

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)
на тему:

«Платформа для пошуку страв»

Виконав студент групи ІПЗп-21-1
БЛОХА Ярослав Сергійович

Керівник:
ЄФРЕМОВ Юрій Миколайович

Рецензент:
ЧИЖМОТРА Олексій Володимирович

Житомир – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення
Тетяна Вакалюк
«01» серпня 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **БЛОХА Ярослав Сергійович**

Керівник роботи: **ЄФРЕМОВ Юрій Миколайович**

Тема роботи: **«Платформа для пошуку страв»,**

затверджена Наказом закладу вищої освіти від **01.08.2025р., №270/с**

Об'єктом дослідження є використання інформаційних технологій для автоматизації процесів пошуку страв.

Предметом дослідження є застосування технологій розробки вебзастосунків для створення платформи для пошуку страв.

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Єфремов Ю.М.	02.08.2025	02.08.2025
2	Єфремов Ю.М.	02.08.2025	02.08.2025
3	Єфремов Ю.М.	02.08.2025	02.08.2025

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з веборієнтованої платформи для пошуку страв та пояснювальної записки. Пояснювальна записка до кваліфікаційної роботи містить 58 сторінок, 34 ілюстрацій, 29 таблиць, 1 додаток та список із 31 використаного джерела. Загальний обсяг – 89 сторінок.

Метою роботи є проектування та розробка платформи для пошуку страв з використанням сучасних вебтехнологій та штучного інтелекту для надання рекомендацій. Платформа дозволяє користувачам швидко переглядати страви та отримувати поради від ШІ, щодо приготування страв.

У роботі було поставлено основні завдання на розробку платформи. Було проведено аналіз аналогічних платформ, визначено їхні переваги та недоліки. Обрано клієнт-серверну архітектуру, де серверна частина реалізована на базі Node.js з використанням Supabase, а клієнтська частина – за допомогою фреймворку Next.js та обґрунтовано вибір інструментів для розробки.

КЛЮЧОВІ СЛОВА: ВЕБЗАСТОСУНОК, ПЛАТФОРМА, ПОШУК СТРАВ, АНАЛІЗ ХАРЧОВОЇ ЦІННОСТІ, АДМІНІСТРУВАННЯ, ШТУЧНИЙ ІНТЕЛЕКТ.

					ІПЗ.КР.Б – 121 – 25 – ПЗ				
Змн.	Арк.	№ докум.	Підпис	Дата					
Розроб.		Блоха Я.С.			Платформа для пошуку страв	Лім.	Арк.	Аркушів	
Керівник		Єфремов Ю.М.					3	89	
						Житомирська політехніка, група ІПЗп-21-1			
Рецензент		Чижмоторя О.В.							
Зав. каф.		Вакалюк Т.А.							

ABSTRACT

The final bachelor's thesis consists of a web-based platform for finding dishes and an explanatory note. The explanatory note to the qualification work contains 58 pages, 34 illustrations, 29 tables, 31 references. The total volume is 89 pages.

The aim of the work is to design and develop a platform for searching for dishes using modern web technologies and artificial intelligence to provide recommendations. The platform allows users to quickly browse dishes and get advice from AI on how to prepare them.

The paper sets the main tasks for the development of the platform. We analyzed similar platforms and identified their advantages and disadvantages. A client-server architecture has been chosen, where the server side is implemented on the basis of Node.js using Supabase, and the client side is implemented using the Next.js framework, and the choice of development tools has been substantiated.

KEYWORDS: WEB APPLICATION, PLATFORM, FOOD SEARCH, NUTRITIONAL VALUE ANALYSIS, ADMINISTRATION, ARTIFICIAL INTELLIGENCE.

ЗМІСТ

РЕФЕРАТ	3
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	6
ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ ПЛАТФОРМИ ДЛЯ ПОШУКУ СТРАВ	9
1.1. Постановка задачі	9
1.2. Аналіз аналогів програмного продукту.....	11
1.3. Вибір архітектури платформи для пошуку страв.....	16
1.4. Обґрунтування вибору інструментальних засобів	17
Висновки до першого розділу	20
РОЗДІЛ 2. ПРОЄКТУВАННЯ ПЛАТФОРМИ ДЛЯ ПОШУКУ СТРАВ.....	22
2.1. Визначення варіантів використання та структури платформи	22
2.2. Розробка бази даних	29
2.3. Проектування та реалізація алгоритмів роботи платформи.....	34
2.4. Реалізація платформи для пошуку страв.....	36
Висновки до другого розділу.....	39
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ПЛАТФОРМОЮ ДЛЯ ПОШУКУ СТРАВ	40
3.1. Порядок встановлення та налаштування системи.....	40
3.2. Структура інтерфейсу та порядок взаємодії з платформою для пошуку страв	41
3.3. Тестування роботи програмного продукту	50
Висновки до третього розділу	53
ВИСНОВКИ.....	54
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	55
ДОДАТКИ.....	58

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API - Application Programming Interface.

БД - база даних.

JSON - JavaScript Object Notation.

JWT - JSON Web Token.

CRUD – Create, Read, Update, Delete.

REST - Representational State Transfer.

HTTP – Hyper Text Transfer Protocol.

URL - Uniform Resource Locator.

CORS - Cross-Origin Resource Sharing.

SPA - Single Page Application.

ШІ – Штучний інтелект.

ВСТУП

Сучасний світ відомий різкими темпами розвитку інформаційних технічних систем та їх розширенням на усі сфери людської діяльності. Кулінарія як одна з найдавніших і найбільш важливої галузей людської культури не стоїть осторонь цих процесів. Сучасні тенденції охорони здоров'я та свідомого харчування також мають свій відбиток на розвиток кулінарії. Користувачі все більше потребують детальної інформації про харчову цінність продуктів, калорійність страв і можливість їх індивідуального налаштування під власні потреби.

Актуальність теми визначається спектром невирішених проблем у сфері харчової діяльності та застосування інформаційних технологій для оптимізації процесів пошуку, аналізу та приготування страв. Нові напрямки використання інформаційних технологій у кулінарії охоплюють різні галузі: від особистого планування харчування і керування дієтами — до ресторанного бізнесу та організації масового харчування.

Незважаючи на значні досягнення в галузі, все ще існує ряд важливих технічних та функціональних проблем: недостатня інтеграція AI-технологій із базами даних харчових продуктів, відсутність цілісного підходу до аналізу харчової цінності та малоефективні алгоритми пошуку страв за наявними інгредієнтами.

Метою роботи є проєктування й розробка платформи для пошуку страв, яка зможе аналізувати харчову цінність продуктів і підбирати рецепти з використанням штучного інтелекту на основі доступних інгредієнтів.

Для досягнення цієї мети передбачено виконання таких завдань:

1. Провести аналіз існуючих рішень і сервісів, оцінити їх функціонал, архітектуру та користувацькі інтерфейси.
2. Визначити й обґрунтувати вибір технологій для реалізації клієнтської і серверної частин платформи, а також бази даних.
3. Сформулювати функціональні й нефункціональні вимоги до системи, створити сценарії її використання відповідно до ролей користувачів.

4. Розробити та реалізувати базу даних для ефективної обробки й зберігання інформації про страви, інгредієнти, користувачів, коментарі, рейтинги, ШП-чати.
5. Створити зручний, адаптивний та зрозумілий інтерфейс з функціоналом пошуку, фільтрації, персональними колекціями, ШП-чатом, а також окремий інтерфейс для адміністратора з системою управління контентом і модерацією.
6. Провести тестування клієнтської та серверної частин платформи.

Об'єктом дослідження є процес пошуку страв і обробки кулінарної інформації з використанням вебтехнологій та ШП, а також інтеграція із зовнішніми сервісами.

Предметом дослідження є платформа, яка забезпечує інтерактивну взаємодію з користувачами та автоматизує процес підбору страв за допомогою штучного інтелекту.

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ ПЛАТФОРМИ ДЛЯ ПОШУКУ СТРАВ

1.1. Постановка задачі

Мета роботи — розробка платформи для пошуку страв із використанням сучасних веб технологій та інтеграцією штучного інтелекту для надання персоналізованих рекомендацій щодо приготування страв на основі наявних інгредієнтів. Система спрямована на створення зручного та функціонального інструменту як для початківців, так і для досвідчених кулінарів, надаючи можливість не тільки знаходити страви, а й ділитися власним кулінарним досвідом та здійснювати аналіз харчової цінності приготовлених страв.

Першим модулем є розробка системи управління користувачами, яка включає модуль реєстрації та авторизації з підтримкою різних рівнів доступу. Система повинна забезпечувати реєстрацію нових користувачів з верифікацією електронної пошти, надійну авторизацію з можливістю відновлення паролю, а також розподіл ролей між звичайними користувачами й адміністраторами платформи.

Наступним модулем платформи є розробка інтуїтивного та функціонального інтерфейсу для роботи зі стравами, який дозволить користувачам не тільки переглядати існуючий контент, а й активно взаємодіяти з ним. Система повинна забезпечувати можливість детального перегляду страв з покроковими інструкціями та фотографіями етапів приготування.

Функціонал взаємодії включає систему оцінювання страв, можливість залишати коментарі та відгуки. Користувачі повинні мати можливість додавати власні страви завантажуючи фотографії, що сприятиме створенню активної спільноти кулінарів.

Ключовим модулем платформи є розробка інтелектуальної системи рекомендацій на базі сучасних технологій штучного інтелекту, зокрема великих мовних моделей типу Gemini. Система має аналізувати наявні у користувача інгредієнти та пропонувати оптимальні страви для їх

використання, враховуючи індивідуальні переваги, дієтичні обмеження та рівень кулінарної майстерності.

III-асистент повинен забезпечувати інтерактивну взаємодію з користувачами через чат, надаючи поради щодо приготування страв і пропозиції щодо заміни інгредієнтів. Система повинна зберігати історію спілкування для покращення якості майбутніх рекомендацій і забезпечення цілісності діалогу.

Платформа має включати систему пошуку та фільтрації страв за різними критеріями, як-от назву страви, тип кухні (італійська, азійська, українська тощо).

Компонент управління колекціями страв дозволить користувачам організовувати знайдені та власні страви за категоріями, планувати меню на тиждень або місяць і створювати списки покупок на основі обраних страв. Система підтримує створення тематичних колекцій, наприклад, "Святкові страви", "Швидкі обіди" і "Дієтичні рецепти".

Аналіз харчової цінності продуктів є критично важливим компонентом, що досягається через інтеграцію з міжнародними API харчових баз даних, такими як Edamam. Необхідно розраховувати калорійність страв, вміст білків, жирів, вуглеводів, вітамінів та мінералів на основі інгредієнтів та їх кількості у рецептах.

Панель адміністратора забезпечує управління платформою та включає інструменти для модерації користувацького контенту, управління стравами, моніторингу активності користувачів і аналізу статистичних даних.

Реалізація поставлених завдань передбачає поетапний підхід, що включає початкову фазу аналізу та проектування, розробку базової функціональності, інтеграцію III-компонентів, тестування та оптимізацію системи. Кінцевим результатом має стати вебплатформа, яка демонструє поєднання сучасних вебтехнологій зі штучним інтелектом для створення інноваційного рішення в сфері кулінарії.

Успішна реалізація проєкту забезпечить створення продукту, який може служити основою для комерційного впровадження та подальшого розвитку в напрямку створення комплексної екосистеми кулінарних сервісів.

1.2. Аналіз аналогів програмного продукту

Розробка платформи для пошуку рецептів потребує детального аналізу вже існуючих рішень на ринку. Сучасний ринок кулінарних сервісів вирізняється високою конкуренцією та різними підходами до пошуку, організації й рекомендації рецептів. Щоб визначити найкращий функціонал і знайти слабкі сторони поточних рішень, було проведено комплексний аналіз платформ, які охоплюють різні сегменти ринку кулінарних додатків.

Основна мета аналізу полягає у визначенні можливостей кулінарних платформ, визначити їхні сильні й слабкі сторони, а також виявити потенціал для інновацій та подальшого розвитку. Особливу увагу приділено користувацькому досвіду, технічним рішенням, бізнес-моделям та можливостям інтеграції.

Одним з таких пратформ є YABPOELA (рис. 1.1).

YABPOELA [1] це класична кулінарна платформа, орієнтована на простоту використання та доступність контенту. Вона зосереджена на базових функціях: перегляд, додавання та обговорення рецептів, без складних технологічних рішень.

До функціоналу платформи входить каталог рецептів із розподілом за категоріями, типами страв і кухнями, детальні описи рецептів з покроковими інструкціями, можливість коментування, а також додавання власних рецептів. Також доступні базові фільтри пошуку та система тегів для зручної навігації.

Переваги платформи включають безкоштовний доступ до всього контенту без підписок, докладні рецепти з чіткими інструкціями, інтуїтивно зрозумілий інтерфейс. Додатково варто відзначити логічну структуру контенту, сортування страв за категоріями, мінімалістичний дизайн, що сприяє швидкому завантаженню, і стабільну роботу без технічних проблем.

До недоліків можна віднести відсутність можливості створювати персональні добірки страв, брак функцій ШІ для підказок на основі наявних

інгредієнтів, а також неможливість зберігати улюблені рецепти для швидкого доступу пізніше.

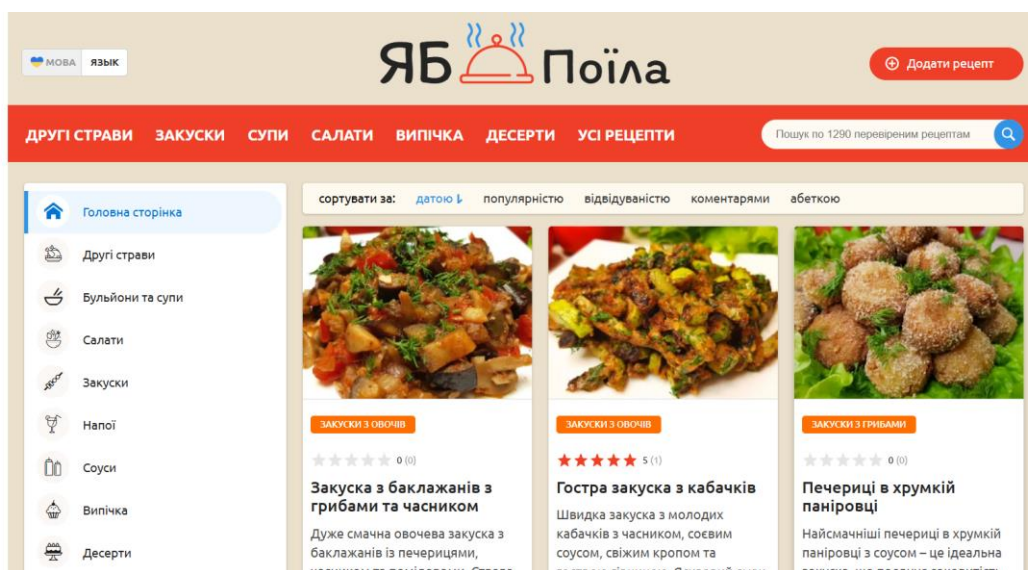


Рис.. 1.1 - YABPOELA

Cookorama.net [2] (рис. 1.2) – це кулінарна платформа, яка об'єднує контент як від звичайних користувачів, так і від професійних кухарів. Вона пропонує ширший функціонал порівняно з базовими платформами, зокрема систему рейтингів і підтримку кількох мов.

Серед переваг варто виділити безкоштовний доступ до основних функцій, наявність детальних рецептів з покроковими інструкціями та фото. Зручна система категорій дозволяє легко орієнтуватися у контенті, а система рейтингів допомагає оцінювати якість рецептів. Завдяки участі професійних кухарів контент платформи є різноманітним і цікавим.

До недоліків належать застарілий інтерфейс, що не відповідає сучасним візуальним стандартам, відсутність аналізу харчової цінності — зокрема даних про білки, жири та вуглеводи, а також повільне завантаження сторінок, спричинене неоптимізованим кодом і великими зображеннями. Крім того, немає інтеграції з ШІ для автоматичних рекомендацій на основі наявних інгредієнтів.

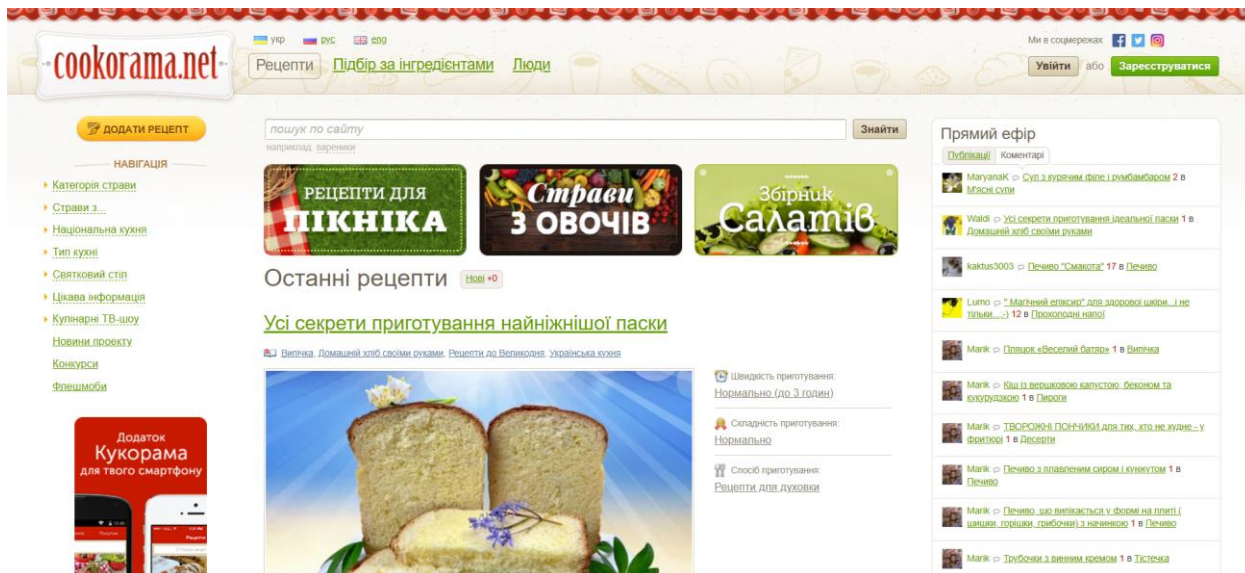


Рис. 1.2 – Cookorama.net

Cookpad [3] (рис. 1.3) позиціонує себе як соціальна мережа для кулінарів, де користувачі можуть не лише шукати рецепти, а й активно ділитися власним досвідом приготування страв. Основний акцент платформи — на взаємодії між користувачами та формуванні кулінарної спільноти.

До функціоналу платформи входять персональні профілі користувачів, можливість підписки на улюблених авторів рецептів, система коментарів і відгуків з фото результатів. Також доступні інструменти для створення та редагування рецептів, система тегів і категорій для зручної організації контенту, а ще функція ведення кулінарного блогу.

Серед переваг можна відзначити інтуїтивний інтерфейс з логічним розташуванням елементів, зручну систему відгуків, що дозволяє оцінювати якість рецептів. Простий процес реєстрації з мінімальними вимогами, а також активна спільнота користувачів, яка постійно наповнює платформу новим контентом — також серед плюсів.

Недоліки платформи полягають у відсутності аналізу харчової цінності продуктів та готових страв, відсутності ШІ-помічника для рекомендацій на основі інгредієнтів. Крім того, відсутність створення власних колекцій для планування харчування.

