МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Острозька академія»
Навчально-науковий інститут інформаційних технологій та бізнесу
Кафедра інформаційних технологій та аналітики даних

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавра

на тему: «**Розробка інтерактивного сайту «YUMTIME»»**

Виконав: студент 4 курсу, групи КН-41
першого (бакалаврського) рівня вищої освіти
спеціальності 122 Комп'ютерні науки
освітньо-професійної програми «Комп'ютерні науки»
Круш Ірина Вікторівна

Керівник: *викладач кафедри інформаційних технологій та аналітики даних*
Мацевич Денис Володимирович

Рецензент: *кандидат технічних наук, доцент, доцент кафедри прикладної математики Донецького національного університету імені Василя Стуса*
Загоруйко Любов Василівна

РОБОТА ДОПУЩЕНА ДО ЗАХИСТУ

Завідувач кафедри інформаційних технологій та аналітики даних

_____ (проф., д.е.н. Кривицька О.Р.)

Протокол № 11 від « 20 » травня 2026 р.

Острог, 2026

АНОТАЦІЯ

кваліфікаційної роботи

на здобуття освітнього ступеня бакалавра

Тема: **Розробка інтерактивного сайту «YUMTIME»**

Автор: Круш Ірина Вікторівна

Науковий керівник:

викладач кафедри кафедри інформаційних технологій та аналітики даних

Мацевич Денис Володимирович

Захищена «……»…… 20__ року.

Пояснювальна записка до кваліфікаційної роботи: *72(кількість сторінок роботи) с., 16 (кількість рисунків) рис., 1 (кількість таблиць) табл., 4 (кількість додатків) додатків, 20 (кількість джерел) джерел.*

Ключові слова: *ВЕБЗАСОСТОСУНОК, .NET 8, NEXT.JS, CLEAN ARCHITECTURE, ШТУЧНИЙ ІНТЕЛЕКТ, POSTGRESQL, CQRS, ГЕНЕРАЦІЯ РЕЦЕПТІВ.*

Короткий зміст праці:

Кваліфікаційна робота присвячена розробці інтерактивної інформаційної системи «YUMTIME» для управління кулінарними рецептами. У роботі проведено аналіз предметної області, спроєктовано архітектуру системи та реалізовано вебзастосунок із використанням сучасних технологій.

Серверна частина базується на платформі .NET 8 із застосуванням принципів Clean Architecture та патерну CQRS, що забезпечує високу масштабованість. Клієнтська частина реалізована на фреймворку Next.js із використанням TypeScript та Tailwind CSS для створення адаптивного інтерфейсу. Особливістю системи є інтеграція з моделями штучного інтелекту для генерації унікальних рецептів на основі наявних інгредієнтів та використання інтерактивної карти світу для візуалізації національних кухонь. Реалізовано систему безпеки на основі JWT-токенів та автоматичний розрахунок поживної цінності страв. Практична цінність роботи полягає у створенні функціонального інструменту для персоналізації процесу приготування їжі.

ABSTRACT

of the qualification

work for a Bachelor's degree

Topic: Development of the "YUMTIME" interactive website

Author: Iryna Viktorivna Krush

Scientific Supervisor: Denys Volodymyrovych Matsevych, Lecturer at the Department of Information Technology and Data Analytics

Defended on "....."..... 20__.

Explanatory note to the qualification work: *72 (number of pages) p., 16 (number of figures) figs., 1 (number of tables) tables, 4 (number of appendices) appendices, 20 (number of sources) sources.*

Keywords: *WEB APPLICATION, .NET 8, NEXT.JS, CLEAN ARCHITECTURE, ARTIFICIAL INTELLIGENCE, POSTGRESQL, CQRS, RECIPE GENERATION.*

Abstract: *This qualification work is dedicated to the development of the "YUMTIME" interactive information system for culinary recipe management. The work involves a subject area analysis, system architecture design, and the implementation of a web application using modern technologies.*

The backend is built on the .NET 8 platform utilizing Clean Architecture principles and the CQRS pattern, ensuring high scalability. The frontend is implemented using the Next.js framework with TypeScript and Tailwind CSS to create a responsive user interface. A distinctive feature of the system is its integration with artificial intelligence models to generate unique recipes based on available ingredients and the use of an interactive world map for visualizing national cuisines. A security system based on JWT tokens and an automatic nutritional value calculation for dishes have been implemented. The practical value of the work lies in creating a functional tool for personalizing the cooking process.

Зміст

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

Скорочення	Розшифровка
БД	База даних
ІС	Інформаційна система
КБЖУ	Калорії, Білки, Жири, Вуглеводи (енергетична та харчова цінність)
ПЗ	Програмне забезпечення
ПП	Програмний продукт
СКБД	Система керування базами даних
ШІ	Штучний інтелект
AI	Artificial Intelligence (штучний інтелект)
API	Application Programming Interface (інтерфейс прикладного програмування)
CQRS	Command Query Responsibility Segregation (розділення відповідальності за команди та запити)
CRUD	Create, Read, Update, Delete (базові операції керування даними: створення, читання, оновлення, видалення)
CSS	Cascading Style Sheets (каскадні таблиці стилів)
DI	Dependency Injection (впровадження залежностей)
DTO	Data Transfer Object (об'єкт переносу даних)
ER	Entity-Relationship (модель «сутність-зв'язок» для проєктування баз даних)

HTML	HyperText Markup Language (мова розмітки гіпертексту)
HTTP	HyperText Transfer Protocol (протокол передачі гіпертексту)
IDE	Integrated Development Environment (інтегроване середовище розробки)
ISR	Incremental Static Regeneration (інкрементальна статична регенерація)
JSON	JavaScript Object Notation (текстовий формат обміну даними)
JWT	JSON Web Token (маркер доступу у форматі JSON)
LLM	Large Language Model (велика мовна модель)
MVC	Model-View-Controller (архітектурний шаблон «Модель-Представлення-Контролер»)
ORM	Object-Relational Mapping (об'єктно-реляційне відображення)
OSI	Open Systems Interconnection (базова еталонна модель взаємодії відкритих систем)
REST	Representational State Transfer (архітектурний стиль взаємодії компонентів вебзастосунку)
SDK	Software Development Kit (комплект засобів розробки програмного забезпечення)
SEO	Search Engine Optimization (пошукова оптимізація)

SPA	Single Page Application (односторінковий вебзастосунок)
SSG	Static Site Generation (генерація статичних сайтів)
SSR	Server-Side Rendering (серверний рендеринг)
TCP	Transmission Control Protocol (протокол керування передачею даних транспортного рівня)
UI / UX	User Interface / User Experience (інтерфейс користувача / користувацький досвід)
URL	Uniform Resource Locator (єдиний вказівник ресурсу)

ВСТУП

Актуальність теми. У сучасному світі інформаційні технології стають невід'ємною частиною повсякденного життя, зокрема у сфері харчування та кулінарії. Велика кількість доступних рецептів в інтернеті часто створює проблему "надлишку вибору", коли користувачу важко знайти страву, що відповідає його вподобанням або наявним інгредієнтам. Розробка інтерактивного сайту «YUMTIME» є актуальною, оскільки вона поєднує класичну базу рецептів із новітніми технологіями штучного інтелекту для генерації персоналізованого контенту, що значно спрощує процес планування харчування та задовольняє потреби сучасного користувача в інтерактивності та зручності.

Мета дослідження - розробка програмного продукту у вигляді інтерактивного веб-сайту для пошуку, збереження та генерації кулінарних рецептів за допомогою сучасних методів веброзробки та інтеграції елементів штучного інтелекту.

Задачі дослідження:

- **аналіз предметної області**: дослідження існуючих кулінарних платформ та виявлення недоліків у їхній функціональності щодо взаємодії з користувачем;
- **проектування інформаційної системи**: розробка архітектури бази даних, створення логіки взаємодії клієнтської та серверної частин сайту;
- **вибір інструментарію, методів реалізації та тестування ПП**:
  для розробки інтерактивного сайту **«YUMTIME»** було обрано сучасний технологічний стек, що дозволяє поєднати високу продуктивність серверної частини із динамічним інтерфейсом користувача:
  - **Back-end: Платформа .NET (C#) та ASP.NET Core.** Вибір обґрунтований надійністю платформи, високою швидкістю обробки запитів та наявністю потужних інструментів, як-от Entity Framework Core для роботи з базою даних рецептів. Архітектура побудована за

- принципами Clean Architecture, що дозволяє чітко розділити бізнес-логіку від технічних деталей реалізації.
- **Front-end: Фреймворк Next.js (React).** Даний інструмент обрано для забезпечення швидкого рендерингу сторінок (SSR/ISR), що є критичним для SEO-оптимізації кулінарного порталу. Використання компонентного підходу дозволяє створювати складні інтерактивні елементи, зберігаючи при цьому чистоту коду.
- **Реалізація інтерфейсу користувача.** Для верстки використовується Tailwind CSS, що дозволяє швидко втілювати сучасний дизайн із використанням м’яких форм, закруглених кутів та складної тіньової ієрархії для карток рецептів, забезпечуючи адаптивність під будь-які пристрої.
- **Інтеграція ШІ.** Програмний продукт передбачає взаємодію із зовнішніми API штучного інтелекту для генерації рецептів на основі вільного текстового вводу користувача, що додає системі високого рівня інтерактивності.
- **Методи тестування.** Для перевірки роботи програмного продукту використано модульне та ручне функціональне тестування. За допомогою бібліотеки xUnit перевірено коректність роботи серверної логіки. Ручне тестування включало перевірку форм вводу, навігації між сторінками та адаптивності інтерфейсу на різних пристроях.

Об’єкт дослідження – інформаційна система у вигляді інтерактивного сайту «YUMTIME», що забезпечує користувачу доступ до бази рецептів та інструментів їх обробки.

Предмет дослідження – методи проектування реляційних баз даних, технології розробки веб-інтерфейсів, методи інтеграції зовнішніх API (зокрема штучного інтелекту для генерації рецептів) та інструменти тестування програмного забезпечення.

РОЗДІЛ 1
ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1 Опис предметного середовища

Предметним середовищем даного дослідження є сфера кулінарних інформаційних технологій, що охоплює процеси пошуку, структурування та генерації рецептів приготування страв. Сучасний ринок потребує не просто статичних баз даних, а інтерактивних систем, що здатні адаптуватися до індивідуальних потреб користувача.

Класифікація інформаційних об'єктів у системі «YUMTIME»

Для функціонування інтерактивної платформи «YUMTIME» розроблено структуру інформаційних об'єктів, які відображають усі аспекти взаємодії користувача з кулінарним контентом. Кожна одиниця має чітко визначений набір властивостей, що забезпечують цілісність даних та можливість їх автоматизованої обробки.

Основними об'єктами системи є:

- **Користувач (User):** Центральний суб'єкт системи. Крім ідентифікаційних даних (ім'я, email), об'єкт містить рольову модель (Role), що дозволяє розмежовувати права доступу між звичайними користувачами та адміністраторами.
- **Рецепт (Recipe):** Головний інформаційний блок, який об’єднує в собі опис страви, покрокові інструкції та медіаконтент. Важливою особливістю об'єкта є наявність обчислювальних властивостей, таких як загальна калорійність (TotalCalories) та середній рейтинг (AverageRating), які розраховуються динамічно на основі пов'язаних інгредієнтів та оцінок.
- **Інгредієнт (Ingredient):** Базовий елемент, що містить дані про поживну цінність (білки, жири, вуглеводи, калорії) на одиницю виміру. Це дозволяє системі не просто виводити список продуктів, а й проводити нутриціологічний аналіз кожної страви.

- **Оцінка (Rating):** Об'єкт зворотного зв'язку, що пов'язує користувача з рецептом. Він дозволяє формувати систему рейтингів від 1 до 5 зірок, що є основою для алгоритму вибору популярних страв.
- **Кухня (Cuisine):** Спеціалізований об'єкт для географічної прив'язки рецептів. Містить координати (Latitude, Longitude), що дозволяє інтегрувати рецепти в інтерактивну мапу світу та фільтрувати їх за країною походження.
- **Категорія (Category):** Інструмент для систематизації контенту (наприклад, "Сніданки", "Десерти"), що полегшує навігацію та пошук у системі.

Функціональна модель та процеси діяльності:

Функціональна модель системи «YUMTIME» визначає логіку взаємодії користувача з програмним продуктом та описує ключові процеси, які забезпечують автоматизацію кулінарної діяльності. Основною метою моделі є мінімізація зусиль користувача на етапах пошуку, планування та приготування страв.

До основних функціональних процесів системи належать:

- **Процес персоналізованого доступу:** Користувач взаємодіє з системою через механізм аутентифікації (JWT). Залежно від ролі (адміністратор або користувач), система надає доступ до певних функцій: перегляду, створення або видалення контенту.
- **Процес інтелектуальної генерації рецептів (AI-модуль):** Це один із ключових інноваційних процесів. Користувач вводить довільний набір інгредієнтів у текстове поле, після чого система передає ці дані до модуля штучного інтелекту. Результатом процесу є структурований рецепт, який інтегрується в інтерфейс сайту.
- **Процес географічної візуалізації (Інтерактивна мапа):** Система використовує координати (Latitude, Longitude) із сутності Cuisine для відображення маркерів на карті світу. Користувач, обираючи регіон, ініціює запит до бази даних, яка повертає перелік традиційних рецептів обраної країни.

- **Процес управління кулінарним контентом:** Включає повний цикл роботи з рецептом (CRUD): від завантаження фотографії через сервіс Cloudinary до автоматичного розрахунку харчової цінності на основі обраних інгредієнтів.
- **Процес оцінювання та зворотного зв'язку:** Користувач виставляє рейтинг, який автоматично перераховує властивість AverageRating у рецепті. Цей процес безпосередньо впливає на логіку вибору «Страви дня», яка формується на основі найвищих показників популярності.
- **Процес формування списку покупок:** Система аналізує обраний рецепт і трансформує його інгредієнти у зручний для користувача перелік продуктів, що дозволяє автоматизувати етап підготовки до приготування.

1.2 Огляд наявних аналогів

Для обґрунтування доцільності розробки та визначення унікальних переваг проекту «YUMTIME» було проведено аналіз найбільш релевантних рішень на українському та міжнародному ринках кулінарних сервісів. Аналіз проводився за критеріями функціональності, зручності інтерфейсу (UI/UX) та наявності інтелектуальних інструментів підтримки користувача.

1. **Klopotenko.com (Україна).**

Даний ресурс є одним із лідерів українського сегменту авторських кулінарних проектів.

- **Переваги**: Висока якість медіаконтенту, професійно розроблені рецепти, адаптовані під локальні продукти, та сучасний візуальний стиль.
- **Недоліки**: Система має відносно жорстко задану структуру "від автора до читача", що певною мірою може обмежувати користувача у персоналізації. Також відсутні інструменти автоматичного підбору страв за наявними інгредієнтами та сучасні методи інтерактивної взаємодії, такі як чат-боти або ШІ-генератори.

2. **Cookpad (Міжнародна платформа).**

Одна з найбільших світових спільнот для обміну домашніми рецептами.

- **Переваги**: Величезна база даних та потужна соціальна складова (можливість публікувати власні рецепти та коментувати чужі).
- **Недоліки**: Через велику кількість користувацького контенту якість рецептів та фотографій може бути нерівномірною. Інтерфейс інколи виглядає перевантаженим рекламними елементами та навігаційними компонентами, що може ускладнювати швидкий пошук інформації. Система пошуку працює за класичними фільтрами і не завжди пропонує інтелектуальний синтез нових ідей.

