

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:

«Сервіс формування меню та рецептів страв»

Виконав студент групи ІПЗ-21-4
ДЕНИСЮК Сергій Михайлович

Керівник роботи:
ВЛАСЕНКО Олег Васильович

Рецензент:
ТОЛСТОЙ Ігор Анатолійович

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

«26» лютого 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ДЕНИСЮК Сергій Михайлович**

Керівник роботи: **ВЛАСЕНКО Олег Васильович**

Тема роботи: **«Сервіс формування меню та рецептів страв»,**

затверджена Наказом закладу вищої освіти від **«14» лютого 2025 р., №61/с**

Вихідні дані для роботи: **вебзастосунок для перегляду кулінарних рецептів з динамічним розрахунком вартості страв.**

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

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Власенко О.В.	26.02.2025р.	26.02.2025р.
2	Власенко О.В.	26.02.2025р.	26.02.2025р.
3	Власенко О.В.	26.02.2025р.	26.02.2025р.

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з вебзастосунку для перегляду кулінарних рецептів з динамічним розрахунком вартості страв та пояснювальної записки. Пояснювальна записка до випускної роботи містить 58 сторінок, 35 ілюстрацій та 22 таблиці.

Метою роботи є розробка сучасного вебзастосунку, який дозволяє користувачам переглядати кулінарні рецепти та отримувати актуальну вартість кожної страви на основі щоденного моніторингу цін інгредієнтів.

Архітектура поділена на дві основні частини. Перша — скрапер, який щоденно отримує актуальні дані про продукти, формує базу інгредієнтів та зберігає інформацію. Друга — вебзастосунок, що надає клієнтам зручний інтерфейс для перегляду рецептів та динамічного розрахунку вартості приготування страв.

Додаток орієнтований переважно на мобільні пристрої і забезпечує доступність і комфорт користування.

КЛЮЧОВІ СЛОВА: ВЕБЗАСТОСУНОК, РЕЦЕПТ, СКРАПІНГ, СЕРВІС, СТРАВА

					ІПЗ.КР.Б-121-25-ІПЗ				
Змн.	Арк.	№ докум.	Підпис	Дата					
Розроб.		Денисюк С.М.			Сервіс формування меню та рецептів страв	Літ.	Арк.	Аркушів	
Керівник		Власенко О.В.					3		
Н. контр.		Тичина О.П.							
Зав. каф.		Вакалюк В.А.				Житомирська політехніка, група ІПЗ-21-4			

ABSTRACT

The bachelor's qualification work consists of a web application for viewing culinary recipes with dynamic cost calculation of dishes, along with an explanatory note. The explanatory note of the final project contains 58 pages, 35 illustrations, and 22 tables.

The purpose of the project is to develop a modern web application that allows users to browse culinary recipes and obtain up-to-date pricing for each dish based on daily monitoring of ingredient prices.

The architecture is divided into two main parts. The first is a scraper that collects current product data daily, generates an ingredient database, and stores the data. The second is a web application which provides users with a convenient interface to view recipes and dynamically calculate the cost of preparing dishes.

The application is primarily optimized for mobile devices and ensures accessibility and user comfort.

KEYWORDS: DISH, RECIPE, SCRAPING, SERVICE, WEB APPLICATION

					ИП3.КР.Б-121-25-ИЗ	Арк.
						4

ЗМІСТ

ВСТУП.....	8
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РОЗРАХУНКУ ВАРТОСТІ СТРАВ ТА ПЛАНУВАННЯ СПОЖИВЧОГО БЮДЖЕТУ	10
1.1 Постановка задачі	10
1.2 Аналіз аналогів програмного продукту	12
1.3 Вибір архітектури програмного комплексу	17
1.4 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення	20
Висновки до першого розділу	22
РОЗДІЛ 2. ПРОЄКТУВАННЯ СЕРВІСУ ФОРМУВАННЯ МЕНЮ ТА РЕЦЕПТІВ	24
2.1. Визначення варіантів використання та об'єктно-орієнтованої структури системи	24
2.2. Розробка бази даних системи	31
2.3. Проєктування та реалізація алгоритмів роботи системи	36
2.4. Реалізація функціоналу системи	42
Висновки до другого розділу	47
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ СЕРВІСУ ФОРМУВАННЯ МЕНЮ ТА РЕЦЕПТІВ	48
3.1. Порядок встановлення та налаштування параметрів платформи	48
3.2. Структура інтерфейсу	50
3.3. Тестування платформи	59
Висновки до третього розділу	63
ВИСНОВКИ	64

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....66

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API — прикладний програмний інтерфейс

БД — база даних

ETL — Extract-Transform-Load (витягнення, перетворення, вантаження)

HTTP — Hypertext Transfer Protocol (протокол передачі гіпертексту)

IAM — Identity and Access Management (керування ідентифікацією та доступом)

IQR — Interquartile Range (міжквартильний розмах)

JSON — JavaScript Object Notation (нотація об'єктів JavaScript)

LLM — Large Language Model (велика мовна модель)

SDK — Software Development Kit (комплект для розробки програмного забезпечення)

SEO — Search Engine Optimization (оптимізація для пошукових систем)

SSR — server-side rendering (серверний рендеринг)

TLS — Transport Layer Security (безпека транспортного рівня)

TS — TypeScript

URL — Uniform Resource Locator (уніфікований покажчик ресурсу)

UX — User Experience (користувацький досвід)

ВСТУП

Актуальність теми полягає у зростаючій потребі користувачів в отриманні оперативної та достовірної інформації щодо вартості приготування страв з урахуванням реальних цін на продукти. У сучасних умовах швидкого зростання цін на продукти харчування та змін у ринку споживання, особливо важливо мати інструмент, який допомагає планувати покупки ефективно та економно. Розробка вебзастосунку, який поєднує рецепти та автоматичний розрахунок вартості страв на основі актуальних даних, сприяє цифровізації побутових процесів та впровадженню сучасних вебтехнологій у повсякденне життя.

Метою випускної кваліфікаційної роботи бакалавра є розробка вебзастосунку для перегляду кулінарних рецептів з автоматичним розрахунком актуальної вартості страв на основі щоденних цін інгредієнтів.

Для реалізації поставленої мети потрібно виконати наступні завдання

1. Провести аналіз наявних рішень у сфері кулінарних платформ та інструментів розрахунку вартості страв.
2. Обґрунтувати вибір технологій для реалізації програмного забезпечення.
3. Реалізувати скрапер для продуктів разом та їх цін з відкритих джерел і формування БД інгредієнтів.
4. Розробити повнофункціональний вебзастосунок з адаптацією під мобільні пристрої.
5. Реалізувати динамічне оновлення вартості страв на основі зібраних даних.

Об'єктом дослідження є процес інтеграції реальних цінових даних у вебзастосунок для приготування страв.

Предметом дослідження є функціональні можливості вебплатформи, що дозволяє динамічно розраховувати вартість страв на основі актуальних даних про ціни інгредієнтів.

Дослідження також охоплює питання оптимізації взаємодії між користувачем і системою, ефективності оновлення даних у БД, а також зручності представлення цієї інформації на клієнтській стороні застосунку.

Методи аналізу та інформаційна база дослідження. У процесі роботи було використано методи аналізу функціональних вимог, системного підходу до проєктування програмного забезпечення, порівняльного аналізу існуючих рішень та метод моделювання програмної архітектури. Інформаційною базою дослідження стали відкриті джерела з цінами на продукти харчування (API, сайти супермаркетів), технічна документація до обраних технологій, а також наукові публікації у сфері розробки вебзастосунків

Практичне значення одержаних результатів полягає у створенні функціонального вебзастосунку, який може бути використаний як у повсякденному житті для планування витрат на продукти, так і як основа для подальших розробок у сфері автоматизації побутових рішень. Застосунок може стати корисним інструментом для широкого кола користувачів, включаючи домогосподарства, студентів, дієтологів та кулінарні блоги. Крім того, структура проєкту дозволяє масштабувати його функціональність — наприклад, додавати підтримку різних валют, персоналізовані раціони або аналітику витрат.

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РОЗРАХУНКУ ВАРТОСТІ СТРАВ ТА ПЛАНУВАННЯ СПОЖИВЧОГО БЮДЖЕТУ

1.1 Постановка задачі

Метою кваліфікаційної роботи бакалавра є створення інтерактивного вебзастосунку для перегляду та створення кулінарних рецептів із можливістю автоматичного розрахунку актуальної вартості страв на основі реальних цін інгредієнтів. Основна ідея полягає у наданні користувачеві зручного інструменту для ведення кулінарних записів, планування витрат та ознайомлення з рецептурами інших користувачів, із врахуванням актуальної економічної ситуації.

Сучасні кулінарні сервіси зазвичай обмежуються функцією перегляду рецептів або мають закриту структуру БД, де лише адміністрація додає нові матеріали. Це знижує актуальність вмісту та не дозволяє користувачам вносити власний внесок. У той же час, стрімке зростання цін на продукти харчування та нестабільність ринку створюють потребу в інтеграції економічного розрахунку у щоденні побутові процеси. Запропонований застосунок дозволяє вирішити обидві проблеми, поєднуючи функції соціальної кулінарної платформи та автоматизованого інструменту для оцінки вартості страв.

Програмний комплекс повинен включати такі функціональні модулі:

1. Модуль користувача – забезпечує реєстрацію, авторизацію, профіль користувача та систему ролей. Авторизовані користувачі можуть створювати нові рецепти, редагувати або видаляти власні, залишати відгуки з оцінками до чужих рецептів.
2. Модуль керування рецептами – дозволяє створювати рецепти з описом, фотографіями, кроками приготування та списком інгредієнтів у зазначених кількостях.

3. Модуль розрахунку вартості – автоматично вираховує вартість кожної страви на основі актуальних цін інгредієнтів, що зберігаються у базі даних.
4. Модуль збору цін (скрапер) – щодня автоматично отримує ціни з відкритих джерел, оновлює значення в БД та забезпечує актуальність фінансової інформації.
5. Модуль відгуків і рейтингу – дозволяє користувачам оцінювати рецепти за п'ятибальною шкалою та залишати текстові коментарі.
6. Адаптивний інтерфейс – реалізований з урахуванням зручного використання на мобільних пристроях.

Очікуваний результат – масштабована, зручна у використанні вебплатформа, яка об'єднує кулінарну спільноту, дозволяє користувачам взаємодіяти між собою, ділитися знаннями, а також планувати витрати на приготування страв на основі реальних цін:

Основні задачі, які необхідно вирішити у процесі реалізації проєкту:

1. Провести дослідження предметної області, сформулювати вимоги до функціональності системи.
2. Проаналізувати існуючі рішення (аналогічні платформи) та визначити їх недоліки, які буде усунено в даному проєкті.
3. Спроектувати архітектуру клієнтської та серверної частини застосунку, визначити формат зберігання даних.
4. Реалізувати систему реєстрації, автентифікації, авторизації та управління правами доступу.
5. Розробити інтерфейс для створення, редагування, перегляду та пошуку рецептів.
6. Реалізувати модуль скрапінгу цін та інтегрувати його з рецептурною базою.
7. Реалізувати динамічний розрахунок вартості страв.
8. Впровадити систему рейтингу та коментування рецептів.

9. Провести тестування всіх компонентів системи та оптимізувати продуктивність.

У результаті реалізації вищенаведених задач буде створено вебзастосунок нового покоління, що поєднує зручність користування, відкритість до користувацького контенту та фінансову точність, яка забезпечується завдяки щоденному оновленню цін. Це дозволить не лише планувати харчування, а й оптимізувати витрати на продукти, що є особливо актуальним у сьогоденних соціально-економічних умовах.

1.2 Аналіз аналогів програмного продукту

Для формування функціональних вимог до майбутнього вебзастосунку було проаналізовано існуючі кулінарні сервіси та суміжні платформи. Розглядалися системи, що реалізують інструменти створення, пошуку та представлення рецептів, а також сервіси, що працюють із ціновими даними продуктів.

Cookpad — міжнародна платформа для обміну кулінарними рецептами. Українська версія представлена повністю локалізованим інтерфейсом.