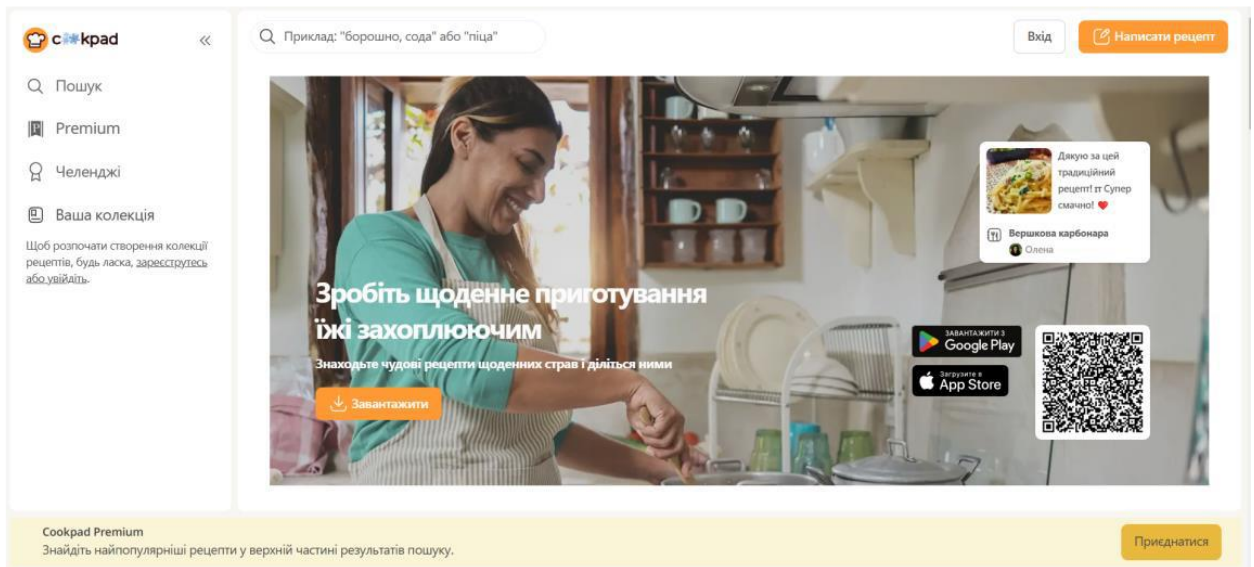


Рис. 1.3 - Cookpad

Smachno.ua [4] (рис. 1.4) – це платформа, яка пропонує рецепти зі знаменитих телевізійних шоу телеканалів СТБ і Нового каналу, таких як “МастерШеф”, “Все буде смачно”, “На ножах”, “Пекельні борошна” та інші. Професійні кулінари діляться своїми фірмовими рецептами та корисними порадами.

Серед переваг платформи можна виділити зручний пошук страв за інгредієнтами, що дозволяє ефективно використовувати наявні продукти, а також інтеграція з соціальними мережами, яка дає змогу легко ділитися улюбленими рецептами та залучати ширшу аудиторію.

Переваги платформи включають зручну інтеграцію з популярними соціальними мережами для простого поширення улюблених рецептів та функціональний пошук за інгредієнтами, що допомагає ефективно використовувати наявні продукти

До недоліків ресурсу можна віднести технічну неоптимізованість сайту, через яку сторінки завантажуються повільно. Крім того, повністю відсутній аналіз харчової цінності продуктів та страв, що ускладнює планування збалансованого раціону. Також немає можливості створення персональних колекцій улюблених рецептів для швидкого доступу до них у майбутньому.

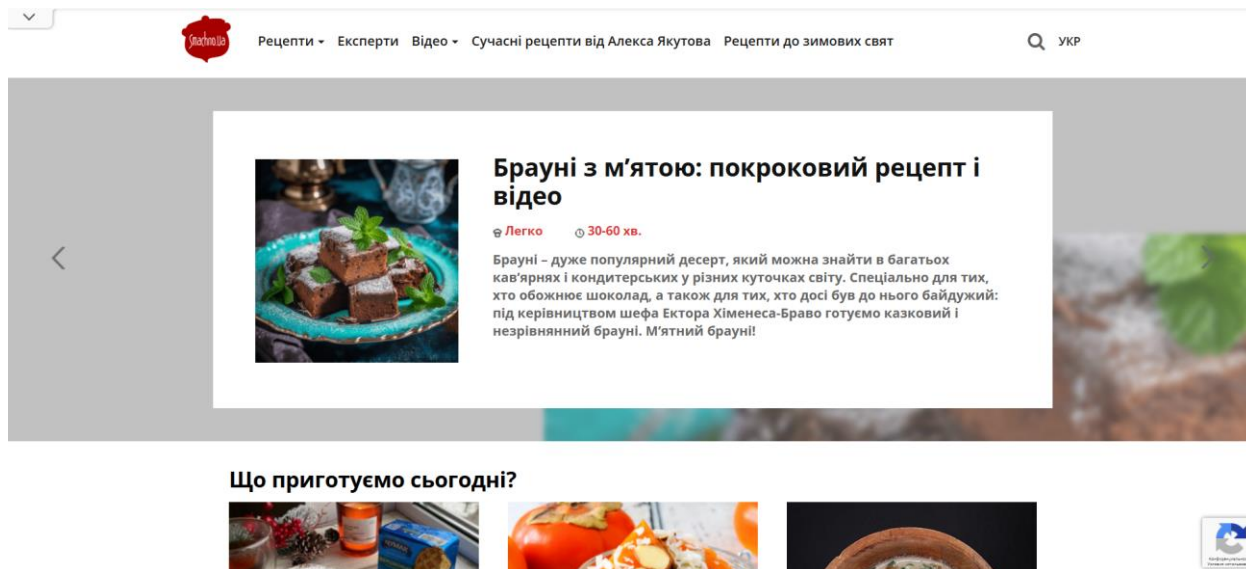


Рис. 1.4 - Smachno.ua

Результати аналізу аналогів дозволило виявити які є переваги кожного з рішень, так і недоліки, що потребують вирішення при розробці власної платформи.

Порівняння функціональних можливостей досліджених продуктів сформовано в таблиці, що наведена нижче (табл. 1.1).

Таблиця 1.1

Функціональні можливості платформ для пошуку страв

Функціональні можливості	YABPOELA	Cookorama	Cookpad	Smachno.ua	Розроблена платформа
Безкоштовний доступ	Так	Так	Так	Так	Так
Система рейтингів	Ні	Так	Так	Так	Так
Аналіз харчової цінності	Так	Ні	Ні	Ні	Так
Інтеграція ІІІ-чату	Ні	Ні	Ні	Ні	Так
Персональні колекції	Ні	Так	Так	Ні	Так
Адміністрування ролей користувачів	Ні	Ні	Ні	Ні	Так
Модерація страв	Ні	Ні	Так	Ні	Так

Аналіз наведених у таблиці платформ чітко показує, що жодна з них не здатна повністю задовольнити потреби користувачів. Найсуттєвішими недоліками є відсутність інтеграції зі штучним інтелектом і обмежені можливості аналізу харчової цінності.

Розроблена платформа для пошуку страв розробляється з метою усунення виявлених недоліків та об'єднання найкращих переваг існуючих рішень. Це має стати основою для формування стійких конкурентних переваг.

1.3. Вибір архітектури платформи для пошуку страв

Для розробки платформи з пошуку страв було обрано клієнт-серверну архітектуру [5], як найбільш ефективне рішення, що поєднує високу продуктивність, масштабованість та зрозумілий розподіл відповідальностей між компонентами системи. Такий підхід дозволяє чітко розмежувати користувацький інтерфейс і бізнес-логіку, спрощуючи розгортання й подальшу підтримку.

Клієнтська частина відповідає за взаємодію з користувачем: перевірку введених даних, кешування, відображення контенту та надсилання запитів на сервер. Вона реалізована на базі фреймворку Next.js [11], побудованого на React [12], що дозволяє поєднувати переваги серверного рендерингу зі швидкістю односторінкових застосунків (SPA) [14]. Це забезпечує як зручний інтерфейс, так і швидке завантаження та хорошу індексацію сторінок у пошукових системах.

Кожен елемент інтерфейсу реалізований як незалежний, наприклад, картка страви, форма пошуку або інтерактивний чат. Компонентна архітектура забезпечує гнучкість, повторне використання коду та спрощення тестування.

Серверна частина реалізована на Node.js [8] з використанням Fastify [9] як вебфреймворку. Такий стек дозволяє ефективно обробляти асинхронні запити, витримувати високі навантаження та будувати швидкі REST-сервіси. Архітектура побудована на модульному принципі: контролери відповідають за HTTP-запити та маршрутизацію, сервіси реалізують бізнес-логіку, а інтеграційні модулі взаємодіють з зовнішніми API, такими як Edamam Food API [22], Edamam Nutrition Analysis API [23], Google Gemini [24].

Кожен функціональний блок платформи (користувачі, рецепти, чат з ШІ, харчова аналітика) винесений в окремий модуль із власними маршрутами, моделями та сервісами. Це забезпечує низький рівень зв'язності між модулями та спрощує підтримку і масштабування платформи в майбутньому.

Обмін даними між клієнтом і сервером відбувається через REST API [7], що базується на стандартних HTTP-методах (GET, POST, PUT, DELETE). Такий підхід полегшує інтеграцію з іншими системами, забезпечує передбачувану поведінку й зручну документацію.

У підсумку, обрана архітектура дозволяє створити масштабовану, зручну в обслуговуванні платформу, яка легко адаптується до змін навантаження, підтримує сучасні інтерфейси та готова до впровадження нових функцій.

На наступний сторінці на рисунку 1.5, розташованому на наступній сторінці, зображено, як працює ця архітектура.

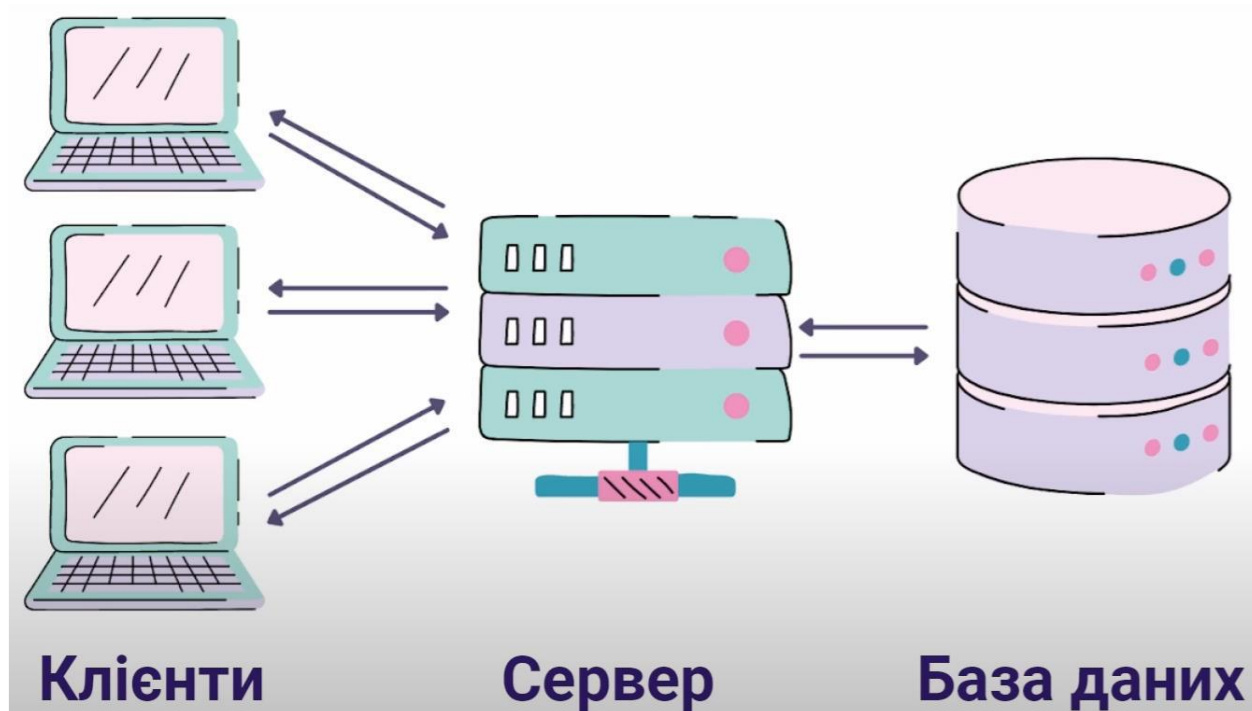


Рис. 1.5 - Клієнт-серверна архітектура

1.4. Обґрунтування вибору інструментальних засобів

Для реалізації платформи з пошуку страв було обрано сучасний технологічний стек, який поєднує ефективність розробки, високу

продуктивність, масштабованість і простоту подальшої підтримки. Архітектура побудована за класичним принципом поділу на клієнтську та серверну частини, які взаємодіють через REST API.

Клієнтська частина реалізована за допомогою фреймворку Next.js на базі бібліотеки React. Його обрано завдяки можливості поєднувати переваги серверного рендерингу та функціональності (SPA), що забезпечує швидке завантаження сторінок. Компонентна модель React дозволяє створювати повторно використовувані UI-елементи з чіткою логікою та станом, що полегшує тестування, підтримку й розширення проєкту..

Для стилізації інтерфейсу використано Tailwind CSS [12], що забезпечує гнучкий підхід до адаптивного дизайну. Tailwind добре інтегрується з React і Next.js та значно прискорює розробку завдяки готовим класам для типових сценаріїв..

Мовою програмування на фронтенді обрано TypeScript [15]. Вона дозволяє впровадити строгішу типізацію в порівнянні з JavaScript, що позитивно впливає на якість коду, зменшує кількість помилок під час компіляції та покращує командну розробку.

Серверна частина реалізована на Node.js — асинхронному середовищі для JavaScript, що добре масштабується під великі навантаження. Використано фреймворк Fastify, який підтримує модульну архітектуру та валідацію даних через схеми — це дозволяє ефективно будувати REST API.

Для автентифікації застосовується Supabase Auth [16], що підтримує JWT-токени та безпечний вхід, у тому числі через сторонні сервіси, як-от Google. Supabase [17] також виконує роль хмарної СУБД на основі PostgreSQL [29], забезпечуючи надійне зберігання даних, підтримку реляційних зв'язків, JSON-структур, резервне копіювання та масштабування.

Для зберігання файлів використовується Supabase Storage [18], що дозволяє легко керувати контентом і медіа-ресурсами, які створюють користувачі.

Ключові характеристики СУБД

Характеристика	MongoDB	PostgreSQL	MySQL
Модель даних	Документо-орієнтована (JSON/BSON)	Реляційна + JSONB	Реляційна
Масштабованість	Горизонтальна (sharding)	Горизонтальна та вертикальна	Переважно вертикальна
Гнучкість схеми	Динамічна, без строгих обмежень	Чітко визначена схема, JSONB для напівструктур	Чітко визначена схема
Типові сценарії продуктивності	Швидка робота з неструктурованими даними	Ефективна обробка складних зв'язків та транзакцій	Максимальна швидкість простих SELECT/INSERT
Робота з JSON	Природна, основний формат	JSONB-колонки з індексами	Зберігається як текст, обмежені функції

Щоб оцінити, наскільки кожна система відповідає вимогам проєкту (велика кількість одночасних користувачів, зміна структури даних, робота з API), розглянемо таблицю (табл. 1.2.).

Таблиця 1.3

Відповідність ключовим вимогам

Вимога проєкту	MongoDB	PostgreSQL	MySQL
Гнучка зміна структури	Висока	Середня (через міграції)	Низька
Масштабування під навантаженням	Горизонтальне	Горизонтальне + реплікація	Переважно вертикальне
Обробка не-реляційних даних	Дуже ефективна	Обмежена (JSONB)	Складно реалізувати
Продуктивність при великих даних	Відмінна	Висока	Висока
Легка інтеграція з JS-стеком	Пряма (Mongoose, Native driver)	Через ORM (Prisma/TypeORM)	Через ORM
Складні реляційні зв'язки	Обмежені	Повні	Повні

Нарешті, для комплексної оцінки врахуємо обмеження на розмір, резервне копіювання та захист даних (див. табл. 1.3.).

Ємність і безпека

Параметр	MongoDB	PostgreSQL	MySQL
Максимальний обсяг БД	~16 ТБ на колекцію (без обмежень у кластері)	Практично необмежений	Практично необмежений
Ліміт документа / рядка	16 МБ	1.6 ТБ на таблицю	4 ГБ на таблицю
Масштабованість	Sharding «з коробки»	Реплікація + шардинг (Citus)	Кластер NDB / InnoDB
Шифрування даних	TLS + field-level TDE	TLS + розширення TDE	TLS, TDE через сторонні засоби
Резервне копіювання	Вбудовані snapshot-и та oplog	pg_dump / стрімінг	mysqldump, hot-backup

Як видно в таблицях 1.1 – 1.3, PostgreSQL демонструє оптимальне поєднання надійності, розширених можливостей SQL і гнучкої підтримки JSON-даних. Завдяки можливості горизонтального масштабування та інтеграції з хмарною платформою Supabase, ця СУБД чудово підходить для архітектури на базі Fastify. Враховуючи вимоги проєкту, саме PostgreSQL було обрано як найкращий варіант серед MongoDB і MySQL.

Обраний технологічний стек дає змогу створити масштабовану, інтерактивну й зручну у використанні платформу, яка поєднує сучасні вебтехнології зі штучним інтелектом для покращення кулінарної обізнаності користувачів. Такий підхід також відкриває перспективи для подальшого розвитку проєкту — включаючи розширення функціоналу, адаптацію до мобільних пристроїв і інтеграцію з новими сервісами.

Висновки до першого розділу

У першому розділі було проаналізовано ключові аспекти, що стосуються створення платформи для пошуку страв із використанням сучасних вебтехнологій та штучного інтелекту. Було визначено основну мету проєкту, описано функціональні модулі системи та їх взаємозв'язок, а також обґрунтовано доцільність використання інтелектуального асистента.

Проведений аналіз існуючих кулінарних платформ дозволив визначити найкращі галузеві практики, виявити слабкі місця конкурентних рішень та

сформулювати вимоги до майбутньої системи. Зокрема, серед недоліків аналогів було виявлено відсутність персоналізації, повноцінної інтеграції з AI-сервісами, обмежений функціонал збереження рецептів та відсутність аналізу харчової цінності.

На основі порівняння архітектурних підходів було обґрунтовано використання клієнт-серверної моделі, яка забезпечує чіткий розподіл обов’язків між фронтендом і бекендом, високу масштабованість, стабільність і спрощену інтеграцію зі сторонніми сервісами.

Також у розділі детально обґрунтовано вибір технологічного стеку, до якого увійшли Next.js, React, TypeScript, Tailwind CSS, Node.js, Fastify та Supabase. Порівняльний аналіз СУБД показав, що PostgreSQL найбільше відповідає вимогам платформи завдяки поєднанню реляційної моделі з підтримкою JSON-структур, високій продуктивності та готовності до масштабування.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ПЛАТФОРМИ ДЛЯ ПОШУКУ СТРАВ

2.1. Визначення варіантів використання та структури платформи

Платформу для пошуку страв реалізовано як багаторівневу систему з чітким поділом функціональності в залежності від ролі користувача. Додаток складається з модулів, що відповідають за перегляд каталогу страв, управління персональними колекціями страв користувача, взаємодію з ШІ-асистентом для отримання рекомендацій для приготування, аналіз харчової цінності інгредієнтів, а також адміністрування контенту. Вся інформація зберігається в одній базі даних і забезпечує цілісність та доступність інформації будь-якому стані користувача в системі.

Підтримуються три основні категорії користувачів, кожна з яких має специфічні потреби та рівні доступу до функціональності системи.

Неzareєстровані користувачі (Гість) представляють категорію відвідувачів платформи, які ще не створили облікового запису або тимчасово користуються системою без автентифікації. Ця категорія користувачів має обмежений, але достатній для ознайомлення доступ до базової функціональності платформи, що дозволяє їм оцінити цінність сервісу перед реєстрацією.

Зареєстровані користувачі являють собою основну цільову аудиторію платформи, яка отримує доступ до повного спектру функцій. Ця категорія включає як любителів кулінарії, так і професійних кухарів, які активно використовують платформу для пошуку, збереження та створення кулінарного контенту.