3. **SuperCook (США/Міжнародний).**

Спеціалізований агрегатор, що фокусується на пошуку рецептів за списком інгредієнтів.

- **Переваги**: Потужний алгоритм зіставлення продуктів із великою кількістю рецептів із різних джерел.
- **Недоліки**: Технічно орієнтований дизайн, який може бути дещо складним для пересічного користувача. Основним обмеженням є відсутність гнучкості: система переважно видає готові збіги з бази даних і не завжди здатна згенерувати унікальну інструкцію у випадку нестандартних комбінацій продуктів користувача.

Таблиця 1.1

Порівняльний аналіз кулінарних веб-ресурсів та «YUMTIME»

Критерії порівняння	**Klopotenko.com**	**Cookpad**	**SuperCook**	**YUMTIME**
Наявність українського контенту	Так	Так	Обмежено	**Так**
Автоматичний розрахунок КБЖУ	Ні	Ні	Ні	**Так**
Інтерактивна мапа кухонь світу	Ні	Ні	Ні	**Так**
Генерація рецептів через ШІ	Ні	Ні	Ні	**Так**
Додавання власних рецептів користувачами	Ні	Так	Ні	**Так**
Формування списку покупок	Ні	Ні	Так	**Так**
Відсутність рекламного перевантаження	Так	Ні	Ні	**Так**

Отже, аналіз наявних аналогів показує, що кожен із сервісів має свої обмеження. Платформи або пропонують лише готовий авторський контент без можливості персоналізації, або мають незручний і перевантажений рекламою інтерфейс. При цьому класичний пошук за фільтрами не завжди дозволяє ефективно підібрати страву під індивідуальний набір продуктів.

Веб-ресурс «YUMTIME» створюється для того, щоб об'єднати ці функції в одному місці. На відміну від конкурентів, система пропонуватиме автоматичний калькулятор КБЖУ, інтерактивну мапу для вивчення кухонь світу та генерацію унікальних рецептів за допомогою штучного інтелекту.

1.3 Постановка задачі

На основі проведеного аналізу предметної області та огляду існуючих аналогів, було виявлено необхідність створення інтелектуального веб-ресурсу, який би поєднував у собі функції класичної кулінарної бази даних та сучасні інструменти обробки інформації.

Обґрунтування проблеми: Більшість сучасних користувачів стикаються з проблемою нераціонального використання часу на пошук рецептів та неефективного управління наявними продуктами. Відсутність персоналізації та інтерактивності в існуючих системах створює бар'єр між користувачем та кулінарним процесом.

Метою даного дослідження є проектування та розробка інтерактивного веб-сайту «YUMTIME», який автоматизує процес підбору страв, розрахунку їх поживної цінності та забезпечує можливість генерації контенту за допомогою алгоритмів штучного інтелекту.

Для досягнення поставленої мети необхідно вирішити наступні задачі:

1. **Проектування архітектури та бази даних:**
   - Розробити реляційну модель бази даних, яка б забезпечувала зберігання інформації про користувачів, рецепти, інгредієнти, категорії та кухні світу.
   - Спроектувати бізнес-логіку системи за принципами Clean Architecture для забезпечення масштабованості та легкого супроводу коду.
2. **Розробка функціональних модулів:**
   - Створити систему автентифікації та авторизації користувачів на основі JWT-токенів із розмежуванням прав доступу.
   - Реалізувати повноцінний CRUD-інтерфейс для керування рецептами (додавання, редагування, видалення) з інтеграцією хмарного сховища Cloudinary для медіафайлів.
   - Розробити інтерактивну мапу кухонь світу на базі бібліотеки Leaflet.
   - Інтегрувати модуль штучного інтелекту для генерації рецептів на основі вільного текстового вводу інгредієнтів.

3. **Забезпечення інтерактивності та аналітики:**
   - Впровадити алгоритми автоматичного розрахунку КБЖУ (калорії, білки, жири, вуглеводи) для кожної страви.
   - Реалізувати систему оцінювання та рейтингів для формування динамічного блоку «Страва дня».
   - Створити функціонал автоматичного формування списку покупок на основі обраних рецептів.
4. **Дослідження та тестування:**
   - Дослідити ефективність інтеграції зовнішніх API для роботи з ШІ.
   - Провести тестування програмного продукту (модульне та функціональне) для гарантування стабільності роботи та коректності візуального відображення інтерфейсу.

Об'єктом дослідження є процес автоматизації взаємодії користувача з кулінарним контентом у веб-середовищі.

Предметом дослідження є методи проектування веб-систем, технології .NET Core та Next.js, алгоритми штучного інтелекту для генерації текстів та інструменти візуалізації картографічних даних.

Висновки до розділу 1

У першому розділі було проведено всебічний аналіз предметного середовища кулінарних інформаційних систем та обґрунтовано необхідність розробки інтерактивного веб-сайту «YUMTIME». На основі виконаного дослідження можна зробити наступні висновки:

1. **Проаналізовано предметну область**, яка характеризується переходом від статичного збереження кулінарних рецептів до інтерактивної взаємодії з користувачем. Визначено ключові інформаційні об'єкти системи (користувачі, рецепти, інгредієнти, кухні світу), що дозволяє забезпечити цілісність даних та автоматизувати складні процеси, такі як розрахунок КБЖУ.

2. **Побудовано функціональну модель системи**, яка охоплює повний цикл роботи з контентом: від авторизації користувача та управління рецептами (CRUD) до інтелектуальної генерації нових страв за допомогою ШІ. Виокремлено унікальні процеси, такі як географічна візуалізація кухонь світу через інтерактивну мапу та формування персоналізованих рекомендацій.
3. **Проведено порівняльний аналіз наявних аналогів** (Klopotenko.com, Cookpad, SuperCook), який показав відсутність комплексних рішень, що поєднують сучасний UX-дизайн із гнучкими інструментами штучного інтелекту. Це підтвердило наявність вільної ніші та актуальність розробки «YUMTIME».
4. **Сформульовано мету та задачі дослідження**, що полягають у створенні сучасного програмного продукту на базі технологій (.NET Core та Next.js). Визначено, що використання хмарних сервісів (Cloudinary) та бібліотек для роботи з картами (Leaflet) забезпечить масштабованість та зручність системи.

Таким чином, результати першого розділу стали теоретичною та аналітичною основою для подальшого проєктування архітектури та технічної реалізації системи «YUMTIME», що буде розглянуто в наступних розділах роботи.

РОЗДІЛ 2
ІНФОРМАЦІЙНЕ ТА МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Аналіз предметної області

У даному підрозділі виконується детальний розгляд об’єктів дослідження та опис правил, що регулюють цілісність даних у системі «YUMTIME». Інформаційна модель базується на сутностях доменної області, а механізми перевірки даних реалізовані за допомогою бібліотеки FluentValidation[9], що дозволяє чітко регламентувати вхідні параметри.

Опис інформаційних об’єктів та їх характеристик

На основі архітектури доменної області (Domain Layer) у системі «YUMTIME» виділено основні інформаційні об’єкти, які забезпечують зберігання, обробку та взаємодію даних.

1. **Користувач (User)** - об’єкт, що відповідає за роботу користувача із системою та забезпечує механізми автентифікації й авторизації. Містить персональні дані користувача, електронну пошту, хеш пароля, роль у системі та посилання на аватар. Також зберігає інформацію про створені рецепти та залишені оцінки.
2. **Рецепт (Recipe)** - основний інформаційний об’єкт системи, що містить дані про кулінарну страву. Включає назву, опис, інструкцію приготування, зображення, орієнтовний час приготування та дату створення. Рецепт пов’язаний із користувачем, категорією, кухнею світу, інгредієнтами та оцінками користувачів. Додатково об’єкт забезпечує автоматичний розрахунок КБЖУ та середнього рейтингу страви.
3. **Категорія (Category)** - об’єкт для групування рецептів за типом страви. Містить назву категорії та список рецептів, що до неї належать. Використовується для реалізації фільтрації та навігації по сайту.
4. **Кухня (Cuisine)** - об’єкт, що використовується для географічної класифікації рецептів. Містить назву країни та координати для відображення на

інтерактивній мапі. Дає можливість користувачу переглядати рецепти відповідно до кухонь світу.

5. **Інгредієнт (Ingredient)** - об’єкт, який описує окремий продукт, що використовується у рецептах. Містить назву інгредієнта, одиницю вимірювання та показники поживної цінності: калорії, білки, жири та вуглеводи.
6. **Складовий інгредієнт рецепта (RecipeIngredient)** - проміжна сутність, що реалізує зв’язок між рецептом та інгредієнтом. Містить інформацію про кількість конкретного інгредієнта у рецепті та використовується для автоматичного розрахунку харчової цінності страви.
7. **Оцінка (Rating)** - об’єкт, призначений для реалізації механізму зворотного зв’язку. Містить кількість зірок, які користувач поставив рецепту, а також забезпечує формування середнього рейтингу страви.

Опис обмежень на вхідні та вихідні дані

Для забезпечення коректної роботи системи «YUMTIME» та збереження цілісності даних у програмному продукті реалізовано механізми валідації вхідної інформації. Перевірка даних здійснюється перед обробкою та збереженням інформації в базі даних, що дозволяє мінімізувати кількість помилок та некоректних записів.

Основні обмеження на вхідні дані:

- **Дані користувача**

  Для імені, прізвища та електронної пошти встановлено перевірку на обов’язкове заповнення та допустиму довжину символів. Email повинен відповідати коректному формату електронної адреси. Пароль користувача має містити мінімальну кількість символів, а роль користувача обмежується значеннями «Admin» або «User».

- **Дані рецептів**

  Назва рецепта, опис та інструкція приготування є обов’язковими полями та мають обмеження на мінімальну і максимальну довжину тексту. Час

приготування повинен бути додатнім числом і не перевищувати встановленого максимального значення.

- **Дані інгредієнтів**

  Для інгредієнтів перевіряється наявність назви та одиниці вимірювання. Значення калорійності, білків, жирів та вуглеводів не можуть бути від'ємними.

- **Дані оцінювання**

  Кількість зірок для оцінювання рецепта обмежується діапазоном від 1 до 5.

- **Дані кухонь світу та категорій**

  Для категорій та кухонь світу перевіряється коректність назв та обов'язковість введення основних параметрів.

Вихідні дані системи формуються у структурованому вигляді та відображаються користувачу через вебінтерфейс. Результати роботи системи включають інформацію про рецепти, інгредієнти, рейтинги, харчову цінність страв та результати пошуку й фільтрації.

2.2 Проектування системи

Концептуальне проектування системи «YUMTIME» базується на принципах багатошарової архітектури (**Clean Architecture)**[18]**,** що дозволяє відокремити бізнес-логіку від інфраструктурних деталей та зовнішніх інтерфейсів. Це забезпечує високу гнучкість, тестованість та легкість підтримки програмного продукту.

Архітектурна модель проекту (Clean Architecture)

Програмна реалізація системи «YUMTIME» базується на принципах Clean Architecture, що відображено у розподілі проекту на чотири функціональні рівні

1. **Domain (Ядро):** Центральний рівень, що містить сутності (Cuisine, Recipe, User тощо) та базові типи даних. Цей рівень не залежить від жодних зовнішніх бібліотек чи баз даних. У папках сутностей також визначено Strongly Typed IDs (CuisineId.cs), що гарантує цілісність посилань між об'єктами.

2. **Application (Логіка):** Рівень прикладних сервісів, де реалізовано патерн **CQRS** (Command Query Responsibility Segregation).

- **Commands:** Містять логіку зміни стану (створення, оновлення, видалення категорій, рецептів тощо).
- **Validators:** Використовують FluentValidation для перевірки даних перед виконанням команд.
- **Interfaces:** Тут визначено контракти для репозиторіїв (ICategoryRepository) та сервісів (IJwtTokenGenerator), що забезпечує інверсію залежностей.

3. **Infrastructure (Інфраструктура):** Реалізація інтерфейсів, визначених у рівні Application.
   - **Persistence:** Містить ApplicationDbContext, міграції та конфігурації сутностей (Configurations), які ми розглянули раніше.
   - **Repositories:** Конкретна реалізація методів роботи з базою даних.
   - **Services:** Інтеграція із зовнішніми сервісами, наприклад, CloudinaryPhotoService.cs для збереження зображень[7].
4. **Api (Представлення):** Зовнішній рівень, реалізований як ASP.NET Core Web API.
   - **Controllers:** Приймають HTTP-запити та передають їх на рівень Application.
   - **Modules:** Містять специфічні налаштування для обробки помилок (ErrorHandlers) та глобальну валідацію DTO.
   - **DTOs (Data Transfer Objects) -** спеціальні об’єкти для передачі даних між клієнтською та серверною частинами системи. Вони використовуються для формування запитів і відповідей API та дозволяють передавати лише необхідні дані.

Проектування схеми бази даних (ER-модель)

На основі сутностей розроблено реляційну модель даних **(рис.2.1)** . Ключовими типами зв'язків у системі є:

1. **Один до багатьох (1:N):**

User - Recipe: Один користувач може бути автором багатьох рецептів, тоді як кожен рецепт належить лише одному користувачу. У базі даних цей зв’язок реалізовано через зовнішній ключ UserId у таблиці рецептів.

Category - Recipe: Одна категорія може містити багато рецептів, але кожен рецепт належить лише до однієї категорії. При видаленні категорії рецепти не видаляються автоматично (DeleteBehavior.Restrict).

Cuisine - Recipe: Одна кухня світу може бути пов’язана з багатьма рецептами. Кожен рецепт може належати до певної кухні або не мати прив’язки до неї. При видаленні кухні значення CuisineId у рецепті встановлюється в null (DeleteBehavior.SetNull).

User - Rating: Один користувач може залишати багато оцінок для різних рецептів.

Recipe - Rating: Один рецепт може містити багато оцінок від різних користувачів.

2. **Багато до багатьох (M:N):**

Recipe - Ingredient: Один рецепт містить декілька інгредієнтів, а один інгредієнт може використовуватися у багатьох рецептах. Для реалізації цього зв’язку використовується проміжна таблиця RecipeIngredient, яка додатково зберігає кількість інгредієнта (Quantity) у рецепті.

3. **Зв’язки зворотного зв’язку та взаємодії:**

Rating: Сутність «Оцінка» забезпечує взаємодію між користувачем та рецептом. Вона зберігає інформацію про рейтинг рецепта та дозволяє реалізувати механізм обчислення середнього рейтингу страви. Система передбачає, що один користувач може залишити лише одну оцінку для конкретного рецепта.

