.

ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Аналіз вимог до системи

Перед початком проєктування було проведено аналіз вимог до системи. Вимоги поділяються на функціональні та нефункціональні.

Функціональні вимоги визначають, що система повинна робити:

1. Система повинна дозволяти користувачу завантажувати одне або кілька зображень харчових продуктів;
2. Система повинна автоматично розпізнавати інгредієнти на завантажених зображеннях;
3. система повинна генерувати не менше трьох рецептів на основі розпізнаних інгредієнтів;
4. користувач повинен мати можливість вручну додавати та видаляти інгредієнти зі списку;
5. система повинна підтримувати повторну генерацію рецептів після редагування списку інгредієнтів;
6. система повинна забезпечувати реєстрацію та автентифікацію користувачів;
7. автентифікований користувач повинен мати можливість зберігати рецепти до особистого кабінету;
8. користувач повинен мати можливість переглядати та видаляти збережені рецепти.

Нефункціональні вимоги визначають якісні характеристики системи:

1. час відповіді на запит розпізнавання не повинен перевищувати 10 секунд за нормальних умов навантаження;
2. інтерфейс повинен бути адаптивним та коректно відображатись на мобільних пристроях з роздільною здатністю екрану від 320px, планшетах та десктопних браузерах з підтримкою останніх версій Chrome, Firefox та Safari;

3. зберігання паролів користувачів повинно відповідати стандартам безпеки: паролі зберігаються виключно у хешованому вигляді за алгоритмом bcrypt, персональні дані передаються лише через захищений протокол HTTPS;
4. система повинна бути розгорнута у хмарному середовищі та доступна через публічний URL.

2.2 Проєктування архітектури системи

Розроблений веб-додаток побудований за трирівневою клієнт-серверною архітектурою, яка включає рівень представлення, рівень бізнес-логіки та рівень даних. Крім того, система взаємодіє із зовнішнім AI API. Загальна архітектура системи зображена на рис. 2.1.

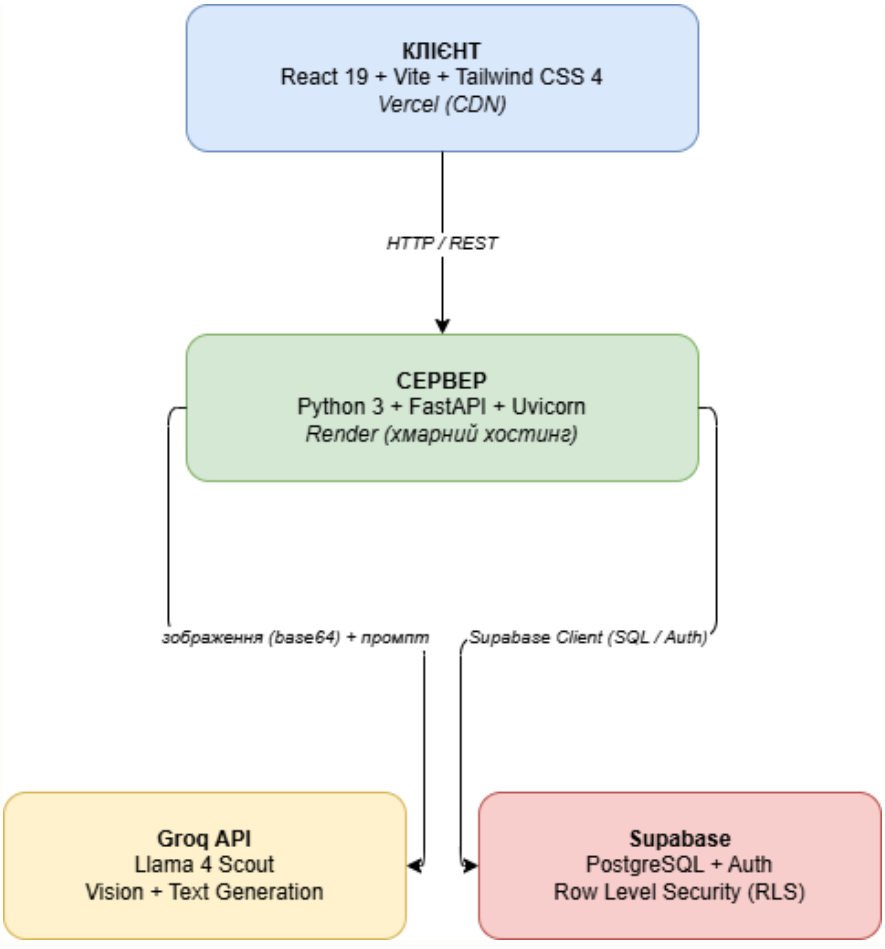


Рис. 2.1 Загальна архітектура веб-додатку

Рівень представлення реалізований у вигляді односторінкового застосунку на базі бібліотеки React [9] версії 19. Збірка проєкту здійснюється інструментом Vite, що забезпечує швидкий цикл розробки завдяки нативній підтримці ES-модулів. Стилізація компонентів виконана з використанням Tailwind CSS версії 4 - утилітарного CSS-фреймворку, що дозволяє описувати зовнішній вигляд елементів безпосередньо в JSX-розмітці без необхідності створювати окремі CSS-файли. Frontend розгорнуто на платформі Vercel, яка забезпечує глобальну CDN-дистрибуцію статичних файлів та автоматичне розгортання при оновленні репозиторію.

Взаємодія між рівнем представлення та рівнем бізнес-логіки відбувається виключно через HTTP-запити до REST API. Frontend не має прямого доступу до backend логіки - він лише надсилає запити та обробляє відповіді у форматі JSON. Єдиним винятком є взаємодія з Supabase [12], яка відбувається безпосередньо з клієнтського коду через офіційну JavaScript-бібліотеку, захищену механізмом Row Level Security.

Рівень бізнес-логіки реалізований у вигляді REST API на базі фреймворку FastAPI (Python). FastAPI обраний через нативну підтримку асинхронного програмування, автоматичну генерацію OpenAPI-документації та зручну роботу з Pydantic-моделями для валідації вхідних даних. Сервер запускається через ASGI-сервер Uvicorn, що забезпечує високу продуктивність завдяки асинхронній обробці запитів. Backend розгорнуто на платформі Render у вигляді веб-сервісу.

Рівень бізнес-логіки відповідає за дві ключові операції: кодування зображення у формат base64 та формування запитів до Groq API, а також обробку відповідей моделі - очищення від markdown-обгорток та парсинг JSON. Таким чином, весь складний код взаємодії з AI інкапсульований у backend і недоступний клієнту.

Рівень даних реалізований на базі хмарної платформи Supabase, яка надає керований екземпляр PostgreSQL, вбудовану систему автентифікації (Supabase Auth) та клієнтську JavaScript-бібліотеку для взаємодії з базою даних

безпосередньо з frontend без необхідності передавати запити через власний backend.

Взаємодія з AI здійснюється через Groq API - хмарний сервіс, що надає доступ до відкритих LLM-моделей з прискореною інференцією на спеціалізованих LPU-процесорах.

Зовнішній AI API - Groq API - забезпечує доступ до мультимодальної моделі Llama 4 Scout з прискореною інференцією на спеціалізованих LPU-процесорах. Groq API інтегрується виключно на рівні backend - frontend ніколи не взаємодіє з Groq напряму, що забезпечує захист API-ключа від витоку у клієнтський код.

Така архітектура забезпечує чіткий поділ відповідальності між компонентами: frontend відповідає за відображення та взаємодію з користувачем, backend - за бізнес-логіку та інтеграцію з AI, Supabase - за зберігання даних та автентифікацію. Кожен компонент може масштабуватись та замінюватись незалежно від інших.

Backend розгорнуто на платформі Render у вигляді веб-сервісу, що запускається командою uvicorn.

2.3 Обґрунтування вибору технологічного стеку

Вибір технологічного стеку є одним із ключових архітектурних рішень при розробці веб-застосунку. У даному проєкті кожна технологія обиралась на основі порівняльного аналізу альтернатив за критеріями продуктивності, зручності розробки, екосистеми та відповідності задачі.

Вибір фреймворку для backend - Основними альтернативами для реалізації серверної частини на Python були Flask та Django. Flask є мінімалістичним мікрофреймворком з широкими можливостями розширення, однак не має вбудованої підтримки асинхронного програмування та автоматичної валідації вхідних даних - обидві ці можливості потребують підключення сторонніх бібліотек. Django є повнофункціональним фреймворком з вбудованою ORM, адміністративною панеллю та системою автентифікації, однак його архітектура

орієнтована на монолітні застосунки і є надлишковою для легковагового REST API з двома ендпоінтами. FastAPI обраний завдяки нативній підтримці асинхронного програмування через Python asyncio, автоматичній валідації вхідних даних через Pydantic-моделі, автоматичній генерації інтерактивної OpenAPI-документації (Swagger UI) та вищій продуктивності порівняно з Flask і Django. Крім того, FastAPI має мінімальний поріг входу - простий ендпоінт записується у кілька рядків коду без зайвої конфігурації.

Стосовно вибору бібліотеки для frontend - серед провідних JavaScript-фреймворків розглядались Vue.js та Angular. Angular є повноцінним фреймворком з вбудованим управлінням станом та системою впровадження залежностей (dependency injection), однак має значний поріг входу та надмірну складність для проєктів середнього масштабу - він орієнтований на великі корпоративні застосунки з великими командами розробників. Vue.js відзначається простотою та поступовим порогом навчання, однак має меншу екосистему бібліотек та спільноту порівняно з React, що ускладнює пошук рішень для нестандартних задач. React обраний завдяки найбільшій екосистемі, широкому поширенню у індустрії, компонентній архітектурі що природно відображає структуру інтерфейсу застосунку. Важливим фактором також є офіційна підтримка Supabase клієнтської бібліотеки для React.

Вибір інструменту збірки frontend**.** Традиційним вибором для React-проєктів є Create React App (CRA), однак цей інструмент є застарілим і більше не отримує активної підтримки від спільноти. Vite обраний як сучасна альтернатива завдяки значно швидшому старту сервера розробки завдяки нативній підтримці ES-модулів у браузері, миттєвому оновленню (HMR - Hot Module Replacement) без повного перезавантаження, та оптимізованій збірці на основі Rollup.

Також обов’язковою частиною проєкту був вибір CSS фреймворку. Розглядались Bootstrap та звичайний CSS. Bootstrap надає готові компоненти інтерфейсу, однак генерує характерний "bootstrap-вигляд" що важко кастомізувати без перевизначення стилів. Tailwind CSS [15] обраний як утилітарний фреймворк, що дозволяє описувати зовнішній вигляд безпосередньо в JSX-розмітці через

короткі класи-утиліти. Це усуває необхідність перемикання між файлами компонентів та стилів і спрощує підтримку консистентного дизайну.

Вибір платформи для бази даних та автентифікації. Основною альтернативою є Firebase від Google. Firebase є зрілою платформою з великою спільнотою та широким набором сервісів, однак використовує власну NoSQL базу даних Firestore з документо-орієнтованою моделлю даних. Для застосунку з чіткими реляційними зв'язками між користувачами та їх рецептами реляційна модель є природнішим вибором. Supabase обраний завдяки використанню PostgreSQL - найпотужнішої відкритої реляційної СУБД з підтримкою Row Level Security, що дозволяє реалізувати ізоляцію даних між користувачами безпосередньо на рівні бази даних без додаткової логіки у коді застосунку. Додатковою перевагою є відкритий вихідний код Supabase та можливість самостійного розгортання у майбутньому.

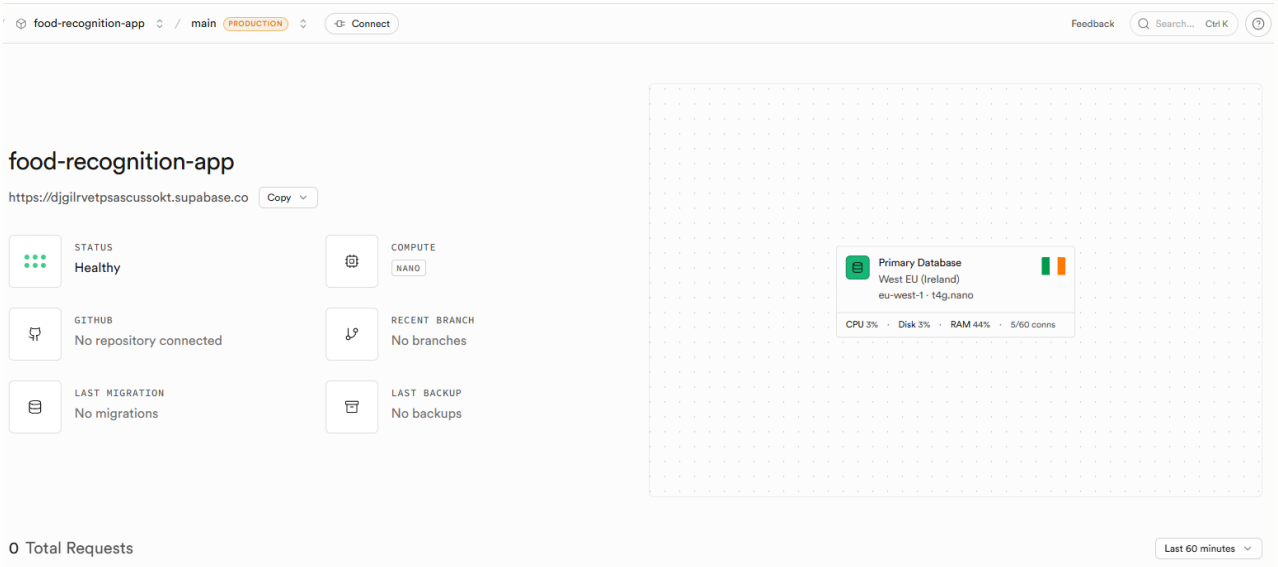


Рис. 2.2 Дашборд проєкту Supabase

Для frontend розглядались Netlify та GitHub Pages. GitHub Pages підтримує лише статичні сайти без серверного рендерингу і не інтегрується з системами збірки так зручно як спеціалізовані платформи. Netlify є повноцінним конкурентом Vercel, однак Vercel є офіційно рекомендованою платформою для React та Vite проєктів з найглибшою інтеграцією. Для backend розглядались Railway та Fly.io. Railway має зручний інтерфейс але менш щедрий безкоштовний рівень. Render

обраний завдяки простоті налаштування Python-сервісів та наявності безкоштовного рівня достатнього для навчального проєкту.