Адміністратори забезпечують управління платформою, модерацію контенту та підтримку якості сервісу. Ця категорія користувачів має розширені права доступу до адміністративних функцій та інструментів аналітики.

Неzareєстровані користувачі мають доступ до базового функціоналу перегляду та пошуку контенту. Платформа повинна забезпечувати можливість перегляду каталогу страв з базовою інформацією про страви, включаючи назву, фотографію, час приготування та рівень складності. Пошукова система

має підтримувати базові фільтри за категоріями страв, типами кухні та основними інгредієнтами.

Функціонал перегляду детальної інформації про рецепти включає доступ до повного списку інгредієнтів, покрокових інструкцій приготування та додаткових рекомендацій від автора. Користувачі можуть переглядати калорійність страви, але без доступу до коментарів та відгуків інших користувачів.

Система реєстрації та автентифікації має бути легко доступною з будь-якої сторінки платформи, з мінімальними вимогами до персональних даних та підтримкою різних методів реєстрації, включаючи електронну пошту.

Зареєстровані користувачі отримують доступ до розширеного функціоналу персоналізації та взаємодії з контентом. Система управління персональними колекціями повинна дозволяти створення, редагування та організацію власних збірок страв та особистих нотаток.

Функціонал взаємодії з ШІ-асистентом включає можливість отримання персоналізованих рекомендацій рецептів на основі наявних інгредієнтів, дієтичних обмежень. Система повинна зберігати історію взаємодії з ШІ для покращення якості майбутніх рекомендацій.

Функції платформи включають можливість оцінювання страви, залишення коментарів та відгуків. Система повинна підтримувати завантаження фотографій власних кулінарних експериментів та їх прив'язку до відповідних рецептів.

Функціонал створення та публікації власних страв має включати інтуїтивний редактор з можливістю додавання контенту, розрахунку харчової цінності через інтеграцію з Edamam API та налаштування рівнів доступу до опублікованого контенту.

Система аналізу харчової цінності повинна розраховувати калорійність, вміст білків, жирів, вуглеводів та основних мікроелементів для рецептів, з можливістю адаптації порцій та порівняння альтернативних інгредієнтів.

Панель адміністратора повинна забезпечувати управління всіма аспектами платформи. Система модерації контенту включає інструменти для перевірки та затвердження нових страв, модерації коментарів та відгуків.

Функціонал управління користувачами охоплює можливості перегляду та редагування профілів користувачів, управління правами доступу, блокування та розблокування облікових записів.

Система повинна забезпечувати час відгуку не більше 2 секунд для основних операцій перегляду контенту та пошуку рецептів при нормальному навантаженні. Для операцій, що потребують інтеграції з зовнішніми API (аналіз харчової цінності, ШІ-рекомендації), максимальний час відгуку не повинен перевищувати 10 секунд.

Платформа має підтримувати одночасну роботу до 1000 активних користувачів без деградації продуктивності. Система завантаження сторінок повинна бути оптимізована для швидкого відображення контенту з підтримкою завантаження та кешування.

На рисунку 2.1 наведено діаграму взаємодії користувачів з системою розроблено діаграму варіантів використання, що відображає основні сценарії роботи з платформою для кожної категорії користувачів.

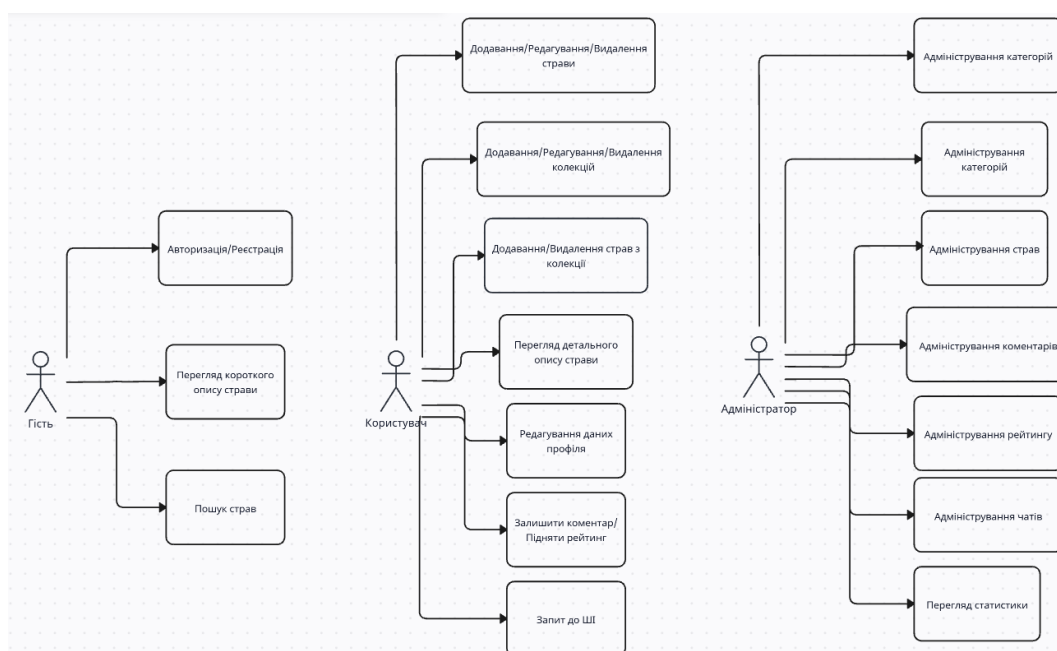


Рис. 2.1 – Діаграма варіантів використання

Опис основних варіантів використання наведено нижче в таблицях 2.1 – 2.11.

Таблиця 2.1

Опис прецеденту «Реєстрація/Вхід»

Поле	Опис
Назва варіанту використання	Реєстрація/Вхід
Основні актори	Гість, Користувач, Адміністратор
Короткий опис	Гість, користувач, адміністратор здійснюють реєстрацію та вхід для доступу до платформи.

Таблиця 2.2

Опис прецеденту «Пошук страв»

Поле	Опис
Назва варіанту використання	Пошук страв
Основні актори	Гість, Користувач, Адміністратор
Короткий опис	Гість, користувач, адміністратор здійснюють пошук страв за назвою, категоріями або іншими критеріями з використанням системи фільтрів.

Таблиця 2.3

Опис прецеденту «Перегляд страв»

Поле	Опис
Назва варіанту використання	Перегляд детальної інформації про страву
Основні актори	Гість, Користувач
Короткий опис	Гість, користувач переглядають інформацію про страву, включаючи інгредієнти, інструкції приготування та харчову цінність.

Таблиця 2.4

Опис прецеденту «Розрахунок харчової цінності»

Поле	Опис
Назва варіанту використання	Розрахунок харчової цінності страви
Основні актори	Гість, Користувач
Короткий опис	Гість, Користувач надсилає запит на розрахунок харчової цінності страви з можливістю адаптації під кількість порцій.

Таблиця 2.5

Опис прецеденту «Запит до ІІІ»

Поле	Опис
Назва варіанту використання	Отримання ІІІ-рекомендацій та кулінарних порад
Основні актори	Користувач
Короткий опис	Користувач надсилає запит штучному інтелекту та отримує відповідь

Таблиця 2.6

Опис прецеденту «Управління персональними колекціями»

Поле	Опис
Назва варіанту використання	Створення та управління колекціями страв
Основні актори	Користувач
Короткий опис	Користувач створює, редагує, видаляє персональні колекції зі стравами.

Таблиця 2.7

Опис прецеденту «Управління персональними колекціями»

Поле	Опис
Назва варіанту використання	Створення та управління колекціями страв
Основні актори	Користувач
Короткий опис	Користувач створює, редагує, видаляє персональні колекції зі стравами.

Таблиця 2.8

Опис прецеденту «Додавання власної страви»

Поле	Опис
Назва варіанту використання	Додавання та публікація власного рецепту
Основні актори	Користувач
Короткий опис	Користувач створює та публікує власну страву з можливістю додавання фотографій та детальних інструкцій приготування.

Таблиця 2.9

Опис прецеденту «Оцінювання та коментування страв»

Поле	Опис
Назва варіанту використання	Оцінювання та коментування страв
Основні актори	Користувач
Короткий опис	Користувач залишає оцінки, коментарі до страв, ділиться власним досвідом приготування страв.

Таблиця 2.10

Опис прецеденту «Модерація контенту»

Поле	Опис
Назва варіанту використання	Перевірка та затвердження користувацького контенту
Основні актори	Адміністратор
Короткий опис	Адміністратор перевіряє нові страви, модерує коментарі, забезпечує якість контенту на платформі.

Таблиця 2.11

Опис прецеденту «Управління користувачами»

Поле	Опис
Назва варіанту використання	Адміністрування облікових записів користувачів
Основні актори	Адміністратор
Короткий опис	Адміністратор управляє обліковими записами користувачів, налаштовує права доступу, видаляє користувачів.

Аналіз функціональних вимог та варіантів використання платформи дав змогу визначити основні сутності, необхідні для реалізації повного функціоналу системи. Кожна сутність представляє ключовий об'єкт предметної області з власною структурою даних, поведінкою та взаємозв'язками з іншими компонентами системи.

На основі проведеного аналізу було ідентифіковано основні сутності, які формують моделі платформи та забезпечують реалізацію всіх визначених функціональних вимог. Ці сутності безпосередньо відображають реальну структуру бази даних, яка була спроектована з урахуванням оптимізації продуктивності та забезпечення цілісності даних.

2.2. Розробка бази даних

Для зберігання даних про страви, користувачів, інгредієнти та кулінарні колекції, ШІ-чати, коментарі, категорії у додатку використовується база даних PostgreSQL на платформі Supabase. Supabase являє собою платформу, яка надає потужні інструменти для розробки додатків, включаючи управління базами даних, автентифікацію користувачів, real-time підписки та API-інтерфейси. PostgreSQL в екосистемі Supabase забезпечує високу надійність та продуктивність при роботі з структурованими даними, а також надає додаткові можливості безпеки даних на рівні рядків.

Вибір PostgreSQL з Supabase для даного проекту обумовлений особливостями предметної області, де необхідно підтримувати складні зв'язки між стравами, інгредієнтами, категоріями та користувачами.

База даних системи складається з 12 колекцій:

1. profiles - колекція для зберігання персональної інформації користувача, профільні дані та ролі доступу.
2. dishes – колекція для зберігання інформації про страви, а саме, назви, описи, зображення, кількість порцій та статус модерації.
3. dish_ingredients – колекція для зберігання інгредієнтів кожної страви, Забезпечує можливість розрахунку калорійності.
4. dish_steps – колекція для зберігання покрокових інструкцій приготування, що містить послідовність дій, часові рамки.
5. dish_categories – колекція для зберігання категорій страв (закуси, основні страви, десерти, напої тощо), дана колекція допомагає класифікувати страви для пошуку та навігації по платформі.
6. dish_category_relations – проміжна колекція для зв'язків між стравами та категоріями, що дозволяє одній страві належати до кількох категорій одночасно.
7. dish_collections – колекція для зберігання персональних колекцій страв користувачів, забезпечує можливість створення добірок рецептів (улюблені страви, планування меню, сезонні рецепти).

8. dish_collection_items - колекція для зберігання конкретної страви в персональній колекції.
9. dish_comments – колекція для зберігання інформації коментарів користувачів до страв.
10. dish_ratings – колекція для зберігання оцінок страв користувачами, що реалізує систему лайків для формування рейтингу страви. Забезпечує основу для рекомендаційних алгоритмів та сортування контенту.
11. ai_chat_sessions – колекція для зберігання спілкування зі ШІ, зберігає історію запитань. Забезпечує безперервність діалогу та можливість повернення до попередніх обговорень.
12. ai_chat_messages – колекція для зберігання повідомлень у чатах з ШІ, що містить як запитання користувачів, так і відповіді асистента.

Опис кожної колекції бази даних наведено нижче в таблицях 2.12 – 2.23.

Таблиця 2.12

Колекція profiles

Назва поля	Тип даних	PK	FK	Примітка
id	UUID	+	-	Унікальний ідентифікатор користувача
email	String	-	-	Електронна пошта користувача
full name	String	-	-	Повне ім'я користувача
avatar url	String	-	-	URL аватара користувача
role	Enum (user/admin)	-	-	Роль користувача
profile tag	String	-	-	Унікальний тег профілю
created at	Date	-	-	Дата створення профілю
updated at	Date	-	-	Дата останнього оновлення

Таблиця 2.13

Колекція dishes

Назва поля	Тип даних	PK	FK	Примітка
1	2	3	4	5
id	UUID	+	-	Унікальний ідентифікатор страви
user_id	UUID	-	+	Унікальний ідентифікатор автора страви

1	2	3	4	5
title	String	-	-	Назва страви
description	String	-	-	Опис страви
main_image_url	Enum (user/admin)	-	-	URL головного зображення
servings	Int	-	-	Кількість порцій
status	String	-	-	Статус модерації
rejection_reason	String	-	-	Причина відхилення модерації
created_at	Date	-	-	Дата створення профілю
updated_at	Date	-	-	Дата останнього оновлення
moderated_at	Date	-	-	Дата модерації

Таблиця 2.14

Колекція dish_ingredients

Назва поля	Тип даних	PK	FK	Примітка
id	UUID	+	-	Унікальний ідентифікатор інгредієнта
dish_id	UUID	-	+	Унікальний ідентифікатор страви
name	String	-	-	Назва інгредієнта
amount	String	-	-	Кількість
unit	Enum (user/admin)	-	-	Одиниця вимірювання
edamam_food_id	UUID	-	-	Унікальний ідентифікатор інгредієнта

Таблиця 2.15

Колекція dish_steps

Назва поля	Тип даних	PK	FK	Примітка
id	UUID	+	-	Унікальний ідентифікатор кроку
dish_id	UUID	-	+	Електронна пошта користувача
step_number	String	-	-	Номер кроку
description	String	-	-	Опис
image_url	String	-	-	URL зображення
duration_minutes	String	-	-	Тривалість кроку
created_at	Date	-	-	Дата створення

Таблиця 2.16

Колекція dish_categories

Назва поля	Тип даних	PK	FK	Примітка
id	UUID	+	-	Унікальний ідентифікатор категорії
name	String	-	-	Назва категорії
description	String	-	-	Опис категорії
created_at	Date	-	-	Дата створення
updated_at	Date	-	-	Дата оновлення

Таблиця 2.17

Колекція dish_category_relations

Назва поля	Тип даних	PK	FK	Примітка
id	UUID	+	-	Унікальний ідентифікатор зв'язку
dish_id	UUID	-	+	Ідентифікатор страви
category_id	UUID	-	+	Ідентифікатор категорії
created_at	Date	-	-	Дата створення

Таблиця 2.18

Колекція dish_collections

Назва поля	Тип даних	PK	FK	Примітка
id	UUID	+	-	Унікальний ідентифікатор колекції
user_id	UUID	-	+	Ідентифікатор страви
owner_id	UUID	-	+	Ідентифікатор категорії
name	String	-	-	Назва колекції
description	String	-	-	Опис колекції
created_at	Date	-	-	Дата створення
updated_at	Date	-	-	Дата оновлення

Таблиця 2.19

Колекція dish_collection_items

Назва поля	Тип даних	PK	FK	Примітка
id	UUID	+	-	Унікальний ідентифікатор коментаря
collection_id	UUID	-	+	Ідентифікатор колекції
dish_id	UUID	-	+	Ідентифікатор страви
user_id	UUID	-	+	Ідентифікатор користувача
added_at	Date	-	-	Дата додавання до колекції

Таблиця 2.20

Колекція dish_comments

Назва поля	Тип даних	PK	FK	Примітка
1	2	3	4	5
id	UUID	+	-	Унікальний ідентифікатор зв'язку
dish_id	UUID	-	+	Ідентифікатор колекції
user_id	UUID	-	+	Ідентифікатор страви
content	UUID	-	-	Ідентифікатор користувача
is_deleted	Boolean	-	-	Ознака видалення коментаря
created_at	Date	-	-	Дата створення
updated_at	Date	-	-	Дата оновлення

Таблиця 2.21

Колекція dish_ratings

Назва поля	Тип даних	PK	FK	Примітка
id	UUID	+	-	Унікальний ідентифікатор рейтингу
dish_id	UUID	-	+	Ідентифікатор колекції
user_id	UUID	-	+	Ідентифікатор страви
content	UUID	-	-	Ідентифікатор користувача
rating_type	int	-	-	Тип оцінки (0 - дизлайк, 1 - лайк)
created_at	Date	-	-	Дата створення
updated_at	Date	-	-	Дата оновлення

Таблиця 2.22

Колекція ai_chat_sessions

Назва поля	Тип даних	PK	FK	Примітка
id	UUID	+	-	Унікальний ідентифікатор сесії
user_id	UUID	-	+	Ідентифікатор користувача
title	String	-	-	Назва чату
created_at	Date	-	-	Дата створення
updated_at	Date	-	-	Дата оновлення

Таблиця 2.23

Колекція ai_chat_messages

Назва поля	Тип даних	PK	FK	Примітка
id	UUID	+	-	Унікальний ідентифікатор повідомлення
session_id	UUID	-	+	Ідентифікатор сесії
title	Enum(user/assistant)	-	-	Роль автора
content	String	-	-	Текст повідомлення
created_at	Date	-	-	Дата створення
updated_at	Date	-	-	Дата оновлення

Таким чином, виглядає розроблена структура бази даних на основі дванадцяти колекцій, які забезпечують повний функціонал платформи для пошуку страв, управління користувачів, модерації страв та коментарів, інтеграції зі ШІ.

Структуру бази даних зображено на рисунку 2.2 на наступній сторінці.

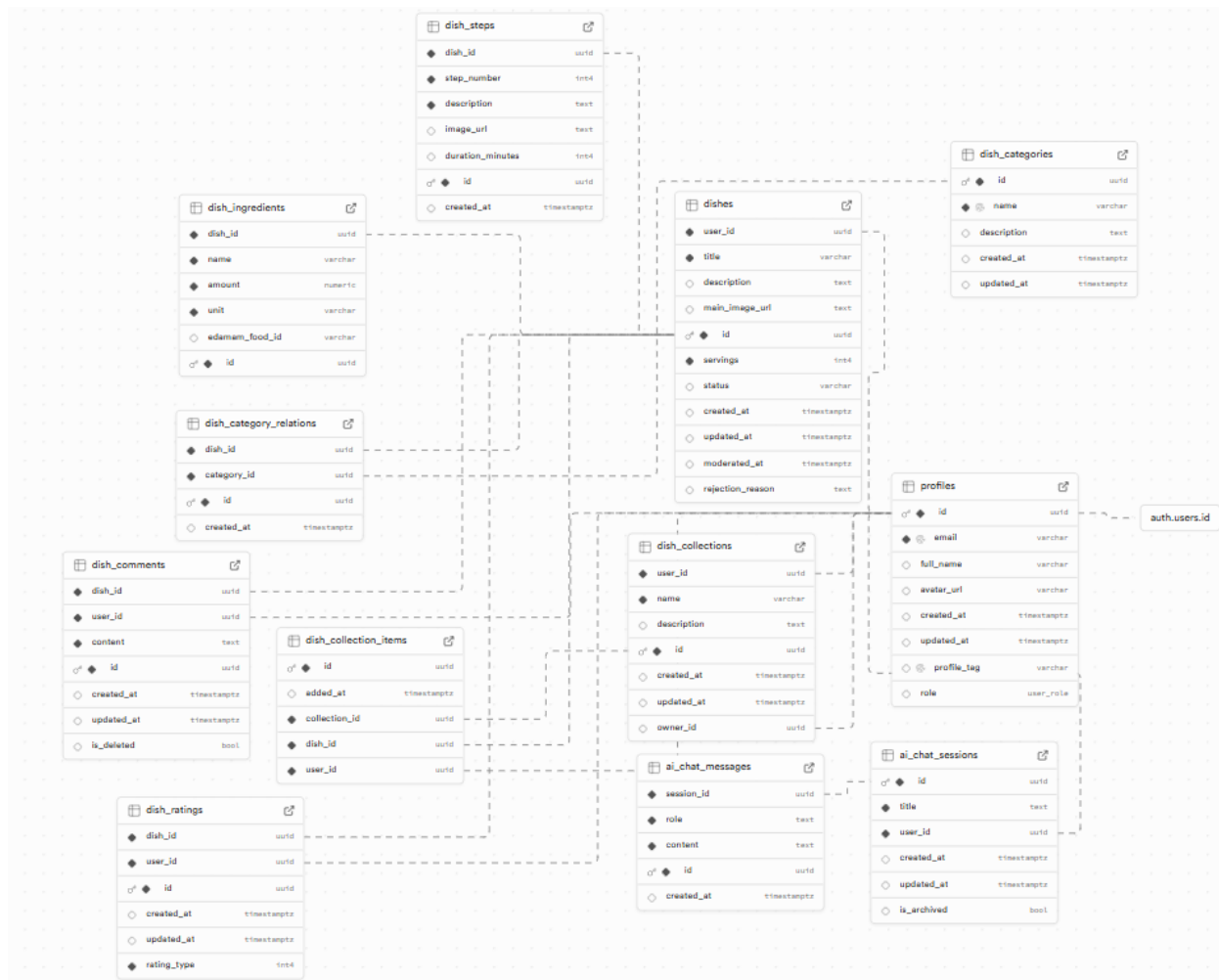


Рис. 2.2 - Структура бази даних

2.3. Проєктування та реалізація алгоритмів роботи платформи

Для зображення загальної логіки роботи платформи, було побудовано загальний алгоритм роботи систем (рис. 2.3), який демонструє основні процеси виконання залежно від ролі користувача.