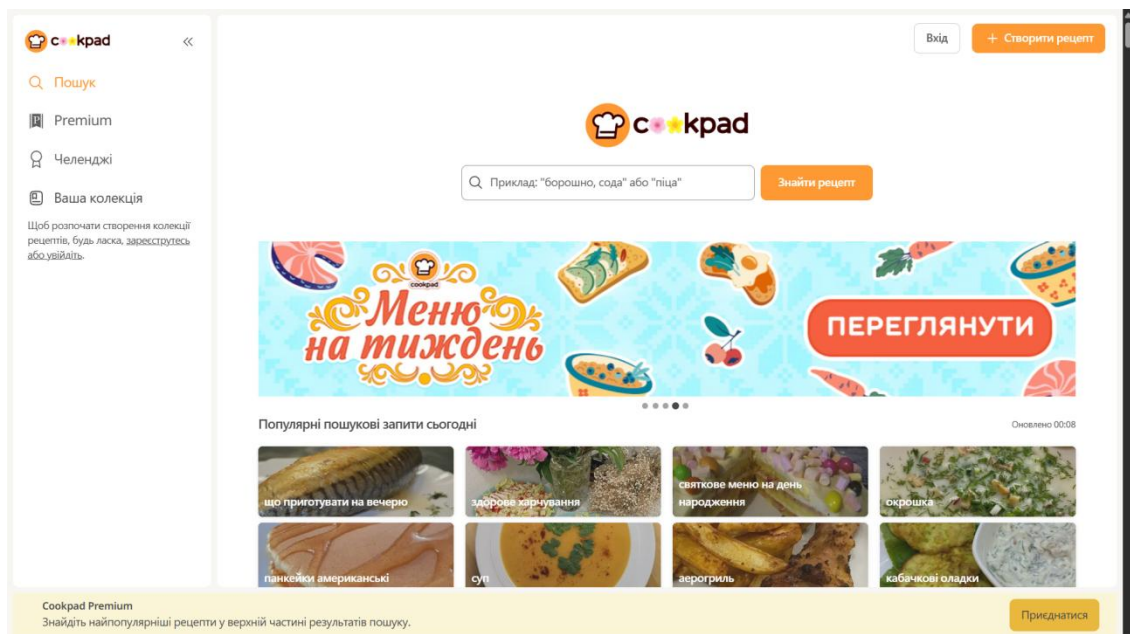


Рис.1.1. Кулінарна платформа Cookpad[1].

Платформа орієнтована на активну взаємодію користувачів, які можуть створювати рецепти, зберігати їх, коментувати й додавати реакції у вигляді

емоцій. Під кожним рецептом передбачена кількість порцій. Доступний пошук за назвою та інгредієнтами, а також система рекомендацій на основі популярних запитів і категорій. Додатково реалізовано систему підписки на авторів рецептів і можливість публікувати фото страви після її приготування. Інтерфейс дозволяє скажитися на рецепт, друкувати його або ділитися з іншими користувачами. Також передбачена функція перегляду вже приготованих рецептів. У профілі користувача доступна статистика активності. Преміум-версія платформи надає розширені фільтри пошуку, функцію формування тижневого плану харчування, створення власних папок для рецептів та пріоритетне відображення популярних рецептів.

Allrecipes — одна з наймасштабніших кулінарних платформ у глобальному інтернет-просторі.

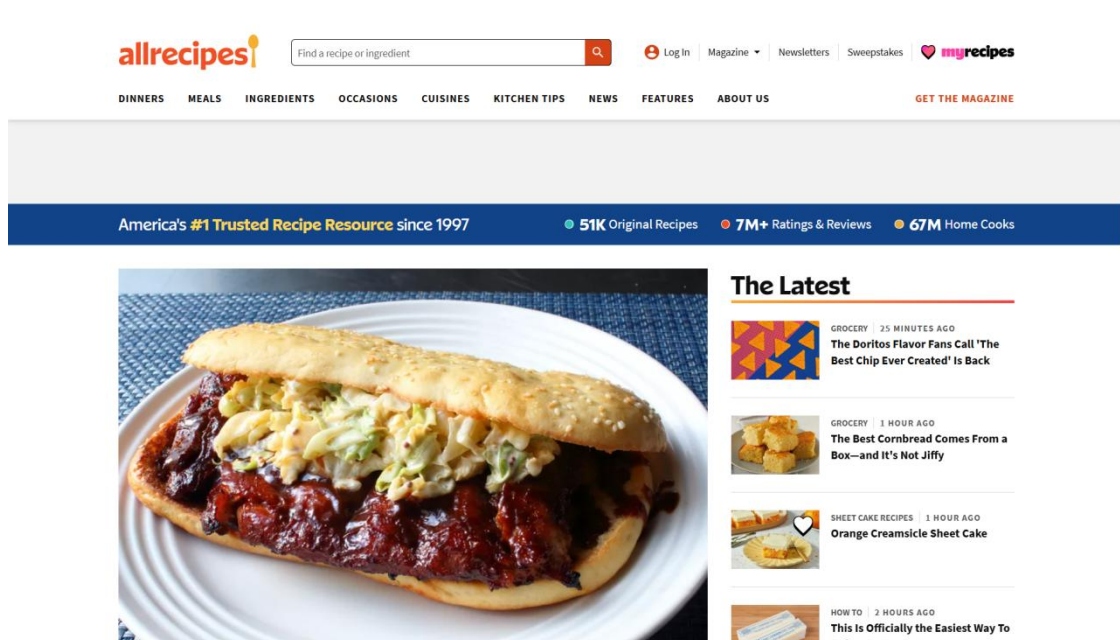


Рис.1.2. Кулінарна платформа AllRecipes[2].

Сервіс поєднує соціальний та редакційний контент. Користувачі мають змогу створювати рецепти з додаванням нотаток, вказанням метаданих (час підготовки, приготування, загальний час, кількість порцій), прикріпленням відеоінструкцій та зазначенням калорійності страви. Передбачено масштабування інгредієнтів залежно від кількості порцій. Кожен рецепт можна зберегти, поділитися ним, роздрукувати та залишити на нього відгук з

оцінкою (від 1 до 5) і фото. Реалізовано режим перегляду рецепта без вимкнення екрана (Cook Mode). Пошукова система підтримує фільтрацію за назвою, інгредієнтами, категоріями та типом кухні. У рамках соціальної взаємодії користувачі можуть ставити лайки на відгуки, переглядати активність інших користувачів і задавати запитання спільноті. Додатково доступні тематичні статті, рекомендації та розділ з найчастішими запитаннями.

ЯБпоїла — українська кулінарна платформа з акцентом на повсякденну кухню.

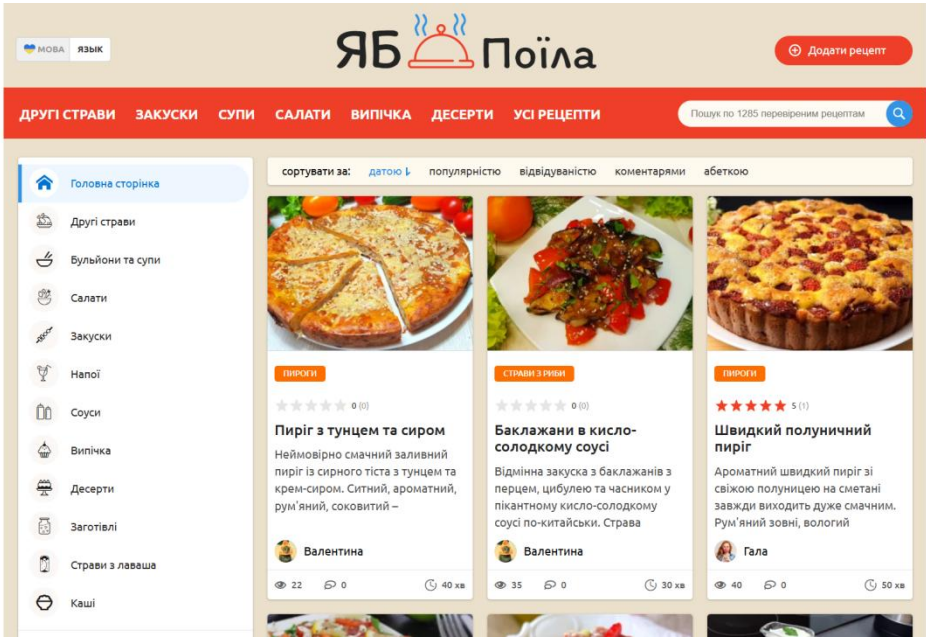


Рис.1.3. Кулінарна платформа ЯБпоїла[3].

На сайті реалізовано створення рецептів із покроковим описом, можливістю додати опитування, визначити час підготовки й приготування, вказати кількість порцій. Автоматично визначається поживна цінність страви. Пошук здійснюється за назвою рецепта. Платформа підтримує функцію масштабування інгредієнтів, аналіз поживної цінності рецепта або окремих інгредієнтів, через вбудований аналізатор. Користувачі можуть оцінювати рецепти лайком або дизлайком, залишати коментарі, зберігати та ділитися рецептами. У профілі передбачена можливість надсилати приватні

повідомлення. У розділі “непрочитані рецепти” зібрано нові або ще не переглянуті користувачем публікації.

«Де дешевше» — український мобільний застосунок, що порівнює ціни на продукти харчування з великих супермаркетів.

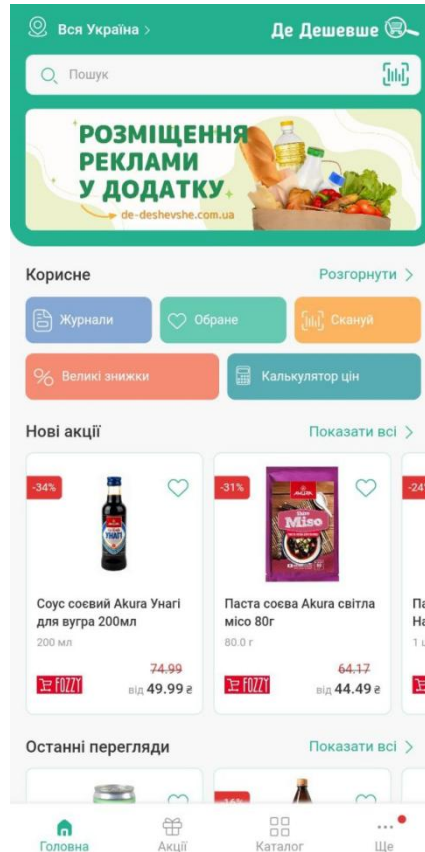


Рис.1.4. Мобільний додаток «Де дешевше»[4].

Додаток забезпечує пошук продуктів харчування за назвою або штрихкодом. Реалізовано відображення товарів за категоріями та фільтрацію результатів за магазинами й регіонами. Для кожного товару вказано дату оновлення ціни, а також доступна інформація про харчову цінність. Користувачі можуть зберігати обрані товари, ділитися ними та переглядати тематичні статті. У преміум-функціоналі передбачено низку калькуляторів: порівняння цін між товарами, розрахунок ціни за одиницю ваги чи об’єму, а також перевірка терміну придатності.

На основі проведеного дослідження було сформовано таблицю, що ілюструє наявність функціональних можливостей у розглянутих аналогах.

Таблиця 1.1

Таблиця порівняння характеристик аналогових програмних продуктів

Характеристика	Cookpad	Allrecipes	ЯБпоїла	«Де дешевше»
Створення рецептів	+	+	+	-
Пошук за назвою	+	+	+	+
Пошук за інгредієнтом	+	+	-	+
Система оцінювання (рейтинги / лайки / дизлайки)	+	+	+	-
Пошук за інгредієнтами	+	+	+	+
Збереження до обраного	+	+	+	+
Поділитися рецептом	+	+	+	+
Масштабування інгредієнтів	-	+	+	-
Відеоінструкції	-	+	-	-
Аналіз калорійності/поживних речовин	-	+	+	+
Взаємодія зі спільнотою (питання, повідомлення, підписки)	+	+	+	-
Категоризація	+	+	+	+
Порівняння цін, історія оновлення	-	-	-	+
Роздрук рецепта	+	+	-	-

Проведений аналіз дозволив зібрати повну картину функціональних можливостей, реалізованих у популярних кулінарних платформах та сервісах моніторингу цін. Виявлено, що більшість систем реалізують базові функції

на кшталт створення рецептів, пошуку за інгредієнтами, збереження та коментування, тоді як більш специфічні можливості — такі як масштабування інгредієнтів, розрахунок поживної цінності, або інтеграція з актуальними цінами — зустрічаються лише частково і в окремих рішеннях.

Зокрема, жодна з аналізованих платформ не поєднує в собі весь спектр функцій, що може бути важливим для користувача в умовах сучасного динамічного ринку. Це стосується, зокрема, автоматичного розрахунку вартості рецепта на основі цін з супермаркетів, а також поєднання кулінарного та економічного функціоналу в єдиному інтерфейсі. Більшість платформ також орієнтуються або на соціальну взаємодію, або на інформаційний контент, уникаючи комплексних рішень.

Таким чином, результати аналізу підтвердили доцільність створення нового вебзастосунку, який би об'єднав основні очікування користувачів — простоту додавання рецептів, їх збереження, комунікацію — з економічно орієнтованими функціями, такими як динамічний розрахунок вартості, актуальність цін і підбір найбільш вигідних варіантів приготування. Це дозволить покрити функціональні прогалини існуючих аналогів та запропонувати цілісне рішення.

1.3 Вибір архітектури програмного комплексу

Розробка сучасного вебзастосунку передбачає чітке розділення відповідальностей між його складовими частинами. Від вибору архітектури залежить масштабованість, стабільність, підтримуваність та зрозумілість системи як для розробника, так і для кінцевого користувача[5]. З урахуванням цілей і функціоналу, поставлених у межах кваліфікаційної роботи, було проведено порівняння поширених архітектурних підходів та обрано найбільш доцільний.