Порівняльний аналіз технологічних альтернатив зведено у таблиці 2.1.

Таблиця 2.1

Порівняльний аналіз технологічних альтернатив

Категорія	**Обраний варіант**	**Альтернативи**	**Ключова перевага обраного**
Backend фреймворк	FastAPI	Flask, Django	Асинхронність, автовалідація, OpenAPI
Frontend бібліотека	React	Vue.js, Angular	Екосистема, компонентна архітектура
Інструмент збірки	Vite	Create React App	Швидкість розробки, HMR
CSS-фреймворк	Tailwind CSS	Bootstrap, CSS	Утилітарний підхід, кастомізація
База даних	Supabase (PostgreSQL)	Firebase (Firestore)	Реляційна модель, RLS
Frontend хостинг	Vercel	Netlify, GitHub Pages	CI/CD, CDN, Vite-інтеграція
Backend хостинг	Render	Railway, Fly.io	Простота, безкоштовний план

2.4 Проєктування бази даних

База даних системи розгорнута на платформі Supabase (PostgreSQL). Схема бази даних включає дві основні сутності: користувачі (users) та збережені рецепти (saved_recipes).

Управління користувачами повністю делеговано вбудованій системі Supabase Auth. Ця система автоматично створює та підтримує таблицю auth.users зі стандартними полями: унікальний ідентифікатор користувача (UUID), електронна

адреса, хешований пароль, дата реєстрації та метадані сесії. Розробнику не потрібно самостійно реалізовувати хешування паролів, управління токенами або підтвердження електронної пошти - вся ця логіка інкапсульована у Supabase Auth.

Таблиця збережених рецептів saved_recipes спроєктована наступним чином та описана у таблиці 2.2.

Таблиця 2.2

Структура таблиці saved_recipes

Поле	Тип	Обмеження	Опис
id	int8	PRIMARY KEY, GENERATED BY DEFAULT AS IDENTITY	Унікальний ідентифікатор запису
user_id	uuid	NULLABLE, FK - auth.users(id)	Ідентифікатор користувача-власника
name	text	NULLABLE	Назва рецепту
ingredients	text	NULLABLE	Список інгредієнтів
instructions	text	NULLABLE	Інструкція приготування
created_at	timestamptz	DEFAULT now()	Дата та час збереження

Зовнішній ключ user_id забезпечує зв'язок між рецептом та його власником і дозволяє застосовувати механізм Row Level Security (RLS) - вбудований у PostgreSQL механізм безпеки на рівні рядків. Завдяки RLS кожен користувач може читати, створювати та видаляти виключно власні рецепти, навіть якщо запити до бази даних надходять безпосередньо з клієнтського JavaScript-коду без

проходження через серверний backend. ER-діаграма бази даних зображена на рис. 2.3.

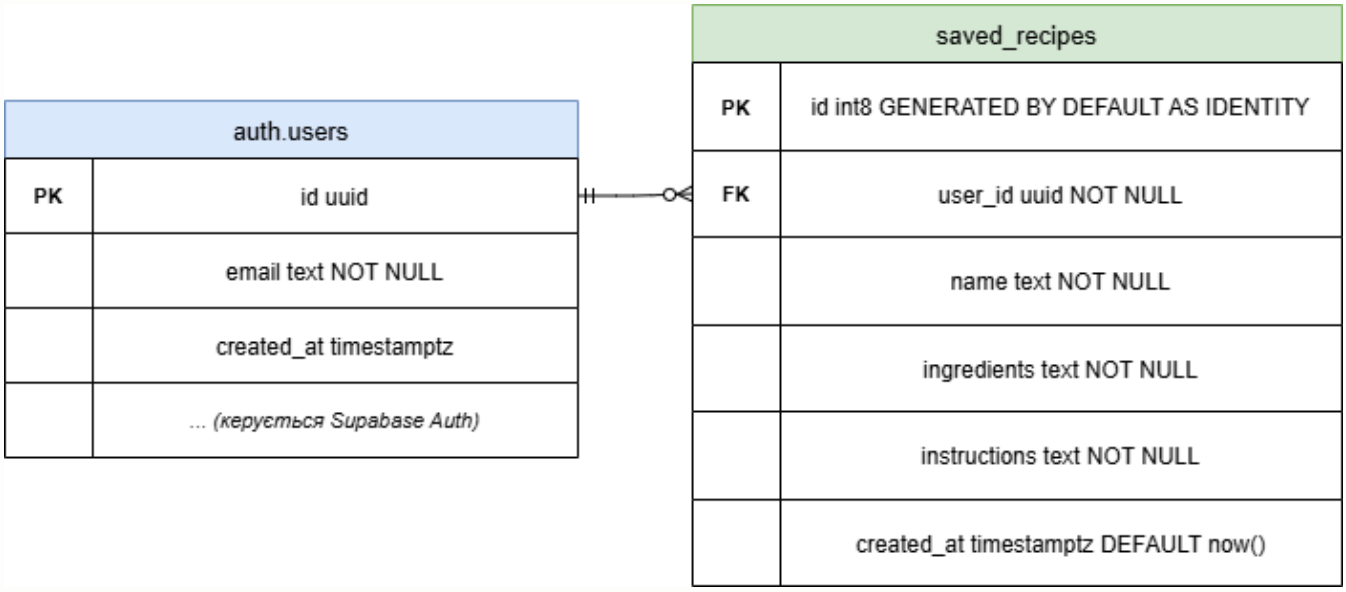


Рис. 2.3 ER-діаграма бази даних

2.5 Проєктування REST API

Серверна частина надає два ендпоінти, що реалізують основну бізнес-логіку системи. Кожен ендпоінт відповідає за окремий етап двоетапного конвеєру обробки - розпізнавання інгредієнтів та генерацію рецептів. Специфікація API описана у таблиці 2.3.

Таблиця 2.3

Специфікація REST API

Метод	**Ендпоінт**	**Вхідні дані**	**Відповідь**	**Опис**
POST	/analyze-image	multipart/form-data: file (image), language (string)	JSON: { result: string }	Розпізнавання інгредієнтів на зображенні

Продовження таблиці 2.3

Метод	Ендпоінт	Вхідні дані	Відповідь	Опис
POST	/get-recipes	JSON: { ingredients: string, language: string, priority_ingredients }	JSON: { recipes: array }	Генерація рецептів за списком інгредієнтів

Ендпоінт POST /analyze-image приймає зображення у форматі multipart/form-data разом з параметром мови language. Backend кодує отриманий файл у base64 та передає до Groq API разом із текстовим промптом. Модель аналізує зображення та повертає JSON-об'єкт де кожен ключ є назвою виявленого інгредієнта мовою інтерфейсу, а значення - порожній рядок. Backend повертає цей рядок клієнту загорнутим у поле result без додаткового парсингу. Приклад відповіді:

{ "result": "{\"томат\": \"\", \"огірок\": \"\", \"куряче філе\": \"\"}" }

Парсинг зі значення поля result виконується на стороні frontend через JSON.parse.

Ендпоінт POST /get-recipes приймає JSON-тіло з трьома полями: ingredients - рядок з переліком інгредієнтів через кому, language - мова відповіді, та priority_ingredients - необов'язкове поле (за замовчуванням порожній рядок). Модель генерує три рецепти та повертає масив об'єктів. Кожен об'єкт містить поля name (назва страви), ingredients (інгредієнти з кількостями), instructions (покрокова інструкція), calories (калорійність) та time (час приготування). Приклад відповіді:

```
json
{
  "recipes": [
    {
      "name": "Запечені овочі з часником",
```

```
      "ingredients": "томат 2шт, огірок 1шт, ...",
      "instructions": "1. Розігріти духовку до 200°C. 2. ...",
      "calories": "280 ккал",
      "time": "35 хвилин"
    }
  ]
}
```


Розподіл задачі розпізнавання та генерації рецептів на два окремі ендпоінти обумовлений кількома причинами. По-перше, це дозволяє послідовно обробляти кілька зображень та об'єднувати результати перед генерацією рецептів. По-друге, користувач може редагувати список інгредієнтів після розпізнавання і повторно викликати лише другий ендпоінт без повторного аналізу зображень. По-третє, такий підхід відповідає принципу єдиної відповідальності та спрощує тестування кожного компонента окремо.

2.6 Проєктування інтерфейсу користувача

Інтерфейс користувача спроєктований як односторінковий застосунок з трьома основними екранами: екраном автентифікації, головним екраном та екраном збережених рецептів. При проєктуванні інтерфейсу застосовувались принципи мінімалізму та інтуїтивності - користувач має отримати результат з мінімальною кількістю дій.

Екран автентифікації відображається для неавтентифікованих користувачів і містить форму входу або реєстрації. Перемикання між режимами входу та реєстрації здійснюється без перезавантаження сторінки. Форма містить поля для введення електронної адреси та паролю, кнопку підтвердження та посилання для перемикання між режимами. Після успішної реєстрації відображається повідомлення з проханням підтвердити електронну адресу. Після успішного входу користувач автоматично перенаправляється на головний екран. Екран автентифікації веб-додатку зображено на рис. 2.4.

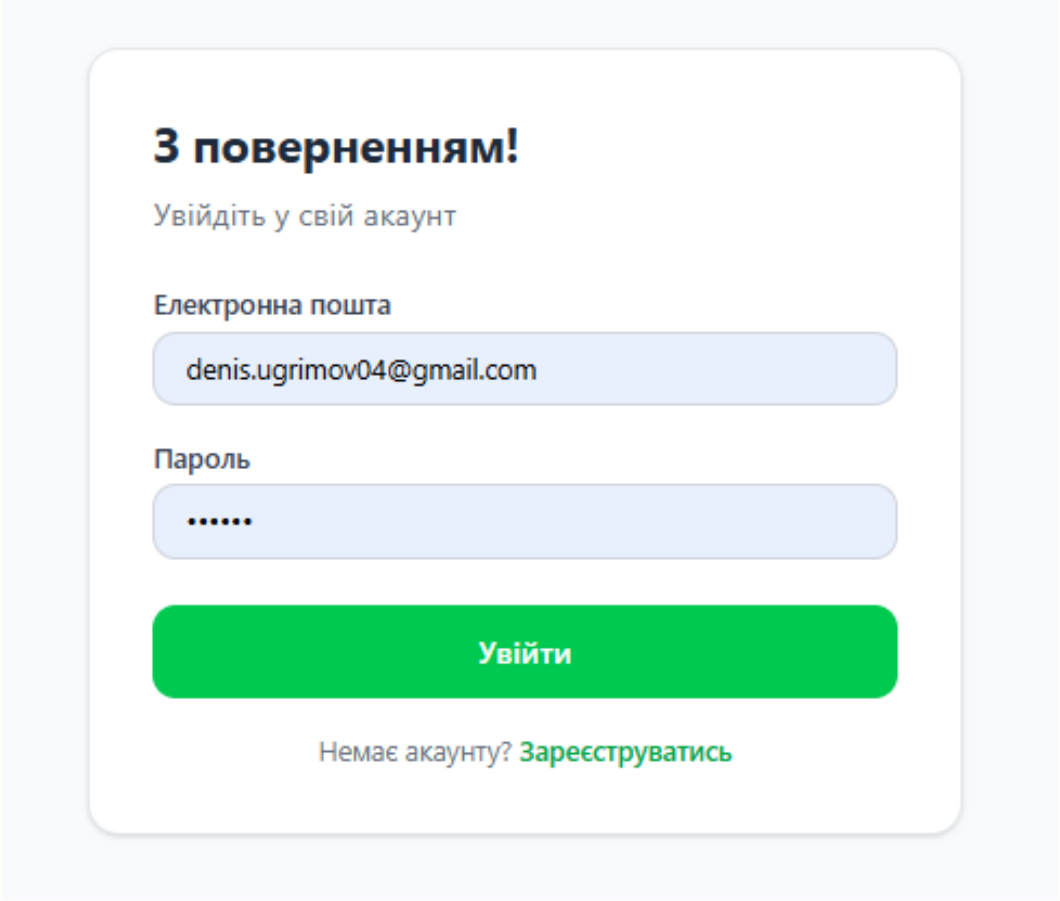


Рис. 2.4 Екран автентифікації

Головний екран є центральним елементом інтерфейсу і складається з трьох логічних блоків що відображаються послідовно в міру взаємодії користувача з системою. Перший блок - завантаження зображень - відображається одразу при відкритті застосунку. Він містить елемент вибору файлів з підтримкою множинного вибору, область попереднього перегляду завантажених зображень та кнопку запуску аналізу. Користувач може видалити будь-яке із завантажених зображень до початку аналізу натиснувши кнопку видалення на мініатюрі. Головний екран застосунку з блоком завантаження зображень зображено на рис. 2.5.