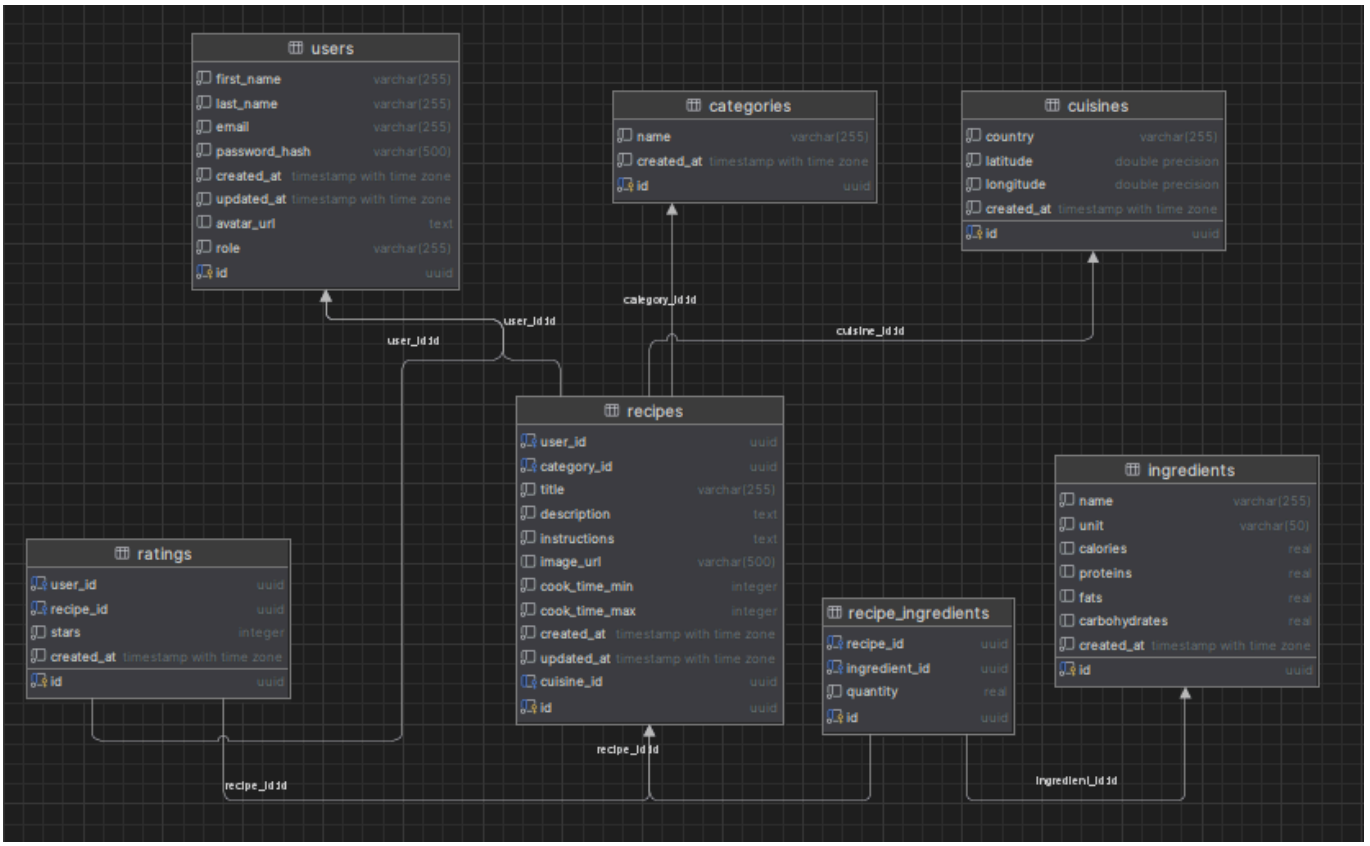

Рис. 2.1. Діаграма база даних

Проєктування користувацького інтерфейсу (UI/UX)

Важливим етапом проєктування інформаційної системи «YUMTIME» є розробка її користувацького інтерфейсу (UX/UI дизайну). Візуальний каркас та стилістичне оформлення системи базуються на прототипі, який було попередньо спроєктовано та досліджено у межах курсової роботи в середовищі Figma.

Для забезпечення єдиного стилю застосунку в середовищі Figma було сформовано базову дизайн-систему проєкту (рис. 2.2), яка містить такі ключові елементи:

- **Типографіка (Typography):** визначено єдину гарнітуру шрифту та зафіксовано шкалу розмірів заголовків і основного тексту для спрощення подальшої верстки.
- **Колористика (Colors):** розроблено фірмову палітру кольорів із градацією відтінків, де для важливих UI-елементів та блоків обрано ніжно-рожевий.

- **Іконографіка (Icons):** уніфіковано використання наборів іконок (Bootstrap та Font Awesome) у контурному та залитому варіантах для відображення різних станів інтерфейсу.

Розробка цієї дизайн-системи дозволила зафіксувати готові дизайн-токени (кольори, відступи та шрифти), що забезпечило швидку реалізацію фронтенд-частини застосунку за допомогою фреймворку Next.js.

Під час розробки цієї кваліфікаційної роботи початковий дизайн було модернізовано та адаптовано до сучасних стандартів веб-розробки:

- **Ергономіка та колористика:** Для системи обрано чисту, світлу палітру з використанням фірмових акцентних кольорів. Зокрема, для блоків кулінарних майстер-класів та популярних рецептів застосовано ніжно-рожевий відтінок, що забезпечує візуальну цілісність сторінок **(рис. 2.3)**.
- **Адаптивність:** Інтерфейс гарантує коректне та зручне відображення всіх елементів (карток, списків, інтерактивної карти) як на широкоформатних моніторах, так і на екранах мобільних телефонів **(рис. 2.4)**.
- **Інтуїтивність навігації:** Макет адаптовано для швидкого доступу користувача до ключових функцій: пошуку, автоматичного розрахунку КБЖУ, створення списку покупок та взаємодії з картою **(рис. 2.5)**.

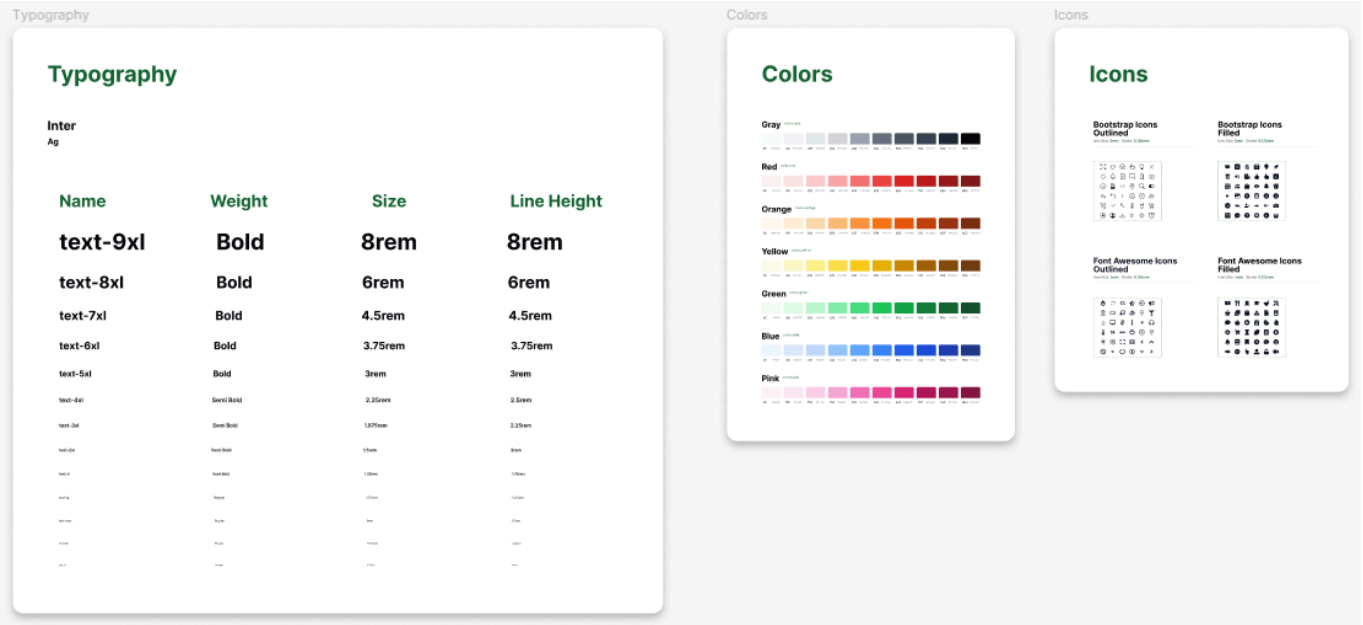

Рис. 2.2. Елементи дизайн-системи проєкту «YUMTIME»

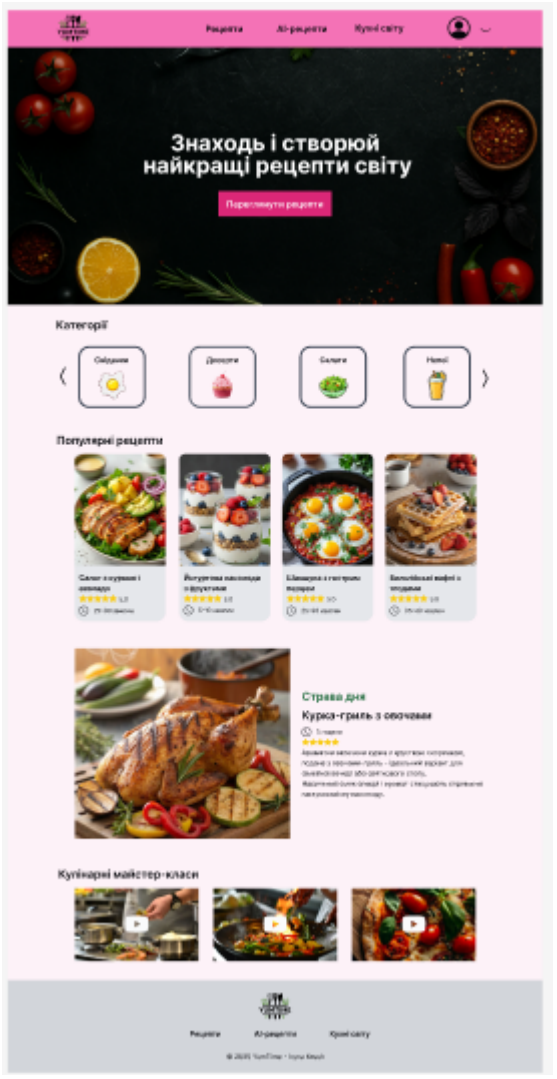

Рис. 2.3. Макет Головної сторінки інформаційної системи «YUMTIME»

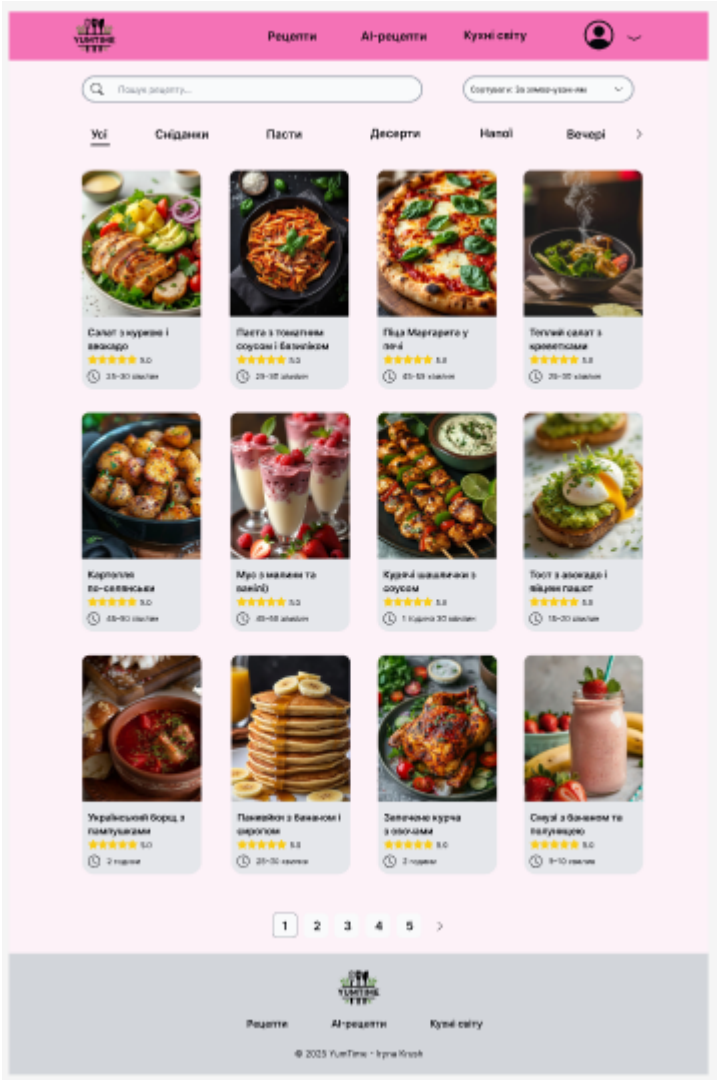

Рис. 2.4. Макет сторінки перегляду кулінарних рецептів системи «YUMTIME»

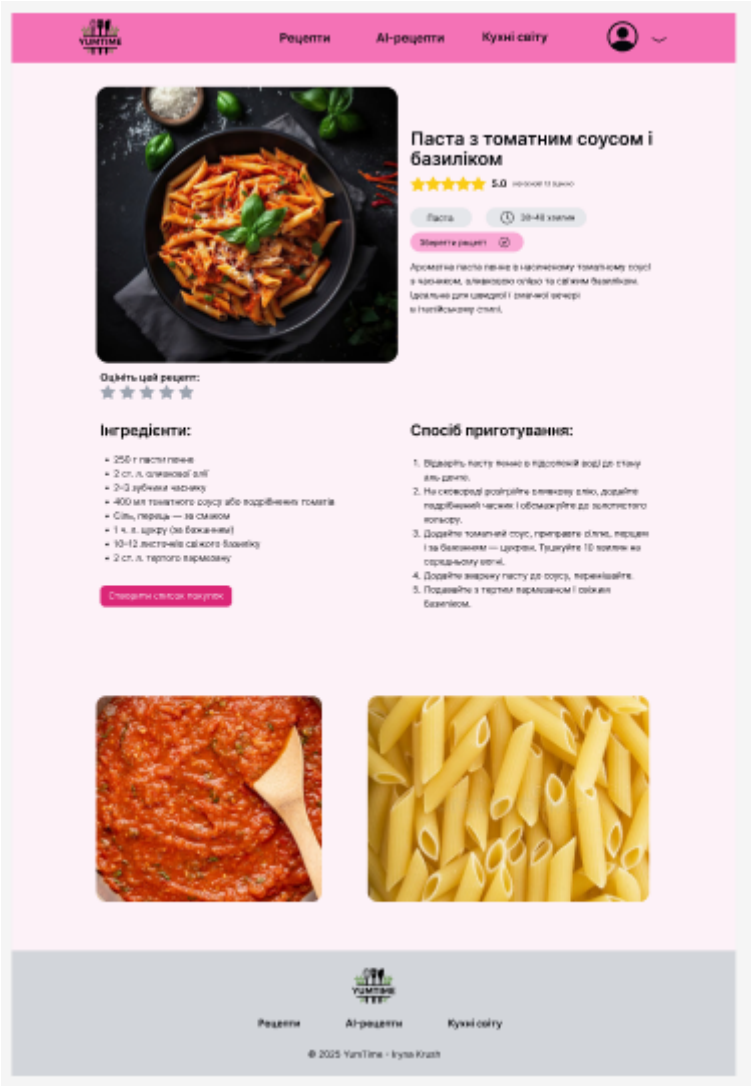

Рис. 2.5. Макет сторінки з детальною інформацією про кулінарний рецепт

2.3 Математичне та алгоритмічне забезпечення

У даному підрозділі представлено опис моделей та алгоритмів, що забезпечують функціонування системи «YUMTIME». Математичне забезпечення системи базується на автоматичному розрахунку поживної цінності страв, статистичній обробці рейтингів та алгоритмах пошуку і фільтрації рецептів.