Серед основних архітектур, які розглядалися на етапі проєктування, варто відзначити монолітну архітектуру, класичну клієнт-серверну модель та мікросервісну. Монолітна архітектура характеризується простотою реалізації,

однак має низку обмежень, що пов'язані з масштабуванням і незалежною підтримкою окремих компонентів[6]. Для простих застосунків така модель може бути ефективною, але у випадках, коли необхідно обробляти асинхронні процеси, взаємодіяти з декількома джерелами даних або мати окрему логіку збору даних, вона стає неефективною. Мікросервісна архітектура передбачає побудову системи як набору незалежних сервісів, кожен з яких виконує окрему бізнес-функцію та може працювати автономно[7][8]. Такий підхід забезпечує гнучке масштабування, зручне оновлення окремих модулів і можливість використання різних технологій у межах однієї системи. Однак у випадку даного проєкту реалізація мікросервісної архітектури виявилася надлишковою. Компоненти системи (зокрема скрапер і вебінтерфейс) хоча й мають окремі функції, не вимагають складної взаємодії між собою або високої незалежності в розгортанні. Дані, які обробляє скрапер, надходять до бази у фоновому режимі й не потребують миттєвої синхронізації з користувацькою частиною. Через це витрати на впровадження мікросервісного підходу — зокрема, організацію міжсервісної комунікації, логування, моніторинг — були б невиправданими для проєкту такого масштабу[9]. Клієнт-серверна модель забезпечує чіткий розподіл між інтерфейсною частиною (клієнт) та логікою обробки даних (сервер), що добре підходить для більшості вебсистем[10]. Водночас вона не покриває задач із фоновією обробкою даних або періодичним отриманням інформації ззовні без взаємодії з користувачем.

З урахуванням вищезазначеного, було прийнято рішення реалізувати архітектуру, що поєднує клієнт-серверну модель з окремим модулем фоновієї обробки даних за принципом ETL[11][12]. Такий підхід дозволяє задовольнити як вимоги до інтерактивної взаємодії користувача із застосунком, так і потребу в автоматизованому зборі та обробці зовнішніх даних, не перевантажуючи основну систему. У межах обраної архітектури вебінтерфейс виконує функції класичного клієнта, який відповідає за рендеринг сторінок, обробку подій, відображення рецептів та динамічний

розрахунок вартості страв на основі даних, що зберігаються у БД. Серверна частина застосунку обробляє запити клієнта, здійснює валідацію введених даних, забезпечує доступ до БД і відповідає за автентифікацію, авторизацію та логіку керування рецептами. Ця частина системи працює в постійному режимі та обслуговує запити в реальному часі. Фоновий компонент архітектури — скрапер — реалізує ETL-конвеєр. Він періодично звертається до зовнішніх джерел (зокрема сайтів супермаркетів), отримує необроблену інформацію про ціни (Extract), перетворює її в уніфікований формат із нормалізованими назвами інгредієнтів та одиницями виміру (Transform), а потім зберігає підготовлені дані до спільної БД (Load). Цей процес є повністю автоматизованим і не потребує прямої взаємодії з користувачем. Важливо, що скрапер працює ізольовано та не залежить від стану чи доступності клієнтської частини системи, що сприяє стабільності всього комплексу. Обидва модулі — вебінтерфейс і скрапер — працюють у незалежних середовищах виконання, що забезпечується засобами віртуалізації. Це дозволяє легко налаштовувати, тестувати й розгортати кожен компонент окремо. Комунікація між ними відбувається опосередковано через спільну БД, без необхідності прямого обміну запитами. Такий підхід відповідає принципам слабкої зв'язності між компонентами, що підвищує надійність системи та полегшує її супровід[13] . Архітектурне рішення також створює умови для подальшого розширення системи. Наприклад, у майбутньому можна додати модулі з персоналізованими рекомендаціями, аналітикою витрат або підпискою на оновлення. Завдяки відокремленню функцій та застосуванню асинхронної обробки даних це розширення не вимагатиме кардинальної перебудови основної структури застосунку.

Таким чином, обрана архітектура — поєднання клієнт-серверного підходу з ETL-моделлю — забезпечує баланс між гнучкістю, масштабованістю та простотою впровадження. Вона дозволяє ефективно виконувати фонову обробку динамічних цінових даних і водночас надає

користувачам зручний доступ до актуальної інформації у звичному форматі взаємодії з вебінтерфейсом.

1.4 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Вибір інструментальних засобів розробки здійснювався з урахуванням функціональних потреб системи, вимог до масштабованості, швидкодії, зручності розгортання, а також актуальності технологій у сучасному веброзробницькому середовищі. Усі компоненти застосунку реалізовано із дотриманням принципу розділення відповідальностей, що відображається у використанні різних мов програмування, середовищ виконання та супутніх інструментів для кожної складової системи.

Клієнтську та серверну частину вебінтерфейсу реалізовано з використанням Next.js — сучасного фреймворку для розробки вебзастосунків на базі JavaScript та React. Його було обрано через підтримку SSR, що позитивно впливає на швидкість завантаження, покращує користувацький досвід та дозволяє ефективно працювати з повільними з'єднаннями. Окрім того, Next.js має гнучку систему маршрутизації, дозволяє легко інтегрувати API та добре масштабуються в умовах зростання функціональності. Важливим аргументом на користь цього фреймворку є також його оптимізація для пошукових систем (SEO): SSR забезпечує повноцінне індексування вмісту сторінок пошуковими ботами, що є критично важливим для проєктів, пов'язаних з контентом і просуванням[14].

Для типізації застосовано TS — мову програмування на основі JavaScript, яка забезпечує статичну типізацію, підвищену надійність коду та зручність при масштабуванні проєкту. Завдяки TS, зменшується кількість помилок на етапі виконання, а також покращується інтеграція з зовнішніми API[15].

Для реалізації модуля збору даних обрано мову програмування Python, яка є однією з найпоширеніших у сфері автоматизації, обробки текстів і

вебскрапінгу[16]. Python має велику кількість готових бібліотек для взаємодії з браузером (Selenium), обробки HTTP-запитів (requests) і роботи з регулярними виразами (re), що дозволяє ефективно реалізувати ETL-процеси. Також більшість сучасних LLM видають найбільш якісний, читабельний та підтримуваний код саме на мові Python[17], що спрощує процес автоматизації або генерації окремих частин логіки (наприклад, шаблонів для скраперів) і пришвидшує розробку.

Усі структурні компоненти застосунку взаємодіють з базою даних MongoDB, яка була обрана через її гнучку документно-орієнтовану модель. Це дозволяє зберігати інформацію у вигляді вкладених структур, які добре відповідають природній ієрархії кулінарних рецептів. MongoDB також підтримує швидкий пошук, індексацію, агрегацію даних та масштабування, що робить її зручною для динамічних проєктів зі змінною структурою даних[18].

Для управління середовищем виконання та забезпечення незалежності компонентів один від одного використовується система контейнеризації Docker. Контейнери дозволяють створювати ізольовані середовища з чітко визначеними залежностями, що робить процес розгортання прогнозованим, стабільним і однаковим на будь-якому пристрої або сервері. Це особливо актуально для застосунків, які складаються з кількох логічно відокремлених частин, як у цьому випадку — вебінтерфейс, скрапер і БД[19]. Конфігурація контейнерів здійснюється за допомогою файлу `docker-compose.yml`, що міститься в кореневій директорії проєкту. Це забезпечує одночасний запуск усіх необхідних сервісів командою `docker compose up`, що значно спрощує як локальну розробку, так і подальше розгортання в продакшн-середовищі згідно з принципами безперервної інтеграції[20].

Для розгортання проєкту в реальному середовищі було обрано хмарну інфраструктуру Google Cloud Platform. Цей вибір обумовлений наявністю зручних сервісів для запуску ізольованих контейнерів, зокрема Google Cloud Run, який дозволяє автоматично масштабувати вебзастосунок і скрапер

відповідно до навантаження без необхідності ручного керування інфраструктурою. Крім того, для зберігання зображень використовується Google Cloud Storage — об’єктне сховище, яке забезпечує високу доступність, ефективну інтеграцію з Node.js API та оптимізацію зберігання медіаданих. Використання хмарної платформи також дозволяє централізовано керувати обліковими даними, правами доступу та секретами через систему IAM і налаштування середовищ виконання. Такий підхід забезпечує надійність, стабільність і масштабованість розгортання всіх компонентів системи в продакшн-умовах[21].

З огляду на невисокі вимоги до ресурсів скрапера та оптимізованість клієнтської частини, загальні апаратні вимоги до системи залишаються помірними. Для локального середовища розробки достатньо машини з такими характеристиками: Процесор: 4-ядерний (наприклад, Intel Core i5 або аналог); Оперативна пам’ять: від 8 ГБ; Накопичувач: SSD-диск об’ємом не менше 50 ГБ; Операційна система: будь-яка, що підтримує Docker (Linux, macOS, Windows). На сервері для розгортання у продуктивному середовищі достатньо мінімального віртуального сервера з 2–4 ядрами процесора та 4–8 ГБ оперативної пам’яті, з урахуванням потенційного масштабування залежно від навантаження.

Таким чином, обрані інструменти забезпечують ефективну реалізацію завдань, поставлених у межах кваліфікаційної роботи, і дозволяють створити гнучку, масштабовану та зручну для підтримки систему, що відповідає сучасним стандартам веброзробки.

Висновки до першого розділу

У першому розділі було проведено огляд предметної області, визначено актуальність проблеми та сформульовано основну мету розробки. Проаналізовано особливості існуючих кулінарних вебзастосунків, зокрема їх функціональні можливості, принципи побудови контенту, підходи до організації пошуку, категоризації рецептів, а також наявність додаткових

сервісів, таких як підрахунок калорій, відгуки, збереження до обраного та формування списку покупок. Проведений аналіз дозволив виокремити базовий функціонал, який має бути обов’язково реалізований у межах проєкту, а також визначити можливі шляхи розширення функціоналу через додавання унікальних можливостей, серед яких — автоматичний розрахунок вартості страви на основі актуальних цін з онлайн-магазинів.

Особливу увагу було приділено вивченню структур даних і взаємодії між користувачами різних ролей — як із контентом, так і між собою. На основі цього було сформульовано вимоги до системи: як функціональні (реєстрація, створення рецептів, відгуки, оцінки, скарги тощо), так і нефункціональні (швидкодія, адаптивність, безпека, безперервна доступність). Ці вимоги лягли в основу визначення варіантів використання системи, архітектурного поділу компонентів та логіки доступу до даних.

Крім того, у межах розділу було обґрунтовано вибір основних інструментальних засобів. Для реалізації вебзастосунку використано фреймворк Next.js у поєднанні з TS, що забезпечує підтримку SSR та типобезпечність. Для зберігання вкладених даних застосовано MongoDB, яка зручно працює з динамічними структурами. Компоненти платформи, включаючи вебінтерфейс і окремий модуль збору цін (скрапер), ізольовано за допомогою Docker. Розгортання відбувається через Google Cloud, який дозволяє стабільно запускати обидві частини проєкту. Скрапер, реалізований на Python, періодично отримує оновлення цін із сайтів супермаркетів.

Таким чином, результати цього розділу стали основою для технічного проєктування системи та безпосередньої реалізації програмного забезпечення, що буде розглянуто у наступних розділах.

РОЗДІЛ 2. ПРОЄКТУВАННЯ СЕРВІСУ ФОРМУВАННЯ МЕНЮ ТА РЕЦЕПТІВ

2.1. Визначення варіантів використання та об'єктно-орієнтованої структури системи

Вебзастосунок для перегляду кулінарних рецептів з автоматичним розрахунком вартості страв має підтримувати різні сценарії використання в залежності від ролі користувача. Визначено чотири основні типи акторів системи: гість, авторизований користувач, модератор та адміністратор.

Гість є неавторизованим користувачем, який має доступ виключно до функції перегляду опублікованих рецептів. Також йому доступна реєстрація та вхід у систему. Авторизований користувач має розширений функціонал: створення, редагування та видалення власних рецептів, додавання відгуків з оцінками, а також подання скарг на неприйнятний вміст у рецептах чи відгуках інших користувачів. Модератор має всі можливості авторизованого користувача, але також може керувати чужим контентом: редагувати або видаляти рецепти, відгуки, переглядати список скарг, а також позначати інгредієнти як некоректно сформовані. Найвищим рівнем доступу володіє адміністратор, який окрім можливостей модератора, має доступ до повного керування тегами, що використовуються в системі, та має контроль над усім вмістом вебзастосунку. Загальна логіка взаємодії користувачів із системою представлена на діаграмі варіантів використання (рис. 2.1).

Таблиця 2.4.

Опис прецеденту «Скарга на рецепт / відгук»

Назва прецеденту	Створення рецепту
Основні актори	Авторизований користувач, модератор, адміністратор
Короткий опис	Авторизовані користувачі можуть залишати скарги на рецепт або відгук обираючи причину зі списку

Таблиця 2.5.