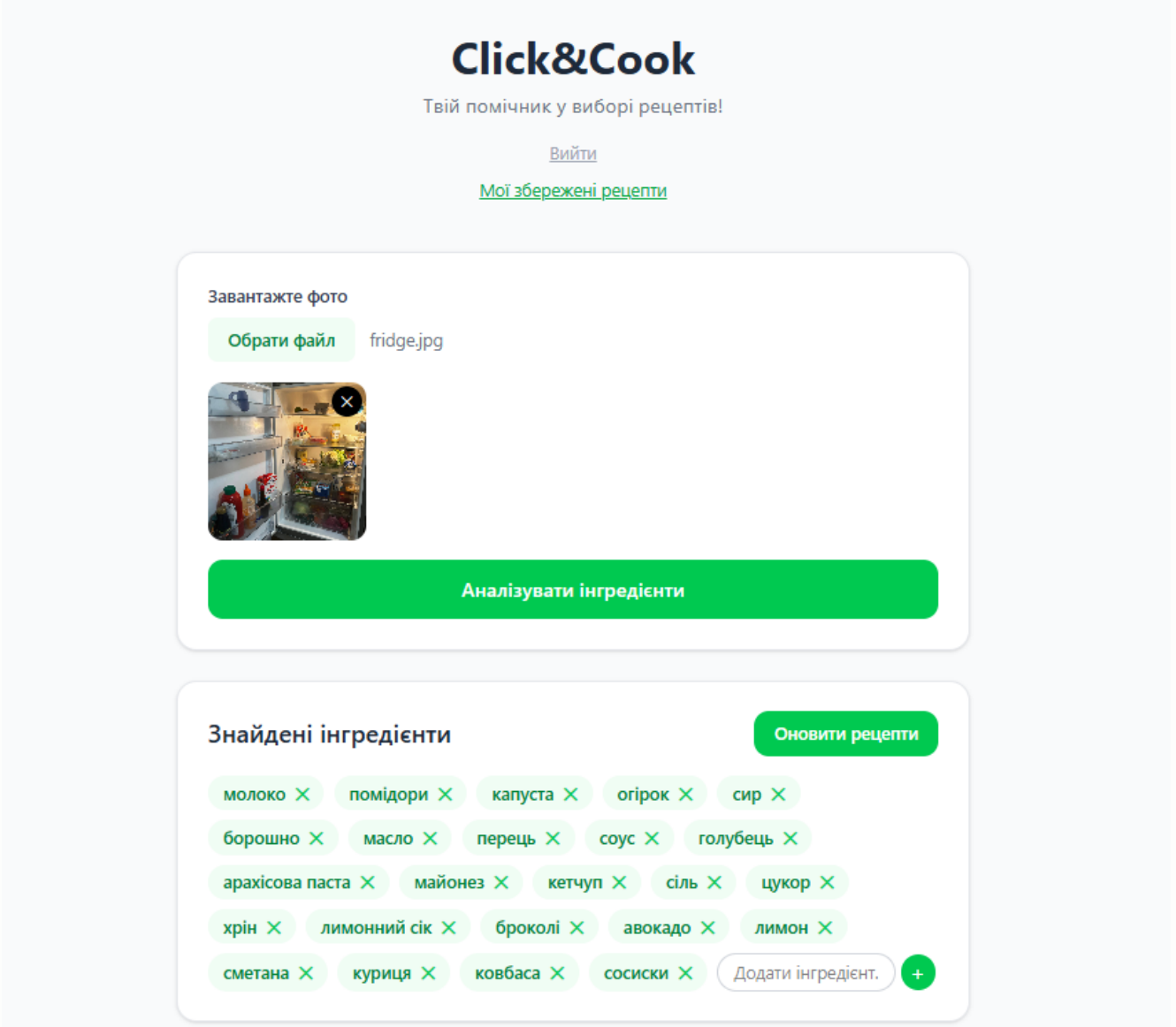


Рис. 2.5 Головний екран

Другий блок - список інгредієнтів - відображається після завершення аналізу. Кожен розпізнаний інгредієнт представлений у вигляді тегу із кнопкою видалення. Під списком інгредієнтів розташоване текстове поле для ручного введення нових інгредієнтів та кнопка оновлення рецептів. Такий підхід дозволяє користувачу коригувати результати автоматичного розпізнавання та отримувати більш точні рецепти.

Третій блок - список рецептів - відображає три згенеровані рецепти у вигляді карток. Кожна картка містить назву страви, час приготування та калорійність у

вигляді бейджів, список інгредієнтів з кількостями, покрокову інструкцію приготування та кнопку збереження. Кнопка збереження автоматично переходить у неактивний стан якщо рецепт вже збережений - це запобігає дублюванню. Картки згенерованих рецептів зображено на рис. 2.6.

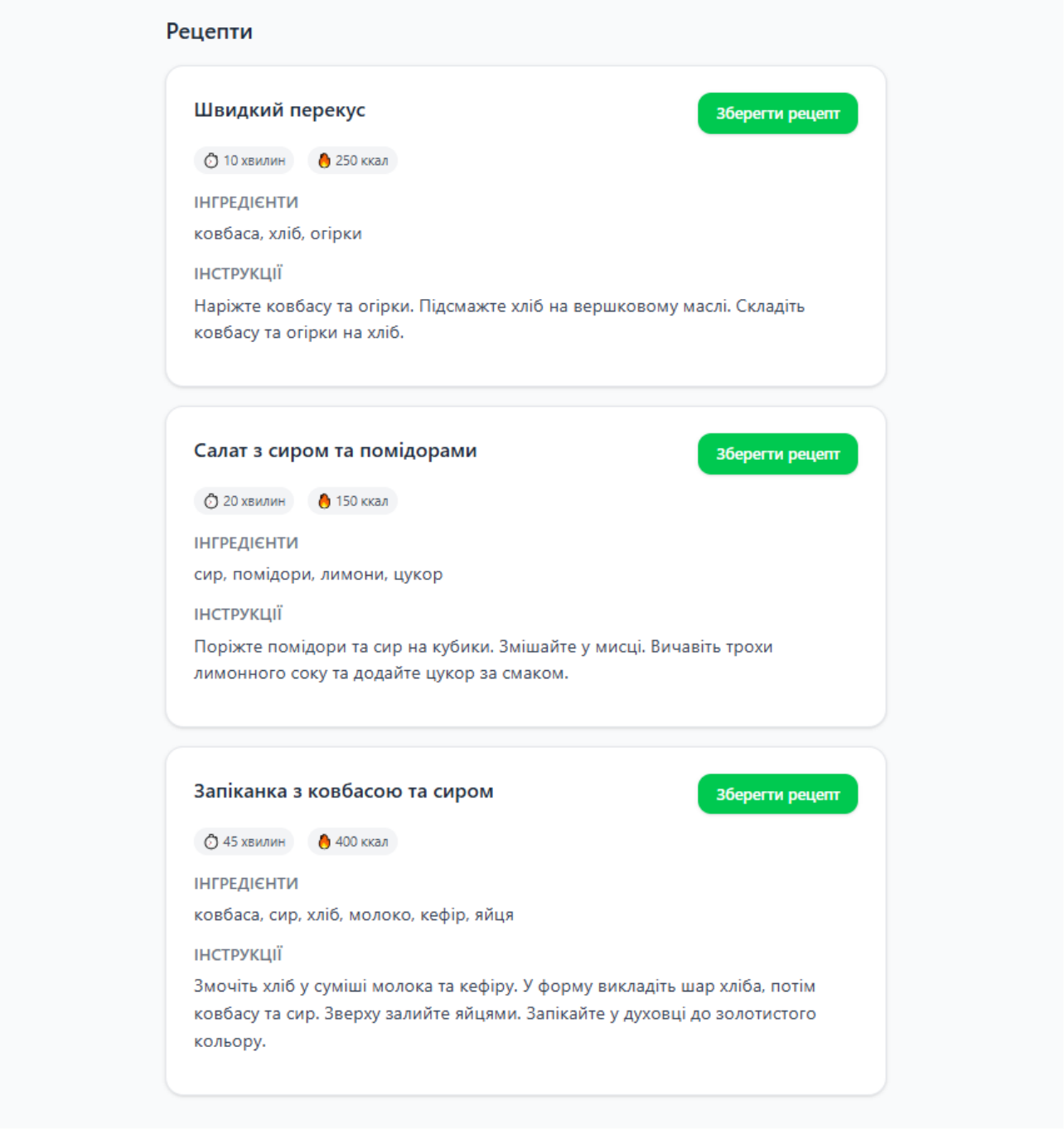


Рис. 2.6 Картки згенерованих рецептів

відображає всі рецепти збережені поточним користувачем у вигляді карток аналогічного дизайну до карток на головному екрані. Картки відсортовані від найновішої до найстарішої. Кожну картку можна видалити натиснувши відповідну кнопку. У верхній частині екрану розташована кнопка повернення до головного екрану. Екран збережених рецептів зображено на рис. 2.7.

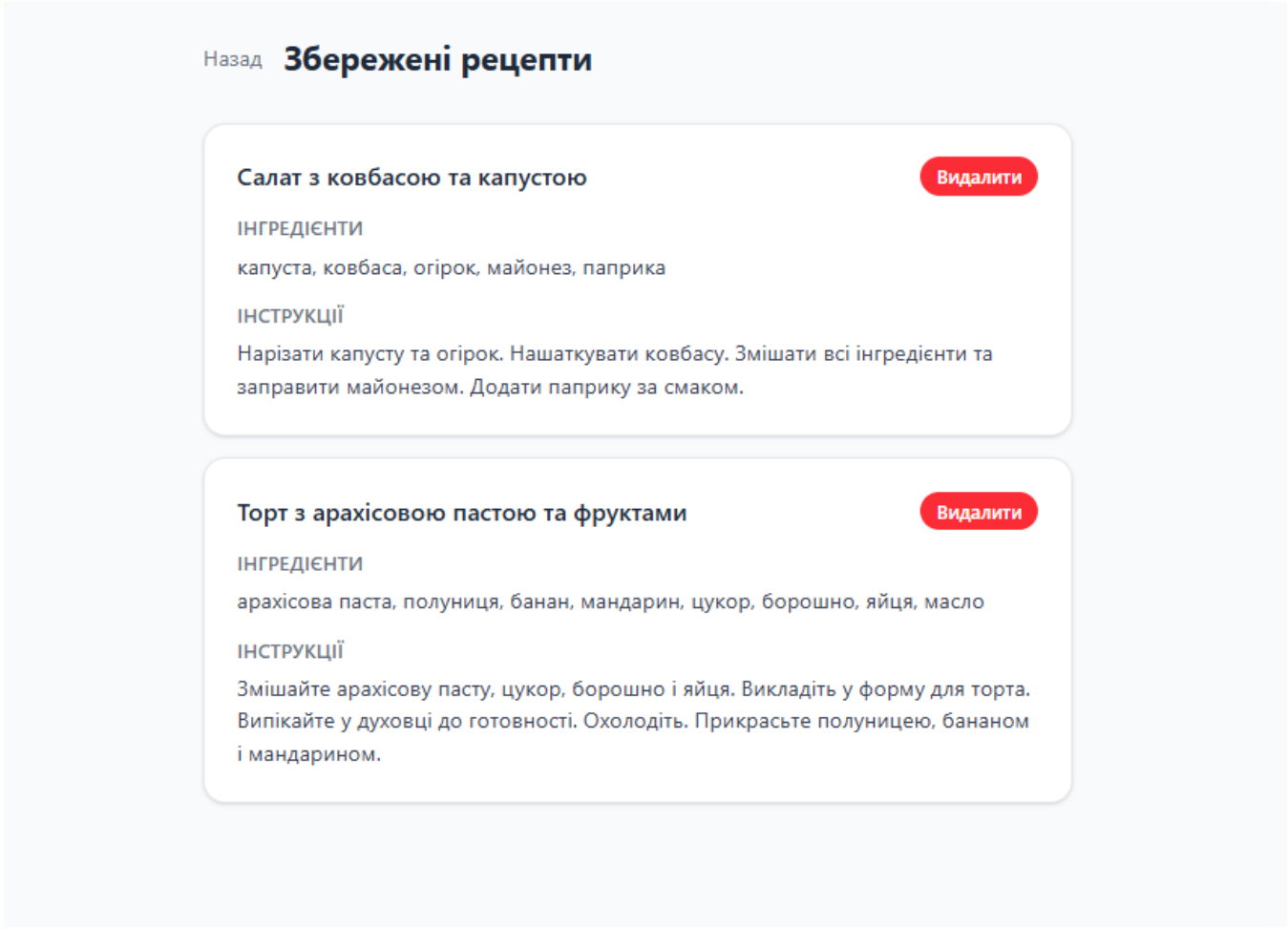


Рис. 2.7 Екран збережених рецептів

Навігація між екранами реалізована без використання бібліотеки маршрутизації - перемикання відбувається через зміну стану компонента App залежно від значень session та showSaved. Діаграма стану на рис. 2.8.

Рис. 2.8 Діаграма стану

Адаптивність інтерфейсу забезпечується утилітарними класами Tailwind CSS. Завдяки системі брейкпоінтів Tailwind інтерфейс коректно відображається як на широких моніторах так і на мобільних пристроях з вузьким екраном. На мобільних пристроях елементи автоматично перебудовуються у вертикальний стек, розміри шрифтів та відступи адаптуються для зручного використання на сенсорному екрані.

У другому розділі сформульовано функціональні та нефункціональні вимоги до системи, спроєктовано трирівневу клієнт-серверну архітектуру веб-додатку. Проведено порівняльний аналіз технологічних альтернатив та обґрунтовано вибір FastAPI для реалізації серверної частини, React та Tailwind CSS для клієнтської частини, Supabase як платформи для бази даних та автентифікації, Groq API для доступу до мультимодальної LLM-моделі, Vercel [14] та Render як платформ розгортання. Розроблено схему бази даних з урахуванням механізму Row Level

Security для забезпечення ізоляції даних між користувачами на рівні СУБД. Спроєктовано REST API з двома ендпоінтами, що реалізують двоетапний конвеєр обробки зображень: розпізнавання інгредієнтів та генерацію рецептів. Визначено структуру інтерфейсу користувача з трьома основними екранами - автентифікації, головним та екраном збережених рецептів.

РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ

3.1 Реалізація серверної частини

Серверна частина веб-додатку реалізована мовою Python з використанням фреймворку FastAPI. Точкою входу є файл main.py, який ініціалізує застосунок, налаштовує CORS-політику та реєструє маршрути.

CORS (Cross-Origin Resource Sharing) налаштовано з явним переліком дозволених джерел: локальний сервер розробки http://localhost:5173 та три продуктивні домени Vercel, на яких розгорнуто frontend. Такий підхід є більш безпечним порівняно з дозволом усіх джерел (*) і відповідає рекомендованій практиці для продуктивних веб-застосунків.