Якщо авторизувався адміністратор:

- після авторизації з'являється посилання до панелі адміністратора;
- адміністратор має можливість переглянути всі страви і підтвердити їх, якщо задовольняють умови користування платформою;

- адміністратор має можливість надавати роль іншим користувачам;
- адміністратор має можливість додавати, редагувати та видаляти категорії;
- адміністратор має можливість переглянути, видалити коментар;
- адміністратор має можливість переглянути статистику, детальну інформацію, видалити ШІ-чати.

Якщо авторизувався користувач:

- користувач має можливість переглянути детальну інформацію страви;
- користувач має можливість залишити коментар та відгук під стравою;
- користувач має можливість створювати, редагувати, видаляти страви;
- користувач має можливість опубліковувати страви які були підтверджені адміністрацією;
- користувач має можливість створювати, редагувати, видаляти колекції;
- користувач має можливість редагувати дані профілю;
- користувач має можливість отримувати поради, щодо приготування страви;

Якщо увійшов гість:

- гість має можливість переглянути страви;
- гість має можливість виконати пошук та фільтрувати страви;
- гість має можливість переглянути детальну інформацію страви;

Представлений алгоритм демонструє розгалуження логіки виконання залежно від ролі користувача в системі. На початковому етапі відбувається ідентифікація користувача через процес авторизації або реєстрації. Після успішної автентифікації система визначає права доступу та доступний функціонал відповідно до призначеної ролі.

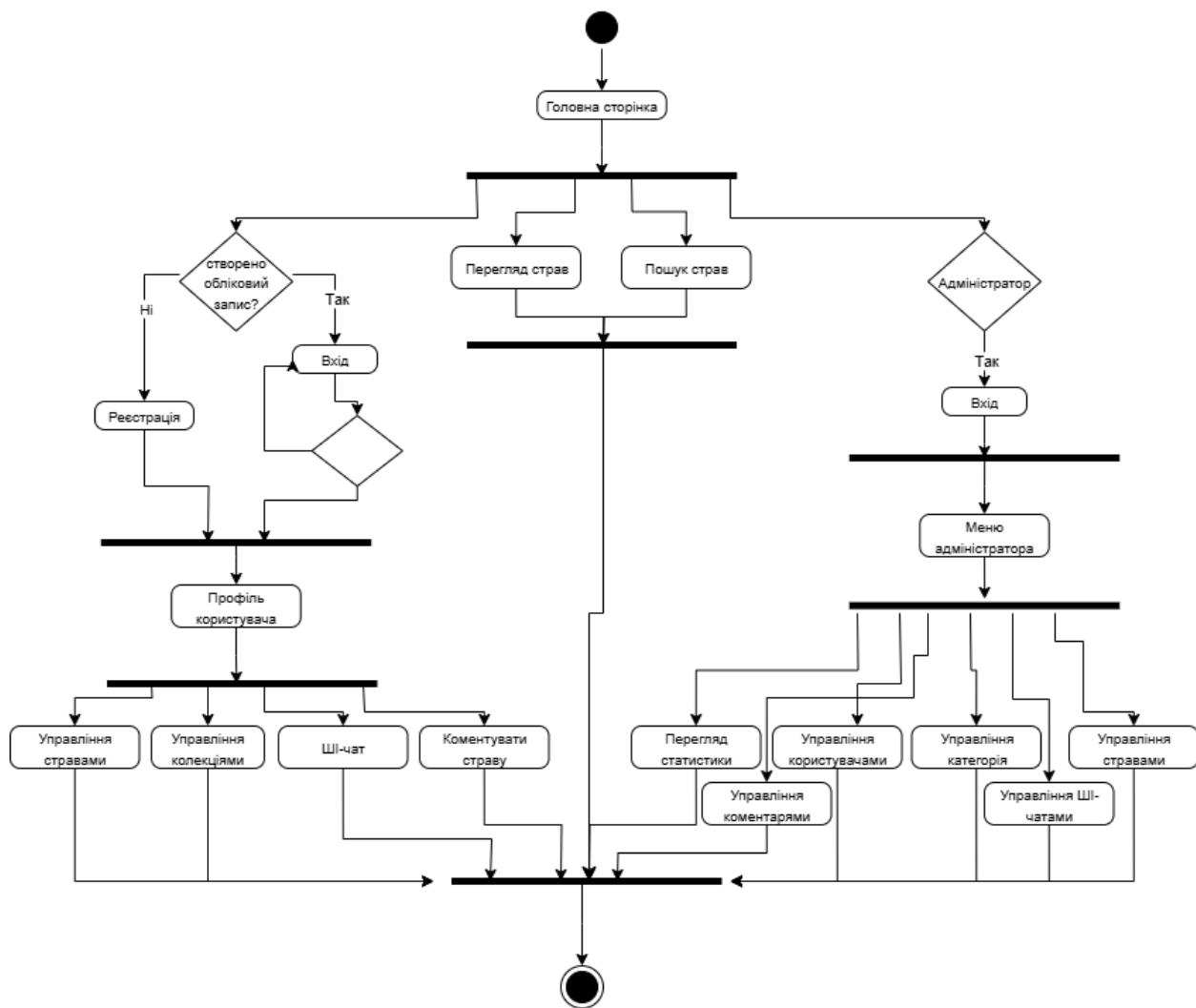


Рис. 2.3 – Діаграма активності

2.4. Реалізація платформи для пошуку страв

Реалізація платформи для пошуку страв базується на клієнт-серверній архітектурі. Клієнтська частина реалізована за допомогою фреймворка Next.js та React, а серверна частина – на node.js з використанням Fastify та Supabase. Даний підхід забезпечує високу продуктивність, масштабованість і зрозумілу логіку розподілу відповідальностей. Комунікація між клієнтом і сервером реалізована через REST API, що базується на стандартизованих HTTP-методах, забезпечуючи передбачуваність поведінки API. Окремо реалізовано функціонал завантаження зображень страв. Для цього застосовано механізми Supabase Storage, які дозволяють зберігати та обробляти медіафайли у хмарному сховищі. Дані про харчову цінність розраховуються через інтеграцію з Edamam Nutrition Analysis API [23], що дозволяє здійснювати

аналіз інгредієнтів та харчову цінніста страви. Для персоналізованих рекомендацій інтегровано Google Gemini AI, який аналізує наявні інгредієнти та формує пропозиції щодо оптимальних страв.

Наведений фрагмент коду, представлений на рисунку 2.4, відповідає за запуск та початкову ініціалізацію серверної частини платформи. Його логіка включає кілька ключових етапів, пов'язаних із підключенням бази даних, налаштуванням збереження медіа-файлів, налаштуванням CORS-політики та перевіркою інтеграції зовнішніх API. Спочатку здійснюється запуск сервера за вказаними даними конфігураційного файлу. Після відбувається підключення до бази даних, на цьому етапі перевіряється доступність сховища, і якщо з'єднання відбулося успішно, сервер повідомляє про готовність до роботи з даними. Далі проводиться ініціалізація механізму збереження медіа-файлів у Supabase Storage. Для цього виконується перевірка наявності спеціальних сховищ (bucket). Якщо вони відсутні, система автоматично їх створює окремі контейнери, для зберігання аватарів користувачів та завантаження фото страв. Обидва сховища налаштовані на роботу з обмеженням за розміром файлів і дозволеними форматами зображень (JPEG, PNG, WebP), що забезпечує контрольоване та безпечне збереження мультимедійного контенту.

```
const start = async () => {
  try {
    await fastify.listen({
      port: config.port,
      host: '127.0.0.1'
    })
    console.log(`Сервер запущено на http://localhost:${config.port}`)

    try {
      const { data: buckets, error } = await supabaseClient.storage.listBuckets()
      if (!error && buckets) {
        console.log('Supabase Storage підключено успішно')

        await ensureAvatarsBucket()

        await ensureDishImagesBucket()
      } else {
        console.log('Supabase Storage недоступно:', error?.message)
      }
    } catch (storageError) {
      console.log('Помилка підключення до Supabase Storage:', storageError.message)
    }

    if (process.env.EDAMAM_APP_FOOD_ID && process.env.EDAMAM_APP_FOOD_KEY) {
      console.log('Edamam Food Database API налаштовано')
    } else {
      console.log('Edamam Food Database API не налаштовано. Додайте EDAMAM_APP_FOOD_ID та EDAMAM_APP_FOOD_KEY до .env файлу')
    }
  }
}
```

Рис. 2.4 – Запуск сервера

Контролер страв відіграє ключову роль у роботі API, забезпечуючи повний набір операцій для керування об'єктами типу страва. Реалізований у вигляді набору асинхронних функцій, кожна з яких виконує конкретне завдання. CreateDish відповідає за створення страви, updateDish оновлює дані страви, deleteDish видаляє страви, функції getDishes getDishesById повертає страви, moderateDish відповідає за оприлюднення адміністрацією страви, uploadDishImage та uploadStepImage завантажують медіафайли страви, updateDishStatus оновлює статус страви, getAllDishesForAdmin отримання всіх даних страви або конкретної getDishesDetailsForAdmin. Обробка помилок стандартизована допоміжними методами _handleError _handleSuccess. Повна реалізація наведена у додатку А. Слід також зазначити, що інші контролери для категорій, колекцій, коментарів, користувачів та інших сутностей – реалізовано у схожому стилі, з дотриманням принципу REST. Наведено фрагмент коду на рисунку 2.5.

```

async createDish(userId, dishData) {
  try {
    const { title, description, category_ids, servings, ingredients, steps, main_image_url } = dishData

    await this._validateCategories(category_ids)

    const { data: dish, error: dishError } = await this.supabase
      .from('dishes')
      .insert({
        user_id: userId,
        title,
        description,
        servings,
        main_image_url,
        status: this.DISH_STATUS.DRAFT
      })
      .select()
      .single()

    if (dishError) {
      throw new Error(dishError.message)
    }

    await this._createDishRelations(dish.id, { category_ids, ingredients, steps })

    if (this.collectionService) {
      await this.collectionService.addDishToSystemCollections(
        dish.id,
        userId,
        dish.status
      )
    }
  }
}

```

Рис. 2.5 – Фрагмент створення страви

На рисунку 2.6, наведено фрагмент коду, який відповідає реєстрацію та підключення маршрутів для різних функціональних модулів платформи. Кожен із маршрутів відповідає за роботу зі своєю сутністю та має чітко визначений префікс для зручності організації API.

```

await fastify.register(authRoutes, { prefix: '/api/auth' })
await fastify.register(edamamRoutes, { prefix: '/api/edamam' })
await fastify.register(aiRoutes, { prefix: '/api/ai' })
await fastify.register(aiChatRoutes, { prefix: '/api/ai/chat' })
await fastify.register(aiAdminRoutes, { prefix: '/api/admin/ai' })
await fastify.register(translationRoutes, { prefix: '/api/translation' })

await fastify.register(userRoutes, { prefix: '/api/users' })
await fastify.register(categoryRoutes, { prefix: '/api/categories' })
await fastify.register(dishRoutes, { prefix: '/api/dishes' })
await fastify.register(commentRoutes, { prefix: '/api/comments' })
await fastify.register(ratingRoutes, { prefix: '/api/ratings' })
await fastify.register(collectionRoutes, { prefix: '/api/collections' })

await fastify.register(userAdminRoutes, { prefix: '/api/admin/users' })
await fastify.register(categoryAdminRoutes, { prefix: '/api/admin/categories' })
await fastify.register(dishAdminRoutes, { prefix: '/api/admin/dishes' })
await fastify.register(commentAdminRoutes, { prefix: '/api/admin/comments' })
await fastify.register(ratingAdminRoutes, { prefix: '/api/admin/ratings' })
await fastify.register(collectionAdminRoutes, { prefix: '/api/admin/collections' })

```

Рис. 2.6 – Реєстрацію та підключення маршрутів

Висновки до другого розділу

У другому розділі було спроектовано архітектуру та структуру платформи для пошуку страв, визначено ключові функціональні модулі та ролі користувачів. На основі аналізу вимог сформовано сценарії використання системи, що охоплюють процеси реєстрації та автентифікації, перегляду й пошуку страв, створення власних страв, взаємодії з ШІ-асистентом, управління персональними колекціями та адміністрування контенту.

Розроблено та описано базу даних, що забезпечує збереження й цілісність усіх необхідних даних: від інформації про страви та інгредієнти до коментарів, оцінок і історії чатів зі штучним інтелектом.

Було спроектовано загальні алгоритми роботи системи з урахуванням різних ролей користувачів.

Приділено увагу реалізації серверної частини та підключенню зовнішніх сервісів. Завдяки використанню Fastify та Supabase забезпечено високу продуктивність і масштабованість, а інтеграція з Edamam API та Google Gemini AI надала можливість реалізувати функції аналізу харчової цінності та формування персоналізованих рекомендацій. Таким чином, було сформовано цілісну модель платформи, яка відповідає визначеним вимогам та здатна забезпечити взаємодію з користувачем та стабільну роботу платформи.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ПЛАТФОРМОЮ ДЛЯ ПОШУКУ СТРАВ

3.1. Порядок встановлення та налаштування системи

Діаграма розгортання, представлена на рисунку 3.1, ілюструє фізичну архітектуру платформи для пошуку страв та демонструє взаємозв'язки між її основними компонентами під час розгортання. На схемі відображено ключові вузли системи: клієнтський веб-інтерфейс, сервер додатку, база даних Supabase, сховище зображень. Така візуалізація дозволяє продемонструвати взаємодію компонентів у реальному середовищі через протоколи HTTP/HTTPS, TCP/IP і API-виклики.

Клієнтська частина платформи реалізована у вигляді браузера, який надсилає HTTP-запити до серверного застосунку, побудованого на Fastify.

Серверна частина обробляє API-запити, виконує бізнес-логіку, управляє сесіями користувачів та взаємодіє з базою даних. Для зберігання та обробки зображень страв використовується інтегроване сховище Supabase Storage, яке забезпечує безпечне завантаження, оптимізацію та швидку доставку медіафайлів через вбудований CDN. Вся архітектура є масштабованою, дозволяє легко додавати нові сервіси та підтримує горизонтальне розширення при збільшенні навантаження. Дана архітектура є масштабованою, що дозволяє легко додавати нові сервіси та підтримує збільшення навантаження.

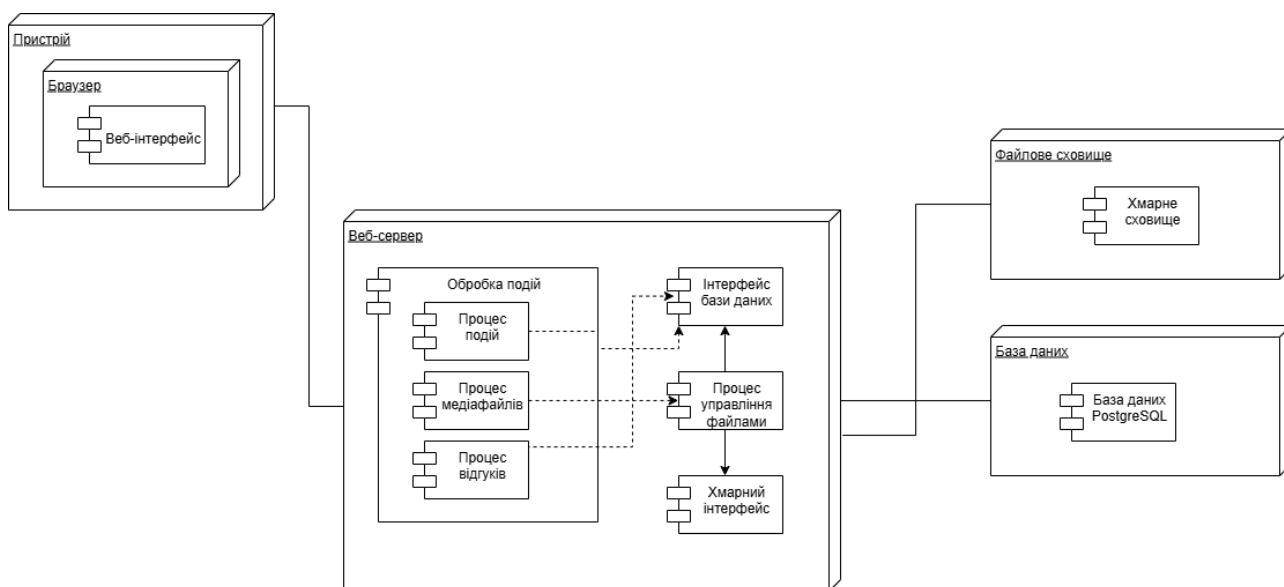


Рис. 3.1 - Діаграма розгортання платформи для пошуку страв

3.2. Структура інтерфейсу та порядок взаємодії з платформою для пошуку страв

Взаємодія користувачів відбувається через вебдодаток. Після відкриття платформи, користувач потрапляє на головну сторінку (рис. 3.2 – рис. 3.3), де його зустрічає сторінка зі стравами, де за допомогою ключових слів та фільтрів дозволяється здійснювати пошук страви. Усі страви виводяться у вигляді карток, які містять обкладинку, назву, короткий опис, кількість порцій, вподобайок, коментарів та автора страви. При наведенні на картку з'являється кнопка для детального перегляду інформації про обрану страву.