Опис прецеденту «Перегляд скарг»

Назва прецеденту	Перегляд скарг
Основні актори	Модератор, адміністратор
Короткий опис	Модератор може переглянути список скарг та відреагувати на них виправленням або видаленням контенту

Таблиця 2.6.

Опис прецеденту «Позначення невірної інгредієнта або прив'язаного продукту»

Назва прецеденту	Позначення невірної інгредієнта або прив'язаного продукту
Основні актори	Модератор, адміністратор
Короткий опис	Модератор може позначити некоректні інгредієнти, після чого вони не відображатимуться під час створення рецепту, або продукти — у такому разі їх буде виключено з обчислення вартості.

Таблиця 2.7.

Опис прецеденту «Керування тегами»

Назва прецеденту	Керування тегами
Основні актори	Адміністратор
Короткий опис	Адміністратор може створювати, редагувати та видаляти теги, які використовуються для категоризації рецептів

Основною метою розробки даного вебзастосунку є створення платформи, що повинна забезпечувати користувачам можливість зручно переглядати кулінарні рецепти та отримувати автоматичний розрахунок вартості приготування страви на основі актуальних цін інгредієнтів. Система

має підтримувати кілька ролей, кожна з яких передбачає певний обсяг функціональних можливостей. Гості, тобто неавторизовані користувачі, повинні мати доступ до рецептурного контенту в режимі перегляду та можливість зареєструватися для розширення функціональності. Авторизовані користувачі мають отримати доступ до створення, редагування й видалення власних рецептів, написання відгуків з оцінками, а також до подання скарг на контент.

Модератори, окрім функцій звичайних користувачів, повинні мати змогу переглядати усі скарги, втручатися в рецепти й відгуки незалежно від авторства та здійснювати маркування інгредієнтів як некоректних. Адміністраторам системи слід надати повний доступ до функціоналу модераторів, а також можливість керування структурою тегів та здійснення інших дій адміністративного рівня.

У системі повинні бути реалізовані механізми реєстрації та авторизації користувачів, функції додавання, редагування, перегляду та видалення рецептів, можливість залишати відгуки з оцінками, а також автоматичне визначення вартості рецепту на основі оновлюваних даних про ціни. Передбачено необхідність існування механізму подання скарг, які повинні бути доступними для перегляду та обробки модераторами. Для адміністраторів має бути створено окремий інтерфейс керування тегами, що дозволить формувати категорії рецептів.

Інтерфейс користувача повинен бути інтуїтивно зрозумілим, з адаптивним дизайном, що дозволяє комфортно користуватися платформою на мобільних пристроях. Швидкодія системи має забезпечувати виконання основних операцій, таких як відкриття рецепту або публікація відгуку, протягом не більше 2–3 секунд. Платформа повинна підтримувати одночасну роботу багатьох користувачів без втрати продуктивності. Усі введені дані слід перевіряти на коректність, а доступ до ресурсів — обмежувати відповідно до рівня авторизації.

Клієнтська частина платформи повинна працювати в сучасному веббраузері, без потреби встановлення додаткового програмного забезпечення. Сумісність має охоплювати основні операційні системи, зокрема Windows, macOS, Linux, Android та iOS. Необхідно забезпечити роботу на пристроях із базовими апаратними характеристиками, таких як 2 ГБ оперативної пам'яті, процесор з частотою від 1.4 GHz та стабільне інтернет-з'єднання.

Серверна частина системи повинна функціонувати на базі Linux-середовища, з використанням контейнеризації Docker. У якості платформи виконання повинні використовуватись сучасні версії Node.js для вебсерверної логіки та Python для скрапінгу цін. Зберігання даних має здійснюватися у базі MongoDB, а мінімальні технічні характеристики сервера повинні включати від 2 до 4 ядер процесора, 4 ГБ оперативної пам'яті та не менше 50 ГБ вільного місця на SSD. Окремо слід забезпечити стабільне підключення до мережі для регулярного оновлення інформації про ціни.

Таким чином, структура системи повинна передбачати логічну розподіленість функцій відповідно до ролей користувачів, швидкодію, безперервну доступність та безпечну обробку даних. Усі зазначені вимоги стали основою для моделювання об'єктної структури, яка реалізована за допомогою інтерфейсів TypeScript та відповідних схем у MongoDB. Такий підхід забезпечує гнучкість при роботі з вкладеними об'єктами, дозволяє зберігати сутності з різним ступенем деталізації та легко масштабувати архітектуру.

На логічному рівні в системі виокремлено кілька ключових об'єктів. Центральне місце займає модель Recipe, яка описує кулінарний рецепт і містить заголовок, опис, складність, кількість порцій, головне фото, список тегів, набір інгредієнтів із зазначенням кількості й одиниці виміру, а також послідовні кроки приготування з ілюстраціями. Кожен рецепт пов'язаний із конкретним користувачем, що представлений сутністю User. Модель

користувача зберігає облікову інформацію, роль у системі, а також використовується для позначення авторства рецептів, відгуків і скарг.

Інгредієнти рецепта базуються на сутності Ingredient — абстрактному описі харчового компонента, що об'єднує групу реальних товарів із подібною сутністю. Кожен інгредієнт пов'язаний із набором об'єктів Product, які мають актуальну ціну та форму упаковки. Саме ці ціни використовуються для динамічного обчислення вартості страв. Для уніфікації обчислень усі ціни нормалізуються до базових одиниць виміру (грам, мілілітр, штука).

Користувачі можуть залишати відгуки (Review) на рецепти, які зберігають як текстову оцінку, так і числовий рейтинг. Це дає змогу формувати агреговану оцінку якості кожної страви. У випадках порушення правил чи зловживань у системі існує можливість створення скарг (Complaint), які можуть стосуватись як рецептів, так і відгуків. Кожна скарга містить тип цільової сутності, опис проблеми та статус розгляду, що дозволяє модераторам централізовано обробляти звернення.

Для зручної класифікації та подальшої фільтрації рецепти пов'язуються з об'єктами Tag, які виконують роль категорій — за типом страви, тематикою або способом приготування. Таким чином, кожна сутність у структурі має чітко визначену роль та мінімально необхідні зв'язки з іншими компонентами системи.

На Рис.2.2 зображено узагальнену модель сутностей та їх взаємозв'язків у системі.

.

принципом нормалізації, але з урахуванням специфіки MongoDB, що дозволяє уникати надмірного дублювання даних, зберігаючи при цьому гнучкість і швидкодію запитів.

2.2. Розробка бази даних системи

У структурі даних вебзастосунку використано колекції MongoDB, що відповідають основним сутностям системи. Зв'язки між об'єктами реалізовано через зберігання їхніх ObjectId, що дозволяє підтримувати типові відношення між документами, зокрема «один до багатьох» та «багато до багатьох».

У системі визначено основні колекції: users, recipes, ingredients, products, reviews, complaints, tags. Кожна реалізована у вигляді окремої моделі Mongoose з чітко визначеною схемою. У деяких документах зберігаються вкладені масиви об'єктів — кроки приготування, інгредієнти з одиницями виміру тощо. Це дозволяє уникати надмірної нормалізації структури та пришвидшити читання документів при завантаженні повного рецепта. У колекціях ingredients і products реалізовано зберігання статистичних даних про ціни, що використовуються для подальшого розрахунку вартості страв.

Опис структур даних, які використовуються в системі наведено в таблицях 2.8 - 2.14.

Таблиця 2.8

Опис колекції users

Поле	Тип	Призначення
_id	ObjectId	Унікальний ідентифікатор користувача
email	string	Електронна пошта
password	string	Хешований пароль
nickname	string	Ім'я користувача
role	enum (Role)	Роль у системі (user, moderator, admin)
avatarUrl	string	Посилання на фото профілю

Таблиця 2.9

Опис колекції recipes

Поле	Тип	Призначення
_id	ObjectId	Унікальний ідентифікатор рецепту
title	string	Назва рецепту
description	string	Опис страви
difficulty	enum (easy, medium, hard)	Рівень складності
author	ObjectId	Ідентифікатор автора
ingredients	Array<IRecipeIngredient>	Масив інгредієнтів із кількістю та обраною одиницею виміру
steps	Array<IRecipeStep>	Кроки приготування. Містять порядковий номер, посилання на фото та опис
tags	Array<ObjectId>	Мітки, що класифікують рецепт
mainPhoto	string	Посилання на фото страви
prepTime	number	Час приготування
servings	number	Кількість порцій
viewCount	number	Лічильник переглядів
createdAt	Date	Дата створення
updatedAt	Date	Дата останнього редагування

Таблиця 2.10

Опис колекції ingredients

Поле	Тип	Призначення
_id	ObjectId	Унікальний ідентифікатор інгредієнта
name	string	Назва інгредієнта
units	Array<Units>	Підтримувані одиниці виміру
products	Array<ObjectId>	Пов'язані товари

Продовження таблиці 2.10

prices	Partial<PriceMap>	Мін/макс/середня ціна по одиницях
invalidProducts	Array<string>	ID товарів, які не відповідають суті
isInvalid	boolean	Чи позначено інгредієнт як некоректний
createdAt	Date	Дата створення
updatedAt	Date	Дата оновлення

Таблиця 2.11

Опис колекції products

Поле	Тип	Призначення
_id	ObjectId	Унікальний ідентифікатор товару
title	string	Назва товару
price	number	Поточна ціна
oldPrice	number	Попередня ціна
classifierSapId	string	Ідентифікатор товару в каталозі
externalProductId	number	Зовнішній ID товару
slug	string	URL-friendly ідентифікатор
sectionSlug	string	Категорія товару
weighted	boolean	Чи товар продається на вагу
displayRatio	string	Рекомендоване відображення одиниці
details	Map	Додаткові характеристики необхідні для групування та формування інгредієнтів
createdAt	Date	Дата створення
lastUpdatedAt	Date	Дата останнього оновлення

Таблиця 2.12

Опис колекції complaints

Поле	Тип	Призначення
_id	ObjectId	Унікальний ідентифікатор скарги
entityModel	“Recipe” “Review”	Цільовий об’єкт
target	ObjectId (recipes reviews)	Ідентифікатор цільового об’єкта
author	ObjectId (users)	Ідентифікатор автора скарги
reason	string	Причина скарги
description	string	Опис
resolved	boolean	Статус
createdAt	Date	Дата створення
updatedAt	Date	Дата оновлення

Таблиця 2.13

Опис колекції reviews

Поле	Тип	Призначення
_id	ObjectId	Унікальний ідентифікатор відгуку
recipe	ObjectId (recipes)	Ідентифікатор рецепту
author	ObjectId (users)	Ідентифікатор автора відгуку
rating	number	Оцінка від 1 до 5
content	string	Контент відгуку
createdAt	Date	Дата створення
updatedAt	Date	Дата оновлення

Опис колекції tags

Поле	Тип	Призначення
_id	ObjectId	Унікальний ідентифікатор тегу
name	string	Назва тегу
color	string	Колір тегу