Взаємодія з Groq API реалізована через офіційну Python-бібліотеку groq. Клієнт ініціалізується з API-ключем, що зчитується зі змінної середовища GROQ_API_KEY через бібліотеку python-dotenv. Використання змінних середовища замість жорстко закодованих значень є обов'язковою практикою безпеки, що унеможливлює потрапляння секретних ключів до репозиторію.

Ендпоінт POST /analyze-image приймає два параметри у форматі multipart/form-data: файл зображення (file) та мову відповіді (language, за замовчуванням "Ukrainian"). Оскільки конвертація зображення у формат JPEG виконується на стороні клієнта засобами Canvas API ще до відправки на сервер, backend отримує вже підготовлений файл і лише кодує його вміст у рядок base64 одним рядком коду.

Промпт для розпізнавання інструктує модель уважно переглянути зображення та визначити всі видимі харчові інгредієнти. Відповідь повинна бути виключно у форматі JSON, де ключ - назва інгредієнта, а значення - порожній рядок. Мова назв інгредієнтів визначається параметром language,відповідно до зафіксованої локалі застосунку. Заборонено використовувати узагальнені назви

типу ingredient_1 - кожен ключ повинен бути реальною назвою продукту.

```
    prompt = f'''Look carefully at this image. Identify ALL visible food ingredients.
Reply ONLY in JSON format. Use the ingredient name as the key, leave the value empty.
Write ingredient names in {language} language.
Example format:
{{
  "молоко": "",
  "яблуко": "",
  "хліб": ""
}}
Be thorough. No recipes, just ingredients. No generic names like ingredient_1.'''
```

Рис. 3.1 Промпт для розпізнавання інгредієнтів

Відповідь моделі перед поверненням клієнту проходить додаткову обробку. Оскільки LLM інколи обгортає JSON у markdown-блоки з потрійними зворотними лапками (```json ... ```), backend явно видаляє ці префікси та суфікси методами removeprefix та removesuffix. Очищений рядок повертається клієнту у вигляді { "result": "<json-рядок>" }, де подальший парсинг JSON виконується на стороні frontend.

Ендпоінт POST /get-recipes приймає JSON-тіло з трьома полями: ingredients - рядок з переліком інгредієнтів через кому, priority_ingredients - необов'язкове поле для пріоритетних інгредієнтів (передбачено архітектурно, але не реалізовано в UI), та language - мова відповіді (за замовчуванням "English").

Промпт для генерації рецептів є центральним елементом системи і містить вісім суворих правил для забезпечення якості та коректності результату:

```
@app.post("/get-recipes")
async def get_recipes(request: IngredientsRequest):
    prompt = f'''You are a professional chef. The user has these ingredients: {request.ingredients}

Basic pantry staples also available: water, salt, black pepper, olive oil, butter, flour, sugar, eggs, garlic, onion, vinegar, common
spices.

STRICT RULES:
1. Every ingredient listed in a recipe's "ingredients" field MUST actually appear and be used in the "instructions" field. No exceptions.
2. Do NOT add ingredients that are not in the user's list or pantry staples.
3. Do NOT invent or reference specific named recipes from chefs or websites — just create original recipes.
4. Each recipe must be different and use different main ingredients from the list.
5. For "calories" field: write ONLY a number followed by "ккал", example: "350 ккал". No other text.
6. For "time" field: write ONLY a number followed by "хвилин", example: "20 хвилин". No other text.
7. Recipes must be culinarily logical and realistic — dishes that actually exist in real cuisine. Do not combine sweet fruits with savory
pasta, cucumbers with fruits, or other unusual combinations that would not appear in a real cookbook. If the available ingredients do not
form a logical meal together, use only the ones that make sense for that recipe.
8. Reply in proper Ukrainian language. Do not transliterate English words — use correct Ukrainian translations (e.g. "лимон" not "лемон",
"торт" not "тарт").

Suggest exactly 3 recipes:
1. A simple quick snack (under 15 minutes)
2. A medium difficulty dish (20-40 minutes)
3. A complex and creative dish (40+ minutes)

Reply in {request.language} language.

Reply ONLY in JSON like this:
{{
  "recipes": [
    {{"name": "", "ingredients": "", "instructions": "", "calories": "", "time": ""}},
    {{"name": "", "ingredients": "", "instructions": "", "calories": "", "time": ""}},
    {{"name": "", "ingredients": "", "instructions": "", "calories": "", "time": ""}}
  ]
}}'''
```

Рис. 3.2 Промпт для генерації рецептів

1. Кожен інгредієнт зі списку ingredients рецепту повинен реально використовуватись в полі instructions - без виключень.
2. Заборонено додавати інгредієнти яких немає у списку користувача або переліку базових продуктів кухні (вода, сіль, чорний перець, оливкова олія, вершкове масло, борошно, цукор, яйця, часник, цибуля, оцет, базові спеції).
3. Рецепти повинні бути оригінальними
4. Кожен з трьох рецептів повинен використовувати різні основні інгредієнти зі списку.
5. Поле calories містить виключно число та одиницю "ккал", наприклад: "350 ккал".
6. Поле time містить виключно число та одиницю "хвилин", наприклад: "20 хвилин".

7. Рецепти повинні бути кулінарно логічними та реалістичними - заборонено поєднання несумісних продуктів (наприклад, солодкі фрукти з макаронами, огірки з фруктами). Якщо наявні інгредієнти не утворюють логічної страви разом - використовуються лише ті, що підходять для конкретного рецепту.

8. Відповідь надається правильною українською мовою без транслітерації англійських слів (наприклад, "лимон", а не "лемон").

Однією з ключових проблем при роботі з великими мовними моделями є явище галюцинацій - генерація правдоподібного але фактично некоректного контенту. У контексті генерації рецептів це може проявлятись у вигляді додавання інгредієнтів яких немає у списку користувача, створення кулінарно нелогічних комбінацій продуктів або порушення формату JSON-відповіді.

Для мінімізації цього явища у промпті реалізовано блок STRICT RULES з вісьма явними заборонами. Дослідження Agarwal та Choudhary (2024) [8] підтверджують що явне визначення заборон є ефективнішим методом контролю виводу моделі порівняно з позитивними інструкціями. Додатково на рівні постобробки реалізовано фільтрацію керуючих символів та очищення від markdown-обгорток, що забезпечує стабільний парсинг JSON-відповіді незалежно від поведінки моделі.

Три рецепти структуровані за рівнем складності: простий швидкий перекус (до 15 хвилин), страва середньої складності (20-40 хвилин) та складна креативна страва (40+ хвилин). Структура JSON-відповіді для кожного рецепту містить поля name, ingredients, instructions, calories та time. Повний промпт наведено на рис. 3.2.

Відповідь моделі проходить більш розширену обробку порівняно з першим ендпоінтом. Окрім видалення markdown-обгорток, виконується додаткова фільтрація керуючих символів - всі символи з кодом нижче пробілу (крім табуляції) замінюються пробілом. Це запобігає помилкам парсингу JSON, що можуть виникати через символи нового рядка всередині рядкових значень JSON, які модель інколи генерує у полі instructions. Після очищення рядок парситься методом json.loads та повертається клієнту як Python-словник, який FastAPI автоматично серіалізує у JSON-відповідь.

3.2 Алгоритм роботи системи

Алгоритм роботи веб-додатку реалізує двоетапний конвеєр обробки зображень. На першому етапі користувач завантажує фотографію харчових продуктів, яка автоматично конвертується у формат JPEG засобами Canvas API та передається на серверний ендпоінт /analyze-image. Сервер формує запит до Groq API з моделлю Llama 4 Scout, яка аналізує зображення та повертає JSON зі списком розпізнаних інгредієнтів. Після отримання результатів користувач може відредагувати список - додати або видалити інгредієнти вручну. На другому етапі відредагований список передається до ендпоінту /get-recipes, який генерує три рецепти різного рівня складності. За бажанням користувач може зберегти рецепт до особистого кабінету в базі даних Supabase. Повний алгоритм роботи системи наведено на рис. 3.3.

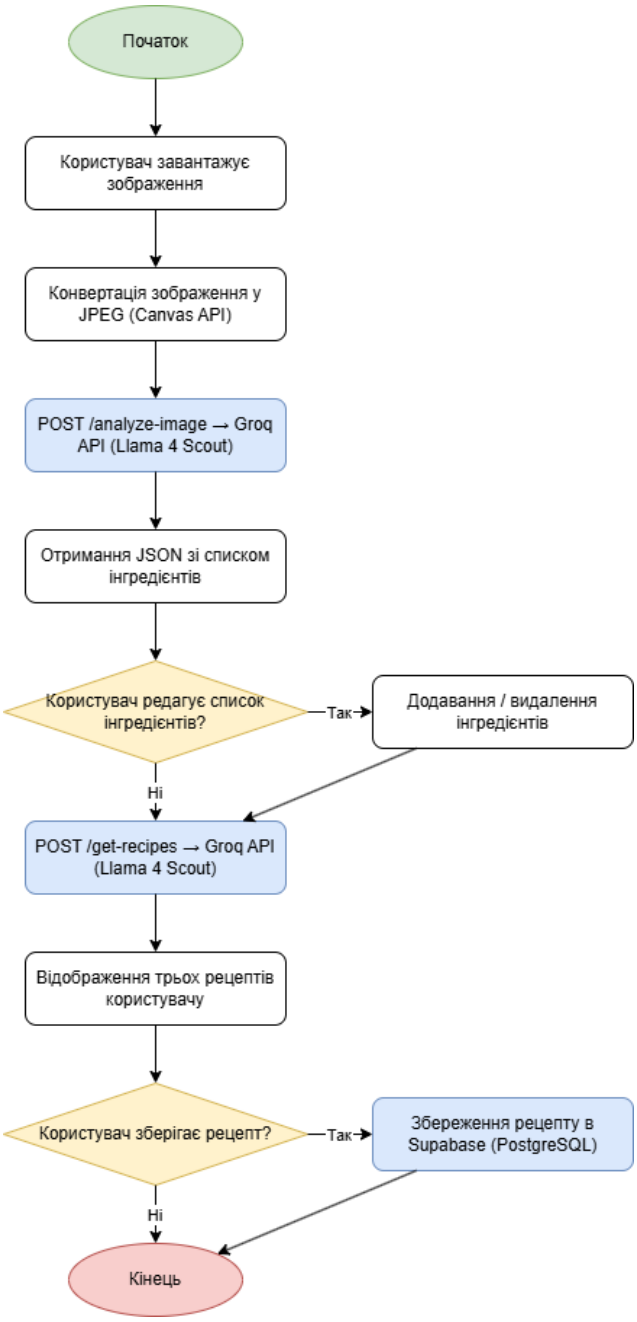


Рис. 3.3 Блок-схема алгоритму роботи системи

Для детальнішого розуміння взаємодії між компонентами системи розроблено діаграму послідовності. Вона відображає обмін повідомленнями між користувачем, frontend, backend, Groq API та Supabase у процесі розпізнавання інгредієнтів та генерації рецептів. Діаграму послідовності взаємодії компонентів системи наведено на рис. 3.3.

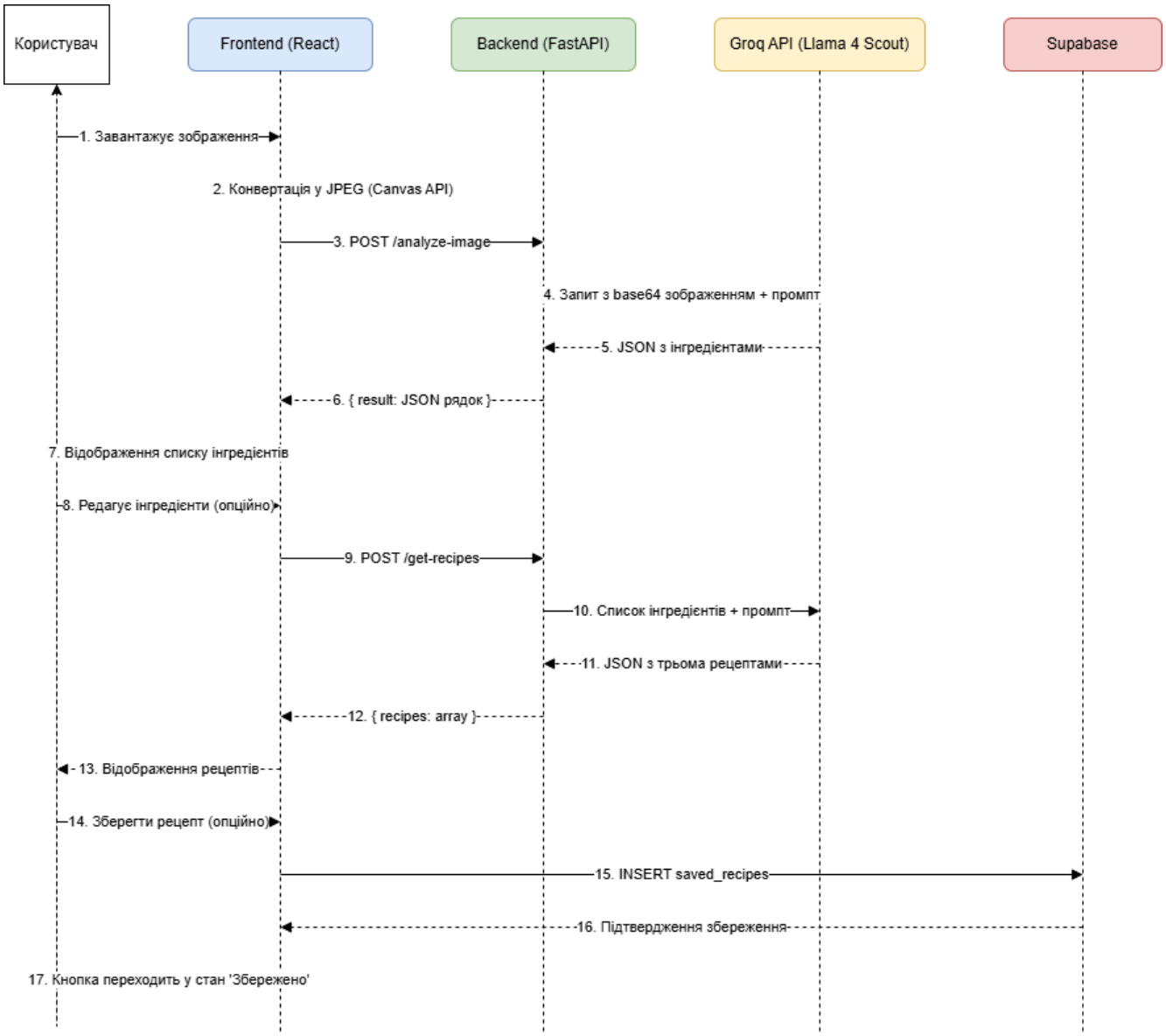


Рис. 3.4 Діаграма послідовності взаємодії компонентів системи

Послідовна обробка кількох зображень реалізована через цикл for...of, що забезпечує стабільну роботу без перевантаження зовнішнього Groq API і зображена на рис. 3.5

```
try {
  const allIngredients = {}

  for (const originalFile of image) {
    const file = await convertToJpeg(originalFile)
    const formData = new FormData()
    formData.append("file", file)
    formData.append("language", t.recipeLanguage)
    const response = await fetch(`${API_URL}/analyze-image`, {
      method: "POST",
      body: formData,
    })
    const data = await response.json()
    const parsed = JSON.parse(data.result)
    Object.entries(parsed).forEach(([key, value]) => {
      if (key !== "recipes") {
        allIngredients[key] = value
      }
    })
  }
```