Математична модель розрахунку поживної цінності (КБЖУ)

Для реалізації функції автоматичного розрахунку поживної цінності використовується модель підсумовування значень усіх інгредієнтів, що входять до складу рецепта. Кожен інгредієнт містить інформацію про калорії, білки, жири та вуглеводи на певну одиницю вимірювання.

Під час розрахунку система враховує кількість кожного інгредієнта у рецепті та додає його поживні показники до загального результату. Таким чином автоматично формується загальна калорійність страви та значення КБЖУ.

У програмній реалізації обчислення виконуються динамічно на основі списку інгредієнтів рецепта, що дозволяє автоматично оновлювати значення при зміні складу страви.

Алгоритм статистичного оцінювання (Average Rating)

Для визначення популярності рецептів у системі використовується механізм середнього рейтингу. Користувачі можуть оцінювати рецепти за шкалою від 1 до 5 зірок, після чого система обчислює середнє арифметичне всіх отриманих оцінок.

Якщо рецепт ще не має оцінок, його рейтинг автоматично встановлюється на значення 0. Для зручності відображення результат округлюється до одного десяткового знака, що забезпечує більш зрозуміле та естетичне представлення інформації в інтерфейсі.

Отриманий рейтинг використовується для сортування рецептів, формування блоку «Страва дня» та відображення популярного контенту.

Алгоритмічне забезпечення пошуку та фільтрації

Пошук рецептів у системі реалізовано за допомогою механізмів фільтрації та групування даних. Користувач може здійснювати пошук за категоріями страв, кухнями світу.

Під час пошуку система аналізує введені параметри та формує перелік рецептів, які відповідають заданим умовам.

Логіка роботи модуля штучного інтелекту (AI Generator)

Окремим алгоритмічним компонентом системи є модуль генерації рецептів на основі штучного інтелекту. Користувач вводить список інгредієнтів у текстовому вигляді, після чого система формує запит для зовнішнього AI-сервісу.

Алгоритм формування списку покупок

Для спрощення підготовки до приготування страв у системі реалізовано алгоритм автоматичного формування списку покупок. Система аналізує інгредієнти обраних рецептів та об’єднує їх у єдиний перелік продуктів із відповідними кількостями. Це дозволяє користувачу швидко сформувати необхідний список для придбання продуктів.

Висновки до розділу 2

У другому розділі було проведено детальне проектування інформаційного та математичного забезпечення системи «YUMTIME». За результатами виконаної роботи можна зробити наступні висновки:

1. **Проведено аналіз предметної області**, у ході якого виділено 7 основних інформаційних об’єктів (User, Recipe, Ingredient, Category, Cuisine, Rating, RecipeIngredient). Визначено їхні характеристики та встановлено жорсткі обмеження на вхідні дані за допомогою механізмів валідації, що гарантує цілісність та достовірність інформації в системі.
2. **Розроблено концептуальну модель системи**, яка базується на архітектурному підході **Clean Architecture**. Розподіл проекту на рівні *Domain, Application, Infrastructure* та *Api* дозволив забезпечити незалежність бізнес-логіки від зовнішніх сервісів та баз даних, що робить систему гнучкою до змін та легкою у тестуванні. Описано реляційну модель бази даних (ER-модель) із визначенням усіх ключових зв’язків між сутностями.
3. **Сформовано математичне та алгоритмічне забезпечення**, яке включає:
    - Математичну модель розрахунку енергетичної цінності страви (КБЖУ) на основі агрегації даних інгредієнтів.
    - Статистичний алгоритм обчислення середнього рейтингу для формування динамічного контенту.

- Логіку взаємодії з модулем штучного інтелекту для генерації рецептів за інгредієнтами.
- Алгоритми фільтрації, пошуку та автоматичного формування списку покупок.

Отримані результати створюють основу для подальшої програмної реалізації та тестування вебсистеми «YUMTIME», що буде розглянуто у наступних розділах роботи.

РОЗДІЛ 3
ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Засоби розробки

1. **Основні технології бекенд-частини**

- **.NET (ASP.NET Core Web API)**[2]

  Використано для реалізації серверної частини системи. Забезпечує обробку HTTP-запитів, роботу з бізнес-логікою та інтеграцію з базою даних.

- **C#**

  Основна мова програмування, яка використовується для реалізації доменної логіки, сервісів, контролерів та алгоритмів обробки даних.

- **Entity Framework Core**[3]

  ORM-технологія для взаємодії з реляційною базою даних. Використовується для створення моделей даних, конфігурації зв’язків та виконання CRUD-операцій.

- **PostgreSQL**[6]

  Реляційна база даних, яка використовується для зберігання основної інформації системи (користувачі, рецепти, інгредієнти, рейтинги тощо).

Архітектурні та програмні підходи

- **Clean Architecture**

  Використовується багаторівнева архітектура (Domain, Application, Infrastructure, Api), що забезпечує розділення бізнес-логіки та інфраструктури.

- **CQRS (Command Query Responsibility Segregation)**

  Розділення операцій читання та запису для підвищення продуктивності та структурованості коду.

- **Dependency Injection (DI)**

  Використовується вбудований контейнер .NET для впровадження залежностей між сервісами та репозиторіями.

- **FluentValidation**[9]

  Бібліотека для перевірки коректності вхідних даних (валідація DTO та команд).

Автентифікація та безпека

- **JWT (JSON Web Token)**[11]

  Використовується для реалізації авторизації користувачів. Токен містить інформацію про користувача та його роль у системі.
- **BCrypt**[14]

  Використовується для хешування паролів, що забезпечує безпечне зберігання облікових даних.

Зовнішні сервіси та інтеграції

- **Cloudinary**[7]

  Використовується для зберігання та обробки зображень рецептів (завантаження, оптимізація, трансформація).

Інструменти та бібліотеки

- **MediatR** - реалізація патерну mediator для організації CQRS[8]
- **Npgsql** - драйвер для роботи з PostgreSQL[10]
- **Swagger** - документація API
- **FluentValidation.AspNetCore** - інтеграція валідації в API
- **Optional** - робота з optional-типами (відсутність значення без null-помилок)

Інфраструктурна організація

Програмна система розділена на логічні рівні:

- **Domain** - бізнес-сутності (Recipe, User, Category тощо)
- **Application** - бізнес-логіка, команди, валідація
- **Infrastructure** - робота з БД, зовнішні сервіси
- **Api** - контролери та HTTP-інтерфейс

Обґрунтування вибору архітектурного та технологічного стеку бекенду:

- **Чому .NET (C#), а не Node.js чи Python (FastAPI)?**

Платформа .NET (ASP.NET Core) була обрана через її високу продуктивність, вбудовану безпеку та сувору типізацію мови C#. На відміну від Node.js, .NET має потужну інфраструктуру для побудови великих застосунків (вбудований DI-контейнер, розвинена екосистема ORM). Порівняно з Python, .NET демонструє значно вищу швидкість обробки HTTP-запитів та кращу оптимізацію використання оперативної пам'яті під високими навантаженнями.

- **Чому Clean Architecture та CQRS, а не класичний тришаровий MVC (Layered)?**

Використання Clean Architecture дозволило повністю ізолювати бізнес-логіку (Domain та Application) від зовнішніх чинників, бази даних (PostgreSQL) та сторонніх сервісів (Cloudinary). Це гарантує, що зміни в інфраструктурі не зламають логіку створення рецептів чи профілів користувачів. Патерн CQRS (за допомогою MediatR) було впроваджено для чіткого розділення операцій створення/модифікації даних (Commands) та їх читання (Queries). Це спрощує масштабування системи, оскільки логіку виведення складних списків рецептів із фільтрацією можна оптимізувати окремо від важкої логіки запису даних.

- **Чому PostgreSQL, а не MySQL чи MongoDB (NoSQL)?**

Для збереження даних у системі «YUMTIME» було обрано PostgreSQL. Дана система керування базами даних добре підходить для роботи зі структурованими даними та великою кількістю зв’язків між сутностями, такими як користувачі, рецепти, інгредієнти та категорії. У порівнянні з MongoDB, PostgreSQL краще забезпечує цілісність реляційних даних і підтримку зв’язків між таблицями. У порівнянні з MySQL, PostgreSQL має ширші можливості для роботи зі складними SQL-запитами, підтримує JSON-типи даних та забезпечує стабільну роботу системи при одночасному виконанні багатьох операцій.

2. Основні технології фронтенд-частини

Фронтенд-частина системи «YUMTIME» реалізована за допомогою сучасного JavaScript-фреймворку Next.js, який побудований на базі бібліотеки React. Використання Next.js забезпечує підтримку серверного рендерингу (SSR), маршрутизації та оптимізації продуктивності вебзастосунку.

Основні технології фронтенд-частини:

- **Next.js** - використовується для побудови клієнтської частини системи, організації маршрутизації сторінок, серверного рендерингу та взаємодії з API. У проекті застосовано App Router та асинхронне завантаження даних [4].
- **React** - бібліотека для створення компонентного користувацького інтерфейсу. Інтерфейс системи побудований із багаторазових UI-компонентів (модальні вікна, списки, картки рецептів, форми тощо) [5].
- **TypeScript** - використовується для статичної типізації коду, що дозволяє зменшити кількість помилок та підвищити надійність програмної реалізації [13].
- **Tailwind CSS** - CSS-фреймворк для стилізації інтерфейсу. Дозволяє швидко створювати адаптивний та сучасний дизайн без написання великої кількості окремих CSS-файлів [12].
- **Fetch API** - використовується для обміну даними між фронтендом та бекендом через HTTP-запити до ASP.NET Core Web API.
- **Middleware Next.js** - застосовується для перевірки JWT-токенів та реалізації базової логіки маршрутизації користувачів між сторінками авторизації.
- **React Hooks** (useState, useRouter, usePathname) - використовуються для керування станом компонентів, навігації та динамічної роботи інтерфейсу.
- **React Icons** - бібліотека іконок для покращення візуального оформлення інтерфейсу.
- **Next/Image та Next/Link** - вбудовані компоненти Next.js для оптимізації зображень та навігації між сторінками.

Фронтенд взаємодіє з бекендом через REST API, отримуючи та надсилаючи дані у форматі JSON. Такий підхід забезпечує незалежність клієнтської та серверної частин системи, що спрощує подальше масштабування та підтримку програмного продукту.

Обґрунтування вибору технологічного стеку фронтенду:

- **Чому Next.js, а не чистий React (SPA)?**

Класичний React-застосунок (Single Page Application) завантажує порожню HTML-сторінку та рендерить інтерфейс на стороні клієнта. Для платформи «YUMTIME» це критично, адже кулінарний портал потребує якісної індексації пошуковими роботами (SEO) та швидкого першого відображення сторінки з рецептом для користувача. Next.js вирішує цю проблему за допомогою серверного рендерингу (SSR) та генерації статичних сторінок (SSG). Крім того, вбудована маршрутизація (App Router) та оптимізація зображень (Next/Image) дозволили уникнути підключення важких сторонніх бібліотек.

- **Чому TypeScript, а не JavaScript?**

Використання TypeScript мінімізує виникнення помилок на етапі розробки (compile-time), забезпечуючи строгу типізацію об'єктів. Це особливо важливо під час типізації складних DTO-моделей, які приходять із бекенду (наприклад, структура об'єкта рецепта із вкладеними масивами інгредієнтів та кроків приготування).

- **Чому Tailwind CSS?**

Tailwind CSS було обрано для розробки користувацького інтерфейсу системи «YUMTIME», оскільки даний фреймворк дозволяє швидко створювати сучасний та адаптивний дизайн. Використання utility-класів забезпечує зручне стилізування компонентів без необхідності написання великої кількості окремих CSS-файлів.
Також Tailwind CSS добре підходить для роботи з адаптивною верстkою, що дозволило забезпечити коректне відображення вебсистеми як на комп’ютерах, так і на мобільних пристроях.

3. Інструменти та середовище розробки

Під час розробки системи «YUMTIME» використовувався набір сучасних інструментів та програмних засобів, що забезпечують ефективне створення, тестування та супровід програмного продукту.

Основні інструменти розробки:

- **JetBrains Rider** - інтегроване середовище розробки (IDE), яке використовувалося для створення серверної частини системи на платформі ASP.NET Core. Rider забезпечує підтримку мови C#, автоматичне форматування коду, налагодження програми, роботу з базою даних та інтеграцію із системою контролю версій.
- **Visual Studio Code** - редактор коду, який використовувався для розробки фронтенд-частини системи на основі Next.js та TypeScript. Завдяки великій кількості розширень VS Code забезпечує зручну роботу з JavaScript/TypeScript, Tailwind CSS та React-компонентами.
- **Swagger (OpenAPI)** - інструмент для автоматичної генерації та тестування документації REST API. Swagger дозволяє переглядати доступні HTTP-методи, структуру запитів і відповідей, а також тестувати API без використання сторонніх програм.
- **GitHub** - система контролю версій та хмарний сервіс для зберігання програмного коду. Використовувався для ведення історії змін проекту, резервного копіювання та організації структури репозиторію.
- **Docker** - платформа контейнеризації, яка використовувалася для ізольованого запуску компонентів системи та спрощення процесу розгортання програмного забезпечення. Docker забезпечує однакове середовище виконання незалежно від операційної системи.

Використання зазначених інструментів дозволило оптимізувати процес розробки, спростити тестування системи та забезпечити зручний супровід програмного продукту.

3.2 Вимоги до технічного та програмного забезпечення

Опис програмного забезпечення системи «YUMTIME» виконано відповідно до стандартної моделі взаємодії відкритих систем OSI. Це дозволяє розділити програми за їхнім призначенням на три основні групи: системне, інструментальне та прикладне ПЗ.

1. **Системне програмне забезпечення**. Це базова основа, на якій працює весь комп'ютер та сервер:
    - **Операційна система:** Windows 10 Pro використовується на комп'ютері під час розробки сайту, а операційна система Linux (в Docker-контейнерах) потрібна для роботи сервера, де розгорнуто сайт;
    - **Мережна СУБД:** Система керування базами даних PostgreSQL. Вона працює в мережі як окремий сервіс і відповідає за надійне зберігання всіх рецептів, користувачів та інгредієнтів;
    - **Сервісні програми супроводу:** вбудовані засоби JetBrains Rider для перегляду структури бази даних, наповнення таблиць та побудови ER-діаграм, а також внутрішня мережа Docker [20] для взаємодії компонентів системи.
2. **Інструментальне програмне забезпечення**. Це програми та інструменти, які потрібні безпосередньо для створення проекту:
    - **Платформи розробника:** .NET SDK 8.0 (для збирання та запуску серверного коду) та Node.js (для запуску клієнтської частини сайту);
    - **Середовища розробки:** Редактори кодів JetBrains Rider та Visual Studio Code;
    - **Допоміжні утиліти:** Інструмент Swagger/OpenAPI [19] для тестування REST API та система контролю версій Git для керування змінами програмного коду.
3. **Функціональне (прикладне) програмне забезпечення**. Це безпосередньо ті програми, які виконують головні завдання сайту «YUMTIME»:

- **Серверна частина (Бекенд):** Веб-додаток на ASP.NET Core 8.0, який обробляє всі запити, рахує КБЖУ та зв'язується зі штучним інтелектом для створення нових рецептів;
- **Клієнтська частина (Фронтенд):** Інтерфейс сайту на фреймворку Next.js, який користувач бачить у себе на екрані (каталог страв, мапа, список покупок);
- **Програмне забезпечення АРМ (робочого місця):** Звичайний веб-браузер (Google Chrome), який використовується як клієнтом, так і адміністратором для управління сайтом.