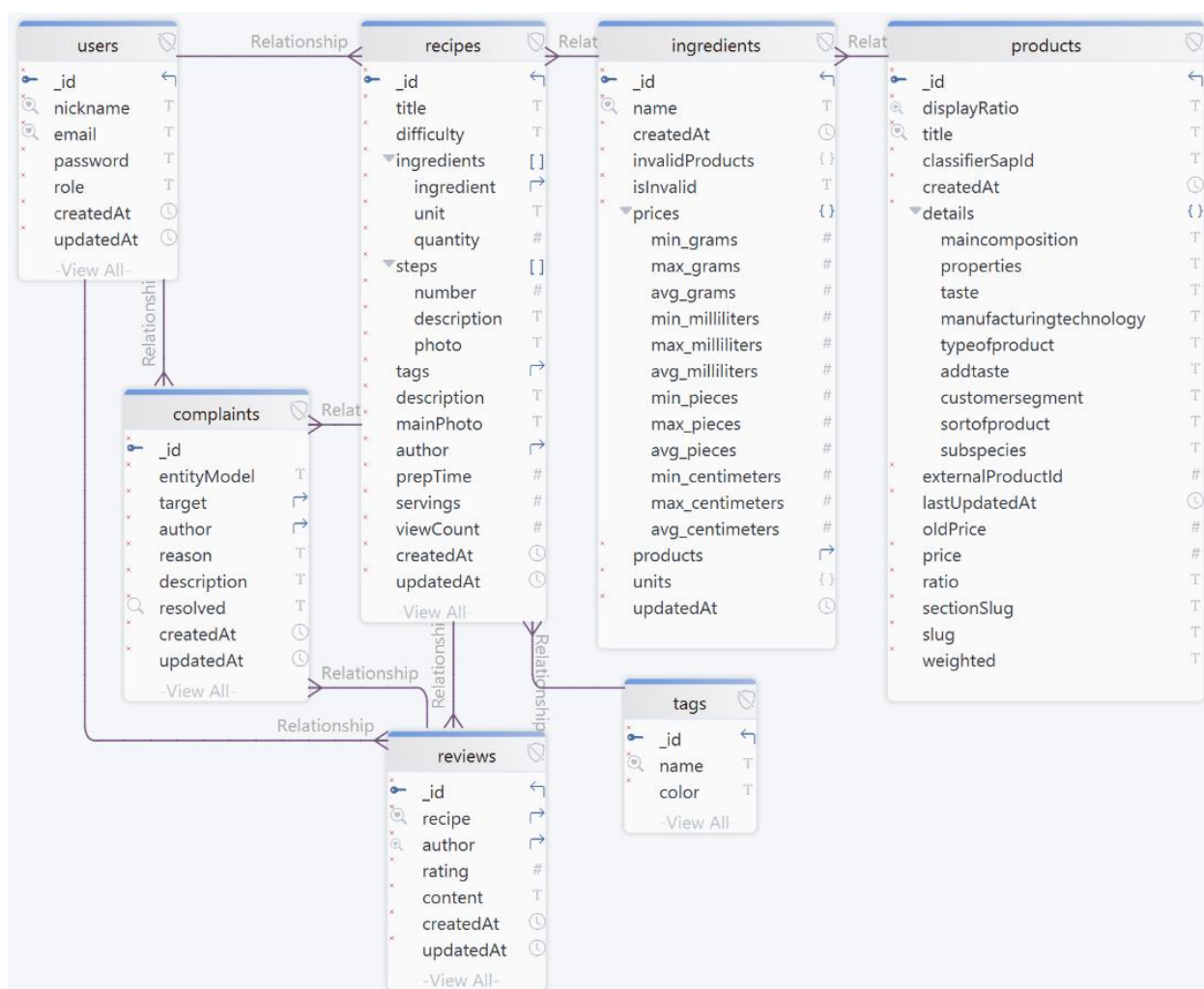


Рис. 2.3. Структура та зв'язки між колекціями бази даних MongoDB

Після аналізу схеми можна простежити кілька ключових взаємозв'язків. Кожен рецепт має обов'язковий зв'язок із автором, що реалізується через поле author і посилається на документ у колекції users. Відгуки (reviews) також зберігають посилання на автора та відповідний рецепт, формуючи двосторонній зв'язок між колекціями users, recipes і reviews. Користувач

може залишити кілька відгуків, але кожен відгук належить лише одному рецепту.

У колекції `recipes` містяться масиви тегів (`tags`) і скарг (`complaints`), які не зберігаються всередині документа, а реалізуються за допомогою зовнішніх зв'язків через зберігання ідентифікаторів. Скарги прив'язуються до певного рецепту або відгуку за допомогою полів `target` і `entityModel`, при цьому кожна скарга також має автора.

Інгредієнти (`ingredients`) зв'язані з товарами (`products`), з яких отримуються актуальні ціни. Один інгредієнт може бути представлений кількома продуктами, а один продукт може бути частиною кількох інгредієнтів. Зв'язок реалізовано через масив `products` у документі інгредієнта, а ціни агрегуються та зберігаються як `min`, `max`, `avg` з поначенням одиниці вимірювання.

Теги (`tags`) прив'язуються до рецептів, забезпечуючи категоризацію. Кожен тег може належати до великої кількості рецептів, а один рецепт може містити багато тегів — таким чином реалізується зв'язок типу «багато до багатьох».

MongoDB також дозволяє індексувати часто використовувані поля, як-от `tags`, `author`, `slug`, `price`, що значно покращує продуктивність пошуку й агрегацій. Гнучкість документної моделі дозволяє легко оновлювати структуру даних у майбутньому без необхідності складних міграцій, характерних для реляційних баз даних. Таким чином, запропонована структура БД забезпечує ефективну підтримку функціональних вимог системи, дозволяє масштабувати дані за потреби та швидко адаптуватися до змін у логіці вебзастосунку.

2.3. Проектування та реалізація алгоритмів роботи системи

Вебзастосунок, який реалізується в межах цієї кваліфікаційної роботи, містить низку бізнес-процесів, реалізованих через чітко визначені алгоритми.

Усі дії користувачів та системні обробки організовані згідно з ролями, що дозволяє забезпечити надійність, гнучкість та захищеність роботи системи.

Однією з базових операцій у системі є авторизація користувача. Після введення електронної пошти та пароля відбувається валідація даних на клієнті, після чого формується запит на сервер. У разі успішної авторизації користувач отримує токен доступу та перенаправляється на головну сторінку. Якщо введено неправильні дані, система виводить повідомлення про помилку. Аналогічно реалізована процедура реєстрації — з тією різницею, що також перевіряється унікальність email-адреси та нікнейму.

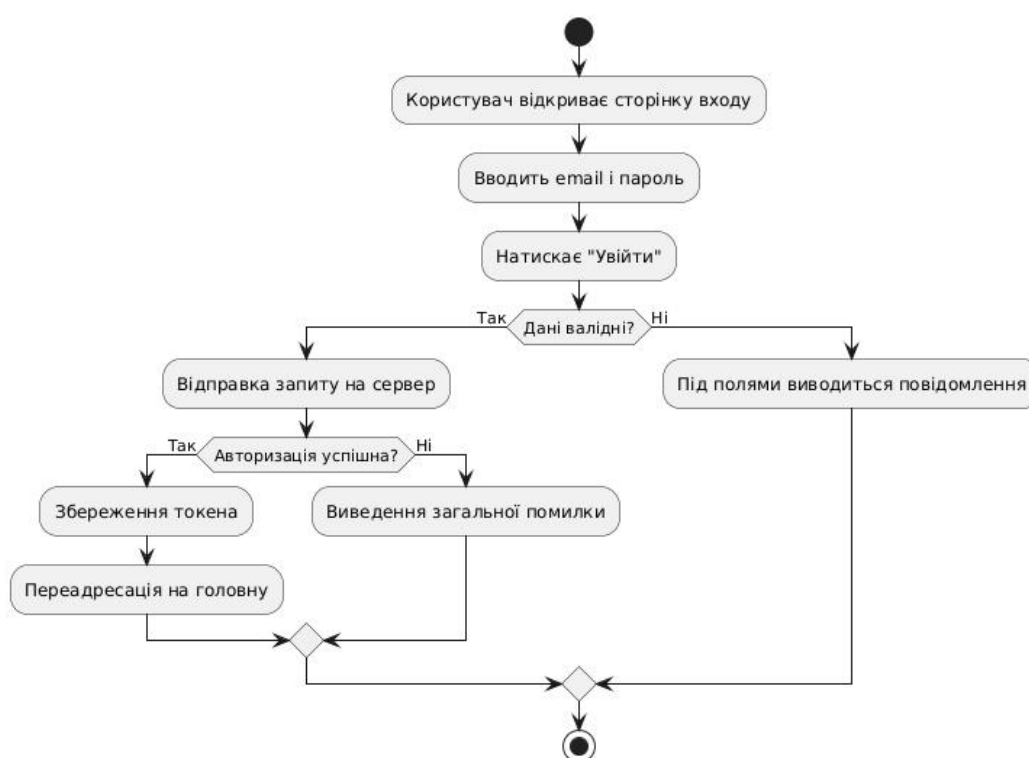


Рис. 2.4. Діаграма активності процесу авторизації користувача

Система побудована за принципом гнучкого доступу: у шапці сайту користувач бачить лише ті посилання, які дозволяє його роль. При спробі вручну перейти на сторінку, яка недоступна відповідній ролі (наприклад, створення тегів для звичайного користувача), користувач буде перенаправлений на сторінку з помилкою 404. Таким чином, на рівні інтерфейсу реалізовано як доступ, так і базову форму захисту маршрутизації.

Створення рецепту — один із найскладніших з погляду інтерактивності процесів у системі. Після переходу на відповідну сторінку, за допомогою SSR, відбувається попереднє завантаження списку тегів, доступних для вибору. Користувач вводить основні параметри рецепту: назву, опис, складність, кількість порцій, час приготування, а також завантажує головне фото. Окремо реалізовано механізм додавання інгредієнтів. Після введення щонайменше двох символів у полі пошуку інгредієнта, з'являється затримка (debounce), після якої відправляється запит на сервер, що повертає список релевантних назв. Користувач обирає інгредієнт зі списку — після чого формується набір можливих одиниць виміру. Користувач вибирає одиницю та вводить кількість. Усі дії можуть бути змінені або скасовані до фінального підтвердження.

Після введення всіх кроків приготування (опис + фото, за потреби), користувач натискає кнопку «Створити рецепт». Відбувається валідація форми: якщо якісь поля не заповнено — вони позначаються повідомленням про помилку. У разі успішної перевірки формується запит до API, фото завантажується в Google Cloud Bucket, а в БД зберігається їхня назва. Після створення документа recipe, користувач автоматично перенаправляється на сторінку щойно створеного рецепту.

Редагування рецепту реалізовано на основі тієї ж форми, що й створення, однак у даному випадку компоненти ініціалізуються даними (initData), отриманими з БД для відповідного рецепту. Доступ до редагування обмежено автором рецепту, а також модераторами й адміністраторами. Усі поля рецепту можуть бути змінені. Якщо при редагуванні було оновлено одне або кілька фото (як основне, так і ті, що належать до кроків), система автоматично видаляє старі файли з Google Cloud Bucket, після чого зберігає нові з унікальними іменами. Як і при створенні, редагований рецепт після успішного збереження перенаправляє користувача на відповідну сторінку рецепту.

Таким чином, форма є уніфікованою для обох сценаріїв — створення та редагування — і відрізняється лише режимом ініціалізації, логікою перевірки прав доступу, а також обробкою раніше завантажених зображень.

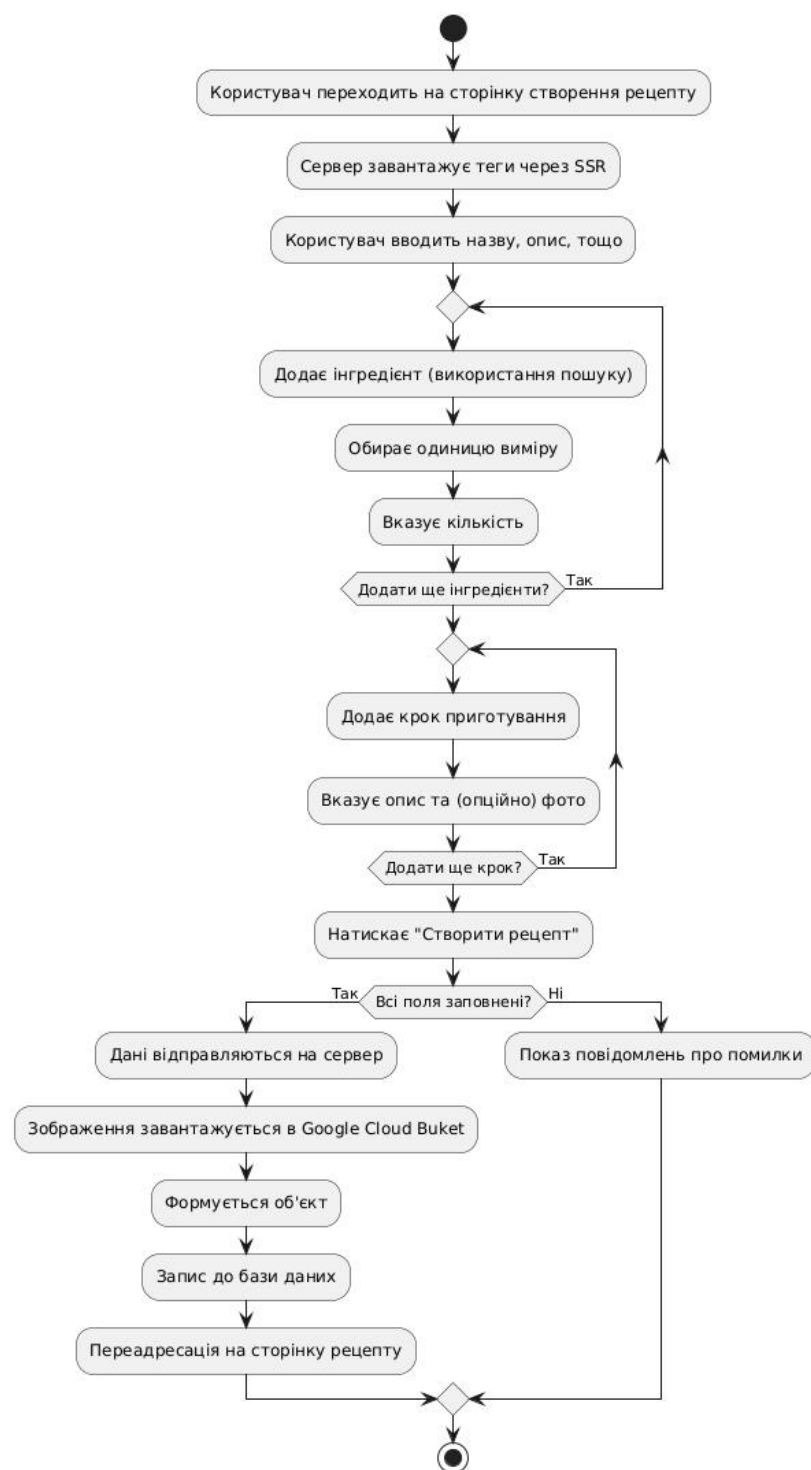


Рис. 2.5. Діаграма активності процесу створення рецепту

Під час завантаження рецепту (або списку рецептів), система виконує агрегаційний запит до БД, який автоматично: підтягує зв'язані документи;

обчислює мінімальну, максимальну та середню ціни рецепту на основі актуальних цін інгредієнтів; підраховує рейтинг за відгуками. Це забезпечує оновлюваність інформації без повторних розрахунків на клієнті. Окремо реалізовано інкремент переглядів — кожен перегляд сторінки рецепту збільшує лічильник viewCount.