Запит до бази даних для отримання збережених рецептів з сортуванням від найновішого реалізований наступним чином:

```
const { data, error } = await supabase
  .from("saved_recipes")
  .select("*")
  .order("created_at", { ascending: false })
```

3.3 Реалізація клієнтської частини

Клієнтська частина реалізована як односторінковий застосунок (Single Page Application) на базі React. Стан застосунку управляється вбудованими хуками React - useState та useEffect. Зовнішні бібліотеки управління станом не використовувались, оскільки складність стану не перевищує можливостей вбудованих засобів. Збірка проєкту виконується інструментом Vite, стилізація - утилітарним CSS-фреймворком Tailwind CSS 4.

Застосунок організований за компонентною архітектурою. Кореневий компонент App відповідає за логіку маршрутизації між екранами залежно від стану

автентифікації. При завантаженні перевіряється наявність активної сесії Supabase: якщо сесія відсутня - відображається компонент Auth; якщо користувач перейшов до збережених рецептів - відображається компонент SavedRecipes; в усіх інших випадках відображається головний екран.

Основні компоненти застосунку:

1. App - кореневий компонент, містить основну бізнес-логіку та стан застосунку;
2. Auth - форма входу та реєстрації з перемиканням між режимами;
3. SavedRecipes - екран зі списком збережених рецептів поточного користувача.

Всі текстові рядки інтерфейсу винесені в окремий файл translations.js у вигляді об'єкту з двома локалями - en (англійська) та ua (українська). Поточна локаль зафіксована як ua у кореневому компоненті: const t = translations["ua"]. Це визначає одночасно три речі: мову елементів інтерфейсу, мову в якій модель повертає назви розпізнаних інгредієнтів (параметр language передається разом із зображенням), та мову генерованих рецептів. Назва застосунку в українській локалі - **Click&Cook**.

Компонент завантаження зображень підтримує вибір кількох файлів одночасно через стандартний елемент <input type="file" multiple>. Після вибору для кожного файлу створюється локальний URL-об'єкт (URL.createObjectURL) для попереднього перегляду без завантаження на сервер. Користувач може видалити будь-яке з вибраних зображень до початку аналізу.

Перед відправкою на backend кожне зображення конвертується у формат JPEG засобами Canvas API браузера. Функція convertToJpeg створює елемент <canvas>, малює на ньому зображення та експортує результат методом canvas.toBlob з якістю 0.9. Конвертація на стороні клієнта забезпечує сумісність з різними вхідними форматами (PNG, WEBP, HEIC тощо) та зменшує обсяг даних, що передаються на сервер. У разі помилки конвертації оригінальний файл передається без змін завдяки обробнику img.onerror.

Зображення обробляються послідовно через цикл for...of. Для кожного файлу виконується конвертація у JPEG, формується об'єкт FormData з файлом та параметром мови, після чого надсилається POST-запит до ендпоінту /analyze-image. Відповідь сервера містить поле result - рядок з JSON-об'єктом, який парситься на клієнті через JSON.parse. Результати кожного зображення об'єднуються в єдиний словник інгредієнтів через Object.entries та перевірку на ключ "recipes" для уникнення конфліктів зі структурою результату. Послідовна обробка обрана замість паралельної для забезпечення стабільної роботи без перевантаження зовнішнього Groq API при завантаженні кількох зображень одночасно.

Після завершення аналізу всіх зображень об'єднаний список інгредієнтів передається до ендпоінту /get-recipes. Відповідь з трьома рецептами зберігається у стані компонента та відображається користувачу.

Після отримання результатів розпізнавання користувач може редагувати список інгредієнтів. Кожен інгредієнт відображається як тег з кнопкою видалення - натискання фільтрує масив інгредієнтів через setIngredients(prev => prev.filter(...)). Для додавання нового інгредієнту вручну реалізовано текстове поле з обробником клавіші Enter та кнопкою з символом +. Після редагування користувач може натиснути кнопку "Оновити рецепти" яка викликає функцію handleRegenerateRecipes - вона надсилає оновлений список до /get-recipes без повторного аналізу зображень.

Автентифікація реалізована через клієнтську бібліотеку Supabase JS. Реєстрація виконується методом supabase.auth.signUp, вхід - supabase.auth.signInWithPassword.

Реалізація функції handleSubmit компонента Auth здійснює виклик відповідного методу Supabase залежно від режиму - входу або реєстрації. Локалізація повідомлень про помилки реалізована через словник errorMessages, який відображає англійські повідомлення Supabase Auth на відповідні українські, що забезпечує зрозумілий зворотній зв'язок для користувача без розкриття технічних деталей. Локалізація повідомлень зображена на рис. 3.6

```
async function handleSubmit() {
  setLoading(true)
  setError(null)

  const { error } = isLogin
    ? await supabase.auth.signInWithPassword({ email, password })
    : await supabase.auth.signUp({ email, password })
  if (!error && !isLogin) {
    setMessage("Перевірте вашу пошту для підтвердження акаунту")
    setLoading(false)
    return
  }

  if (error) {
    const errorMessages = {
      "Invalid login credentials": "Невірна електронна пошта або пароль",
      "Email not confirmed": "Підтвердіть електронну пошту",
      "User already registered": "Користувач вже зареєстрований",
      "Password should be at least 6 characters": "Пароль має бути не менше 6 символів",
      "Unable to validate email address: invalid format": "Невірний формат електронної пошти",
      "missing email or phone": "Не вказано пошту чи телефон",
    }
    setError(errorMessages[error.message] || error.message)
  }
  setLoading(false)
}
```

Рис. 3.6 Локалізація повідомлень про помилки

Важливо зазначити, що реєстрація потребує підтвердження email. Сесія автоматично зберігається у localStorage браузера бібліотекою Supabase і відновлюється при перезавантаженні сторінки.

Для відстеження зміни стану автентифікації використовується підписка supabase.auth.onAuthStateChange у хуку useEffect. Це дозволяє реактивно оновлювати інтерфейс при вході, виході та автоматичному поновленні токену без додаткових запитів до сервера. Підписка існує протягом всього життєвого циклу застосунку.

Збереження рецептів виконується безпосередньо з клієнтського коду через Supabase JS клієнт. Перед збереженням система перевіряє чи рецепт з такою самою назвою вже існує у базі даних поточного користувача через запит select("id").eq("name", recipe.name). Якщо рецепт вже збережений - кнопка переходить у неактивний стан без повторного запису. Якщо рецепту немає -

виконується insert з полями name, ingredients, instructions та user_id. Реалізацію перевірки дублікатів та збереження рецепту наведено на рис 3.5.

```
const { data: existing } = await supabase
  .from("saved_recipes")
  .select("id")
  .eq("user_id", session.user.id)
  .eq("name", recipe.name)
  .single()

if (existing) {
  setSavedIndices(prev => new Set([...prev, index]))
  return
}

const { error } = await supabase
  .from("saved_recipes")
  .insert({
    name: recipe.name,
    ingredients: recipe.ingredients,
    instructions: recipe.instructions,
    user_id: session.user.id
  })
```

Рис. 3.5 Перевірка на дублікати

Перевірка вже збережених рецептів виконується також автоматично після кожної генерації або регенерації рецептів через функцію checkAlreadySaved. Вона отримує всі збережені назви поточного користувача та порівнює їх з поточними рецептами, формуючи множину індексів вже збережених карток. Це дозволяє коректно відображати стан кнопки "Збережено" навіть для рецептів, що були збережені в попередніх сесіях.

Екран збережених рецептів реалізований в окремому компоненті SavedRecipes. При монтуванні компонента виконується запит до Supabase для отримання всіх рецептів поточного користувача, відсортованих за датою створення у порядку від найновішого до найстарішого. Фільтрація рецептів за поточним користувачем відбувається автоматично на рівні бази даних завдяки правилам Row Level Security - запит не потребує явного фільтра user_id у коді клієнта. Видалення

рецепту виконується методом supabase.from("saved_recipes").delete().eq("id", id), після чого локальний стан оновлюється фільтрацією масиву без повторного запиту до бази даних.

3.4 Забезпечення безпеки системи

Безпека є критичним аспектом будь-якого веб-застосунку, що працює з персональними даними користувачів. У розробленій системі реалізовано кілька рівнів захисту.

Всі секретні ключі - GROQ_API_KEY, URL та анонімний ключ Supabase - зберігаються виключно у змінних середовища і ніколи не потрапляють до репозиторію вихідного коду. Файл .env включений до .gitignore. На платформі Render GROQ_API_KEY налаштований як секретна змінна середовища, недоступна у логах сервісу. Ключ Supabase, що використовується на frontend є публічним анонімним ключем (anon key) - він призначений для клієнтського використання і сам по собі не надає привілейованого доступу до бази даних без відповідного JWT-токена [18], [19] автентифікованого користувача.

Система автентифікації повністю делегована Supabase Auth, що реалізує галузеві стандарти безпеки. Паролі зберігаються у хешованому вигляді та ніколи не передаються у відкритому вигляді. Сесії керуються через JWT-токени з обмеженим терміном дії та механізмом автоматичного поновлення. Після реєстрації система надсилає підтвердження на електронну пошту - користувач не може увійти до підтвердження адреси. Мінімальна довжина паролю контролюється на рівні Supabase Auth. Локалізовані повідомлення про помилки автентифікації відображаються користувачу без розкриття технічних деталей помилки.

Механізм Row Level Security забезпечує ізоляцію даних між користувачами безпосередньо на рівні СУБД незалежно від логіки застосунку. RLS-політика визначає що всі операції (ALL) над таблицею saved_recipes дозволені лише якщо user_id запису збігається з auth.uid() - ідентифікатором поточного автентифікованого користувача з JWT-токену. Навіть якщо зловмисник отримає

анонімний ключ Supabase і спробує звернутись до бази даних напряму, RLS відфільтрує чужі рядки на рівні PostgreSQL - вони просто не потраплять у вибірку.

Захист від міжсайтових запитів (CORS) налаштовано на backend з явним переліком дозволених джерел: локальний сервер розробки http://localhost:5173 та три продуктивні домени Vercel. Запити з будь-яких інших доменів блокуються на рівні HTTP-заголовків ще до виконання бізнес-логіки. Це запобігає несанкціонованому використанню backend API сторонніми сайтами.

Валідація вхідних даних - FastAPI автоматично валідує всі вхідні дані через Pydantic-моделі ще до виконання коду ендпоінту. Поле file ендпоінту /analyze-image приймає лише файли типу UploadFile, поле language - рядок з дефолтним значенням. Модель IngredientsRequest визначає типи та значення за замовчуванням для всіх полів запиту до /get-recipes. Запити з некоректною структурою автоматично отримують відповідь HTTP 422 Unprocessable Entity без виконання будь-якої бізнес-логіки.

3.5 Розгортання системи

Система розгорнута у хмарному середовищі та складається з трьох незалежних сервісів, кожен з яких обслуговується окремою платформою. Такий підхід відповідає принципу розподілу відповідальності та дозволяє масштабувати кожен компонент незалежно.

Vercel автоматично виявляє тип проєкту як Vite та налаштовує процес збірки без додаткової конфігурації. Збірка виконується командою vite build, що генерує оптимізовані статичні файли HTML, CSS та JavaScript у директорії dist. Результат збірки розподіляється по глобальній CDN-мережі Vercel з вузлами по всьому світу, що мінімізує затримку для користувачів незалежно від їх географічного розташування. Дашборд розгорнутого frontend на платформі Vercel зображено на рис. 3.3.