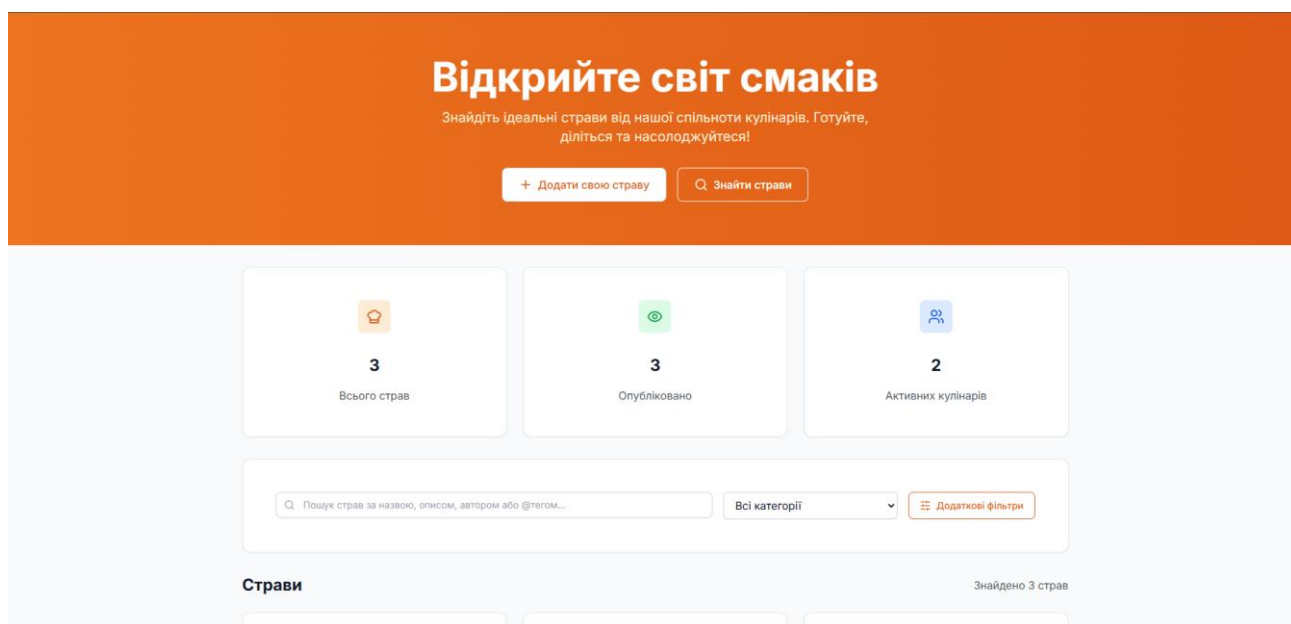


Рис. 3.2 – Головна сторінка платформи

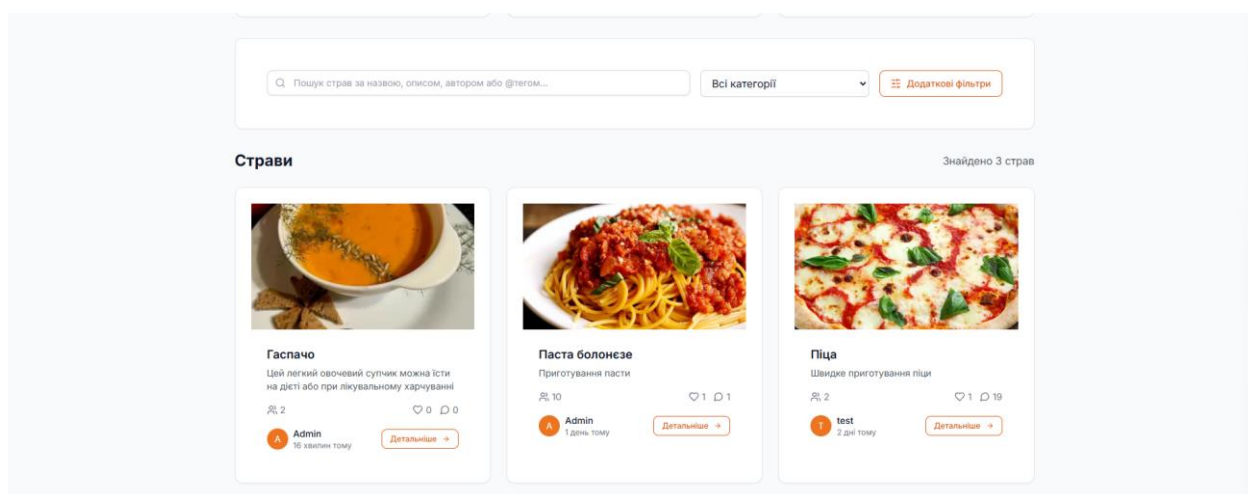


Рис. 3.3 – Головна сторінка платформи

Після обрання страви користувач може перейти до модального вікна страви, де може переглянути більш детальну інформацію про страву.

Авторизований користувач може залишити коментар, поставити вподобайку та поділитися з іншими (рис 3.4 – 3.5)

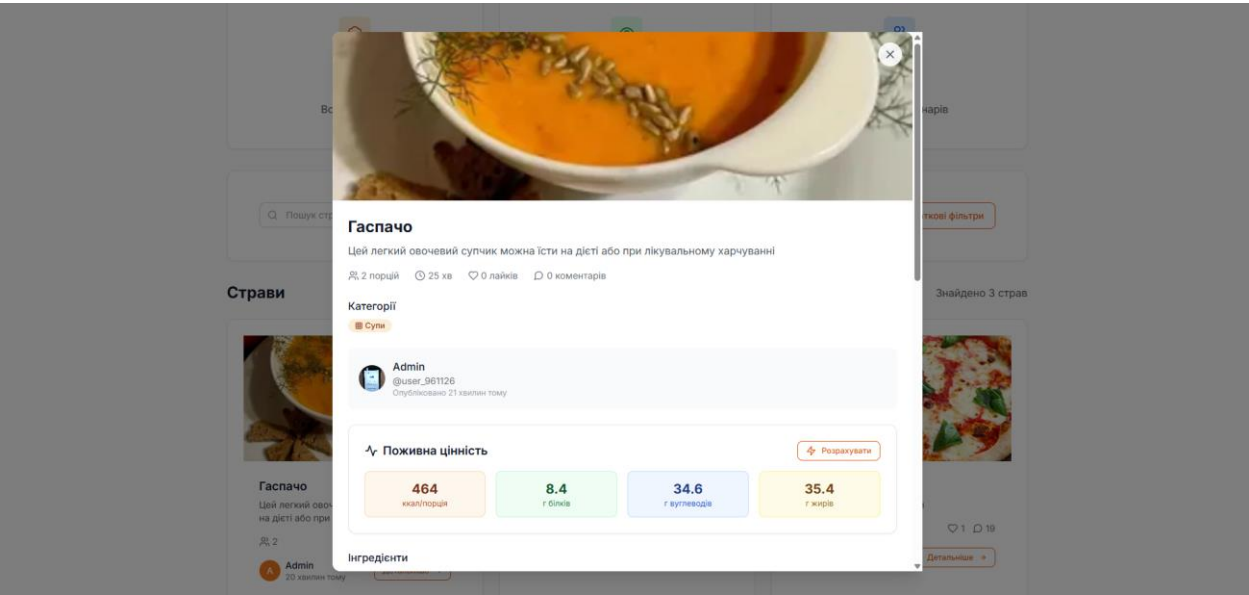


Рис. 3.4 – Модальне вікно страви

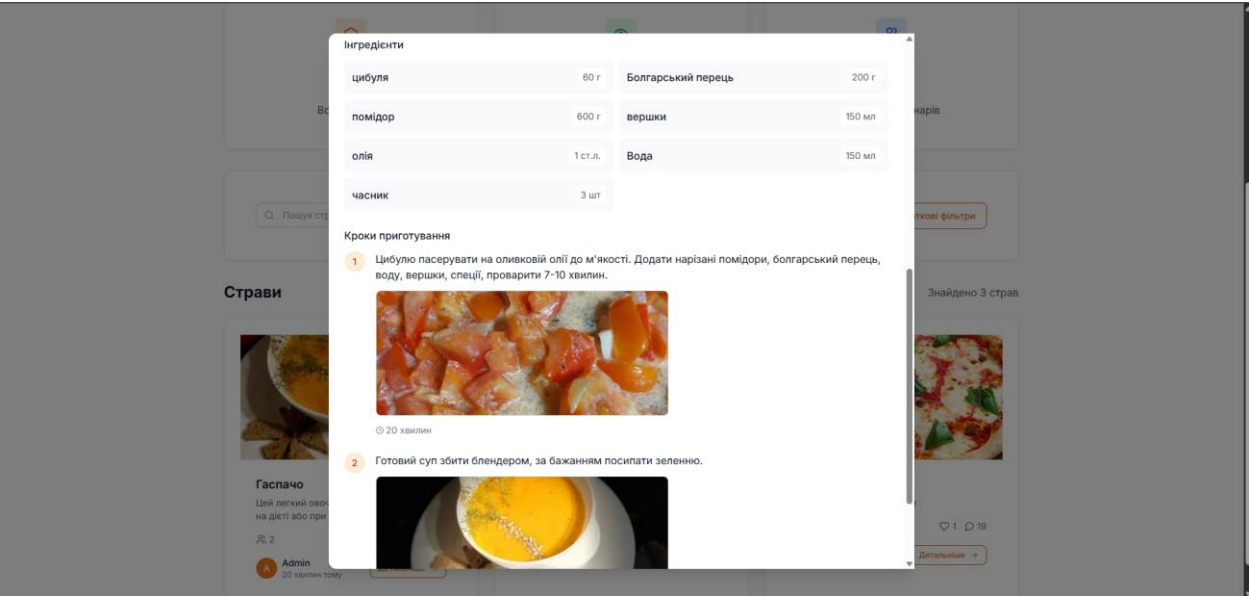


Рис. 3.5 - Модальне вікно страви

Для отримання більших можливостей платформи користувачу необхідно увійти або зареєструвати аккаунт. Відкривши сторінку, буде запропоновано увійти до створеного аккаунту або зареєструвати його. На сторінці з реєстрацією відривається форма з полями для введення даних: повне ім'я, адреса електронної пошти та пароль. Сторінка входу реалізовано окремо – користувач вводить пошту та пароль, або скористатися функцією відновлення паролю (рис. 3.6 – рис. 3.9).

Створіть ваш акаунт
Або увійдіть до існуючого акаунту

Регістрація

Повне ім'я
Введіть ваше повне ім'я

Адреса електронної пошти
Введіть вашу електронну пошту

Пароль
Введіть ваш пароль

Підтвердіть пароль
Підтвердіть ваш пароль

Створити акаунт

Після реєстрації:

- Ви отримаєте вітальний лист на вашу електронну пошту
- Перевірте папку "Спам", якщо не бачите листа

Рис. 3.6 – Сторінка реєстрації

Увійдіть до вашого акаунту
Або створіть новий акаунт

Вхід

Адреса електронної пошти
Введіть вашу електронну пошту

Пароль
Введіть ваш пароль

[Забули пароль?](#) [Повторити підтвердження](#)

Увійти

Проблеми з входом?

- Переконайтеся, що ви підтвердили електронну пошту
- Перевірте папку "Спам" для листа підтвердження
- Переконайтеся, що пароль введено правильно
- Надіслати лист підтвердження повторно

Рис. 3.7 - Сторінка входу

Відновлення пароля

Скидання пароля

Адреса електронної пошти
Введіть вашу електронну пошту

Надіслати інструкції для скидання пароля

[← Повернутися до входу](#)

Рис. 3.8 – Сторінка відновлення паролю

Після авторизації, користувачу відкривається профіль, де можна побачити статистику, дані користувача, переглянути персональні колекції та страви (рис 3.9).

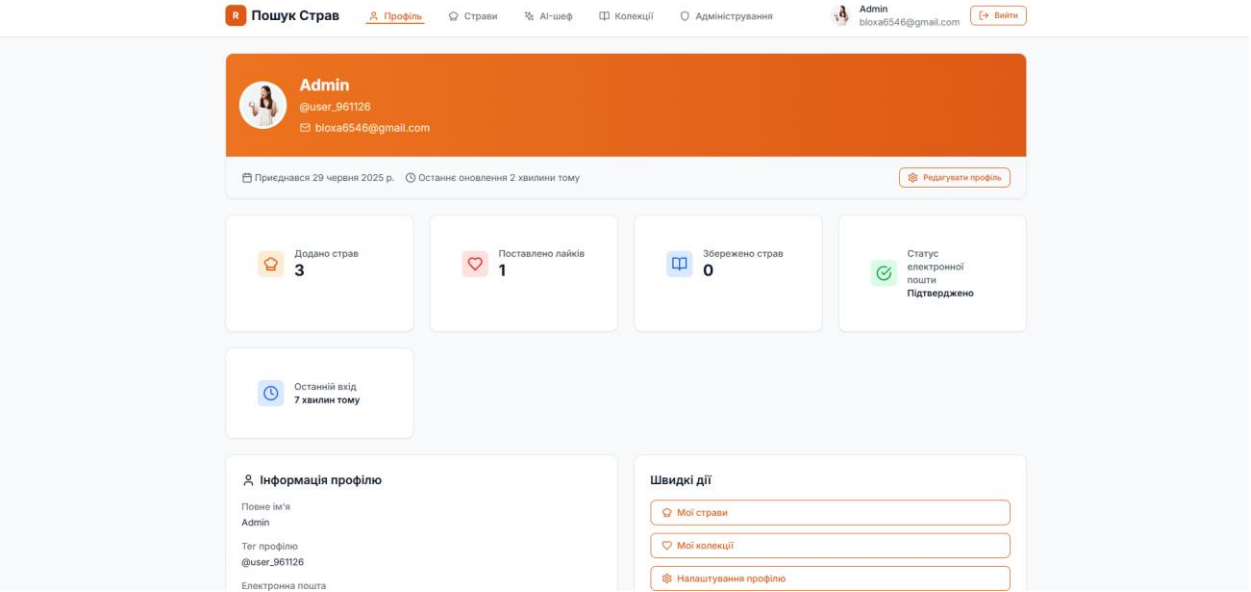


Рис. 3.9 – Сторінка профілю користувача

Також при натисканні кнопки для редагування профілю, перейти до сторінки де користувач може оновити дані профілю, змінити фото профілю, ім'я, тег та змінити пароль (рис. 3.10).

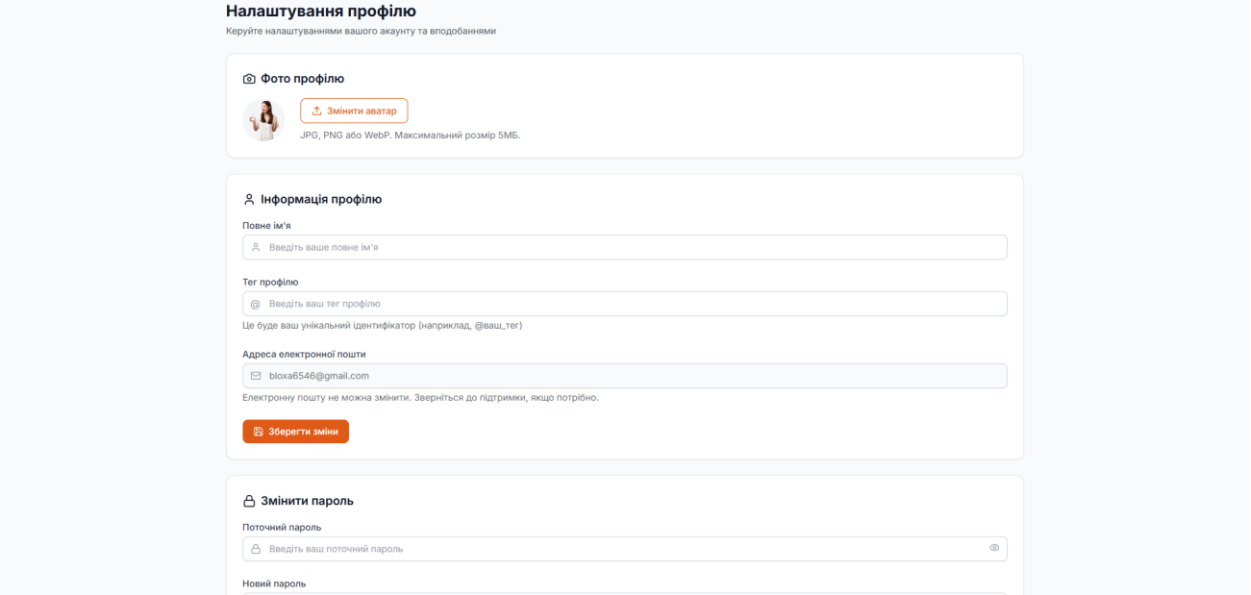


Рис. 3.10 – Сторінка редагування профілю

Перейшовши на сторінку з власними страми, користувач може переглянути стави які додав, переглянути детальну інформацію, видали та додати страву (рис. 3.11).

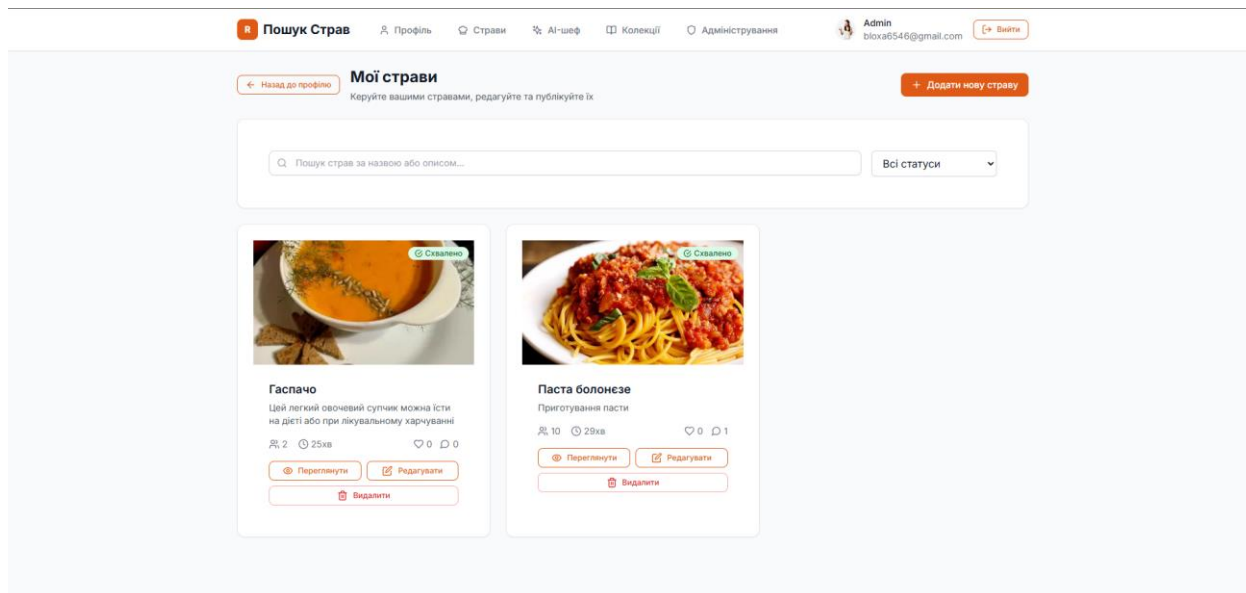


Рис. 3.11 – Сторінка страв користувача

На рисунку 3.12 – 3.14 представлено сторінку додавання страви, відривається форма для додавання страви, де користувач вводить назву страви, опис, кількість порцій, зображення страви, категорії, інгредієнти, кроки приготування.

Рис. 3.12 – Сторінка додавання страви

Рис. 3.13 – Сторінка додавання страви

Кроки приготування *

1 Крок 1

Опис кроку *

Детально опишіть що потрібно зробити на цьому кроці

Час виконання (хвилини)

0

Зображення кроку (необов'язково)

Завантажити зображення для цього кроку

Перетягніть файл сюди або натисніть для вибору

Обрати файл

JPG, PNG або WebP. Максимум 10MB.

Скасувати

Створити страву

Рис. 3.14 – Сторінка додавання страви

Після створення страви, перейшовши до страв користувача, користувач може побачити, що страва додалась і її статус “Чернетка”, тобто тільки користувач може її бачити. Для того, щоб інші користувачі змоги побачити страву потрібно надіслати адміністраторам для перевіри. Після того як адміністратор підтвердить, страва з`явиться для усіх користувачів (рис. 3.15).