Для коректної роботи створення рецептів і агрегаційних запитів система потребує достовірних, структурованих даних про інгредієнти та їхні ціни. Для цього реалізовано окремий серверний компонент, що автоматично формує й оновлює інгредієнти на основі зібраної інформації. Спершу запускається скрапер, який звертається до API супермаркету «Сільпо» для отримання дерева категорій. Після фільтрації залишаються лише релевантні категорії, які можуть бути використані як інгредієнти.

Далі система асинхронно обходить ці категорії та збирає товари, зберігаючи лише необхідні поля (наприклад, назву, ціну, details, ratio). Отримані документи нормалізуються й записуються до колекції products, оновлюючи або додаючи записи. Після цього автоматично запускається модуль ingredients_builder, який опрацьовує зібрані продукти. Назва інгредієнта визначається на основі поля details або sectionSlug, з урахуванням специфічних правил для кожної категорії. Система групує продукти за назвою інгредієнта, парсить одиниці виміру з displayRatio (через регулярні вирази й таблицю відповідностей), і розраховує ціну за базову одиницю.

Для кожного інгредієнта визначаються статистичні показники — мінімальна, максимальна та середня ціна — з виключенням аномальних значень за міжквартильним інтервалом. На завершення формується документ з назвою, продуктами, одиницями виміру, цінами та датою оновлення. Він зберігається або оновлюється в колекції ingredients. У результаті, цей процес виступає як ETL-пайплайн (Extract – Transform – Load), що виконується автоматично, підтримуючи актуальність цін і дозволяючи масштабувати систему або підключати нові джерела даних у майбутньому.



Рис. 2.6. Діаграма активності процесу формування інгредієнтів

Модератори мають окрему сторінку для перевірки правильності інгредієнтів. Вони можуть знайти інгредієнт, відмітити його як некоректний

або вказати, що деякі продукти з ним не пов'язані. Зміни зберігаються локально до моменту підтвердження. Якщо користувач натискає «Скасувати» — зміни відкидаються. Якщо «Зберегти» — вони відправляються на сервер і оновлюють відповідні поля в БД.

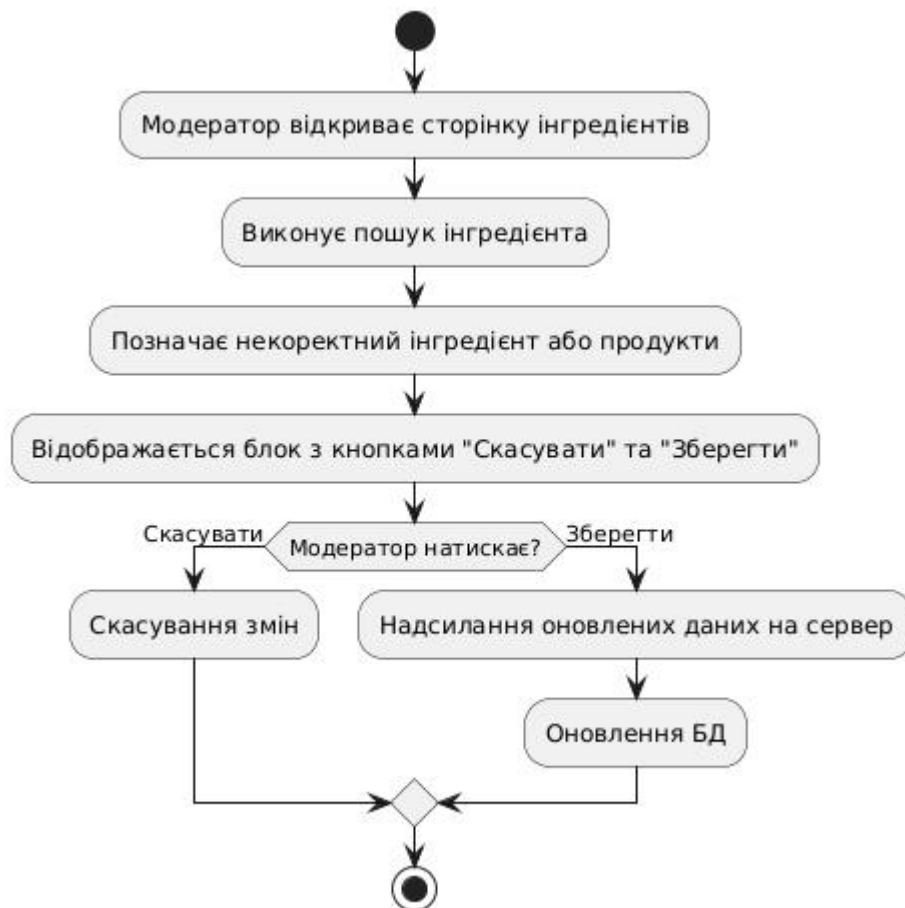


Рис. 2.7. Діаграма активності процесу модерації інгредієнтів

2.4. Реалізація функціоналу системи

У межах розробки програмного забезпечення було реалізовано низку ключових функціональних можливостей, що забезпечують повноцінну взаємодію користувача з платформою, автоматичну обробку даних, гнучку систему керування контентом і динамічні обчислення. Функціонал системи охоплює як клієнтську взаємодію, так і серверну логіку, ETL-процеси та роботу з мультимедійним вмістом. Його структура наочно представлена на діаграмі компонентів (Рис. 2.8).

формування запитів до API з урахуванням фільтрів (наприклад, тип товару, форма випуску, підвид, тощо). Для кожної категорії визначаються відповідні URL-адреси з параметрами пагінації, а також здійснюється покрокова обробка товарів. Список запитів формується динамічно, і система виконує їх асинхронно з логуванням категорій, фільтрів і кількості товарів (Додаток А).

Після завершення етапу збору даних викликається функція `build_ingredients`, яка виконує агрегацію отриманих товарів у сутності інгредієнтів. У цьому процесі система намагається побудувати однозначну назву інгредієнта, використовуючи або категоріальні ознаки (наприклад, «молочні продукти», «м'ясо»), або структуровані поля `detail`, якщо вони наявні. Для різних категорій існують індивідуальні правила: наприклад, для сирів назва формується на основі комбінації полів `typeofproduct` та `subspecies`, тоді як для м'ясних товарів може враховуватися внутрішній параметр `tipsteika`. Якщо ні одна з ознак не дозволяє сформувати назву, використовується транслітерація `sectionSlug` (Додаток Б).

На етапі нормалізації одиниць виміру виконується розбір співвідношення товару (наприклад, «500 г», «1 л», «2 шт») за допомогою регулярного виразу. Кожна одиниця має відповідний коефіцієнт перерахунку до базової одиниці (г, мл, шт), що дозволяє уніфікувати обчислення незалежно від форми упаковки. Наприклад, пляшка води об'ємом 1.5 л та пакет соку на 950 мл будуть приведені до єдиної міри — мілілітрів (Додаток В).

Після приведення цін усіх товарів до єдиної базової одиниці виконується обчислення статистичних метрик. Щоб виключити цінові викиди, застосовується міжквартильний фільтр: розраховується інтерквартильний розмах і відсікаються значення, що виходять за межі $[Q1 - 1.5 \times IQR, Q3 + 1.5 \times IQR]$. На основі очищеного масиву значень система обчислює середню, мінімальну та максимальну ціну[23] (Додаток Г).

У результаті в колекції ingredients зберігається об'єкт, що містить назву інгредієнта, список пов'язаних товарів, нормалізовану одиницю виміру та зведену статистику.

Крім того, на серверній частині реалізовано систему автентифікації та контролю доступу. Авторизація користувачів здійснюється за допомогою бібліотеки NextAuth, яка інтегрується з Next.js та забезпечує зручну обробку сесій, токенів і зовнішніх провайдерів автентифікації[24]. Для контролю доступу до сторінок і API застосовується функція-обгортка withRoles, яка викликається у SSR-хендлерах (getServerSideProps). У разі, якщо користувач не авторизований, система перенаправляє його на сторінку входу; якщо роль не відповідає вимогам — повертається сторінка 404. У протилежному випадку користувач отримує доступ до сторінки з додатковим об'єктом user у props (Додаток Д)..

Цей механізм дозволяє централізовано керувати рівнями доступу: звичайні користувачі можуть створювати і редагувати лише власні рецепти, модератори мають можливість переглядати та обробляти скарги й інгредієнти, а адміністратори отримують доступ до керування тегами.

Під час створення або редагування рецепта дані користувача надсилаються через форму з типом multipart/form-data. Усі поля рецепта передаються у вигляді частин форми, а зображення прикріплюються як окремі файли. На сервері ці дані обробляються за допомогою бібліотеки formidable, яка дозволяє розпарсити як текстову, так і файлову частину запиту в межах одного обробника[25]. Якщо рецепт редагується, виконується перевірка: у разі зміни фотографії, її попередня версія видаляється зі сховища Google Cloud Storage, щоб уникнути накопичення зайвих файлів. Нові зображення завантажуються у хмару, а відповідні URL зберігаються у полях документа рецепта (mainPhoto, steps.photo). Така сама логіка застосовується для кожного кроку рецепта. Зміни зберігаються через PUT-запит до відповідного API-ендпоінту, який оновлює лише ті частини документа, що були змінені (Додаток Е).

Окрему реалізацію має логіка розрахунку вартості рецепта, яка здійснюється виключно на сервері, без додаткових клієнтських обчислень. Під час агрегації в MongoDB, для кожного рецепта виконується \$lookup до колекції ingredients, після чого застосовується етап \$addFields, який додає поле price. Це поле включає три підполя — avg, min і max, кожне з яких містить загальну вартість рецепта за відповідною методикою. Розрахунок виконується за допомогою \$map та \$multiply, де кожна кількість інгредієнта множиться на ціну за одиницю, обрану динамічно на основі поля unit (наприклад, avg_grams, min_pieces). Це значення вилучається через \$getField з відповідного об'єкта prices, що міститься у підключеній інформації ingredientDetails. Такий підхід дозволяє обчислювати повну вартість рецепта на основі актуальних агрегованих цін, незалежно від форми упаковки або кількості товарів. Агрегаційний етап реалізовано у вигляді об'єкта attachPrice, який вставляється у відповідне місце пайплайну (Додаток Ж).

Ключовий функціональний компонент клієнтської частини — це інтерактивна форма створення та редагування рецепту, реалізована як окремий компонент RecipeCreateForm. Вона дозволяє користувачу у зручному візуальному інтерфейсі поступово заповнити всі необхідні дані. Структура рецепта формується поступово у внутрішньому стані компонента, який оновлюється в реальному часі після кожної взаємодії користувача — наприклад, при додаванні чергового кроку або виборі одиниці виміру для інгредієнта. Під час підтвердження форма виконує локальну валідацію всіх обов'язкових полів, після чого збирає дані у форматі FormData. Це дозволяє надіслати на сервер одночасно як текстові значення, так і файли (головне фото страви та фото кожного кроку приготування). Компонент також динамічно керує списками інгредієнтів та тегів — при редагуванні раніше збережені значення підтягуються з сервера і одразу відображаються у відповідному вигляді. Форма підтримує як режим створення, так і режим редагування. У режимі редагування усі поля автоматично заповнюються на основі отриманих даних (initialData), а головне фото і зображення кроків

одразу підтягуються зі сховища, що забезпечує зручний UX для оновлення раніше збережених рецептів. При скасуванні редагування користувач перенаправляється назад до перегляду рецепту (Додаток И).

Таким чином, реалізація функціоналу охоплює як клієнтські сценарії роботи, так і серверну логіку з контролем доступу, обробкою медіа, ETL-процесами та динамічними обчисленнями вартості. Усі ці компоненти злагоджено взаємодіють, забезпечуючи повноцінну роботу платформи.

Висновки до другого розділу

В результаті проєктування платформи було здійснено аналіз основних вимог до функціональності системи, а також визначено її структурну організацію. На основі отриманих вимог було спроектовано архітектуру платформи, що включає клієнтську та серверну частини на Next.js, базу даних MongoDB та ETL-модуль на Python. Детально описано взаємодію між цими компонентами, а також визначено їх ролі в обробці, зберіганні та оновленні інформації.