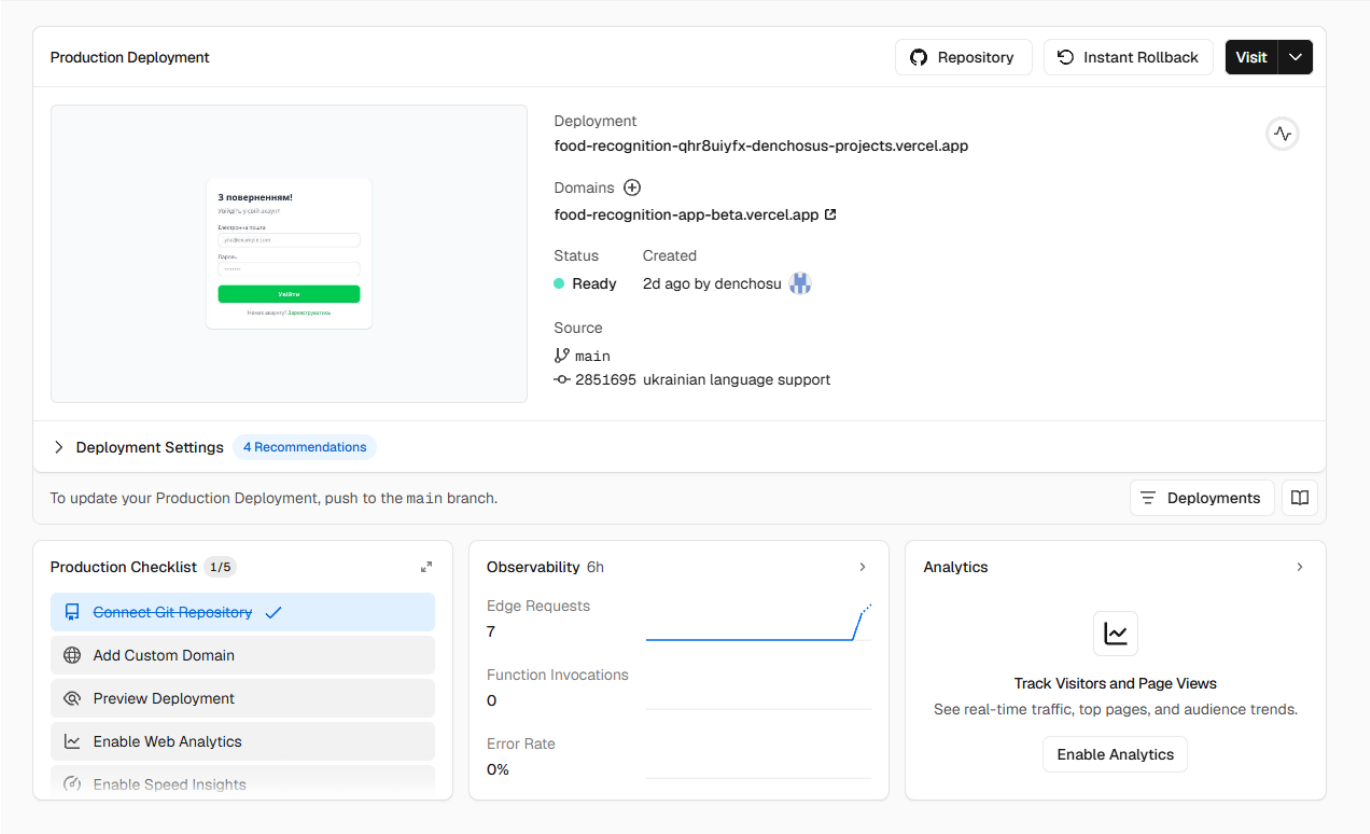


Рис. 3.5 Дашборд Vercel з розгорнутим frontend

Налаштування CI/CD виконується одноразово через підключення репозиторію GitHub до проєкту Vercel. Після цього кожен push до гілки main автоматично запускає новий процес збірки та розгортання. У разі помилки збірки попередня версія залишається активною. Змінні середовища frontend - VITE_API_URL (URL backend на Render) - задаються через панель керування Vercel і вбудовуються у збірку на етапі компіляції через механізм import.meta.env Vite.

Розгортання backend на Render. Render автоматично виявляє Python-проєкт за наявністю файлу requirements.txt у директорії backend. Сервіс запускається командою uvicorn main:app --host 0.0.0.0 --port $PORT, де змінна $PORT надається платформою Render автоматично. Залежності встановлюються з requirements.txt при кожному розгортанні, що гарантує відтворюваність середовища. Дашборд розгорнутого backend на платформі Render зображено на рис. 3.4.

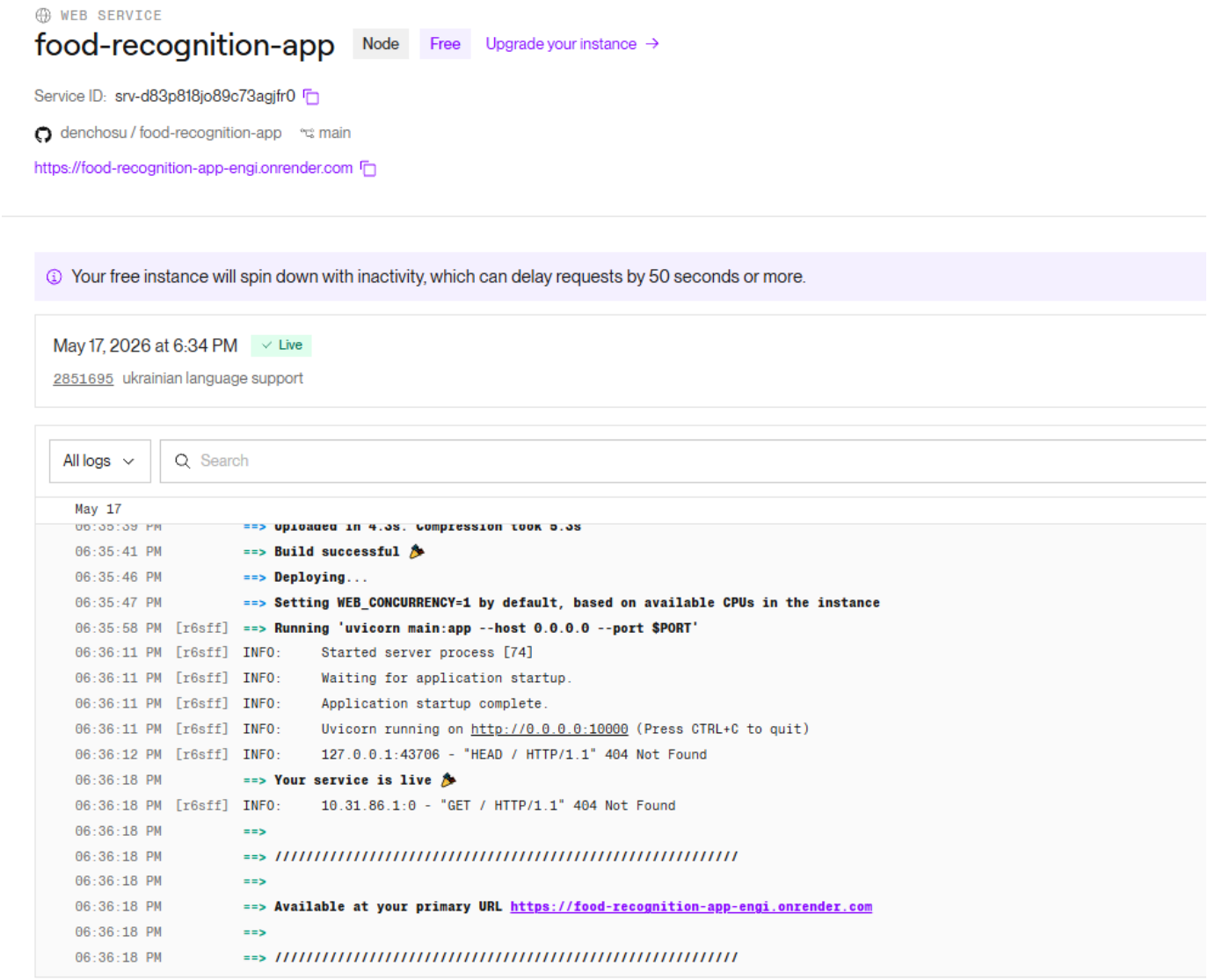


Рис. 3.6 Дашборд Render з розгорнутим backend

Секретна змінна GROQ_API_KEY задається через захищену панель керування Render і недоступна у логах або середовищі збірки. Безкоштовний тарифний план Render передбачає автоматичне "засипання" сервісу після 15 хвилин неактивності з метою економії ресурсів. Перший запит після паузи спричиняє "холодний старт" - затримку до 50 секунд на ініціалізацію контейнера. Це є відомим обмеженням безкоштовного рівня і не впливає на роботу системи при регулярному використанні.

Налаштування Supabase. База даних та система автентифікації налаштовані через веб-консоль Supabase. Таблиця saved_recipes створена через вбудований

SQL-редактор консолі виконанням скрипту наведеного у Додатку А. Правила Row Level Security активовані командою ALTER TABLE saved_recipes ENABLE ROW LEVEL SECURITY та визначена єдиною політикою для всіх операцій (ALL). Таблиця saved_recipes у консолі Supabase зображена на рис. 3.5.

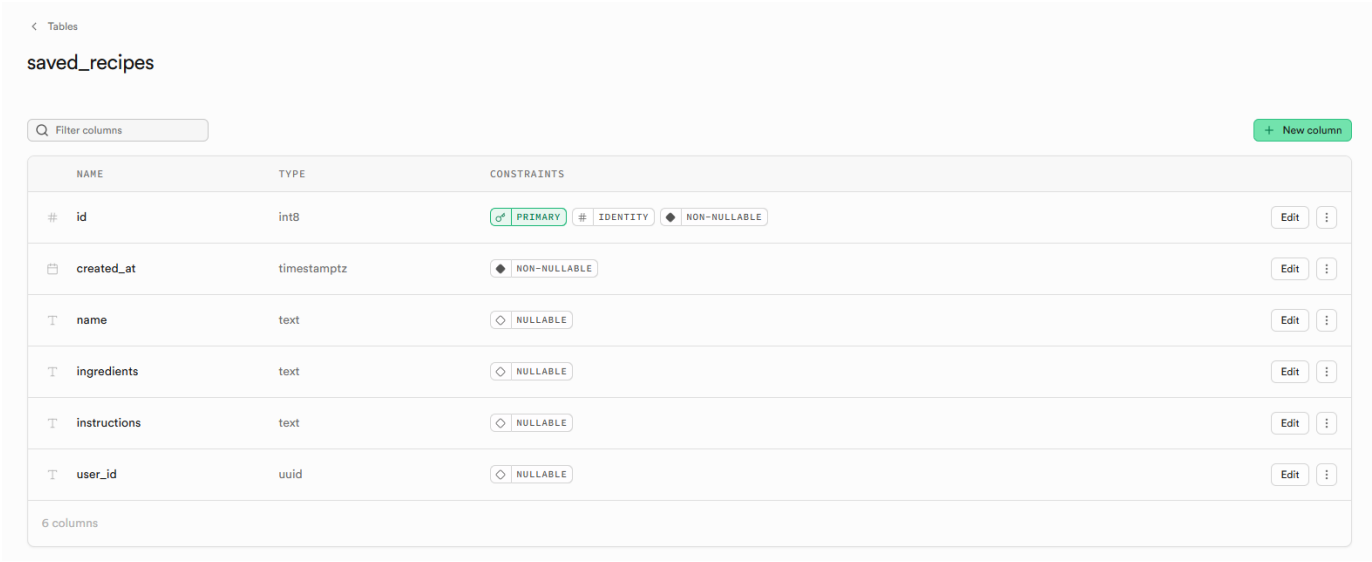


Рис. 3.7 Скріншот сторінки з таблицею saved_recipes у консолі Supabase

Підтвердження електронної пошти при реєстрації налаштоване у розділі Authentication - Email Templates консолі Supabase. URL для перенаправлення після підтвердження вказує на продуктивний домен Vercel. Для локальної розробки URL перенаправлення також включає localhost:5173 у списку дозволених redirect URLs.

Змінні середовища та конфігурація. Система використовує наступні змінні середовища: на backend - GROQ_API_KEY для доступу до Groq API; на frontend - VITE_API_URL для URL backend та публічні ключі Supabase VITE_SUPABASE_URL і VITE_SUPABASE_ANON_KEY. Публічні ключі Supabase є анонімними і безпечними для включення у клієнтський код - реальний захист даних забезпечується механізмом RLS, а не секретністю ключів.

Система контролю версій**.** Вихідний код проєкту розміщено у репозиторії GitHub. Репозиторій організований за монорепозиторним підходом - директорія frontend містить клієнтську частину, директорія backend - серверну. Інтеграція GitHub з платформами Vercel та Render забезпечує автоматичний CI/CD конвеєр -

кожен push до гілки main запускає процес збірки та розгортання без додаткової конфігурації. Репозиторій проєкту на GitHub зображено на рис. 3.6.

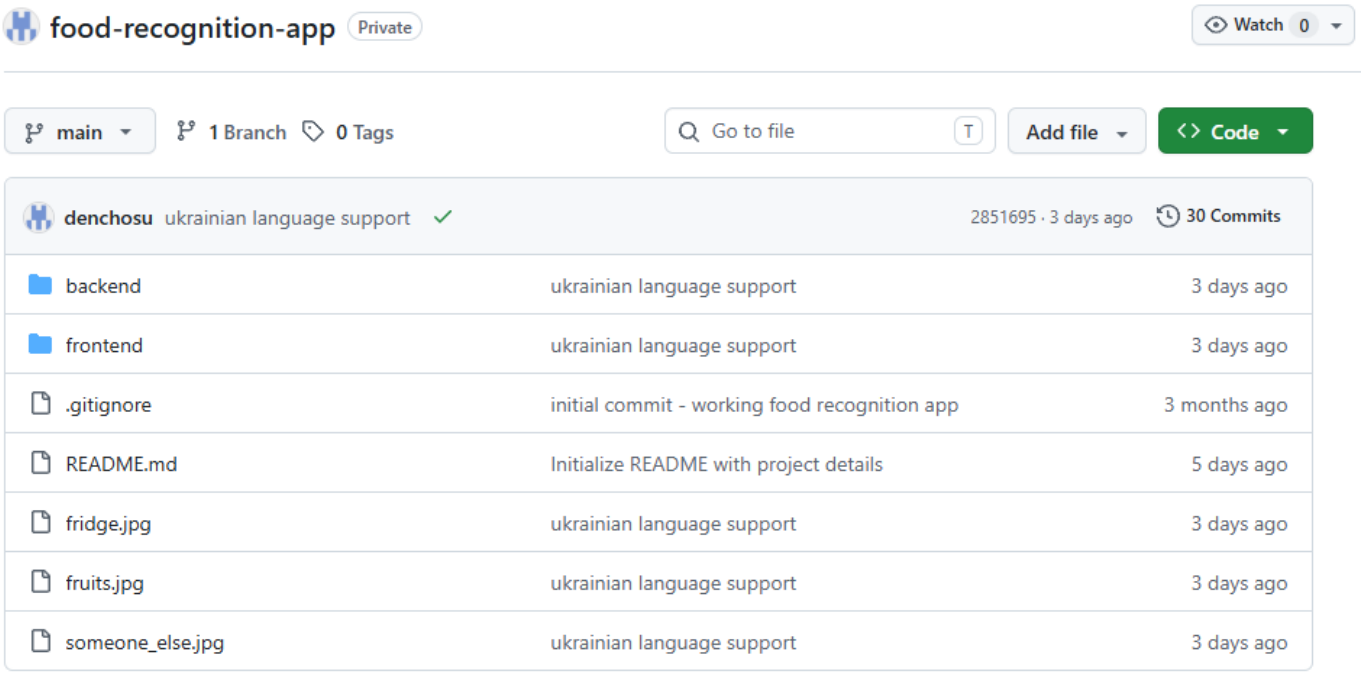


Рис. 3.8 Репозиторій проєкту на GitHub

3.6 Тестування системи

Тестування системи проводилось у кілька етапів: функціональне тестування кожного компонента окремо та інтеграційне тестування повного сценарію використання.