Взаємодія компонентів системи за рівнями моделі OSI

Робота проекту в комп'ютерній мережі просто пояснюється через рівні моделі OSI:

1. **Прикладний рівень:** Браузер користувача відправляє запити на сайт (наприклад, натискання кнопки «Згенерувати рецепт через ШІ»).
2. **Представницький рівень:** Передача даних між сервером та браузером відбувається у простому і зрозумілому форматі JSON.
3. **Сеансовий рівень:** Перевірка прав користувача (хто зайшов: адмін чи звичайний гість) за допомогою безпечних токенів (JWT).
4. **Транспортний рівень:** Надійний обмін повідомленнями між сервером та базою даних PostgreSQL через мережевий протокол TCP.
5. **Нижні рівні (Мережевий, Канальний, Фізичний):** Передача інформації у вигляді пакетів через домашній Інтернет, роутери та мережеві карти.

Таким чином, обране програмне забезпечення забезпечує стабільну роботу системи «YUMTIME», підтримує сучасні підходи до веброзробки та дозволяє реалізувати масштабовану клієнт-серверну архітектуру.

3.3 Опис програмної реалізації

Програмна реалізація системи «YUMTIME» виконана відповідно до принципів багатошарової архітектури Clean Architecture. Такий підхід дозволив відокремити бізнес-логіку від механізмів доступу до даних, зовнішніх сервісів та користувацького інтерфейсу. Система реалізована у вигляді клієнт-серверного вебзастосунку, де бекенд побудований на платформі ASP.NET Core Web API, а фронтенд на основі Next.js.

Структура програмної системи

Програмний продукт складається з чотирьох основних рівнів **(рис. 3.1)**:

1. **Domain**

   Містить основні бізнес-сутності системи (Recipe, User, Category, Cuisine, Ingredient, Rating, RecipeIngredient) та доменні правила.

   На цьому рівні реалізовано:

   - логіку створення та оновлення сутностей;
   - розрахунок КБЖУ рецепта;
   - обчислення середнього рейтингу;
   - strongly typed identifiers (RecipeId, CategoryId тощо).

2. **Application**

   Реалізує прикладну бізнес-логіку системи.

   Основні компоненти:

   - CQRS-команди та запити;
   - валідатори FluentValidation;
   - обробники MediatR;
   - інтерфейси репозиторіїв та сервісів;
   - класи Result для обробки результатів виконання операцій.

3. **Infrastructure**

   Відповідає за взаємодію із зовнішніми сервісами та базою даних.

   На цьому рівні реалізовано:

   - ApplicationDbContext;

- конфігурації Entity Framework Core;
- репозиторії доступу до даних;
- JWT-автентифікацію;
- сервіс хешування паролів BCrypt;
- інтеграцію із Cloudinary для роботи із зображеннями.

4. **Api**

Забезпечує HTTP-взаємодію із клієнтською частиною системи.

Рівень містить:

- REST API контролери;
- DTO-об’єкти;
- middleware;
- Swagger-документацію;
- глобальну валідацію та обробку помилок.

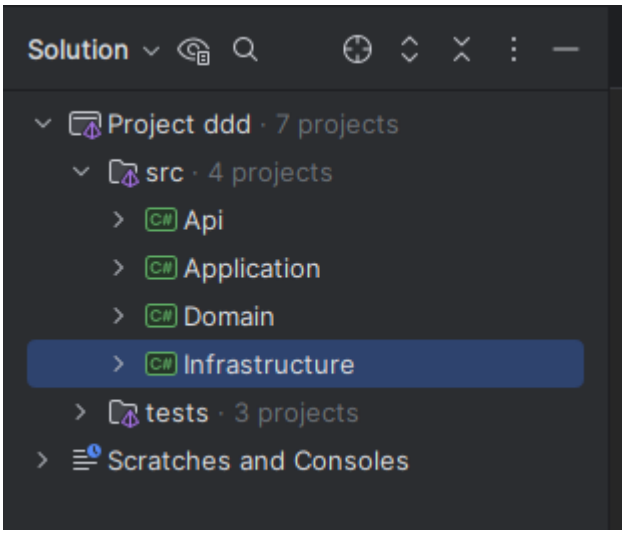

Рис. 3.1. Структура рішень та шарів Clean Architecture серверної частини застосунку

Реалізація серверної частини

Серверна частина побудована на ASP.NET Core Web API та реалізує REST-архітектуру.

Основними функціональними модулями є:

- модуль користувачів;
- модуль рецептів;
- модуль категорій;

- модуль інгредієнтів;
- модуль кухонь світу;
- модуль оцінювання;
- AI-модуль генерації рецептів.

Для взаємодії з базою даних використовується Entity Framework Core та PostgreSQL. Конфігурація зв’язків між сутностями виконується через Fluent API.

Для прикладу, сутність Recipe містить:

- інформацію про автора рецепта;
- категорію;
- кухню світу;
- список інгредієнтів;
- список оцінок;
- методи автоматичного розрахунку КБЖУ;
- метод обчислення середнього рейтингу.

Обробка запитів реалізована за допомогою патерну CQRS. Команди відповідають за зміну стану системи, а запити за отримання даних.

Для забезпечення безпеки використовується JWT-автентифікація. Після успішного входу користувач отримує токен, який використовується для доступу до захищених API-методів.

Паролі користувачів не зберігаються у відкритому вигляді. Для їх захисту застосовується алгоритм BCrypt.

Реалізація клієнтської частини

Клієнтська частина системи створена за допомогою Next.js та React.

Інтерфейс системи побудований за компонентним підходом **(рис. 3.2)**. Основними компонентами є:

- сторінки рецептів;
- форми створення та редагування;
- модальні вікна;

- блоки категорій;
- картки рецептів;
- карта кухонь світу;
- AI-генератор рецептів.

Для стилізації використовується Tailwind CSS, що забезпечує адаптивність та сучасний вигляд інтерфейсу.

Взаємодія з бекендом реалізована через Fetch API. Дані передаються у форматі JSON.

У системі реалізовано:

- динамічне оновлення даних;
- серверне завантаження сторінок;
- middleware-перевірку JWT-токенів;
- глобальні loading-компоненти;
- обробку помилок API.

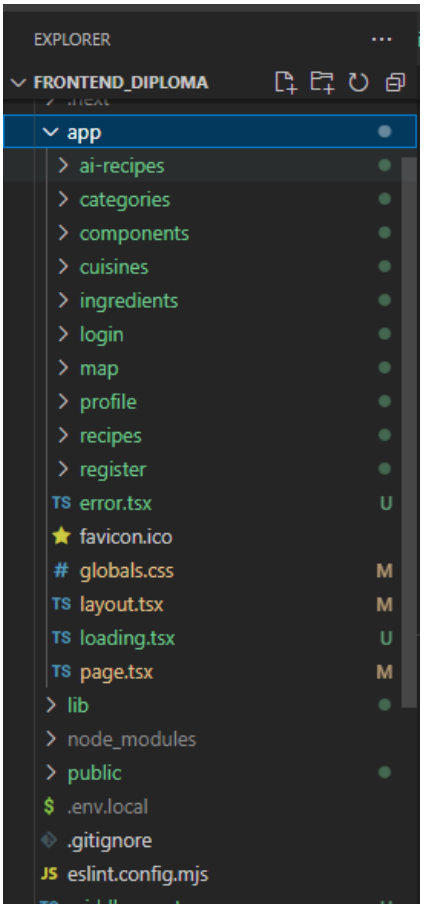

Рис. 3.2. Архітектурна структура папок та маршрутизації клієнтської частини Next.js

Реалізація роботи із зображеннями

Для завантаження та зберігання фотографій рецептів використовується сервіс Cloudinary.

Під час завантаження:

- файл передається на сервер;
- виконується автоматична трансформація зображення;
- зображення масштабується до заданого розміру;
- система отримує URL готового файлу.

Отримане посилання зберігається у базі даних та використовується для відображення рецепта у вебінтерфейсі.

Реалізація валідації даних

Для перевірки коректності вхідних даних використовується FluentValidation.

Валідація виконується:

- для DTO-об’єктів;
- для CQRS-команд;
- перед збереженням інформації у базі даних.

Перевіряються:

- обов’язковість полів;
- довжина рядків;
- допустимі числові значення;
- коректність email;
- межі рейтингу;
- унікальність окремих записів.

Реалізація API-документації

Для тестування та документування REST API використовується Swagger.

Swagger дозволяє:

- переглядати доступні маршрути API;
- тестувати HTTP-запити;

- перевіряти авторизацію JWT;
- переглядати структуру DTO та відповідей сервера.

Це значно спрощує процес тестування та інтеграції фронтенд- і бекенд-частин системи.

3.4 Керівництво користувача

Цей підрозділ містить інструкції щодо взаємодії користувача з інформаційною системою «YUMTIME», опис основних функціональних можливостей та демонстрацію інтерфейсу застосунку.

Реєстрація та автентифікація

Для отримання доступу до функцій створення рецептів та оцінювання, користувач повинен пройти процес реєстрації **(рис. 3.3)**:

1. **Перехід на сторінку реєстрації:** Користувач натискає кнопку «Register» у верхньому меню навігації.
2. **Заповнення даних:** Необхідно ввести ім'я, прізвище, електронну адресу та пароль. Система автоматично перевіряє складність пароля та коректність email за допомогою FluentValidation на бекенді.
3. **Вхід у систему:** Після успішної реєстрації користувач переспрямовується на сторінку «Login», де вводить свої облікові дані для отримання JWT-токена.

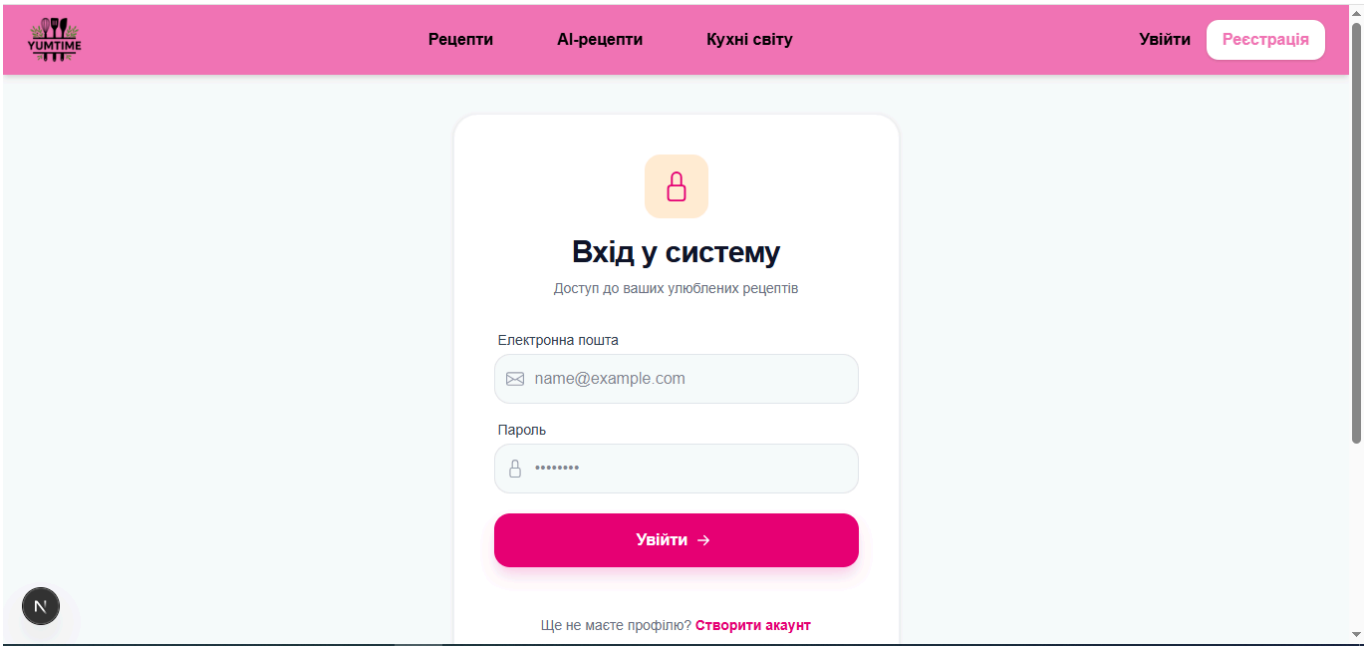

Рис. 3.3. Форма реєстрації та автентифікації користувача

Робота з рецептами

Основний функціонал системи дозволяє користувачам переглядати та додавати власні кулінарні рецепти (**рис. 3.4**, **рис. 3.5**):

1. **Додавання рецепта:** У профілі користувача або через головне меню необхідно обрати «Add Recipe».
2. **Заповнення форми:** Користувач вказує назву, категорію, обирає кухню світу, додає перелік інгредієнтів та покрокову інструкцію.
3. **Завантаження фото:** За допомогою сервісу Cloudinary користувач додає зображення страви, яке автоматично оптимізується для відображення.
4. **Перегляд:** Після збереження рецепт стає доступним у загальному списку та на персональній сторінці автора.