Окрему увагу було приділено моделюванню БД, що містить колекції користувачів, рецептів, тегів, скарг, інгредієнтів, продуктів та інших сутностей, які необхідні для коректного функціонування платформи. Було обґрунтовано вибір окремих структур даних, зокрема для зберігання статистики та фотографій через інтеграцію з Google Cloud Storage.

Крім того, було реалізовано ключовий функціонал системи — автоматичне оновлення цін інгредієнтів на основі даних із зовнішнього API магазину «Сільпо». Цей функціонал забезпечує регулярне оновлення та агрегацію цін, їх нормалізацію та підготовку до подальших розрахунків. Загалом, проєктування дало змогу створити гнучку, масштабовану структуру платформи, яка враховує як функціональні, так і нефункціональні вимоги, а також забезпечує можливість розширення системи в майбутньому.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ СЕРВІСУ ФОРМУВАННЯ МЕНЮ ТА РЕЦЕПТІВ

3.1. Порядок встановлення та налаштування параметрів платформи

Розгортання вебзастосунок охоплює декілька ключових компонентів, що взаємодіють у межах загальної інфраструктури: застосунок на базі Next.js, сервер скрапінгу даних, хмарна база даних MongoDB Atlas, а також Google Cloud Storage як сховище користувацьких зображень. Усі обчислювальні компоненти розгорнуті окремо на сервісі Google Cloud Run.

Вебзастосунок реалізовано на основі фреймворку Next.js, який поєднує в собі клієнтську та серверну частини. Цей застосунок розгортається як єдиний контейнер, що обслуговує інтерфейс користувача, обробку запитів, завантаження медіа, SSR і авторизацію. Для наочності, на Рис.3.1 Next.js-застосунок поділено на логічні складові — "Клієнт" та "Сервер", хоча фізично це один виконуваний процес.

MongoDB Atlas використовується як хмарна база даних для зберігання структурованої інформації: рецепти, інгредієнти, користувачі, відгуки, скарги тощо. Вебзастосунок підключається до неї за допомогою зашифрованого з'єднання (MongoDB+TLS). Окремо, до цієї ж БД підключається модуль скрапінгу, що дозволяє централізовано зберігати зібрані дані.

Скрапер реалізований як окремий Python-застосунок і розгортається незалежно на Google Cloud Run. Його завдання — періодично збирати актуальні дані з API супермаркету, обробляти їх, формувати структури інгредієнтів і зберігати результати в MongoDB. Скрипти запуску скрапінгу та побудови інгредієнтів виконуються послідовно у рамках одного процесу. Для зберігання зображень (фото рецептів, кроків, аватарів користувачів тощо) використовується Google Cloud Storage. Завантаження, перегляд і видалення медіа реалізовані через серверну частину Next.js, яка виступає в ролі проксі до Bucket — прямий доступ користувачів до сховища повністю виключено з

міркувань безпеки[26]. Імена завантажених файлів зберігаються в БД, а передача здійснюється через офіційний SDK Google Cloud.

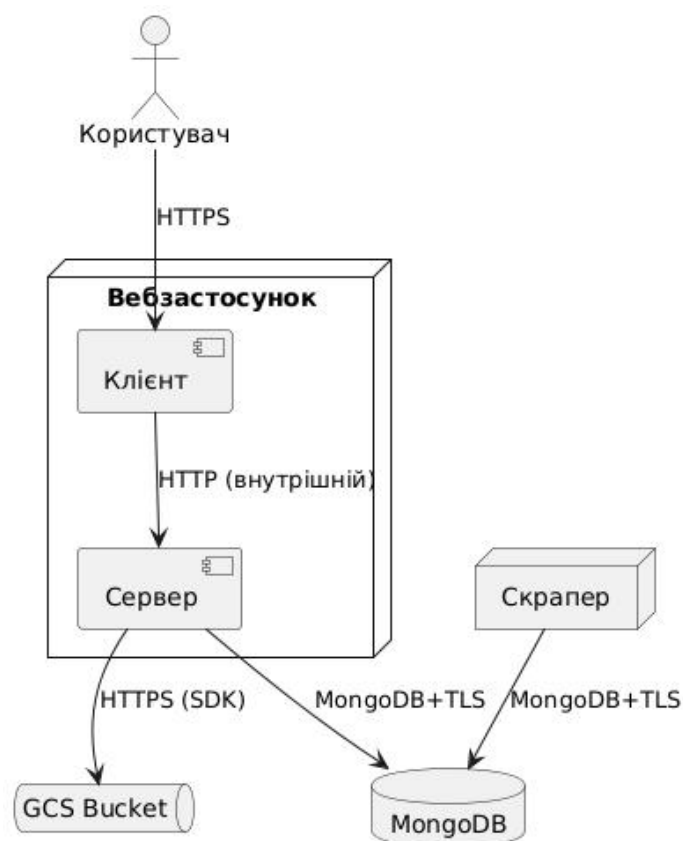


Рис. 3.1. Діаграма розгортання системи.

Запропонована архітектура дозволяє досягти високої гнучкості та масштабованості: усі ключові компоненти розгортаються незалежно, мають чітко визначені зони відповідальності та можуть оновлюватися або масштабуватися без впливу на інші частини системи. Такий підхід також полегшує обслуговування та спрощує подальший розвиток проєкту, зокрема — підключення нових джерел даних, зміну БД або хмарного провайдера.

Для забезпечення безпечної та гнучкої конфігурації кожен з компонентів системи використовує файли середовища `.env`, у яких зберігаються критично важливі параметри доступу до сервісів. Така практика відповідає принципам сучасних веб-застосунків, що рекомендують зберігати конфігурацію в змінних середовища[27]. Файл конфігурації застосунку (`.env`) не входить до репозиторію та повинен бути вручну створений перед запуском відповідного сервісу.

Для вебзастосунку (Next.js) передбачено такі ключові змінні:

1. MONGODB_URI — рядок підключення до бази даних.
2. NEXTAUTH_SECRET — секретний ключ, що використовується для підпису сесій та токенів у NextAuth.
3. NEXTAUTH_URL — повна URL-адреса розгорнутого сайту. Необхідно для правильного функціонування контролю доступу.
4. GCLOUD_STORAGE_BUCKET — ім'я бакету в Google Cloud Storage, у який завантажуються користувацькі зображення.
5. GOOGLE_APPLICATION_CREDENTIALS — шлях до JSON-файлу сервісного акаунта Google, який дозволяє серверу взаємодіяти з GCS через офіційний SDK. При розгортанні в Google Cloud Run не потрібно вказувати.

Для сервісу скрапінгу (Scraper) файл .env містить лише змінні, потрібні для підключення до бази даних:

1. MONGODB_URI — та ж змінна, що й у вебзастосунку, використовується для підключення до тієї самої бази.
2. MONGODB_DB — назва бази даних.

Забезпечення правильного значення кожного з параметрів є обов'язковою умовою для стабільної роботи всієї системи. Будь-яка помилка в конфігурації призводить до неможливості підключення до БД, збоїв авторизації або втрати доступу до медіафайлів.

3.2. Структура інтерфейсу

Основною точкою входу для взаємодії є хедер, який присутній на всіх сторінках платформи. У верхній частині інтерфейсу завжди кнопка пошуку рецептів, яка активує поле з автозаповненням після введення щонайменше двох символів. Натискання на запропоновану опцію переадресовує на відповідну сторінку рецепту, тоді як кнопка пошуку — на сторінку розширеного пошуку з переданими параметрами.

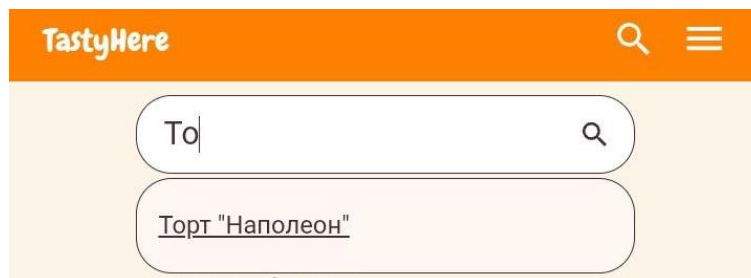


Рис. 3.2. Швидкий пошук рецепту.

Праворуч розміщується бургер-меню, що відкриває список навігаційних посилань, сформований динамічно згідно з роллю користувача. Залежно від статусу сесії, у нижній частині меню відображаються кнопки входу або виходу з акаунту.

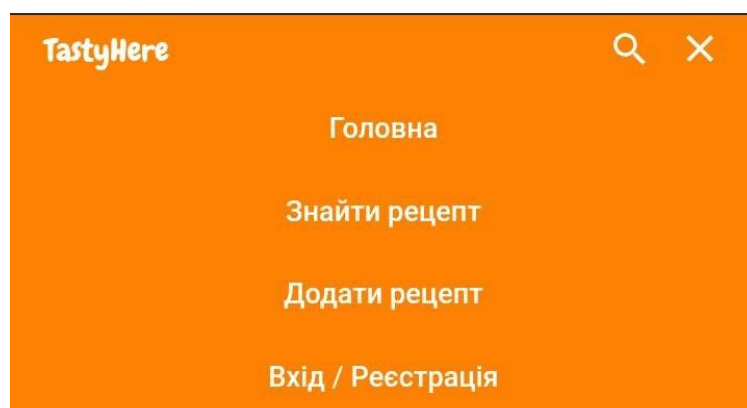


Рис. 3.3. Меню навігації не авторизованого користувача.



Рис. 3.4. Меню навігації адміністратора.

Головна сторінка містить шапку з фоновим зображенням, назвою сервісу, слоганом і кнопкою прокрутки до списку рецептів. Під шапкою — короткий опис місії платформи. Основну частину складають тематичні підбірки рецептів: нові, найпопулярніші та найкращі за рейтингом. Кожна з них містить обмежену кількість карток, які адаптуються до розміру екрана й дозволяють перейти до повного списку через відповідне посилання. Структура картки рецепту у всіх випадках однакова — включає фото, назву, опис, автора, рейтинг, складність, вартість та інші метадані.

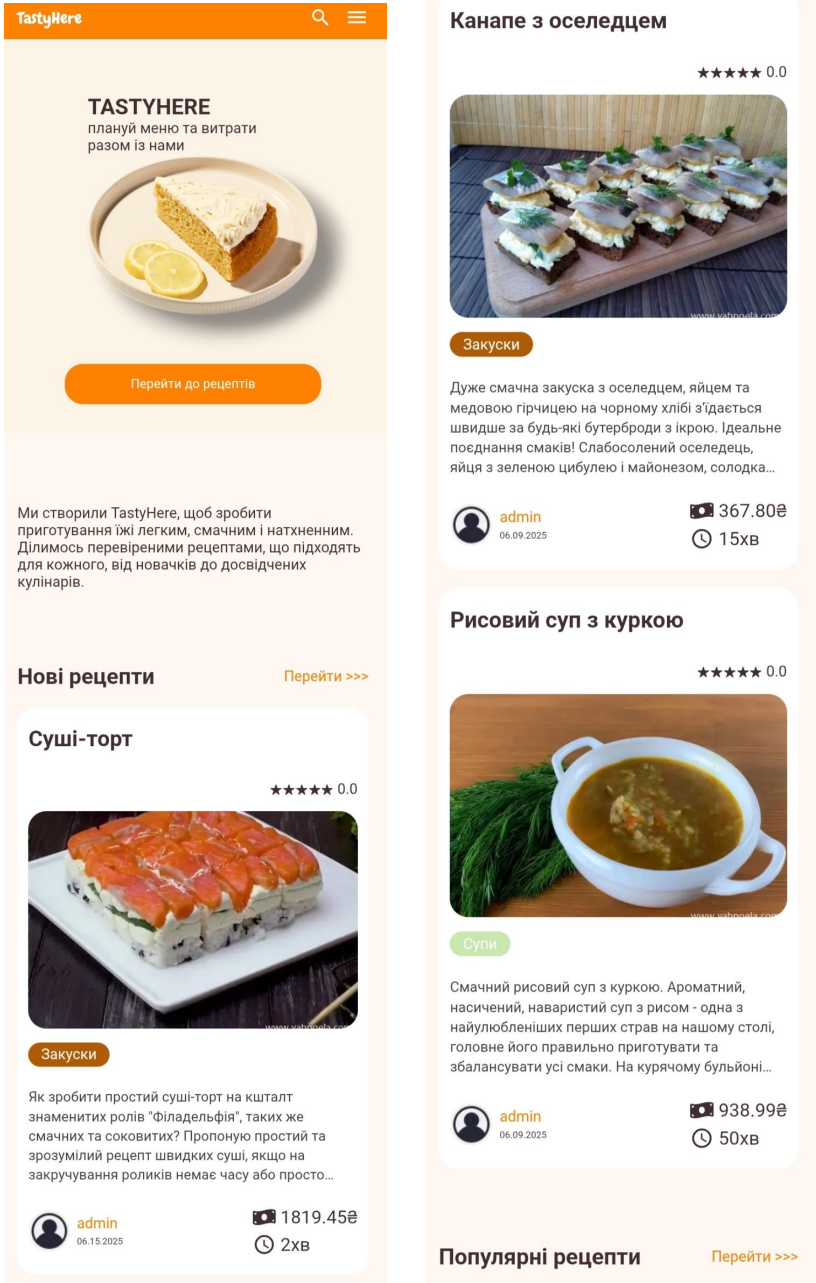


Рис. 3.5-3.6. Мобільна версія головної сторінки з першою підбіркою.