Функціональне тестування backend виконувалось через вбудований інтерфейс Swagger UI, що автоматично генерується FastAPI за адресою /docs. Інтерфейс Swagger UI з ендпоінтами REST API зображено на рис. 3.7.

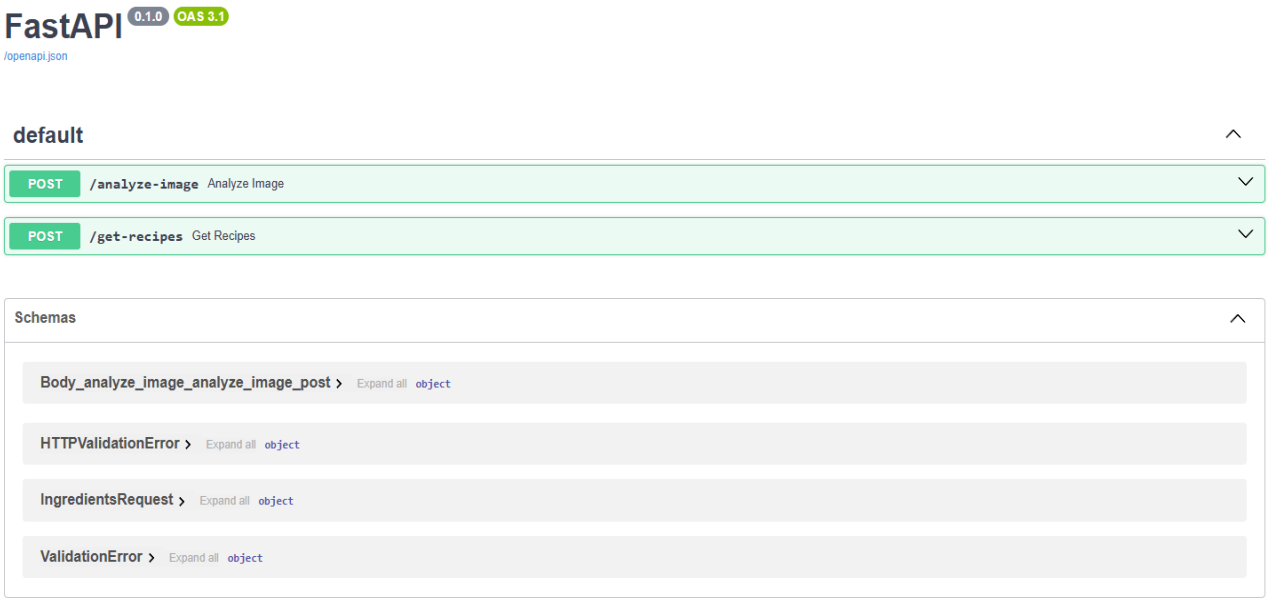


Рис. 3.9 Інтерфейс Swagger UI з ендпоінтами REST API

Для кожного ендпоінту перевірялись: коректність обробки валідних вхідних даних, повернення відповіді у очікуваному форматі JSON, обробка некоректних вхідних даних (не-зображення, порожній список інгредієнтів), коректність HTTP-кодів відповіді.

Тестування розпізнавання інгредієнтів проводилось на наборі тестових зображень різних типів, що зведено у таблиці 3.1.

Таблиця 3.1

Результати тестування розпізнавання інгредієнтів

Тип зображення	**Кількість інгредієнтів на фото**	**Розпізнано коректно**	**Хибні спрацювання**	**Результат**
Овочі на столі	6	6	0	Успішно
М'ясо та спеції	4	4	1	Успішно
Холодильник (відкритий)	12	10	2	Успішно

Продовження таблиці 3.1

Тип зображення	**Кількість інгредієнтів на фото**	**Розпізнано коректно**	**Хибні спрацювання**	**Результат**
Готова страва	5	3	0	Частково
Зображення низької якості	4	2	1	Частково
Порожнє зображення	0	0	0	Успішно

З таблиці видно, що система демонструє високу точність розпізнавання на зображеннях із сирими інгредієнтами та знижену точність на зображеннях готових страв, де окремі складники не є візуально відокремленими.

Тестування генерації рецептів проводилось на різних наборах інгредієнтів для перевірки стабільності промпту та якості результатів. У всіх тестових сценаріях модель повертала рівно три рецепти у коректному JSON-форматі з заповненими полями name, ingredients, instructions, calories та time. При стандартному наборі з 3-5 інгредієнтів система генерувала логічні та різноманітні страви різного рівня складності. При наборі з одного інгредієнта модель коректно доповнювала рецепти базовими продуктами кухні відповідно до інструкції у промпті. При великому наборі з 15 і більше інгредієнтів кожен з трьох рецептів використовував різні основні інгредієнти зі списку, що відповідає четвертому правилу промпту. Єдиним сценарієм з частковою відповідністю правилам виявились набори з кулінарно несумісних продуктів - модель у більшості випадків коректно ігнорувала непідходящі інгредієнти, однак у поодиноких тестах все ж включала їх до одного рецепту.

Тестування автентифікації перевіряло такі сценарії: реєстрація з валідною електронною адресою та паролем, реєстрація з паролем менше шести символів (очікувана помилка Supabase), вхід з коректними даними, вхід з некоректним паролем, автоматичне відновлення сесії після перезавантаження сторінки, коректний вихід із системи.

Таблиця 3.2

Результати тестування автентифікації

Сценарій	**Очікуваний результат**	**Фактичний результат**	**Статус**
Реєстрація з валідними даними	Повідомлення про підтвердження email	Повідомлення відображено	Успішно
Реєстрація з паролем менше 6 символів	Повідомлення про помилку	Повідомлення відображено	Успішно
Реєстрація з уже зареєстрованим email	Повідомлення про дублікат	Повідомлення відображено	Успішно
Вхід з коректними даними	Перехід до головного екрану	Виконано	Успішно
Вхід з невірним паролем	Повідомлення про помилку	Повідомлення відображено	Успішно
Вхід з непідтвердженим email	Повідомлення про непідтверджений email	Повідомлення відображено	Успішно
Вихід із системи	Перехід до екрану автентифікації	Виконано	Успішно

Тестування безпеки на рівні даних підтверджувало коректність роботи RLS-правил: спроба отримати рецепти іншого користувача через прямий SQL-запит з неправильним токеном повертала порожній масив, а не помилку доступу (що є коректною поведінкою PostgreSQL RLS - рядки просто не потрапляють у вибірку, а не блокуються явною помилкою).

Тестування продуктивності вимірювало час відповіді системи за типових умов. Середній час розпізнавання одного зображення через Groq API становив 2-4 секунди. Генерація трьох рецептів займала 3-6 секунд. Таким чином, повний цикл від завантаження зображення до отримання рецептів займає орієнтовно 5-10 секунд, що відповідає встановленій нефункціональній вимозі.

У третьому розділі описано практичну реалізацію всіх компонентів системи. Серверна частина на базі FastAPI реалізує двоетапний конвеєр обробки: розпізнавання інгредієнтів через мультимодальну модель Llama 4 Scout та генерацію рецептів за сформованим списком. Клієнтська частина на React забезпечує інтуїтивний інтерфейс з підтримкою завантаження кількох зображень, редагування інгредієнтів та збереження рецептів. Система успішно розгорнута у хмарному середовищі з використанням Vercel, Render та Supabase. Проведене тестування підтвердило коректність роботи всіх функціональних вимог та відповідність нефункціональним вимогам щодо продуктивності та безпеки. Реалізовано комплексний захист системи на кількох рівнях: захист API-ключів через змінні середовища, автентифікація через JWT-токени, ізоляція даних через Row Level Security та обмеження CORS.

ВИСНОВКИ

У кваліфікаційній роботі вирішено науково-практичне завдання розробки веб-додатку з розпізнаванням харчових продуктів та підбором рецептів на основі штучного інтелекту. За результатами виконаної роботи можна зробити такі висновки.

1. Проведений аналіз існуючих рішень у сфері розпізнавання харчових продуктів та підбору рецептів показав, що жоден з розглянутих публічних сервісів - Yummly, Supercook, Google Lens, Im2Recipe - не реалізує повний цикл «фото, інгредієнти, рецепти» з підтримкою персоналізації та збереження результатів. Це підтвердило актуальність та доцільність розробки власного рішення.

2. Досліджено еволюцію методів комп'ютерного зору - від класичних CNN-архітектур (ResNet, InceptionV3) до сучасних мультимодальних великих мовних моделей. Встановлено, що мультимодальні LLM, зокрема Llama 4 Scout від Meta AI, є оптимальним вибором для задачі розпізнавання довільних побутових інгредієнтів, оскільки не потребують спеціалізованих навчальних датасетів та здатні обробляти відкриту множину категорій продуктів.

3. Спроєктовано трирівневу клієнт-серверну архітектуру веб-додатку, що включає React-застосунок на рівні представлення, FastAPI-сервер на рівні бізнес-логіки та Supabase-платформу на рівні даних. Розроблено схему бази даних з двома сутностями та реалізовано механізм Row Level Security для забезпечення ізоляції даних між користувачами на рівні СУБД.

4. Розроблено серверну частину на базі FastAPI з двоетапним конвеєром обробки: на першому етапі мультимодальна модель аналізує зображення та повертає JSON зі списком інгредієнтів, на другому - генерує три рецепти на основі отриманого списку. Реалізовано промпт-інжиніринг для отримання детермінованих JSON-відповідей від моделі без зайвого тексту. Зображення обробляються послідовно з об'єднанням результатів перед генерацією рецептів, що забезпечує стабільну роботу системи без перевантаження зовнішнього API.

5. Розроблено клієнтську частину на базі React з підтримкою завантаження та попереднього перегляду кількох зображень, редагування списку інгредієнтів, відображення згенерованих рецептів, автентифікації користувачів через Supabase Auth та збереження рецептів до особистого кабінету. Конвертація зображень у формат JPEG засобами Canvas API перед відправкою на сервер забезпечує сумісність з різними вхідними форматами та зменшує обсяг переданих даних.

6. Систему успішно розгорнуто у хмарному середовищі: frontend - на платформі Vercel з автоматичним CI/CD при оновленні репозиторію, backend - на платформі Render, база даних та автентифікація - на платформі Supabase. Проведене тестування підтвердило коректність роботи всіх функціональних вимог. Середній час повного циклу обробки від завантаження зображення до отримання рецептів становить 5-10 секунд, що відповідає встановленій нефункціональній вимозі продуктивності. Точність розпізнавання сирих інгредієнтів на зображеннях прийнятної якості становить близько 85-90%.

Практична цінність розробленого веб-додатку полягає в тому, що він є повністю функціональним, розгорнутим у хмарному середовищі продуктом, готовим до реального використання. Запропонований підхід до інтеграції мультимодальних LLM у веб-застосунок може бути використаний як основа для розробки аналогічних систем в інших предметних галузях, де потрібне поєднання аналізу зображень та генерації текстового контенту.