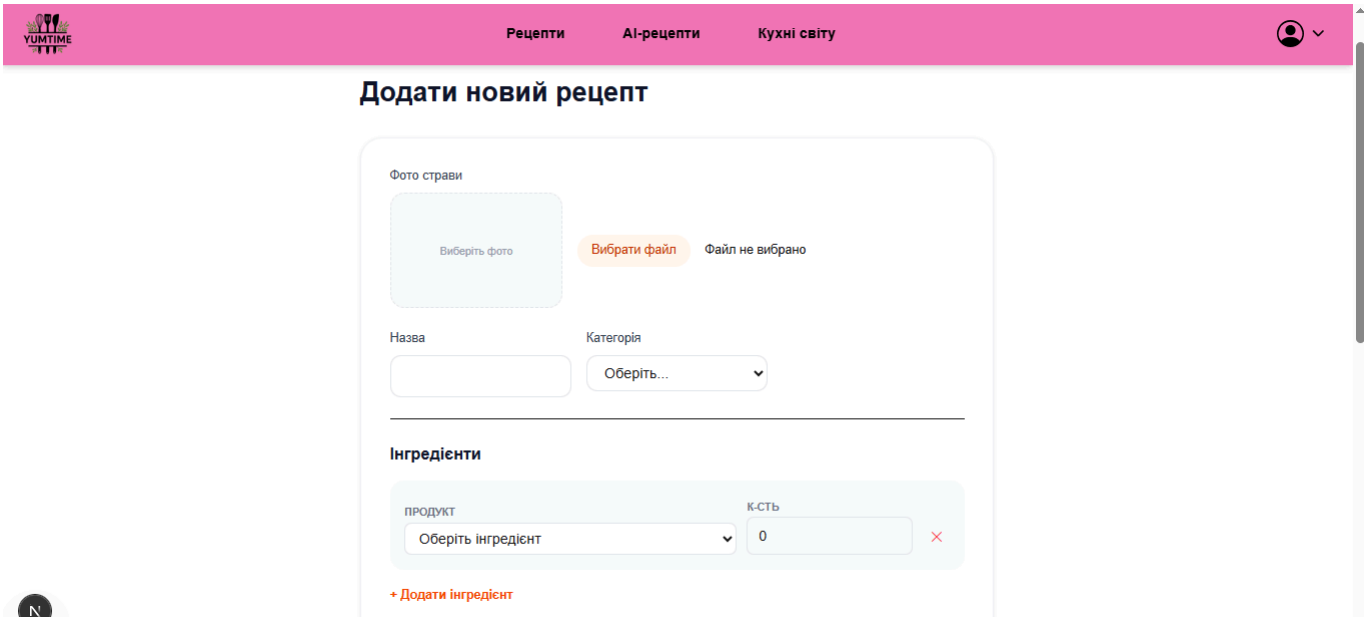

Рис. 3.4. Інтерфейс додавання нового рецепта

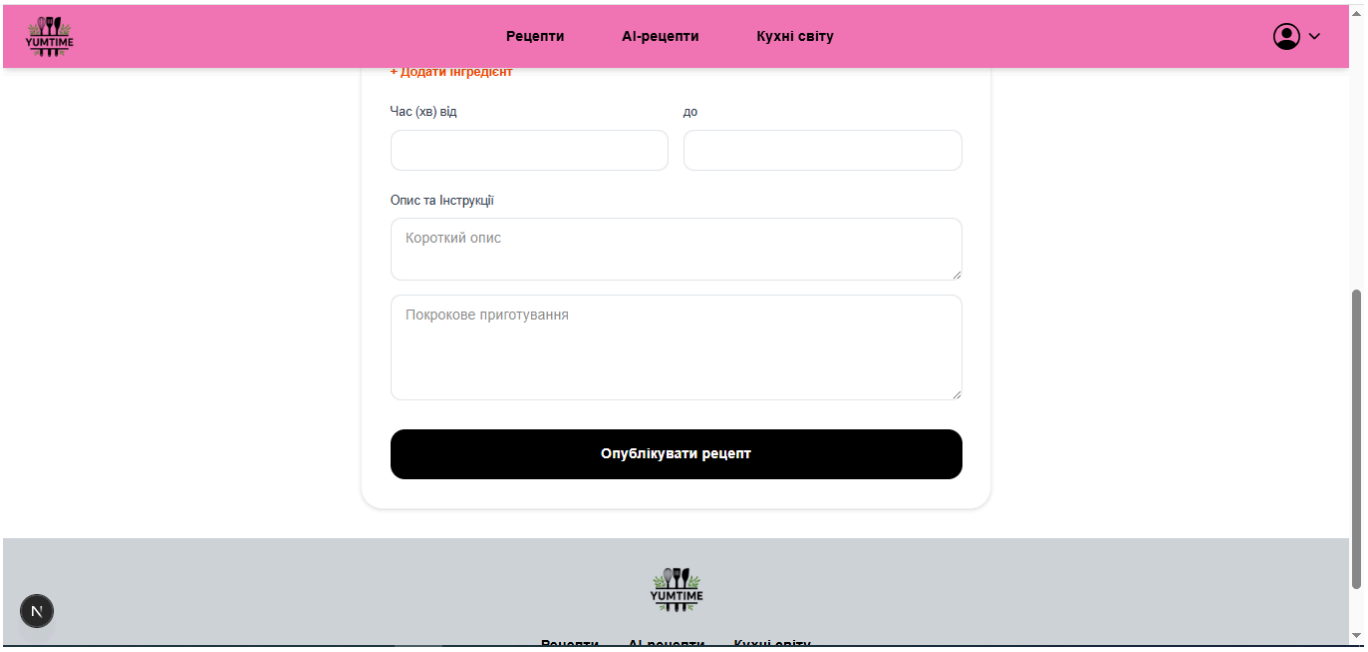

Рис. 3.5. Інтерфейс додавання нового рецепта

Перегляд та пошук рецептів

Після входу в систему користувачу відкривається головна сторінка з усіма доступними рецептами:

1. **Стрічка рецептів:** На головній сторінці відображаються картки рецептів, кожна з яких містить фото, назву, короткий опис, рейтинг та розраховані показники КБЖУ (рис. 3.6).
2. **Детальна інформація:** При натисканні на картку відкривається повна сторінка рецепта (рис. 3.7). Вона містить:
   - Велике зображення страви та її назву.
   - Категорію (наприклад, «Перші страви») та орієнтовний час приготування.
   - Блок із розрахунком енергетичної цінності (калорії, білки, жири, вуглеводи).
   - Детальний список інгредієнтів із зазначенням ваги та калорійності кожного продукту.
   - Покроковий спосіб приготування.

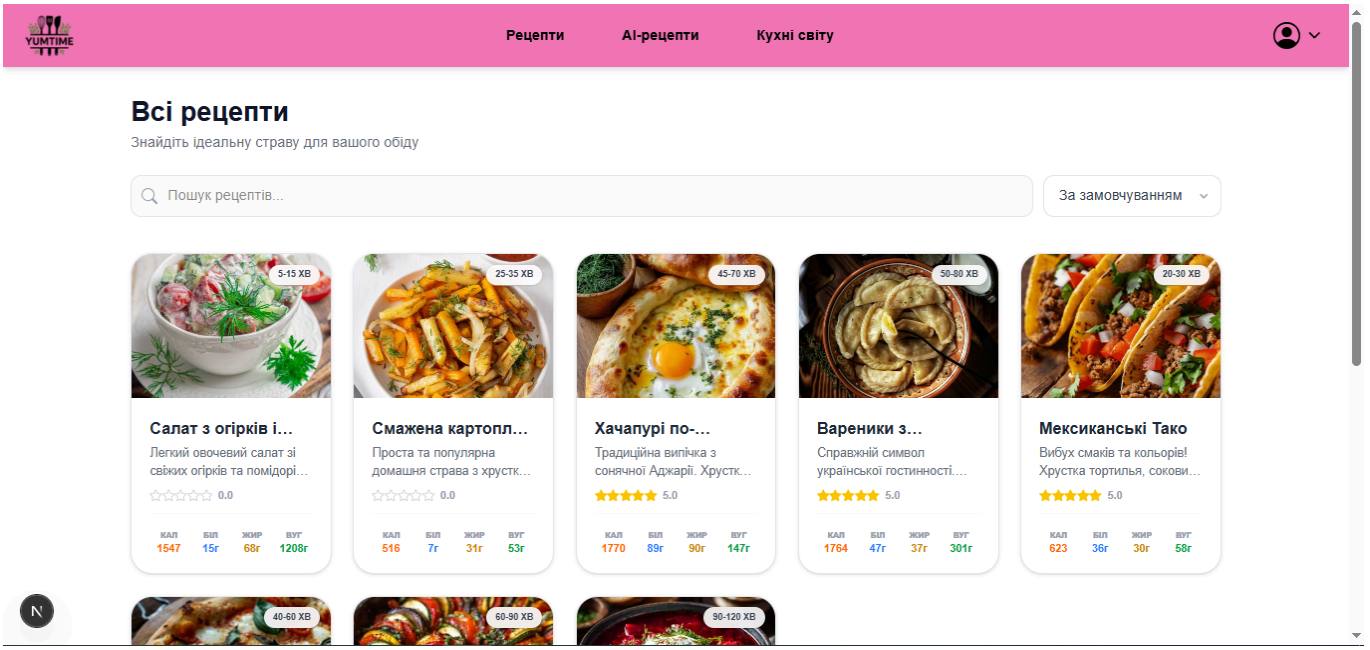

Рис. 3.6. Сторінка каталогу рецептів із відображенням карток страв

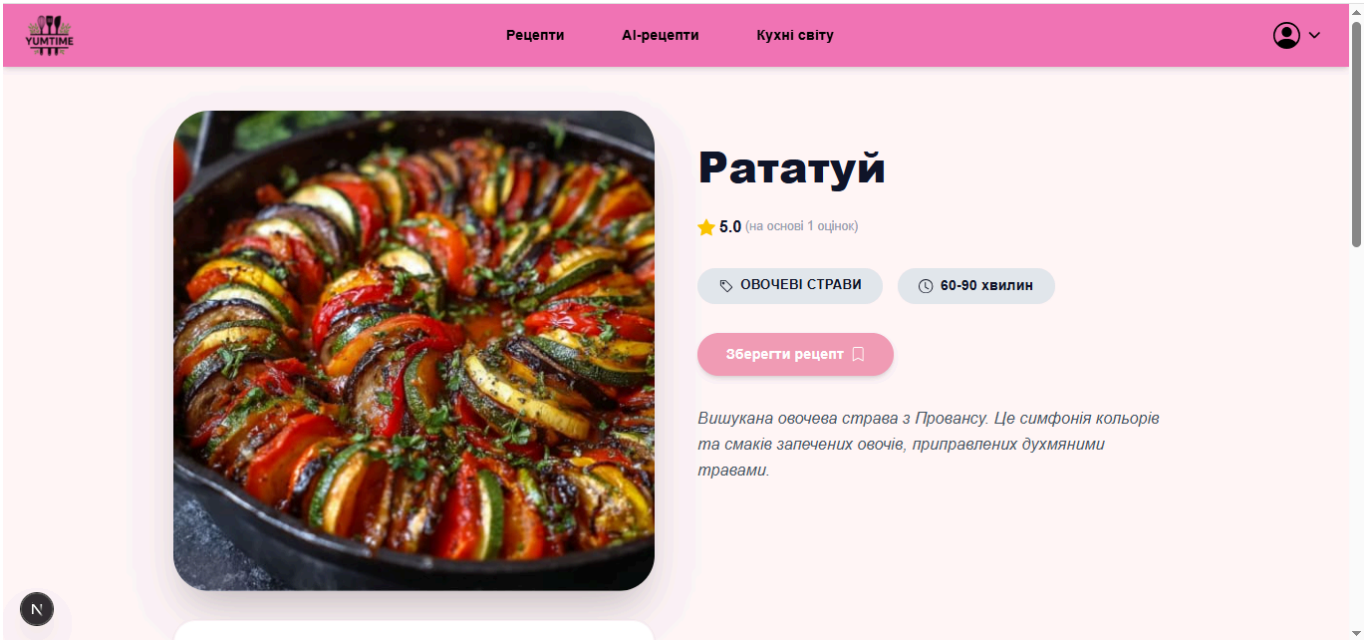

Рис. 3.7. Відображення детальної інформації про рецепт

Оцінювання та інтерактивність

Система «YUMTIME» дозволяє користувачам впливати на рейтинг страв:

1. **Виставлення оцінки:** На сторінці конкретного рецепта користувач може обрати оцінку від 1 до 5 зірок. Система автоматично перераховує середній рейтинг страви на основі всіх відгуків **(рис. 3.8)**.
2. **Список покупок:** Користувач може натиснути кнопку «Створити список покупок», щоб згенерувати перелік необхідних інгредієнтів для походу в магазин **(рис. 3.9, рис. 3.10)**.

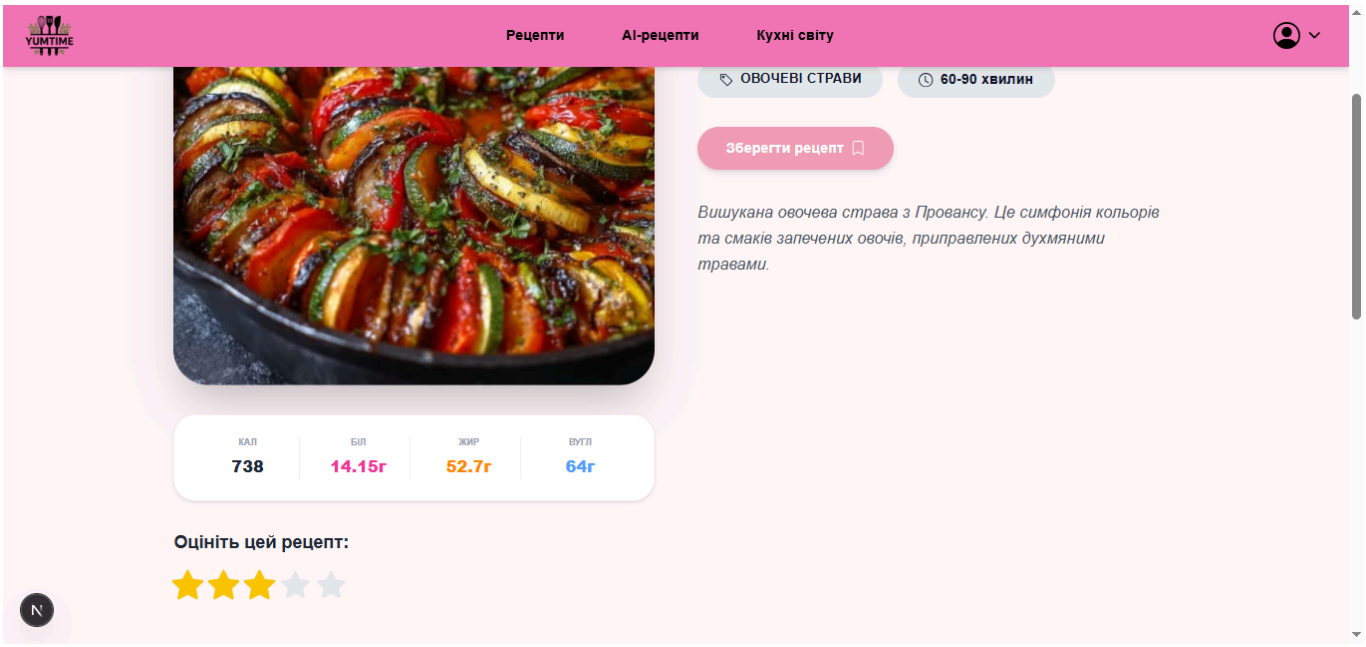

Рис. 3.8. Відображення детальної інформації про рецепт, розрахунок енергетичної цінності та оцінка рецепту