Рис. 3.15 – Результат додавання страви

Рис. 3.16 – Сторінка колекцій

На рисунку 3.16. зображено сторінку з колекціями користувача. Користувач може додати, редагувати, видаляти колекції. Натиснувши

ІПЗ.КР.Б – 121 – 25 - ПЗ

46

“Створити колекцію”, відкривається форма для додавання колекції. Після додавання, з`являється колекція, яку можна переглянути, редагувати, виконати пошук, додати та видалити страви з колекції.

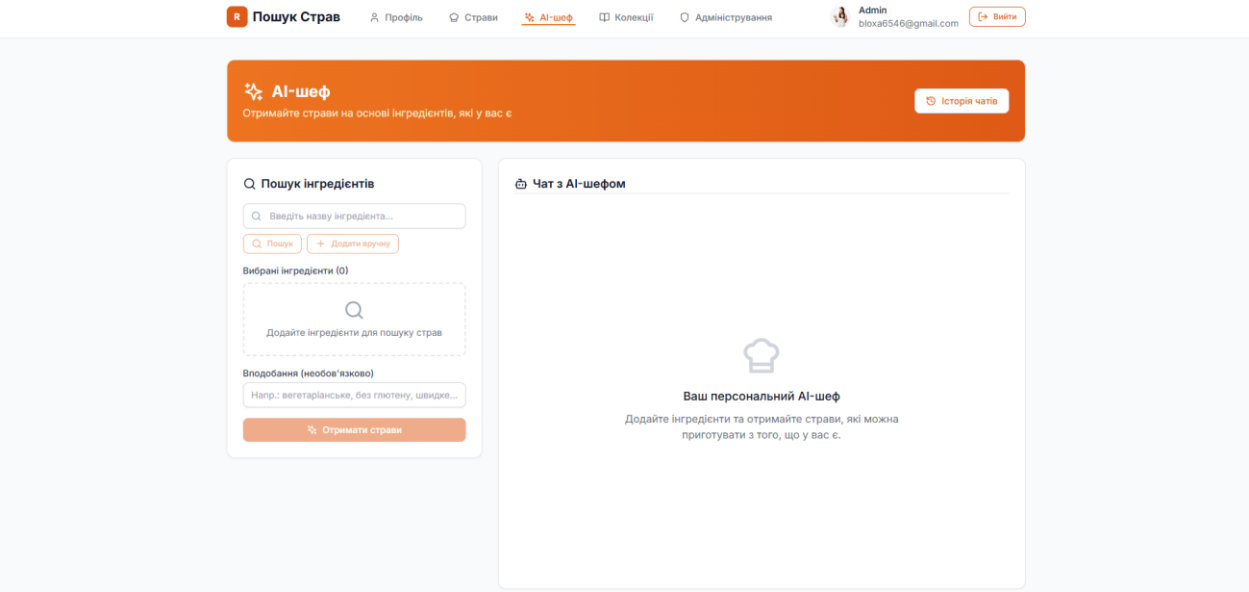


Рис. 3.17 – Сторінка ІШ-чату

На рисунку 3.17 зображено сторінку з ІШ-чатом. Користувач може ввести наявні інгредієнти та вподобання і натиснувши “Отримати страви”, ІШ-асистент згенерує декілька став, які можна приготувати з наявних інгредієнтів.

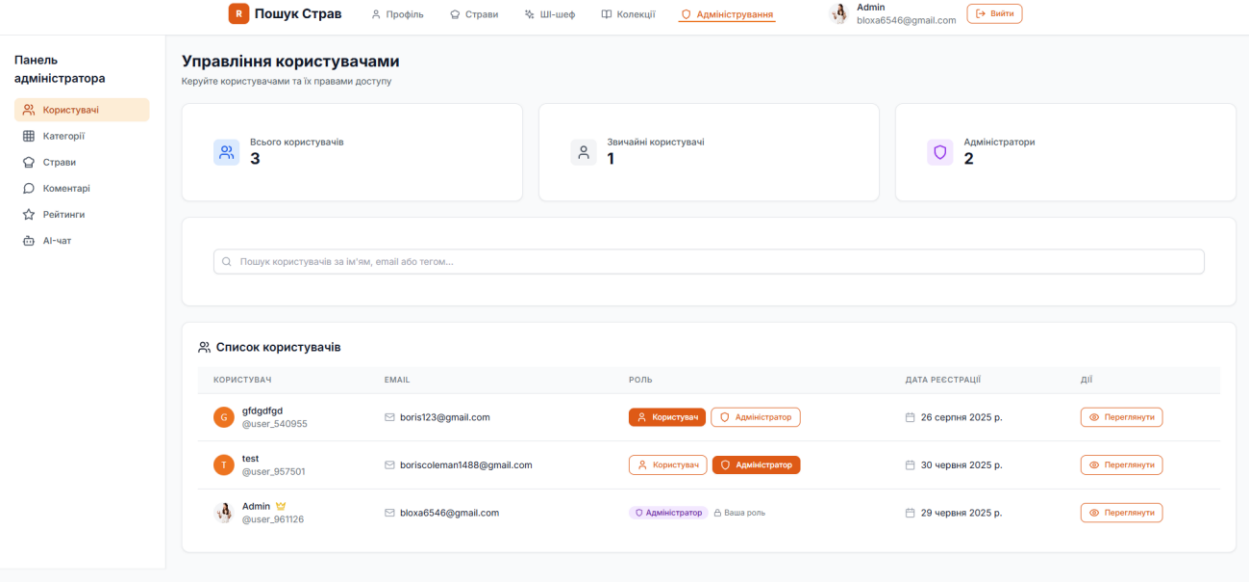


Рис. 3.18 – Сторінка керування користувачами

На рисунку 3.18 зображено сторінку керування користувачами для адміністраторів. Дана сторінка дозволяє адміністратору переглядати кількості користувачів, змінювати роль, переглядати детальну інформацію, видаляти користувача. Також розміщено пошук для швидкого знаходження потрібного

користувача. У таблиці зображується користувач, пошту, роль, дата реєстрації та кнопка для дій над користувачем.

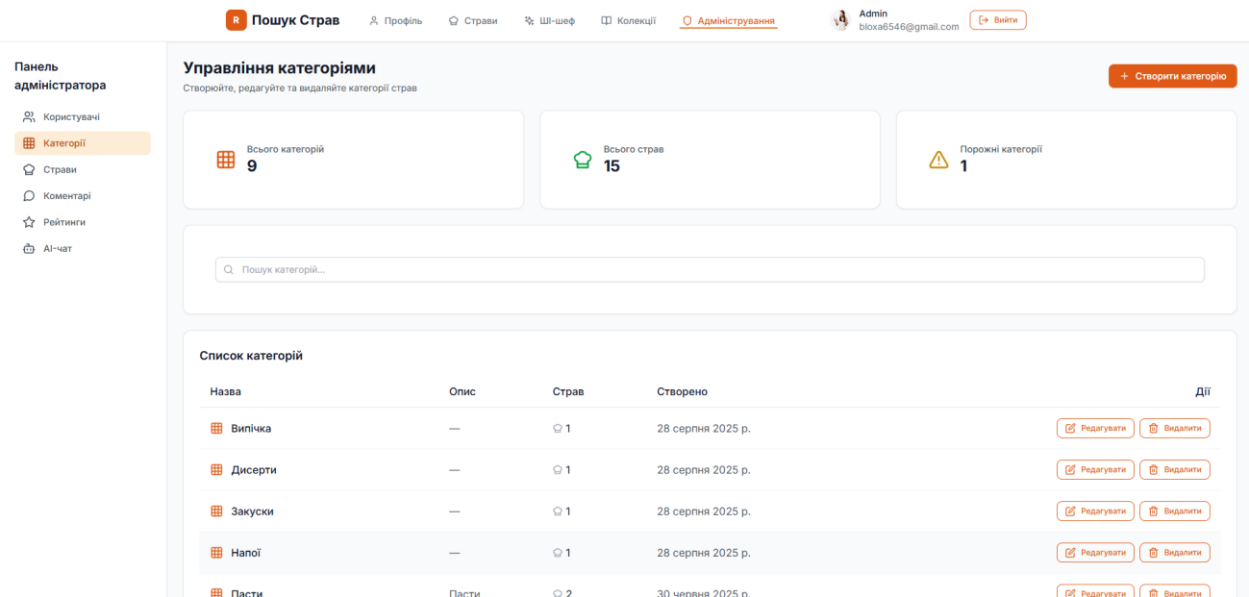


Рис. 3.19 – Сторінка керування категоріями

На рисунку 3.19 зображено сторінку керування категоріями для адміністраторів. Дана сторінка дозволяє адміністратору додати, редагувати та видаляти та переглядати кількість категорій. Також розміщено пошук для швидкого знаходження потрібної категорії. У таблиці зображується назва, опис, кількість страв, дату створення та дії над категорією.

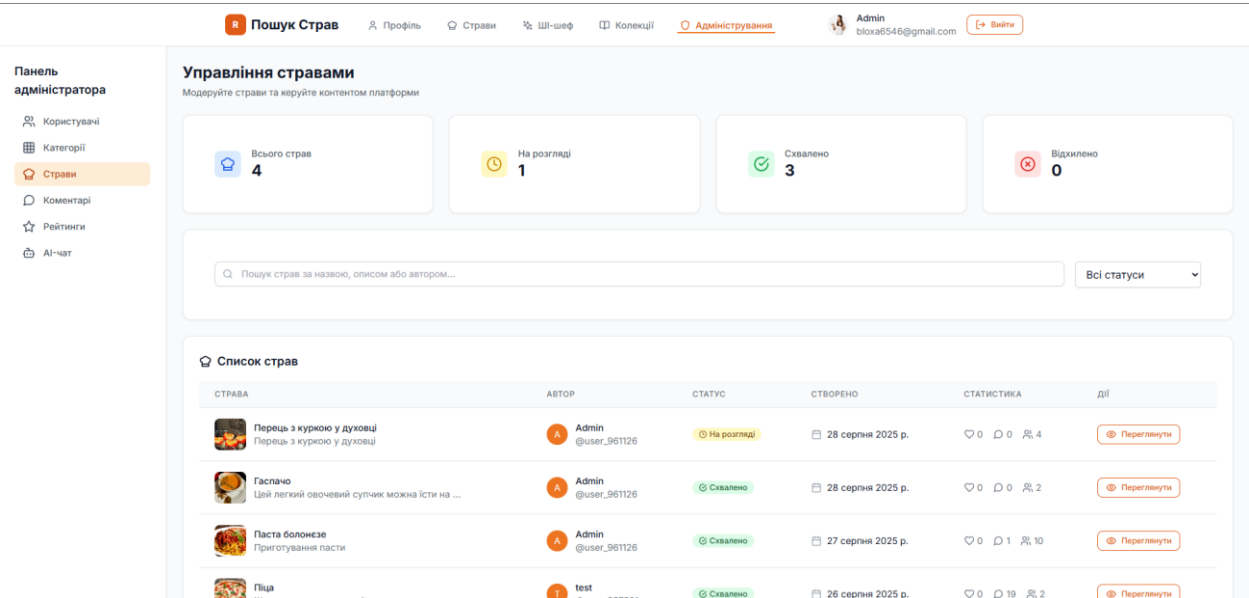


Рис. 3.20 – Сторінка керування стравами

На рисунку 3.20 зображено сторінку керування стравами для адміністраторів. Дана сторінка дозволяє адміністратору модерувати та

переглянути статистику страв, переглянути детальну інформацію про страву, кількість вподобайок та коментарів, інгредієнти, кроки та категорії. Також розміщено пошук для швидкого знаходження потрібної страви. У таблиці зображується фото, назва, автор, статус, дата створення, статистика страви та дії над стравою.

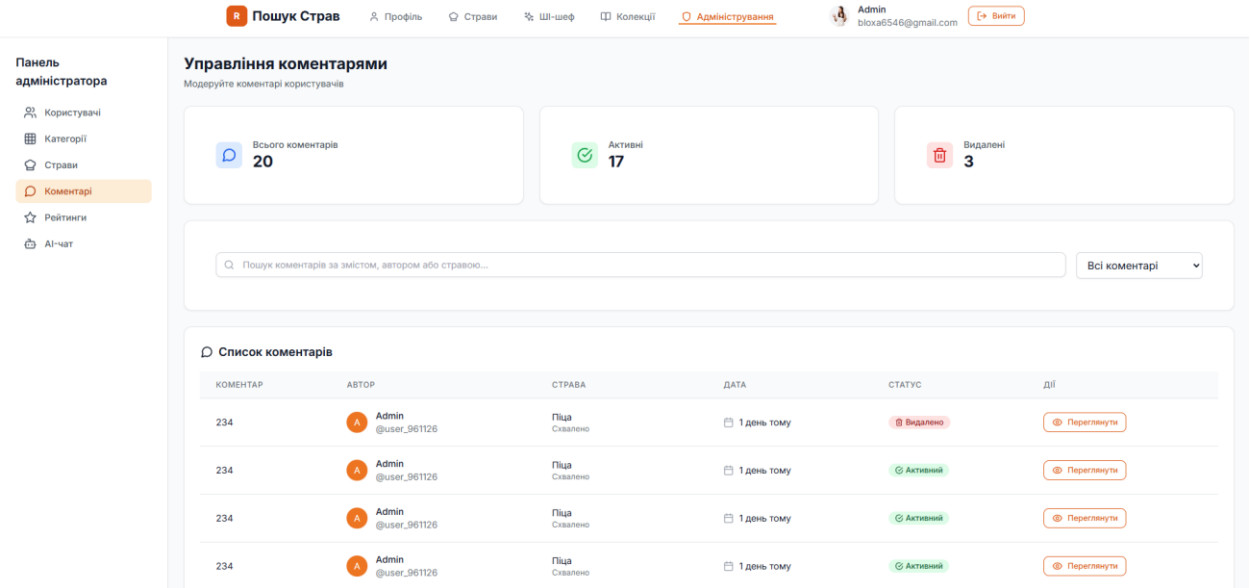


Рис. 3.21 – Сторінка керування коментарями.

На рисунку 3.21 зображено сторінку керування коментарями для адміністраторів. Дана сторінка дозволяє адміністратору переглянути кількість коментарів, детальну інформацію, видаляти та відновлювати коментар. Також розміщено пошук для швидкого знаходження потрібного коментаря. У таблиці зображується назва, опис, кількість страв, дату створення та дії над категорією.

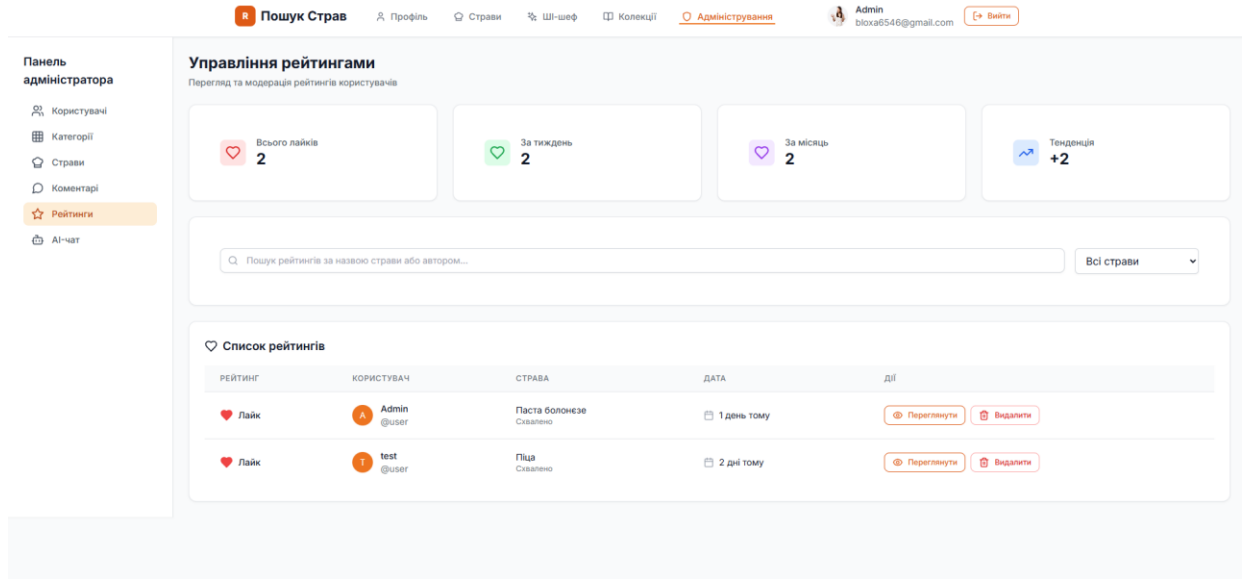


Рис. 3.22 – Сторінка керування рейтингом

На рисунку 3.22 зображено сторінку керування рейтингом для адміністраторів. Дана сторінка дозволяє адміністратору переглянути кількість та детальну інформацію рейтингу. Також розміщено пошук для швидкого знаходження потрібного рейтингу. У таблиці зображується рейтинг, автор, страва, дата та дії над рейтингом.

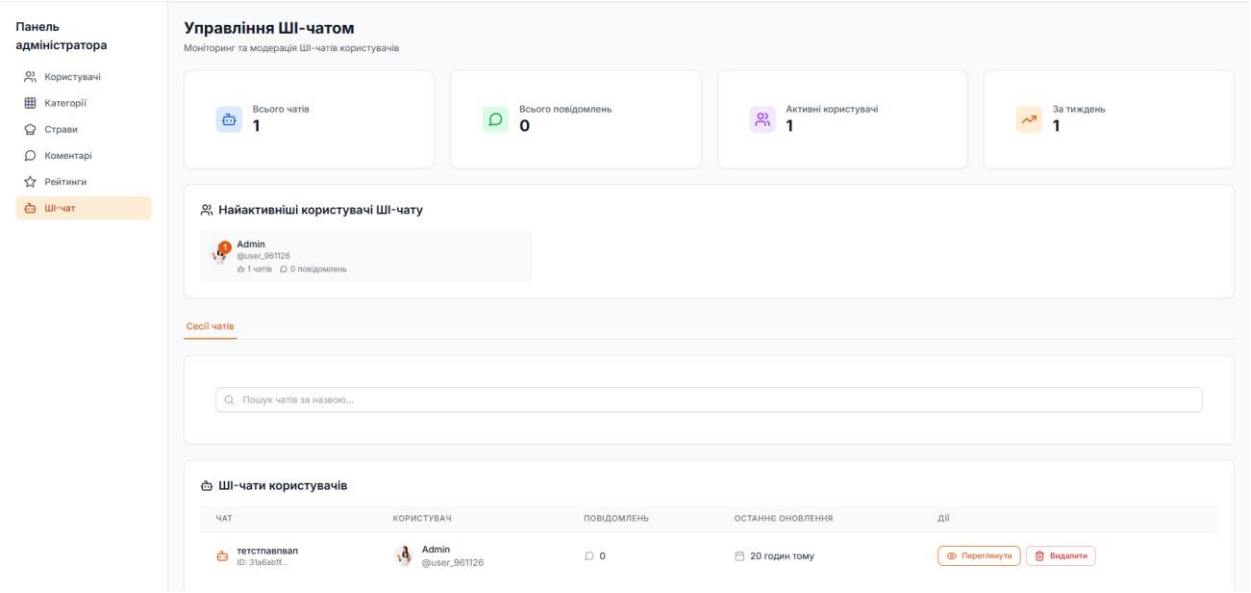


Рис. 3.23 – Сторінка керування ШІ-чатами

На рисунку 3.23 зображено сторінку керування ШІ-чатами для адміністраторів. Дана сторінка дозволяє адміністратору переглянути кількість, детальну інформацію та видалити чат. Також розміщено пошук для швидкого знаходження потрібного чату. У таблиці зображується назва, користувач, кількість повідомлень, останнє оновлення та дії над чатом.

3.3. Тестування роботи програмного продукту

Після завершення розробки платформи для пошуку страв, було проведено тестування, з метою перевірки коректності функціонування всіх компонентів системи, забезпечення стабільності роботи, виявлення та усунення помилок. Процес тестування охоплював клієнтську та серверну частини, взаємодію з базою даних, інтеграцію зі штучним інтелектом Gemini API та зовнішніми сервісами.