ПЕРЕЛІК ПОСИЛАНЬ

1. Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A. N., Kaiser Ł., Polosukhin I. Attention Is All You Need. *Advances in Neural Information Processing Systems 30 (NeurIPS 2017)*. 2017. arXiv:1706.03762. URL: https://arxiv.org/abs/1706.03762

2. Krizhevsky A., Sutskever I., Hinton G. E. ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems 25 (NeurIPS 2012)*. 2012. С. 1097–1105. URL: https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks

3. He K., Zhang X., Ren S., Sun J. Deep Residual Learning for Image Recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2016)*. Лас-Вегас, 2016. С. 770–778. URL: https://openaccess.thecvf.com/content_cvpr_2016/html/He_Deep_Residual_Learning_CVPR_2016_paper.html

4. Bossard L., Guillaumin M., Van Gool L. Food-101 – Mining Discriminative Components with Random Forests. *Computer Vision – ECCV 2014. Lecture Notes in Computer Science*. Т. 8694. Springer, Cham, 2014. С. 446–461. DOI: https://doi.org/10.1007/978-3-319-10599-4_29

5. Dosovitskiy A., Beyer L., Kolesnikov A., Weissenborn D., Zhai X., Unterthiner T., Houlsby N. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *International Conference on Learning Representations (ICLR 2021)*. 2021. arXiv:2010.11929. URL: https://arxiv.org/abs/2010.11929

6. Meta AI. The Llama 4 herd: The beginning of a new era of natively multimodal AI innovation. *Meta AI Blog*. 2025. URL: https://ai.meta.com/blog/llama-4-multimodal-intelligence/

7. Sahoo P., Singh A. K., Sahu S., Jain V., Mondal S., Chadha A. A Systematic Survey of Prompt Engineering in Large Language Models: Techniques and Applications. *arXiv preprint*. 2024. arXiv:2402.07927. URL: https://arxiv.org/abs/2402.07927

8. Agarwal M., Choudhary S. A Survey of Prompt Engineering Methods in Large Language Models for Different NLP Tasks. *arXiv preprint*. 2024. arXiv:2407.12994. URL: https://arxiv.org/abs/2407.12994

9. React. The library for web and native user interfaces : офіц. документація. 2024. URL: https://react.dev/

10. Groq. GroqCloud Developer Console : офіц. документація. 2024. URL: https://console.groq.com/docs/openai

11. Liang C. X., Tian P., Yin C. H. та ін. A Comprehensive Survey and Guide to Multimodal Large Language Models in Vision-Language Tasks. *arXiv preprint*. 2024. arXiv:2411.06284. URL: https://arxiv.org/abs/2411.06284

12. Supabase. The Postgres Development Platform : офіц. документація. 2024. URL: https://supabase.com/docs

13. Min W. та ін. Deep Learning in Food Image Recognition: A Comprehensive Review. *Applied Sciences*. 2025. Т. 15, № 14. DOI: https://doi.org/10.3390/app15147626

14. Vercel. Vercel Documentation : офіц. документація. 2024. URL: https://vercel.com/docs

15. Tailwind CSS. A utility-first CSS framework : офіц. документація. 2024. URL: https://tailwindcss.com/docs

16. Minaee S., Mikolov T., Nikzad N., Chenaghlu M., Socher R., Amatriain X., Gao J. Large Language Models: A Survey. *arXiv preprint*. 2024. arXiv:2402.06196. URL: https://arxiv.org/abs/2402.06196

17. Yin S., Fu C., Zhao S., Li K., Sun X., Xu T., Chen E. A Survey on Multimodal Large Language Models. *National Science Review*. 2024. Т. 11, № 12. nwae403. DOI: https://doi.org/10.1093/nsr/nwae403

18. Bucko A., Vishi K., Krasniqi B., Rexha B. Enhancing JWT Authentication and Authorization in Web Applications Based on User Behavior History. *Computers*. 2023. Т. 12, № 4. DOI: https://doi.org/10.3390/computers12040078

19. Dalimunthe S., Hasri Putra E., Fadhly Ridha M. A. Restful API Security Using JSON Web Token (JWT) With HMAC-Sha512 Algorithm in Session

Management. *IT Journal Research and Development*. 2023. Т. 8, № 1. С. 81–94. DOI: https://doi.org/10.25299/itjrd.2023.12029

20. Jonathan R., Suprihadi. Development of Front-End Web Applications Utilizing Single Page Application Framework and React.js Library. *International Journal Software Engineering and Computer Science*. 2023. Т. 3, № 3. С. 529–536. DOI: https://doi.org/10.35870/ijsecs.v3i3.1943

21. Iqbal M. та ін. Food Detection and Recognition with Deep Learning: *A Comparative Study. IEEE Conference Publication. 2023. DOI:* https://ieeexplore.ieee.org/document/10262523

22. United Nations Environment Programme (UNEP). *Food Waste Index Report 2021*. Найробі, 2021. URL: https://www.unep.org/resources/report/unep-food-waste-index-report-2021

23. Min W. та ін. Large Scale Visual Food Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2023. arXiv:2103.16107. URL: https://arxiv.org/abs/2103.16107

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

(Презентація)

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

«Веб-додаток з розпізнаванням продуктів та підбором рецептів на основі штучного інтелекту»

Виконав студент 4 курсу
групи ІСД-42
Угрімов Денис Володимирович
Керівник роботи:
Ткаленко Оксана Миколаївна

Київ - 2026

МЕТА ТА ЗАВДАННЯ

Мета:

Покращення процесу розпізнавання продуктів та підбору рецептів на основі ШІ

Об’єкт:

Автоматизоване розпізнавання харчових продуктів та генерація персоналізованих рецептів

Предмет:

Методи розробки веб-додатку з інтеграцією AI-моделей для розпізнавання інгредієнтів

Завдання:

Розробити веб додаток на основі штучного інтелекту

ТЕОРЕТИЧНА БАЗА: ЕВОЛЮЦІЯ МЕТОДІВ

CNN Згорткові нейронні мережі класифікують зображення за навченими категоріями, але вимагають великих розмічених датасетів і обмежені фіксованим набором класів.

Transformer Архітектура 2017 року на основі механізму самоуваги - дозволяє моделі враховувати залежності між будь-якими елементами послідовності незалежно від відстані між ними.

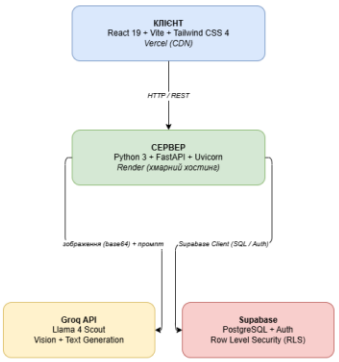
Multimodal LLM Моделі на кшталт Llama 4 Scout обробляють одночасно текст і зображення без спеціалізованого навчання - вирішують задачу розпізнавання на відкритій множині категорій.

ПРОМПТ-ІНЖИНІРИНГ ТА АНАЛІЗ КОНКУРЕНТІВ

Модель отримує роль шеф-кухаря, запит без прикладів і повертає строго JSON

Рішення	Розпізнавання з фото	Генерація рецептів	Відкритий API	Автентифікація
Yummly	Ні	Ні (база даних)	Частково	Так
Supercook	Ні	Ні (база даних)	Ні	Ні
Google Lens	Так	Ні	Ні	Так
Im2Recipe	Так	Так	Ні	Ні
Click&Cook	Так	Так	Так	Так

АРХІТЕКТУРА СИСТЕМИ



Frontend

React на Vercel для UI та роутингу.

AI & Data

Groq API з Llama 4 Scout та Supabase PostgreSQL для даних аутентифікації.

Backend

FastAPI на Render

ТЕХНОЛОГІЧНИЙ СТЕК

React 19

SPA, найбільша екосистема

FastAPI

Async, автовалідація, OpenAPI

Supabase

PostgreSQL + Auth + RLS

Groq API

LPU, Llama 4 Scout

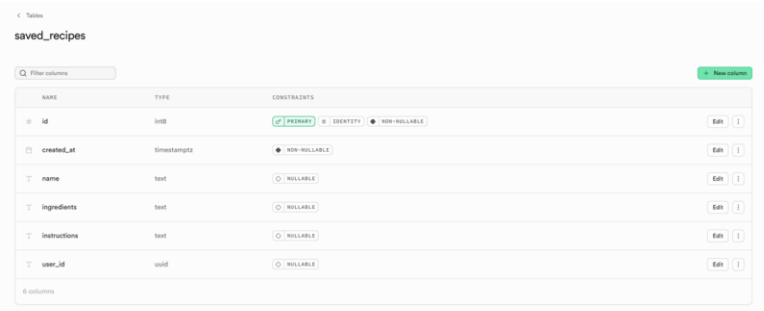
Vercel + Render

CI/CD, CDN, безкоштовний план

Tailwind CSS 4

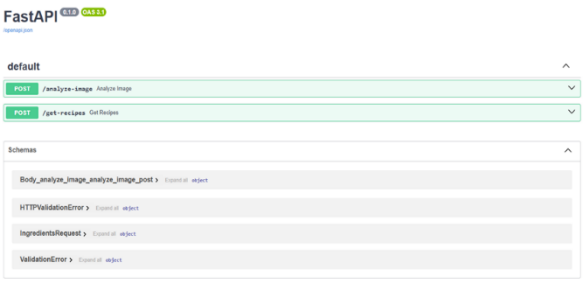
Утилітарний підхід, адаптивність

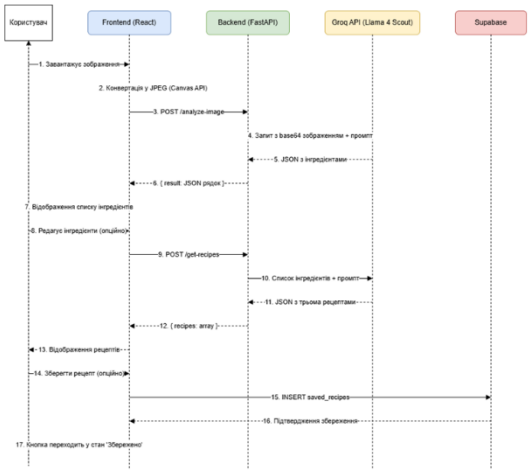
БАЗА ДАНИХ ТА REST API



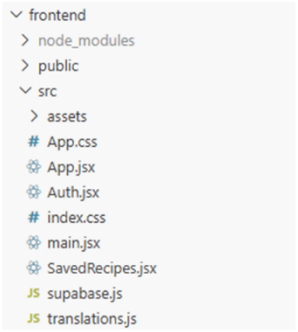
Метод	Ендпоінт	Вхідні дані	Відповідь	Опис
POST	/analyze-image	multipart/form-data: file (image), language (string)	JSON: { result: string }	Розпізнавання інгредієнтів на зображенні
POST	/get-recipes	JSON: { ingredients: string, language: string, priority_ingredients }	JSON: { recipes: array }	Генерація рецептів за списком інгредієнтів

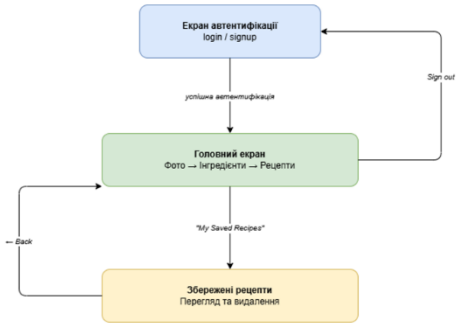
РЕАЛІЗАЦІЯ BACKEND - FASTAPI





РЕАЛІЗАЦІЯ FRONTEND - REACT





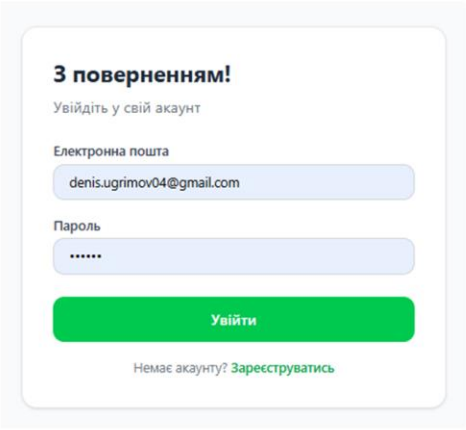
Компонентна структура

App - маршрутизація за станом auth
Auth - вхід / реєстрація
SavedRecipes - збережені рецепти

Особливості

Canvas API: конвертація у JPEG
Послідовна обробка for...of
Tailwind CSS 4, локалізація UA/EN

АВТЕНТИФІКАЦІЯ ТА ОСОБИСТИЙ КАБІНЕТ



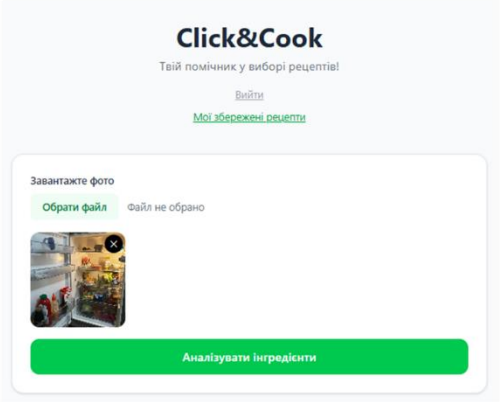
Supabase Auth

1. JWT-токени, підтвердження email
2. bcrypt-хешування паролів
3. Локалізовані повідомлення

Row Level Security

Ізоляція даних на рівні PostgreSQL - кожен користувач бачить лише власні рецепти

CLICK&COOK: ІНТЕРФЕЙС



Завантаження фото, автоматичне розпізнавання інгредієнтів через Llama 4 Scout

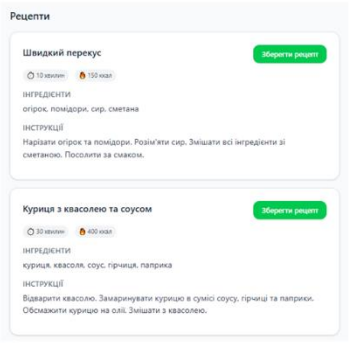
Завантаження

Множинний вибір фото,
попередній перегляд

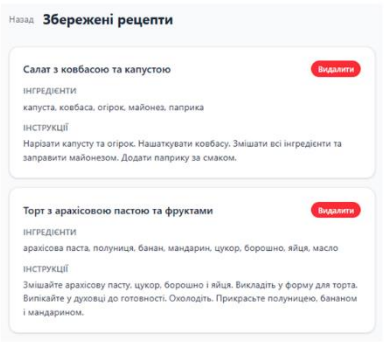
Редагування

Теги з кнопками видалення, ручне додавання

ІНТЕРФЕЙС - РЕЦЕПТИ ТА ЗБЕРЕЖЕНЕ



Назва, калорійність, час, інгредієнти, покрокова інструкція



3 рецепти різної складності: перекус (до 15 хв), середня (20–40 хв), складна (40+ хв)

РОЗГОРТАННЯ ТА ТЕСТУВАННЯ

Тип зображення	Кількість інгредієнтів на фото	Розпізнано коректно	Хибні спрацювання	Результат
Овочі на столі	6	6	0	Успішно
М'ясо та спеції	4	4	1	Успішно
Холодильник (відкритий)	12	10	2	Успішно
Готова страва	5	3	0	Частково
Зображення низької якості	4	2	1	Частково
Порожнє зображення	0	0	0	Успішно

ВИСНОВКИ

1. Заповнено нішу ринку

Жоден сервіс не реалізує повний цикл «фото, інгредієнти, рецепти»

2. Працюючий продукт у хмарі

Vercel + Render + Supabase, CI/CD, точність 85–90%

3. Мультимодальна LLM без датасетів

Llama 4 Scout - довільні інгредієнти, відкрита множина категорій

4. Практична цінність

Готовий до використання, основа для інших предметних галузей

ДЯКУЮ ЗА УВАГУ!