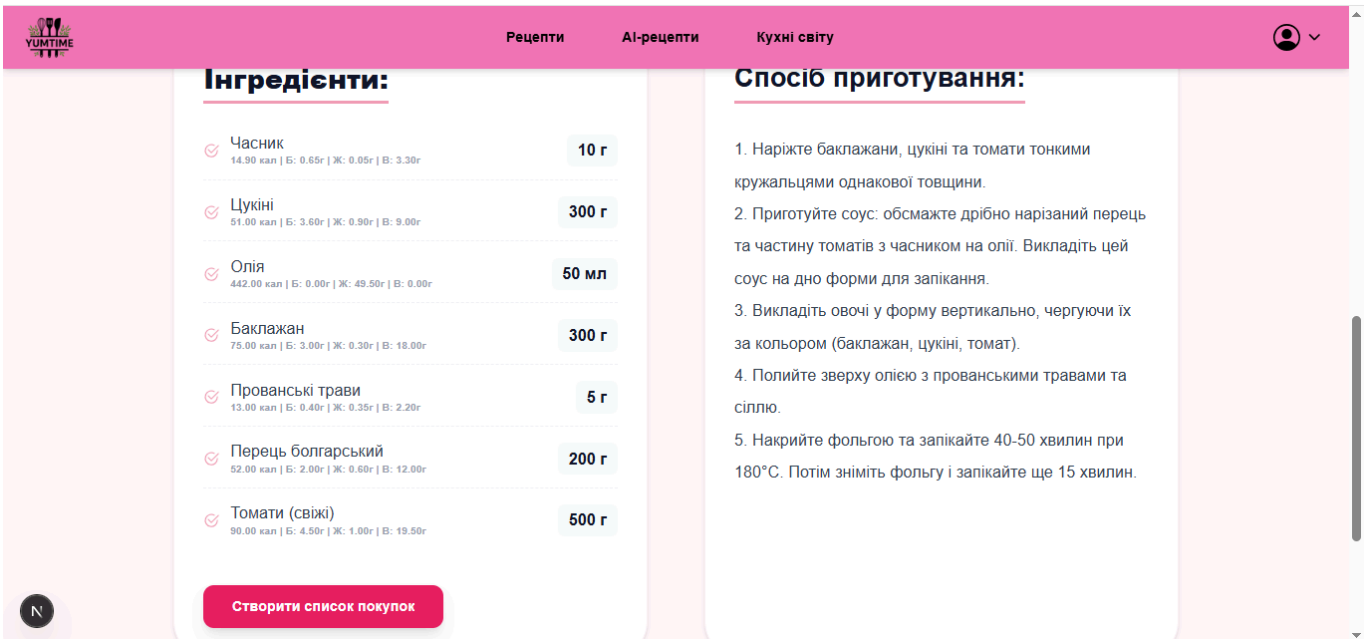

Рис. 3.9. Відображення детальної інформації про рецепт, інгредієнти, спосіб приготування

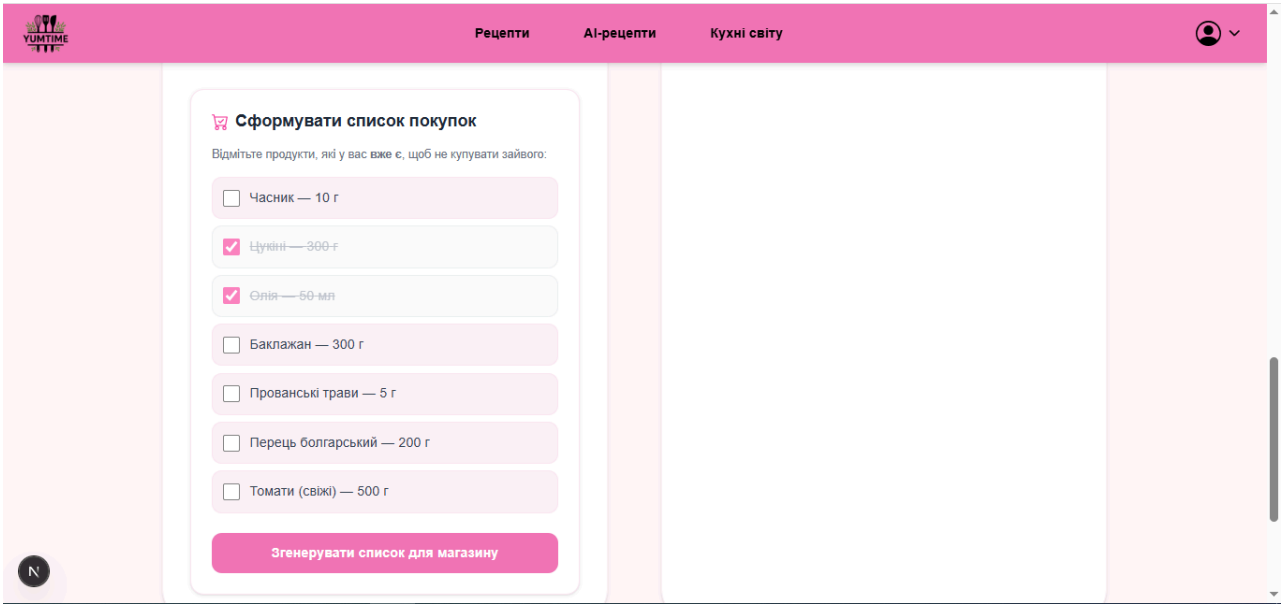

Рис. 3.10. Відображення детальної інформації про рецепт, список покупок

Інтерактивна карта кухонь світу

Для зручного пошуку страв у системі реалізовано модуль карти **(рис. 3.11)**:

1. **Відкриття карти:** Користувач переходить у розділ «Map».
2. **Вибір локації:** На інтерактивній карті (React Leaflet) відображаються маркери різних країн.
3. **Фільтрація:** При натисканні на маркер або країну система відображає список рецептів, що належать до обраної національної кухні.

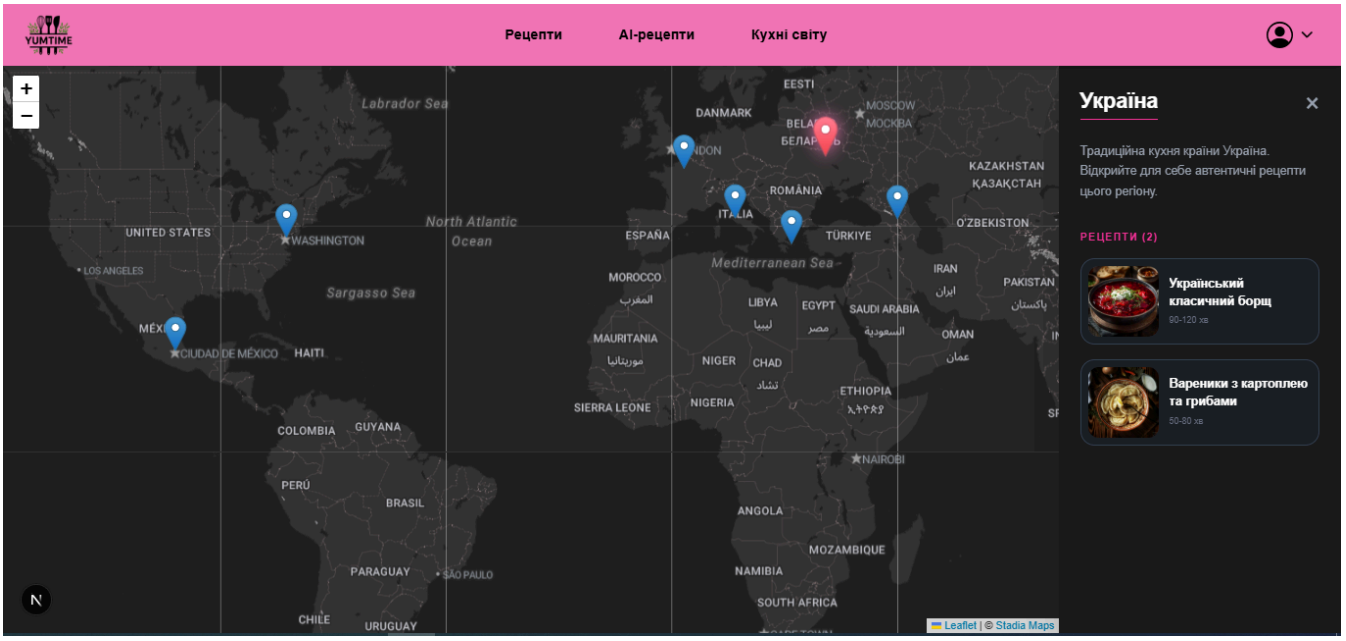

Рис. 3.11. Інтерактивна карта кухонь світу на базі React Leaflet

AI-генерація рецептів

Інноваційною частиною системи є можливість створення рецептів на основі штучного інтелекту (рис. 3.12):

1. **Введення інгредієнтів:** Користувач вводить у текстове поле перелік продуктів, які має в наявності.
2. **Генерація:** Після натискання кнопки «Згенерувати магію» система звертається до AI-модуля.
3. **Результат:** Користувач отримує детальний рецепт із назвою, часом приготування та інструкціями.

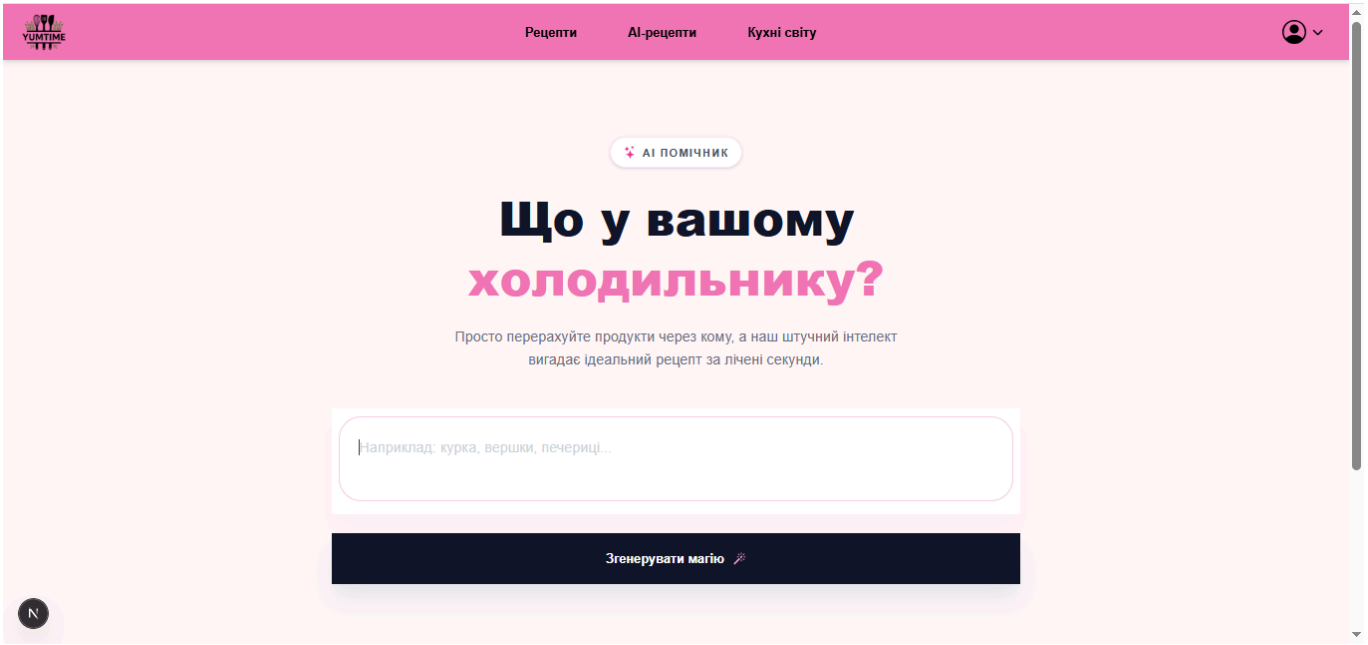

Рис. 3.12. Модуль AI-генерації рецептів за наявними інгредієнтами

Висновки до розділу 3

У третьому розділі було детально розглянуто програмну та технічну реалізацію інформаційної системи «YUMTIME». За результатами проведеної розробки можна зробити наступні висновки:

1. **Обґрунтовано вибір технологічного стека.** Використання платформи **.NET** для бекенду та фреймворку **Next.js** для фронтенду дозволило створити високопродуктивну систему з чітким розділенням зон відповідальності. Вибір

реляційної бази даних **PostgreSQL** забезпечив надійне зберігання структурованих даних користувачів та рецептів.

2. **Реалізовано багаторівневу архітектуру.** Застосування принципів **Clean Architecture** та патерну **CQRS** дозволило відокремити бізнес-логіку від інфраструктурних деталей. Це забезпечило високу тестованість коду, спростило підтримку системи та її подальше масштабування.
3. **Забезпечено високий рівень безпеки та валідації.** Інтеграція **JWT-автентифікації** та хешування паролів за допомогою **BCrypt** гарантує захист персональних даних користувачів. Використання бібліотеки **FluentValidation** на обох рівнях системи мінімізує ризик потрапляння некоректних даних у базу.
4. **Впроваджено сучасні функціональні модулі.** Реалізація інтерактивної карти за допомогою **Leaflet** та модуля **AI-генерації** рецептів підвищує залученість користувачів та виділяє систему серед існуючих аналогів. Використання хмарного сервісу **Cloudinary** дозволило ефективно вирішити питання зберігання та швидкої доставки мультимедійного контенту.
5. **Розроблено інтуїтивно зрозумілий інтерфейс.** Завдяки використанню **Tailwind CSS** та компонентного підходу **React**, створено адаптивний дизайн, який забезпечує комфортну роботу з системою як на стаціонарних комп'ютерах, так і на мобільних пристроях.

Таким чином, розроблена програмна система повністю відповідає поставленим вимогам, є технічно стабільною та готовою до експлуатації користувачами.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було розроблено інформаційну систему «YUMTIME», призначену для зберігання, пошуку, створення та аналізу кулінарних рецептів. Основною метою роботи було створення сучасного вебзастосунку, який поєднує зручний користувацький інтерфейс, функції автоматизованого розрахунку поживної цінності страв, механізми оцінювання рецептів та інтеграцію із сервісами штучного інтелекту. У ході виконання роботи поставлена мета була повністю досягнута.

Під час дослідження було проведено аналіз предметної області кулінарних інформаційних систем, визначено основні функціональні вимоги та сформовано інформаційну модель системи. На основі цього було спроєктовано структуру бази даних, побудовано ER-модель та реалізовано зв’язки між основними сутностями: користувачами, рецептами, категоріями, кухнями світу, інгредієнтами та рейтингами.

У процесі програмної реалізації було використано сучасні технології та архітектурні підходи. Серверну частину системи реалізовано на платформі ASP.NET Core Web API із використанням мови програмування C#, ORM-технології Entity Framework Core та реляційної бази даних PostgreSQL. Для забезпечення масштабованості та підтримуваності програмного коду застосовано принципи Clean Architecture, CQRS та Dependency Injection.

Клієнтську частину системи реалізовано за допомогою Next.js, React та TypeScript. Для створення адаптивного інтерфейсу використано Tailwind CSS. Взаємодія між фронтендом і бекендом здійснюється через REST API у форматі JSON, що забезпечує незалежність компонентів системи та спрощує її подальший розвиток.

Особливу увагу приділено забезпеченню безпеки та коректності обробки даних. У системі реалізовано JWT-автентифікацію та механізм хешування паролів BCrypt. Для перевірки вхідних даних використано бібліотеку FluentValidation, що

дозволило мінімізувати ризик виникнення помилок і некоректних записів у базі даних.

Практичним результатом роботи стала реалізація функціональних модулів системи, серед яких:

- створення та редагування рецептів;
- автоматичний розрахунок КБЖУ;
- система оцінювання та рейтингу страв;
- інтерактивна карта кухонь світу;
- AI-генератор рецептів на основі введених інгредієнтів;
- автоматичне формування списку покупок.

Розроблена система має зручний та інтуїтивно зрозумілий інтерфейс, підтримує адаптивне відображення на різних пристроях та може бути використана як основа для подальшого розвитку кулінарних платформ.

Отримані результати мають практичну цінність та можуть бути використані для створення сучасних вебсервісів у сфері кулінарії, автоматизації зберігання рецептів і персоналізованих рекомендацій харчування. У перспективі система може бути розширена шляхом впровадження повноцінної системи коментарів, рекомендацій на основі вподобань користувача, мобільного застосунку та інтеграції з додатковими AI-сервісами.