Функціональні перевірки включали наступні сценарії: реєстрація та вхід користувача, авторизація адміністратора, пошук і фільтрація страв, перегляд детальної інформації про страви, коментування та оцінювання страв,

створення та управління колекціями, взаємодія з ШІ-чатом для отримання кулінарних рекомендацій, розрахунок харчової цінності страв, і модерація контенту адміністратором.

Також було протестовано захист приватних маршрутів, обмеження прав доступу до адміністративного функціоналу, адаптивність інтерфейсу на різних пристроях, коректне відображення сторінок у популярних браузерях і відсутність критичних помилок при різному навантаженні.

Усі тести проводились за структурованими чек-листами, де фіксувалися: функція, кроки перевірки, очікуваний результат та фактичний статус. Це дозволило структуровано оцінити якість реалізації програмного продукту та виявити незначні недоліки, усунені на завершальному етапі розробки.

Таблиця 3.1

Результати тестування основних функцій платформи для пошуку страв

№ тесту	Функціональність	Кроки перевірки	Очікуваний результат	Статус
1	2	3	4	5
1	Реєстрація користувача	Відкрити сторінку реєстрації, заповнити форму (повне ім'я, email, пароль), натиснути "Зареєструватися"	Користувача створено, надіслано лист підтвердження	Пройдено
2	Авторизація користувача	Ввести email і пароль, натиснути "Увійти"	Перехід до кабінету, ініціалізація сесії	Пройдено
3	Пошук страв	Ввести назву страви в поле пошуку, натиснути "Знайти"	Відображення списку страв, що відповідають запиту	Пройдено
4	Фільтрація за категоріями	Обрати категорію з випадаючого списку, застосувати фільтр	Відображення страв тільки з обраної категорії	Пройдено
5	Перегляд детальної інформації про страву	Натиснути на картку страви	Відкриття детальної сторінки з інгредієнтами, інструкціями, харчовою цінністю	Пройдено
6	Розрахунок харчової цінності	Вказати кількість порцій, натиснути "Розрахувати"	Відображення калорійності, білків, жирів, вуглеводів на задану кількість порцій	Пройдено

Продовження таблиця 3.1

1	2	3	4	5
7	Взаємодія з ШІ-асистентом	Ввести наявні інгредієнти в чат, натиснути "Отримати страви"	ШІ генерує список рекомендованих страв з наявних інгредієнтів	Пройдено
8	Створення персональної колекції	Натиснути "Створити колекцію", ввести назву, натиснути "Зберегти"	Колекція створена та відображається в особистому кабінеті	Пройдено
9	Додавання страви до колекції	Обрати страву, натиснути "Додати до колекції", вибрати колекцію	Страва додана до обраної колекції, відображається в списку	Пройдено
10	Оцінювання страви	Натиснути кнопку "подобається"	Оцінка зафіксована, оновлено загальний рейтинг страви	Пройдено
11	Коментування страви	Ввести коментар в текстове поле, натиснути "Відправити"	Коментар опублікований, відображається під стравою	Пройдено
12	Додавання власної страви	Заповнити форму з назвою, інгредієнтами, інструкціями, завантажити фото	Страва створена, направлена на модерацію	Пройдено

Таблиця 3.2

Результати тестування адміністративної панелі

№ тесту	Функціональність	Кроки перевірки	Очікуваний результат	Статус
1	2	3	4	5
1	Авторизація користувача	Ввести email і пароль, натиснути "Увійти"	Перехід до кабінету, ініціалізація сесії	Пройдено
2	Вхід до адмін-панелі	Авторизуватися з правами адміністратора, перейти в розділ управління	Відкрито меню адміністратора	Пройдено
3	Пошук страв	Ввести назву страви в поле пошуку, натиснути "Знайти"	Відображення списку страв, що відповідають запиту	Пройдено
4	Фільтрація за категоріями	Обрати категорію з випадаючого списку, застосувати фільтр	Відображення страв тільки з обраної категорії	Пройдено

1	2	3	4	5
5	Перегляд детальної інформації про страву	Натиснути на картку страви	Відкриття детальної сторінки з інгредієнтами, інструкціями, харчовою цінністю	Пройдено
6	Розрахунок харчової цінності	Вказати кількість порцій, натиснути "Розрахувати"	Відображення харчової цінності на задану кількість порцій	Пройдено
7	Взаємодія з ІІІ-асистентом	Ввести наявні інгредієнти в чат, натиснути "Отримати страви"	ІІІ генерує список рекомендованих страв з наявних інгредієнтів	Пройдено
8	Створення персональної колекції	Натиснути "Створити колекцію", ввести назву, натиснути "Зберегти"	Колекція створена та відображається в особистому кабінеті	Пройдено

Висновки до третього розділу

У третьому розділі було побудовано та наведено діаграму розгортання, яка відображає взаємодію між клієнтською та серверною частинами, базою даних і зовнішніми сервісами. Було розглянуто особливості інтерфейсу та порядок роботи з платформою для пошуку страв

Проаналізовано структуру інтерфейсу, яка передбачає зручну навігацію, можливість пошуку та фільтрації страв, управління колекціями й інтеграцію з ІІІ-асистентом для отримання рекомендацій.

Проведене тестування функціональності показали стабільність роботи та відповідність заявленим вимогам.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи бакалавра було спроектовано та реалізовано платформу для пошуку страв, було проведено аналіз аналогічних рішень, визначено їхні переваги та недоліки. На основі результатів досліджень та аналізу були визначені вимоги для платформи, яка розроблялася. Обґрунтовано вибір клієнт-серверної архітектури, яка забезпечує чіткий розподіл між інтерфейсом користувача та бізнес-логікою, гнучкість у розширенні функціоналу та зручність у підтримці.

Було спроектовано та реалізовано базу даних, яка забезпечує структуроване зберігання інформації про страви, інгредієнти, користувачів, коментарі та рейтинги. Також реалізовано механізм зберігання медіафайлів, що дозволяє користувачам завантажувати зображення страв.

На основі отриманих результатів обґрунтовано вибір стеку технологій. Для реалізації клієнтської частини обрано Next.js, React, TypeScript і Tailwind CSS, що дозволило створити швидкий та адаптивний інтерфейс. Серверна частина побудована на Node.js і Fastify із використанням Supabase як базу даних.

Також показано та описано порядок роботи з платформою, розглянуто структуру інтерфейсу та проведено тестування основного функціоналу. Додатково розроблено діаграму розгортання, яка демонструє взаємодію між компонентами системи та зовнішніми сервісами.

Результатом кваліфікаційної роботи є готова до експлуатації платформа для пошуку страв, яка поєднує інструменти аналізу харчової цінності, рекомендації від штучного інтелекту та зручний інтерфейс.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. YABPOELA [Електронний ресурс] – Режим доступу до ресурсу:
<https://ua.yabpoela.net/>
2. Cookorama [Електронний ресурс] – Режим доступу до ресурсу:
<https://cookorama.net/uk/>
3. Cookpad [Електронний ресурс] – Режим доступу до ресурсу:
<https://cookpad.com/ua>
4. Smachno.ua [Електронний ресурс] - Режим доступу до ресурсу:
<https://smachno.ua/ua/>
5. Плакса І. О. Архітектура клієнт-сервер: огляд та її застосування в сучасних обчисленнях / І. О. Плакса, В. М. Федорченко // Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління : тези доп. 13-ї міжнар. наук.-техн. конф., 26-27 квітня 2023 р., Баку–Харків–Жиліна : [у 2 т.]. Т. 2 : секція 2 / Нац. ун-т оборони Азербайджанської Республіки [та ін.]. – Харків : Impress, 2023. – С. 8
6. JavaScript and jQuery: Interactive Front-End Web Development [Електронний ресурс] - Режим доступу до ресурсу: <https://www.javascriptbook.com/>
7. REST API [Електронний ресурс] - Режим доступу до ресурсу:
<https://robotdreams.cc/uk/blog/466-vstup-do-rest-api-restful-vebservisi>
8. Документація Node.js [Електронний ресурс] - Режим доступу до ресурсу:
<https://nodejs.org/docs/latest/api/>
9. Документація Fastify [Електронний ресурс] - Режим доступу до ресурсу:
<https://fastify.dev/docs/latest/>
10. Документація Tailwind CSS [Електронний ресурс] - Режим доступу до ресурсу: <https://tailwindcss.com/docs/installation/framework-guides>
11. Документація Next.js [Електронний ресурс] - Режим доступу до ресурсу:
<https://nextjs.org/docs>
12. Документація React [Електронний ресурс] - Режим доступу до ресурсу:
<https://react.dev/learn>
13. CSS [Електронний ресурс] – Режим доступу до ресурсу:
<https://developer.mozilla.org/en-US/docs/Web/CSS>

14. Архітектура SPA [Електронний ресурс] - Режим доступу до ресурсу:
<https://wezom.com.ua/ua/blog/chto-takoe-spa-prilozheniya>
15. Документація TypeScript [Електронний ресурс] - Режим доступу до ресурсу: <https://www.typescriptlang.org/docs/>
16. Supabase Auth [Електронний ресурс] - Режим доступу до ресурсу:
<https://supabase.com/docs/guides/auth>
17. Документація Supabase [Електронний ресурс] - Режим доступу до ресурсу:
<https://supabase.com/docs>
18. Документація Supabase Storage [Електронний ресурс] - Режим доступу до ресурсу: <https://supabase.com/docs/guides/storage>
19. Офіційна документація CORS (Cross-Origin Resource Sharing) [Електронний ресурс] – Режим доступу до ресурсу:
<https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/CORS>
20. Документація HTML5 [Електронний ресурс] – Режим доступу до ресурсу:
<https://developer.mozilla.org/en-US/docs/Glossary/HTML5>
21. Документація Edamam Food API [Електронний ресурс] – Режим доступу до ресурсу: <https://developer.edamam.com/food-database-api>
22. Документація Nutrition Analysis API [Електронний ресурс] – Режим доступу до ресурсу: <https://developer.edamam.com/edamam-nutrition-api>
23. Документація Nutrition Analysis API [Електронний ресурс] – Режим доступу до ресурсу: <https://deepmind.google/models/gemini/>
24. Документація Zod [Електронний ресурс] – Режим доступу до ресурсу:
<https://zod.dev/>
25. Документація React Hook Form [Електронний ресурс] – Режим доступу до ресурсу: <https://react-hook-form.com/>
26. Документація Nodemon [Електронний ресурс] – Режим доступу до ресурсу: <https://github.com/remy/nodemon#nodemon>
27. Документація JavaScript [Електронний ресурс] – Режим доступу до ресурсу: <https://developer.mozilla.org/en-US/docs/Web/JavaScript>
28. Документація PostgreSQL [Електронний ресурс] – Режим доступу до ресурсу: <https://www.postgresql.org/docs/current/>

29. Роберт М. Чиста архітектура: Мистецтво створення програмного забезпечення. Роберт М. [Видання друге] – Харків: Фабула, 2023 – 368 с
30. Документація JSON Web Token [Електронний ресурс] – Режим доступу до ресурсу: <https://jwt.io/introduction>
31. Документація Material-UI [Електронний ресурс] - Режим доступу до ресурсу: <https://mui.com/material-ui/>

ДОДАТКИ

					ІІЗ.КР.Б – 121 – 25 - ІІЗ	
						58

Лістинг dishService.js

```

export class DishService {
  constructor(supabase, logger, collectionService, supabaseAdmin) {
    this.supabase = supabase
    this.supabaseAdmin = supabaseAdmin
    this.logger = logger
    this.collectionService = collectionService

    this.DISH_STATUS = {
      DRAFT: 'draft',
      PENDING: 'pending',
      APPROVED: 'approved',
      REJECTED: 'rejected'
    }

    this.ERRORS = {
      DISH_NOT_FOUND: 'Dish not found',
      PERMISSION_DENIED: 'Permission denied',
      VALIDATION_ERROR: 'Validation error',
      DATABASE_ERROR: 'Database error'
    }

    this.ALLOWED_IMAGE_TYPES = ['image/jpeg', 'image/jpg', 'image/png', 'image/webp']
    this.MAX_IMAGE_SIZE = 10 * 1024 * 1024 // 10MB
  }

  async _validateDishExists(dishId, userId = null) {
    const query = this.supabase
      .from('dishes')
      .select('*')
      .eq('id', dishId)

    if (userId) {
      query.eq('user_id', userId)
    }
  }
}

```

```

const { data: dish, error } = await query.single()

if (error || !dish) {
  throw new Error(userId ? 'Dish not found or access denied' : 'Dish not found')
}

return dish
}

async _validateCategories(categoryIds) {
  if (!categoryIds || categoryIds.length === 0) return true

  const { data: categories, error } = await this.supabase
    .from('dish_categories')
    .select('id')
    .in('id', categoryIds)

  if (error) {
    throw new Error('Failed to validate categories')
  }

  if (categories.length !== categoryIds.length) {
    throw new Error('Some categories do not exist')
  }

  return true
}

_validateImageFile(fileBuffer, mimetype) {
  if (!this.ALLOWED_IMAGE_TYPES.includes(mimetype)) {
    throw new Error('Дозволені тільки файли JPG, PNG та WebP')
  }

  if (fileBuffer.length > this.MAX_IMAGE_SIZE) {
    throw new Error('Розмір зображення не повинен перевищувати 10МБ')
  }
}

```

```

async uploadDishImage(userId, fileBuffer, mimetype, originalFilename) {
  try {
    this._validateImageFile(fileBuffer, mimetype)

    this.logger.info('Starting dish image upload', {
      userId,
      originalFilename,
      fileSize: fileBuffer.length,
      mimetype
    })

    const fileExtension = mimetype.split('/')[1]

    const uniqueFilename = `${userId}-dish-${Date.now()}-${
      Math.random().toString(36).substring(2)}.${fileExtension}`

    const { data: uploadData, error: uploadError } = await this.supabaseAdmin.storage
      .from('dish-images')
      .upload(uniqueFilename, fileBuffer, {
        contentType: mimetype,
        upsert: false
      })

    if (uploadError) {
      this.logger.error('Dish image upload failed', {
        error: uploadError.message,
        userId,
        filename: uniqueFilename
      })
      return {
        success: false,
        error: 'Upload failed',
        message: 'Не вдалося завантажити зображення'
      }
    }

    const { data: urlData } = this.supabaseAdmin.storage
      .from('dish-images')
      .getPublicUrl(uniqueFilename)

```

```

if (!urlData.publicUrl) {
    return {
        success: false,
        error: 'Failed to get public URL',
        message: 'Не вдалося отримати посилання на зображення'
    }
}

this.logger.info('Dish image uploaded successfully', {
    userId,
    filename: uniqueFilename,
    publicUrl: urlData.publicUrl
})

return {
    success: true,
    imageUrl: urlData.publicUrl,
    filename: uniqueFilename,
    message: 'Зображення завантажено успішно'
}

} catch (error) {
    this.logger.error('Dish image upload error', {
        error: error.message,
        userId
    })
    return {
        success: false,
        error: 'Upload failed',
        message: error.message || 'Не вдалося завантажити зображення'
    }
}
}

async uploadStepImage(userId, fileBuffer, mimetype, originalFilename) {
    try {
        this._validateImageFile(fileBuffer, mimetype)
    }
}

```

```

this.logger.info('Starting step image upload', {
  userId,
  originalFilename,
  fileSize: fileBuffer.length,
  mimetype
})

const fileExtension = mimetype.split('/')[1]

const uniqueFilename = `${userId}-step-${Date.now()}-${
  Math.random().toString(36).substring(2)}.${fileExtension}`

const { data: uploadData, error: uploadError } = await this.supabaseAdmin.storage
  .from('dish-images')
  .upload(uniqueFilename, fileBuffer, {
    contentType: mimetype,
    upsert: false
  })

if (uploadError) {
  this.logger.error('Step image upload failed', {
    error: uploadError.message,
    userId,
    filename: uniqueFilename
  })
  return {
    success: false,
    error: 'Upload failed',
    message: 'Не вдалося завантажити зображення кроку'
  }
}

const { data: urlData } = this.supabaseAdmin.storage
  .from('dish-images')
  .getPublicUrl(uniqueFilename)

if (!urlData.publicUrl) {
  return {

```

```

        success: false,
        error: 'Failed to get public URL',
        message: 'Не вдалося отримати посилання на зображення'
    }
}

this.logger.info('Step image uploaded successfully', {
    userId,
    filename: uniqueFilename,
    publicUrl: urlData.publicUrl
})

return {
    success: true,
    imageUrl: urlData.publicUrl,
    filename: uniqueFilename,
    message: 'Зображення кроку завантажено успішно'
}

} catch (error) {
    this.logger.error('Step image upload error', {
        error: error.message,
        userId
    })
    return {
        success: false,
        error: 'Upload failed',
        message: error.message || 'Не вдалося завантажити зображення кроку'
    }
}
}

async getDishes() {
    try {
        const { data: dishes, error } = await this.supabase
            .from('dishes')
            .select(`
                *,

```

```

        profiles:user_id(full_name, profile_tag, avatar_url),
        dish_category_relations(
            dish_categories(id, name)
        ),
        dish_ratings(id, rating),
        dish_comments(id)
    `)
    .eq('status', this.DISH_STATUS.APPROVED)
    .order('created_at', { ascending: false })

if (error) {
    this.logger.error('Public dishes fetch error', { error: error.message })
    return {
        success: false,
        error: 'Database error',
        message: 'Unable to fetch dishes'
    }
}

const processedDishes = (dishes || []).map(dish => ({
    ...dish,
    categories: dish.dish_category_relations?.map(rel => rel.dish_categories) || [],
    ratings: dish.dish_ratings || [],
    comments_count: dish.dish_comments?.length || 0
})))

processedDishes.forEach(dish => {
    delete dish.dish_category_relations
    delete dish.dish_ratings
    delete dish.dish_comments
})

return {
    success: true,
    dishes: processedDishes,
    total: processedDishes.length
}
} catch (error) {

```

```

    this.logger.error('Public dishes fetch error', { error: error.message })
    return {
      success: false,
      error: 'Internal server error',
      message: 'Unable to fetch dishes'
    }
  }
}

```

```

async getUserDishes(userId) {
  try {
    const { data: dishes, error } = await this.supabase
      .from('dishes')
      .select(`
        *,
        profiles:user_id(full_name, email, profile_tag),
        dish_category_relations(
          dish_categories(id, name)
        ),
        dish_ratings(id, rating),
        dish_comments(id),
        dish_ingredients(*),
        dish_steps(*)
      `)
      .eq('user_id', userId)
      .order('created_at', { ascending: false })

    if (error) {
      this.logger.error('User dishes fetch error', { error: error.message })
      return {
        success: false,
        error: 'Database error',
        message: 'Unable to fetch user dishes'
      }
    }
  }
}

```

```

const processedDishes = (dishes || []).map(dish => ({
  ...dish,

```

```

        categories: dish.dish_category_relations?.map(rel => rel.dish_categories) || [],
        ratings: dish.dish_ratings || [],
        comments_count: dish.dish_comments?.length || 0,
        ingredients: dish.dish_ingredients || [],
        steps: dish.dish_steps?.sort((a, b) => a.step_number - b.step_number) || []
    )))

    processedDishes.forEach(dish => {
        delete dish.dish_category_relations
        delete dish.dish_ratings
        delete dish.dish_comments
        delete dish.dish_ingredients
        delete dish.dish_steps
    })

    return {
        success: true,
        dishes: processedDishes,
        total: processedDishes.length
    }
} catch (error) {
    this.logger.error('User dishes fetch error', { error: error.message })
    return {
        success: false,
        error: 'Internal server error',
        message: 'Unable to fetch user dishes'
    }
}
}

async getDishById(dishId) {
    try {
        const { data: dish, error } = await this.supabase
            .from('dishes')
            .select(`
                *,
                profiles:user_id(full_name, profile_tag, avatar_url),
                dish_category_relations(