СПИСОК ДЖЕРЕЛ

1. Microsoft. .NET Documentation. URL: https://learn.microsoft.com/en-us/dotnet/ (дата звернення: 10.03.2026).
2. Microsoft. ASP.NET Core Documentation. URL: https://learn.microsoft.com/en-us/aspnet/core/ (дата звернення: 12.03.2026).
3. Entity Framework Core. EF Core Documentation. URL: https://learn.microsoft.com/en-us/ef/core/ (дата звернення: 15.03.2026).
4. Next.js. Next.js Documentation. URL: https://nextjs.org/docs (дата звернення: 18.03.2026).
5. React. React Documentation. URL: https://react.dev/ (дата звернення: 20.03.2026).
6. PostgreSQL. PostgreSQL Official Documentation. URL: https://www.postgresql.org/docs/ (дата звернення: 22.03.2026).
7. Cloudinary. Cloudinary DotNet SDK. URL: https://cloudinary.com/documentation/dotnet_integration (дата звернення: 25.03.2026).
8. MediatR. MediatR Documentation. URL: https://github.com/jbogard/MediatR/wiki (дата звернення: 28.03.2026).
9. FluentValidation. FluentValidation Documentation. URL: https://docs.fluentvalidation.net/ (дата звернення: 02.04.2026).
10. Npgsql. Npgsql Documentation. URL: https://www.npgsql.org/efcore/ (дата звернення: 05.04.2026).
11. JSON Web Tokens. JWT.io Introduction. URL: https://jwt.io/introduction (дата звернення: 07.04.2026).
12. Tailwind CSS. Tailwind CSS Documentation. URL: https://tailwindcss.com/docs (дата звернення: 10.04.2026).
13. TypeScript. TypeScript Documentation. URL: https://www.typescriptlang.org/docs/ (дата звернення: 12.04.2026).
14. BCrypt.Net. BCrypt.Net Library Documentation. URL: https://github.com/BcryptNet/bcrypt.net (дата звернення: 15.04.2026).

15. Leaflet. Leaflet - an open-source JavaScript library for interactive maps. URL: https://leafletjs.com/ (дата звернення: 18.04.2026).
16. OpenAI. OpenAI API Documentation. URL: https://platform.openai.com/docs/introduction (дата звернення: 20.04.2026).
17. Martin Fowler. Command Query Responsibility Segregation (CQRS). URL: https://martinfowler.com/bliki/CQRS.html (дата звернення: 22.04.2026).
18. Steve Smith. Clean Architecture. URL: https://ardalis.com/clean-architecture/ (дата звернення: 25.04.2026).
19. OpenAPI Initiative. OpenAPI Specification (Swagger). URL: https://spec.openapis.org/oas/latest.html (дата звернення: 28.04.2026).
20. Docker. Docker Documentation. URL: https://docs.docker.com/ (дата звернення: 02.05.2026).

Додаток А

Програмна реалізація доменної сутності кулінарного рецепта (Recipe.cs)

```
using Domain.Users;
using Domain.Categories;
using Domain.Cuisines;
using Domain.Ratings;
using Domain.RecipeIngredients;

namespace Domain.Recipes;

public class Recipe
{
   public RecipeId Id { get; }
   public UserId UserId { get; }
   public User User { get; }
   public CategoryId CategoryId { get; }
   public Category Category { get; }

   public string Title { get; private set; }
   public string Description { get; private set; }
   public string Instructions { get; private set; }
   public string? ImageUrl { get; private set; }
   public int CookTimeMin { get; private set; }
   public int CookTimeMax { get; private set; }
   public DateTime CreatedAt { get; private set; }
   public DateTime UpdatedAt { get; private set; }

   public List<RecipeIngredient> RecipeIngredients { get; private set;
} = new();
   public List<Rating> Ratings { get; private set; } = new();

   public CuisineId? CuisineId { get;private set;}
   public Cuisine? Cuisine { get;private set;}

   private Recipe(
```

Продовження додатку А

```
        RecipeId id,
        UserId userId,
        CategoryId categoryId,
        CuisineId? cuisineId,
        string title,
        string description,
        string instructions,
        string? imageUrl,
        int cookTimeMin,
        int cookTimeMax,
        DateTime createdAt,
        DateTime updatedAt)
    {
        Id = id;
        UserId = userId;
        CategoryId = categoryId;
        CuisineId = cuisineId;
        Title = title;
        Description = description;
        Instructions = instructions;
        ImageUrl = imageUrl;
        CookTimeMin = cookTimeMin;
        CookTimeMax = cookTimeMax;
        CreatedAt = createdAt;
        UpdatedAt = updatedAt;
    }

    public static Recipe New(
        RecipeId id,
        UserId userId,
        CategoryId categoryId,
        CuisineId? cuisineId,
        string title,
        string description,
```

Продовження додатку А

```
        string instructions,
        string? imageUrl,
        int cookTimeMin,
        int cookTimeMax)
        => new(id, userId, categoryId,cuisineId, title, description,
instructions, imageUrl, cookTimeMin, cookTimeMax, DateTime.UtcNow,
DateTime.UtcNow);

    public void UpdateDetails(
        string title,
        string description,
        string instructions,
        string? imageUrl,
        int cookTimeMin,
        int cookTimeMax,
        CuisineId? cuisineId)
    {
        Title = title;
        Description = description;
        Instructions = instructions;
        ImageUrl = imageUrl;
        CookTimeMin = cookTimeMin;
        CookTimeMax = cookTimeMax;
        UpdatedAt = DateTime.UtcNow;
        CuisineId = cuisineId;
        Cuisine = null;
    }
    public float TotalCalories => RecipeIngredients.Sum(ri =>
ri.GetCalories());
    public float TotalProteins => RecipeIngredients.Sum(ri =>
ri.GetProteins());
    public float TotalFats => RecipeIngredients.Sum(ri =>
ri.GetFats());
```

Продовження додатку А

```
    public float TotalCarbohydrates => RecipeIngredients.Sum(ri =>
ri.GetCarbohydrates());

   public double AverageRating => Ratings == null || !Ratings.Any()
       ? 0
       : Math.Round(Ratings.Average(r => r.Stars), 1);

   public int RatingsCount => Ratings?.Count ?? 0;
}
```

Додаток Б

Реалізація патерну CQRS для операції створення кулінарного рецепта

```
using Application.Common;
using Application.Common.Interfaces.Repositories;
using Application.Recipes.Exceptions;
using Domain.Categories;
using Domain.Cuisines;
using Domain.Recipes;
using Domain.Users;
using MediatR;
using Microsoft.AspNetCore.Http;

namespace Application.Recipes.Commands;

public class CreateRecipeCommand : IRequest<Result<Recipe,
RecipeException>>
{
   public required Guid UserId { get; init; }
   public required Guid CategoryId { get; init; }
   public Guid? CuisineId { get; init; }
   public required string Title { get; init; }
   public required string Description { get; init; }
   public required string Instructions { get; init; }
   public IFormFile? ImageFile { get; init; }
   public required int CookTimeMin { get; init; }
   public required int CookTimeMax { get; init; }
}

public class CreateRecipeCommandHandler(
   IRecipeRepository recipeRepository,
   IUserRepository userRepository,
   ICategoryRepository categoryRepository,
   ICuisineRepository cuisineRepository,
   IPhotoService photoService)
```

Продовження додатку Б

```
    : IRequestHandler<CreateRecipeCommand, Result<Recipe,
RecipeException>>
{
   public async Task<Result<Recipe, RecipeException>>
Handle(CreateRecipeCommand request,
       CancellationToken cancellationToken)
   {
       var userId = new UserId(request.UserId);
       var categoryId = new CategoryId(request.CategoryId);
       var cuisineId = request.CuisineId.HasValue ? new
CuisineId(request.CuisineId.Value) : null;


       var user = await userRepository.GetById(userId,
cancellationToken);
       string? uploadedImageUrl = null;
       if (request.ImageFile != null)
       {
           uploadedImageUrl = await
photoService.UploadPhotoAsync(request.ImageFile);
       }

       return await user.Match<Task<Result<Recipe, RecipeException>>>(
           async u =>
           {
               var category = await
categoryRepository.GetById(categoryId, cancellationToken);

               return await category.Match<Task<Result<Recipe,
RecipeException>>>(
                   async c =>
                   {
                       if (cuisineId is null)
                       {
```

Продовження додатку Б

```
                return await CreateEntity(
                        request.Title,
                        request.Description,
                        request.Instructions,
                        uploadedImageUrl,
                        request.CookTimeMin,
                        request.CookTimeMax,
                        userId,
                        categoryId,
                        null,
                        cancellationToken);
                }

                var cuisine = await
cuisineRepository.GetById(cuisineId, cancellationToken);

                return await cuisine.Match<Task<Result<Recipe,
RecipeException>>>(
                    async cu => await CreateEntity(
                        request.Title,
                        request.Description,
                        request.Instructions,
                        uploadedImageUrl,
                        request.CookTimeMin,
                        request.CookTimeMax,
                        userId,
                        categoryId,
                        cuisineId,
                        cancellationToken),
                    () => Task.FromResult<Result<Recipe,
RecipeException>>(
                        new
RecipeCuisineNotFoundException(cuisineId)));
            },
```

Продовження додатку Б

```
                () => Task.FromResult<Result<Recipe, RecipeException>>(
                        new
RecipeCategoryNotFoundException(categoryId)));
            },
            () => Task.FromResult<Result<Recipe, RecipeException>>(
                new RecipeUserNotFoundException(userId)));
    }

private async Task<Result<Recipe, RecipeException>> CreateEntity(
        string title,
        string description,
        string instructions,
        string? imageUrl,
        int cookTimeMin,
        int cookTimeMax,
        UserId userId,
        CategoryId categoryId,
        CuisineId? cuisineId,
        CancellationToken cancellationToken)
    {
        try
        {
            var entity = Recipe.New(
                RecipeId.New(),
                userId,
                categoryId,
                cuisineId,
                title,
                description,
                instructions,
                imageUrl,
                cookTimeMin,
                cookTimeMax);
```

Продовження додатку Б

```
      return await recipeRepository.Add(entity, cancellationToken);
        }
        catch (Exception exception)
        {
            return new RecipeUnknownException(RecipeId.Empty(),
exception);
        }
    }
}
```

Додаток В

Реалізація REST API контролера для обробки HTTP-запитів сутності «Рецепт»

```
using System.Security.Claims;
using Api.Dtos;
using Api.Modules.Errors;
using Application.Common.Interfaces.Queries;
using Application.Recipes.Commands;
using Domain.Recipes;
using MediatR;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

namespace Api.Controllers;

[Route("recipes")]
[ApiController]
public class RecipeController(ISender sender, IRecipeQueries
recipeQueries) : ControllerBase
{
   [HttpGet]
   public async Task<ActionResult<IReadOnlyList<RecipeDto>>> GetAll(
       [FromQuery] Guid? categoryId,
       [FromQuery] string? search,
       [FromQuery] string? sortBy,
       CancellationToken cancellationToken)
   {
       var entities = await recipeQueries.GetAll(categoryId, search,
sortBy, cancellationToken);

       return entities.Select(RecipeDto.FromDomainModel).ToList();
   }

   [HttpGet("{recipeId:guid}")]
   public async Task<ActionResult<RecipeDto>> Get([FromRoute] Guid
recipeId, CancellationToken cancellationToken)
```

Продовження додатку В

```
    {
        var entity = await recipeQueries.GetById(new
RecipeId(recipeId), cancellationToken);

        return entity.Match<ActionResult<RecipeDto>>(
            r => RecipeDto.FromDomainModel(r),
            () => NotFound());
    }

    [HttpPost]
    [Consumes("multipart/form-data")]
    public async Task<ActionResult<RecipeDto>> Create(
        [FromForm] CreateRecipeCommand command,
        CancellationToken cancellationToken)
    {
        var result = await sender.Send(command, cancellationToken);

        return result.Match<ActionResult<RecipeDto>>(
            r => RecipeDto.FromDomainModel(r),
            e => e.ToObjectResult());
    }

    [HttpPut]
    [Consumes("multipart/form-data")]
    public async Task<ActionResult<RecipeDto>> Update(
        [FromForm] UpdateRecipeCommand command,
        CancellationToken cancellationToken)
    {
        var result = await sender.Send(command, cancellationToken);

        return result.Match<ActionResult<RecipeDto>>(
            r => RecipeDto.FromDomainModel(r),
            e => e.ToObjectResult());
    }
```

Продовження додатку В

```
    [Authorize]
    [HttpDelete("{recipeId:guid}")]
    public async Task<ActionResult<RecipeDto>> Delete([FromRoute] Guid
recipeId, CancellationToken cancellationToken)
    {
        var userIdClaim =
User.FindFirstValue(ClaimTypes.NameIdentifier);

        if (string.IsNullOrEmpty(userIdClaim))
        {
            return Unauthorized("User ID claim not found in token");
        }

        var input = new DeleteRecipeCommand
        {
            RecipeId = recipeId,
            CurrentUserId = Guid.Parse(userIdClaim),
            IsAdmin = User.IsInRole("Admin")
        };

        var result = await sender.Send(input, cancellationToken);

        return result.Match<ActionResult<RecipeDto>>(
            r => RecipeDto.FromDomainModel(r),
            e => e.ToObjectResult());
    }


    [HttpGet("user/{userId:guid}")]
    public async Task<ActionResult<IReadOnlyList<RecipeDto>>>
GetByUser(
        [FromRoute] Guid userId,
        CancellationToken cancellationToken)
    {
```

Продовження додатку В

```
 var entities = await recipeQueries.GetByUserId(userId,
cancellationToken);
        return Ok(entities.Select(RecipeDto.FromDomainModel).ToList());
    }

    [HttpGet("daily")]
    public async Task<ActionResult<RecipeDto>>
GetDaily(CancellationToken cancellationToken)
    {
        var result = await
recipeQueries.GetDailyRecipe(cancellationToken);

        return result.Match<ActionResult<RecipeDto>>(
            r => RecipeDto.FromDomainModel(r),
            () => NotFound("Сьогодні страва дня не знайдена"));
    }
}
```

Додаток Г

Програмна реалізація контексту бази даних Entity Framework Core (ApplicationDbContext.cs)

```
using System.Reflection;
using Domain.Users;
using Domain.Categories;
using Domain.Cuisines;
using Domain.Recipes;
using Domain.Ingredients;
using Domain.Ratings;
using Domain.RecipeIngredients;
using Microsoft.EntityFrameworkCore;

namespace Infrastructure.Persistence;

public class
ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) :
DbContext(options)
{
   public DbSet<User> Users { get; set; }
   public DbSet<Category> Categories { get; set; }
   public DbSet<Recipe> Recipes { get; set; }
   public DbSet<Ingredient> Ingredients { get; set; }
   public DbSet<Rating> Ratings { get; set; }
   public DbSet<RecipeIngredient> RecipeIngredients { get; set; }
   public DbSet<Cuisine> Cuisines { get; set; }

   protected override void OnModelCreating(ModelBuilder builder)
   {

builder.ApplyConfigurationsFromAssembly(Assembly.GetExecutingAssembly(
));
       base.OnModelCreating(builder);
   }
}
```