```

```

        dish_categories(id, name)
    ),
    dish_ratings(id, rating, user_id),
    dish_ingredients(*),
    dish_steps(*)
`)
.eq('id', dishId)
.eq('status', this.DISH_STATUS.APPROVED)
.single()

if (error) {
    this.logger.error('Dish fetch error', { error: error.message })
    return {
        success: false,
        error: 'Dish not found',
        message: error.message
    }
}

const processedDish = {
    ...dish,
    categories: dish.dish_category_relations?.map(rel => rel.dish_categories) || [],
    ratings: dish.dish_ratings || [],
    ingredients: dish.dish_ingredients || [],
    steps: dish.dish_steps?.sort((a, b) => a.step_number - b.step_number) || []
}

delete processedDish.dish_category_relations
delete processedDish.dish_ratings
delete processedDish.dish_ingredients
delete processedDish.dish_steps

return {
    success: true,
    dish: processedDish
}
} catch (error) {
    this.logger.error('Dish fetch error', { error: error.message })

```

```

    return {
      success: false,
      error: 'Internal server error',
      message: 'Unable to fetch dish'
    }
  }
}

async getDishByIdForUser(dishId, userId) {
  try {
    const { data: dish, error } = await this.supabase
      .from('dishes')
      .select(`
        *,
        profiles:user_id(full_name, profile_tag, avatar_url),
        dish_category_relations(
          dish_categories(id, name)
        ),
        dish_ratings(id, rating, user_id),
        dish_ingredients(*),
        dish_steps(*)
      `)
      .eq('id', dishId)
      .eq('user_id', userId)
      .single()

    if (error) {
      this.logger.error('User dish fetch error', { error: error.message, dishId, userId })
      return {
        success: false,
        error: 'Dish not found',
        message: 'Dish not found or you do not have permission to access it'
      }
    }

    const processedDish = {
      ...dish,
      categories: dish.dish_category_relations?.map(rel => rel.dish_categories) || [],

```

```

        ratings: dish.dish_ratings || [],
        ingredients: dish.dish_ingredients || [],
        steps: dish.dish_steps?.sort((a, b) => a.step_number - b.step_number) || []
    }

    delete processedDish.dish_category_relations
    delete processedDish.dish_ratings
    delete processedDish.dish_ingredients
    delete processedDish.dish_steps

    return {
        success: true,
        dish: processedDish
    }
} catch (error) {
    this.logger.error('User dish fetch error', { error: error.message, dishId, userId })
    return {
        success: false,
        error: 'Internal server error',
        message: error.message || 'Unable to fetch dish'
    }
}
}

async_createDishRelations(dishId, { category_ids, ingredients, steps }) {
    const operations = []

    if (category_ids && category_ids.length > 0) {
        const categoryRelations = category_ids.map(categoryId => ({
            dish_id: dishId,
            category_id: categoryId
        })))

        operations.push(
            this.supabase
                .from('dish_category_relations')
                .insert(categoryRelations)
        )
    }
}

```

```

    }

    if (ingredients && ingredients.length > 0) {
      const ingredientsData = ingredients.map((ing) => ({
        dish_id: dishId,
        name: ing.name,
        amount: ing.amount,
        unit: ing.unit,
        edamam_food_id: ing.edamam_food_id
      })))

      operations.push(
        this.supabase
          .from('dish_ingredients')
          .insert(ingredientsData)
      )
    }

    if (steps && steps.length > 0) {
      const stepsData = steps.map((step, index) => ({
        dish_id: dishId,
        step_number: index + 1,
        description: step.description,
        image_url: step.image_url,
        duration_minutes: step.duration_minutes
      })))

      operations.push(
        this.supabase
          .from('dish_steps')
          .insert(stepsData)
      )
    }

    const results = await Promise.allSettled(operations)

    const errors = results
      .filter(result => result.status === 'rejected')

```

```

        .map(result => result.reason)

    if (errors.length > 0) {
        this.logger.error('Failed to create dish relations', { errors })
        throw new Error('Failed to create dish relations')
    }
}

async _deleteDishRelations(dishId) {
    const operations = [
        this.supabase.from('dish_ingredients').delete().eq('dish_id', dishId),
        this.supabase.from('dish_steps').delete().eq('dish_id', dishId),
        this.supabase.from('dish_category_relations').delete().eq('dish_id', dishId)
    ]

    await Promise.all(operations)
}

_handleError(error, context, defaultMessage = 'Internal server error') {
    this.logger.error(context, { error: error.message })
    return {
        success: false,
        error: defaultMessage,
        message: error.message
    }
}

_handleSuccess(message, data = {}) {
    return {
        success: true,
        message,
        ...data
    }
}

async createDish(userId, dishData) {
    try {
        const { title, description, category_ids, servings, ingredients, steps, main_image_url } = dishData
    }
}

```

```

await this._validateCategories(category_ids)

const { data: dish, error: dishError } = await this.supabase
  .from('dishes')
  .insert({
    user_id: userId,
    title,
    description,
    servings,
    main_image_url,
    status: this.DISH_STATUS.DRAFT
  })
  .select()
  .single()

if (dishError) {
  throw new Error(dishError.message)
}

await this._createDishRelations(dish.id, { category_ids, ingredients, steps })

if (this.collectionService) {
  await this.collectionService.addDishToSystemCollections(
    dish.id,
    userId,
    dish.status
  )
}

this.logger.info('Dish created successfully', {
  dishId: dish.id,
  userId,
  status: dish.status
})

return this._handleSuccess(
  'Dish saved as draft in your private collection',

```

```

    {
      dish: {
        id: dish.id,
        title: dish.title,
        status: dish.status
      }
    }
  )
} catch (error) {
  return this._handleError(error, 'Dish creation error', 'Unable to create dish')
}
}

async updateDish(dishId, userId, dishData) {
  try {
    await this._validateDishExists(dishId, userId)

    const { title, description, category_ids, servings, ingredients, steps, main_image_url } = dishData

    await this._validateCategories(category_ids)

    const { data: dish, error: dishError } = await this.supabase
      .from('dishes')
      .update({
        title,
        description,
        servings,
        main_image_url,
        status: this.DISH_STATUS.PENDING,
        updated_at: new Date().toISOString()
      })
      .eq('id', dishId)
      .select()
      .single()

    if (dishError) {
      throw new Error(dishError.message)
    }
  }
}

```

```

    await this._deleteDishRelations(dishId)

    await this._createDishRelations(dishId, { category_ids, ingredients, steps })

    this.logger.info('Dish updated successfully', { dishId, userId })

    return this._handleSuccess('Dish updated successfully', { dish })
  } catch (error) {
    return this._handleError(error, 'Dish update error', 'Unable to update dish')
  }
}

async updateDishStatus(dishId, userId, action) {
  try {
    const { data: existingDish, error: fetchError } = await this.supabase
      .from('dishes')
      .select('*')
      .eq('id', dishId)
      .eq('user_id', userId)
      .single()

    if (fetchError || !existingDish) {
      return {
        success: false,
        error: 'Dish not found',
        message: 'The specified dish does not exist or you do not have permission to edit it'
      }
    }

    let newStatus
    let message
    const oldStatus = existingDish.status

    switch (action) {
      case 'submit_for_review':
        newStatus = 'pending'
        message = 'Dish submitted for moderation'
        break

```

```

    case 'make_private':
        newStatus = 'draft'
        message = 'Dish moved to private collection'
        break
    default:
        return {
            success: false,
            error: 'Invalid action',
            message: 'Action must be "submit_for_review" or "make_private"'
        }
    }
}

const { data: dish, error: updateError } = await this.supabase
    .from('dishes')
    .update({
        status: newStatus,
        updated_at: new Date().toISOString()
    })
    .eq('id', dishId)
    .select()
    .single()

if (updateError) {
    this.logger.error('Dish status update failed', { error: updateError.message })
    return {
        success: false,
        error: 'Unable to update dish status',
        message: updateError.message
    }
}

if (this.collectionService) {
    await this.collectionService.updateDishInSystemCollections(
        dishId,
        userId,
        oldStatus,
        newStatus
    )
}

```

```

    }

    this.logger.info('Dish status updated successfully', {
      dishId,
      userId,
      action,
      newStatus
    })

    return {
      success: true,
      message,
      dish
    }
  } catch (error) {
    this.logger.error('Dish status update error', { error: error.message })
    return {
      success: false,
      error: 'Internal server error',
      message: 'Unable to update dish status'
    }
  }
}

async deleteDish(dishId, userId) {
  try {
    const { data: existingDish, error: fetchError } = await this.supabase
      .from('dishes')
      .select('*')
      .eq('id', dishId)
      .eq('user_id', userId)
      .single()

    if (fetchError || !existingDish) {
      return {
        success: false,
        error: 'Dish not found',
        message: 'The specified dish does not exist or you do not have permission to delete it'
      }
    }
  }
}

```

```

    }
  }

  this.logger.info('Starting dish deletion - cleaning up related records', { dishId, userId })

  const { error: commentsError } = await this.supabaseAdmin
    .from('dish_comments')
    .delete()
    .eq('dish_id', dishId)

  if (commentsError) {
    this.logger.error('Failed to delete dish comments', { dishId, error: commentsError.message })
    return {
      success: false,
      error: 'Database error',
      message: 'Unable to delete dish comments'
    }
  }

  const { error: ratingsError } = await this.supabaseAdmin
    .from('dish_ratings')
    .delete()
    .eq('dish_id', dishId)

  if (ratingsError) {
    this.logger.error('Failed to delete dish ratings', { dishId, error: ratingsError.message })
    return {
      success: false,
      error: 'Database error',
      message: 'Unable to delete dish ratings'
    }
  }

  const { error: collectionItemsError } = await this.supabaseAdmin
    .from('dish_collection_items')
    .delete()
    .eq('dish_id', dishId)

```

```

if (collectionItemsError) {
  this.logger.error('Failed to delete dish collection items', { dishId, error: collectionItemsError.message })
  return {
    success: false,
    error: 'Database error',
    message: 'Unable to delete dish from collections'
  }
}

const { error: ingredientsError } = await this.supabaseAdmin
  .from('dish_ingredients')
  .delete()
  .eq('dish_id', dishId)

if (ingredientsError) {
  this.logger.error('Failed to delete dish ingredients', { dishId, error: ingredientsError.message })
  return {
    success: false,
    error: 'Database error',
    message: 'Unable to delete dish ingredients'
  }
}

const { error: stepsError } = await this.supabaseAdmin
  .from('dish_steps')
  .delete()
  .eq('dish_id', dishId)

if (stepsError) {
  this.logger.error('Failed to delete dish steps', { dishId, error: stepsError.message })
  return {
    success: false,
    error: 'Database error',
    message: 'Unable to delete dish steps'
  }
}

const { error: categoryRelationsError } = await this.supabaseAdmin

```

```

        .from('dish_category_relations')
        .delete()
        .eq('dish_id', dishId)

    if (categoryRelationsError) {
        this.logger.error('Failed to delete dish category relations', { dishId, error: categoryRelationsError.message
    })

    return {
        success: false,
        error: 'Database error',
        message: 'Unable to delete dish category relations'
    }
}

const { error: deleteError } = await this.supabaseAdmin
    .from('dishes')
    .delete()
    .eq('id', dishId)

if (deleteError) {
    this.logger.error('Dish deletion failed', { error: deleteError.message })
    return {
        success: false,
        error: 'Unable to delete dish',
        message: deleteError.message
    }
}

this.logger.info('Dish deleted successfully', {
    dishId,
    userId
})

return {
    success: true,
    message: 'Dish deleted successfully'
}
} catch (error) {

```

```

    this.logger.error('Dish deletion error', { error: error.message })
  }
  return {
    success: false,
    error: 'Internal server error',
    message: 'Unable to delete dish'
  }
}
}

```

```

async getAllDishesForAdmin() {
  try {
    const { data: dishes, error } = await this.supabase
      .from('dishes')
      .select(`
        *,
        profiles:user_id(full_name, email, profile_tag),
        dish_category_relations(
          dish_categories(id, name)
        ),
        dish_ratings(id, rating),
        dish_comments(id)
      `)
      .in('status', ['pending', 'approved', 'rejected'])
      .order('created_at', { ascending: false })

    if (error) {
      this.logger.error('Admin dishes fetch error', { error: error.message })
      return {
        success: false,
        error: 'Database error',
        message: 'Unable to fetch dishes for admin'
      }
    }
  }
}

```

```

const processedDishes = (dishes || []).map(dish => ({
  ...dish,
  categories: dish.dish_category_relations?.map(rel => rel.dish_categories) || [],
  ratings: dish.dish_ratings || [],

```

```

        comments_count: dish.dish_comments?.length || 0
    )))

    processedDishes.forEach(dish => {
        delete dish.dish_category_relations
        delete dish.dish_ratings
        delete dish.dish_comments
    })

    return {
        success: true,
        dishes: processedDishes,
        total: processedDishes.length
    }
} catch (error) {
    this.logger.error('Admin dishes fetch error', { error: error.message })
    return {
        success: false,
        error: 'Internal server error',
        message: 'Unable to fetch dishes for admin'
    }
}
}

async getDishDetailsForAdmin(dishId) {
    try {
        const { data: dish, error } = await this.supabase
            .from('dishes')
            .select(`
                *,
                profiles:user_id(full_name, email, profile_tag, created_at),
                dish_category_relations(
                    dish_categories(id, name)
                ),
                dish_ratings(id, rating, profiles!user_id(full_name, profile_tag)),
                dish_comments(content, created_at, profiles!user_id(full_name, profile_tag)),
                dish_ingredients(*),
                dish_steps(*)
            `)
    } catch (error) {
        this.logger.error('Admin dish details fetch error', { error: error.message })
        return {
            success: false,
            error: 'Internal server error',
            message: 'Unable to fetch dish details for admin'
        }
    }
}

```

```

    )
    .eq('id', dishId)
    .single()

    if (error) {
        this.logger.error('Admin dish details fetch error', { error: error.message })
        return {
            success: false,
            error: 'Dish not found',
            message: 'Unable to fetch dish details'
        }
    }

    const processedDish = {
        ...dish,
        categories: dish.dish_category_relations?.map(rel => rel.dish_categories) || [],
        ratings: dish.dish_ratings || [],
        comments: dish.dish_comments || []
    }

    delete processedDish.dish_category_relations
    delete processedDish.dish_ratings
    delete processedDish.dish_comments

    return {
        success: true,
        dish: processedDish
    }
} catch (error) {
    this.logger.error('Admin dish details fetch error', { error: error.message })
    return {
        success: false,
        error: 'Internal server error',
        message: 'Unable to fetch dish details'
    }
}
}

```

```

async moderateDish(dishId, status, rejectionReason = null) {
  try {
    const existingDish = await this._validateDishExists(dishId)

    if (![ 'approved', 'rejected' ].includes(status)) {
      return {
        success: false,
        error: 'Invalid status',
        message: 'Status must be "approved" or "rejected"'
      }
    }

    const updateData = {
      status,
      moderated_at: new Date().toISOString()
    }

    if (status === 'rejected' && rejectionReason) {
      updateData.rejection_reason = rejectionReason
    }

    const { data: dish, error } = await this.supabase
      .from('dishes')
      .update(updateData)
      .eq('id', dishId)
      .select()
      .single()

    if (error) {
      throw new Error(error.message)
    }

    this.logger.info('Dish moderated successfully', { dishId, status })

    return this._handleSuccess(`Dish ${status} successfully`, { dish })
  } catch (error) {
    return this._handleError(error, 'Dish moderation error', 'Unable to moderate dish')
  }
}

```

```

    }

    async deleteDishByAdmin(dishId) {
      try {
        const { data: dish, error: fetchError } = await this.supabaseAdmin
          .from('dishes')
          .select('id, title')
          .eq('id', dishId)
          .single()

        if (fetchError || !dish) {
          this.logger.error('Dish not found for admin deletion', { dishId, error: fetchError?.message })
          return {
            success: false,
            error: 'Dish not found',
            message: 'The dish you are trying to delete does not exist'
          }
        }
      }

      this.logger.info('Starting admin dish deletion - cleaning up related records', { dishId })

      const { error: commentsError } = await this.supabaseAdmin
        .from('dish_comments')
        .delete()
        .eq('dish_id', dishId)

      if (commentsError) {
        this.logger.error('Failed to delete dish comments', { dishId, error: commentsError.message })
        return {
          success: false,
          error: 'Database error',
          message: 'Unable to delete dish comments'
        }
      }

      const { error: ratingsError } = await this.supabaseAdmin
        .from('dish_ratings')
        .delete()

```

```

        .eq('dish_id', dishId)

    if (ratingsError) {
        this.logger.error('Failed to delete dish ratings', { dishId, error: ratingsError.message })
        return {
            success: false,
            error: 'Database error',
            message: 'Unable to delete dish ratings'
        }
    }

    const { error: collectionItemsError } = await this.supabaseAdmin
        .from('dish_collection_items')
        .delete()
        .eq('dish_id', dishId)

    if (collectionItemsError) {
        this.logger.error('Failed to delete dish collection items', { dishId, error: collectionItemsError.message })
        return {
            success: false,
            error: 'Database error',
            message: 'Unable to delete dish from collections'
        }
    }

    const { error: ingredientsError } = await this.supabaseAdmin
        .from('dish_ingredients')
        .delete()
        .eq('dish_id', dishId)

    if (ingredientsError) {
        this.logger.error('Failed to delete dish ingredients', { dishId, error: ingredientsError.message })
        return {
            success: false,
            error: 'Database error',
            message: 'Unable to delete dish ingredients'
        }
    }
}

```

```

const { error: stepsError } = await this.supabaseAdmin
    .from('dish_steps')
    .delete()
    .eq('dish_id', dishId)

if (stepsError) {
    this.logger.error('Failed to delete dish steps', { dishId, error: stepsError.message })
    return {
        success: false,
        error: 'Database error',
        message: 'Unable to delete dish steps'
    }
}

const { error: categoryRelationsError } = await this.supabaseAdmin
    .from('dish_category_relations')
    .delete()
    .eq('dish_id', dishId)

if (categoryRelationsError) {
    this.logger.error('Failed to delete dish category relations', { dishId, error: categoryRelationsError.message
    })

    return {
        success: false,
        error: 'Database error',
        message: 'Unable to delete dish category relations'
    }
}

const { error: deleteError } = await this.supabaseAdmin
    .from('dishes')
    .delete()
    .eq('id', dishId)

if (deleteError) {
    this.logger.error('Admin dish deletion error', { dishId, error: deleteError.message })
    return {

```

```

        success: false,
        error: 'Database error',
        message: 'Unable to delete dish'
    }
}

this.logger.info('Dish deleted by admin', { dishId, dishTitle: dish.title })
return {
    success: true,
    message: 'Dish deleted successfully by admin'
}
} catch (error) {
    this.logger.error('Admin dish deletion error', { dishId, error: error.message })
    return {
        success: false,
        error: 'Internal server error',
        message: 'Unable to delete dish'
    }
}
}

async getDishStats() {
    try {
        const { data: dishes, error } = await this.supabase
            .from('dishes')
            .select('status')

        if (error) {
            this.logger.error('Dish stats fetch error', { error: error.message })
            return {
                success: false,
                error: 'Database error',
                message: 'Unable to fetch dish statistics'
            }
        }

        const total = dishes.length
        const approved = dishes.filter(d => d.status === 'approved').length
        const pending = dishes.filter(d => d.status === 'pending').length
        const rejected = dishes.filter(d => d.status === 'rejected').length
    }
}

```

```

return {
  success: true,
  stats: {
    total,
    approved,
    pending,
    rejected
  }
}
} catch (error) {
  this.logger.error('Dish stats fetch error', { error: error.message })
  return {
    success: false,
    error: 'Internal server error',
    message: 'Unable to fetch dish statistics'
  }
}
}
}

```