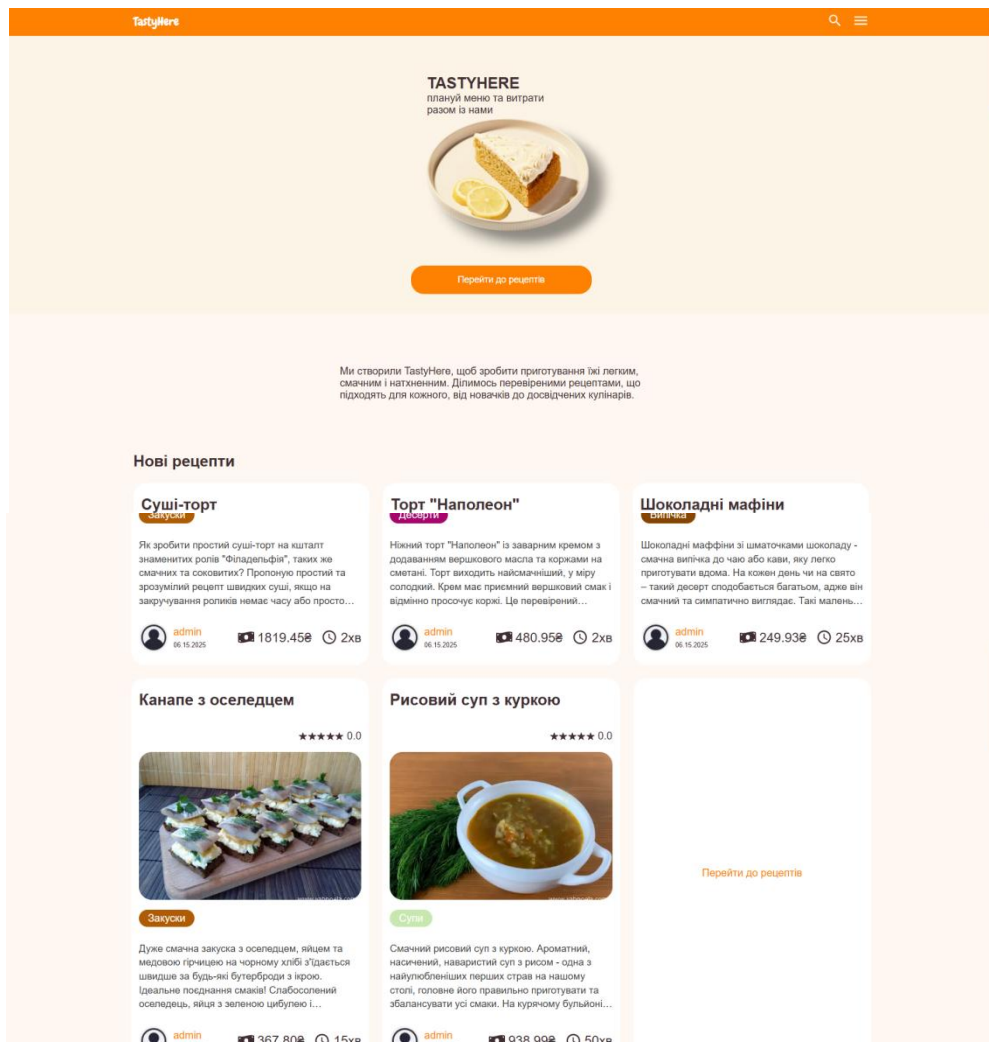


Рис. 3.7-3.8. Десктопна версія головної сторінки з першою підбіркою.

Після переходу на сторінку пошуку рецептів користувач може здійснити повнотекстовий пошук за назвою або частиною опису рецепту. Окрім пошукового рядка, інтерфейс надає фільтри за тегами, рівнем складності та додаткові параметри сортування: за датою, популярністю, рейтингом, вартістю або часом приготування. Результати автоматично оновлюються при зміні параметрів, а для переходу між сторінками результатів реалізована пагінація, що зберігає контекст фільтрації та сортування.

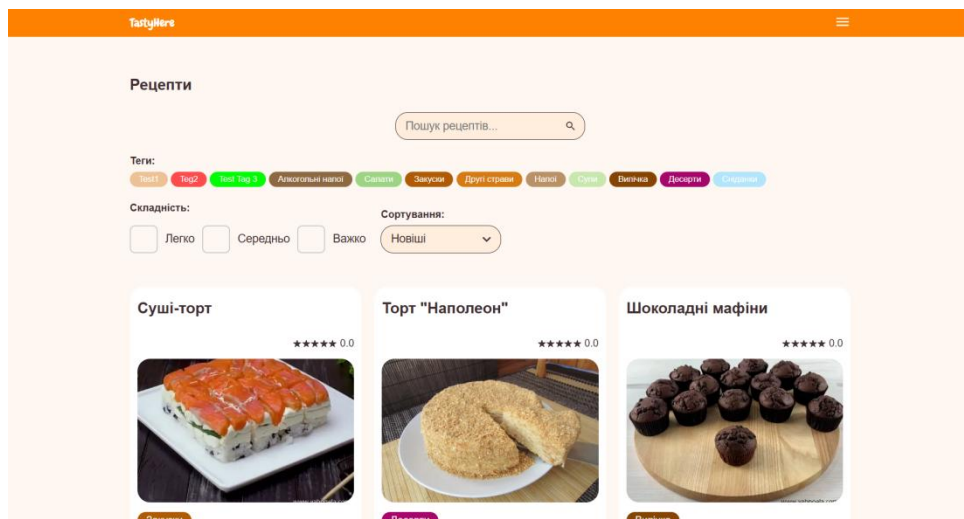


Рис. 3.9. Десктопна версія сторінки пошуку рецептів.

На сторінці перегляду окремого рецепту користувач бачить заголовок страви, а також, залежно від наданих прав, може редагувати або видаляти його. Основний контент включає фото, інформацію про автора, складність, рейтинг, опис, а також інтерактивний список інгредієнтів, де кожен елемент можна позначити як оброблений. В нижній частині реалізована система відгуків, де користувач може залишити оцінку та коментар.

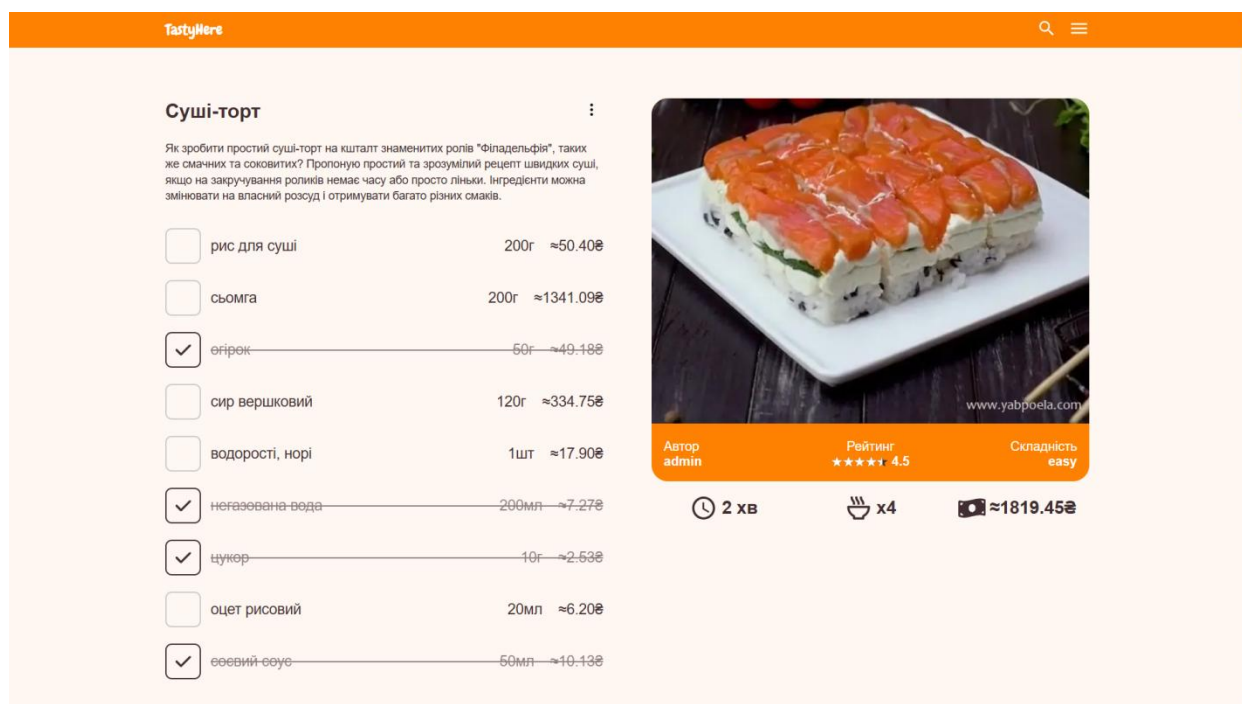


Рис. 3.10. Десктопна версія сторінки рецептів.



Рис. 3.11. Десктопна версія сторінки рецептів. Кроки.

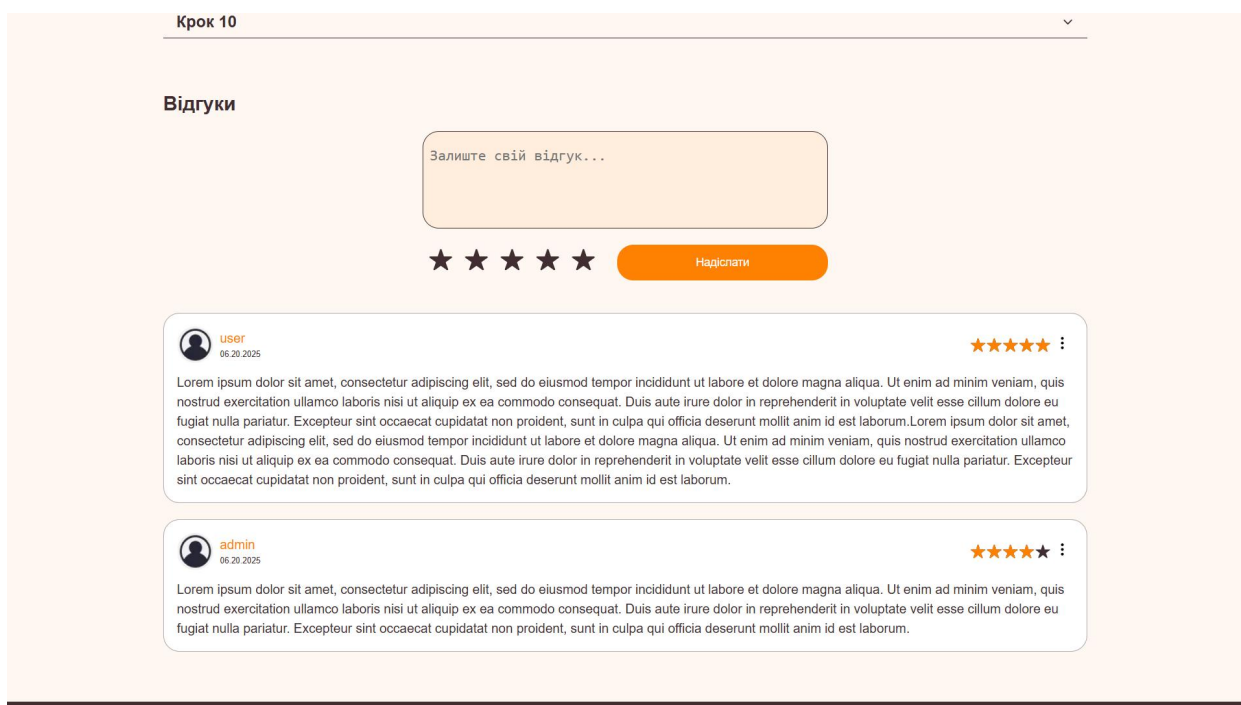


Рис. 3.12. Десктопна версія сторінки рецептів. Відгуки.

Авторизовані користувачі можуть створювати та редагувати рецепти. Інтерфейс реалізований у вигляді форми з підтримкою завантаження фото, вибору тегів, інгредієнтів, складності, введення кроків приготування та опису. Для кожного кроку передбачено додавання фото. Перед надсиланням виконується валідація обов'язкових полів

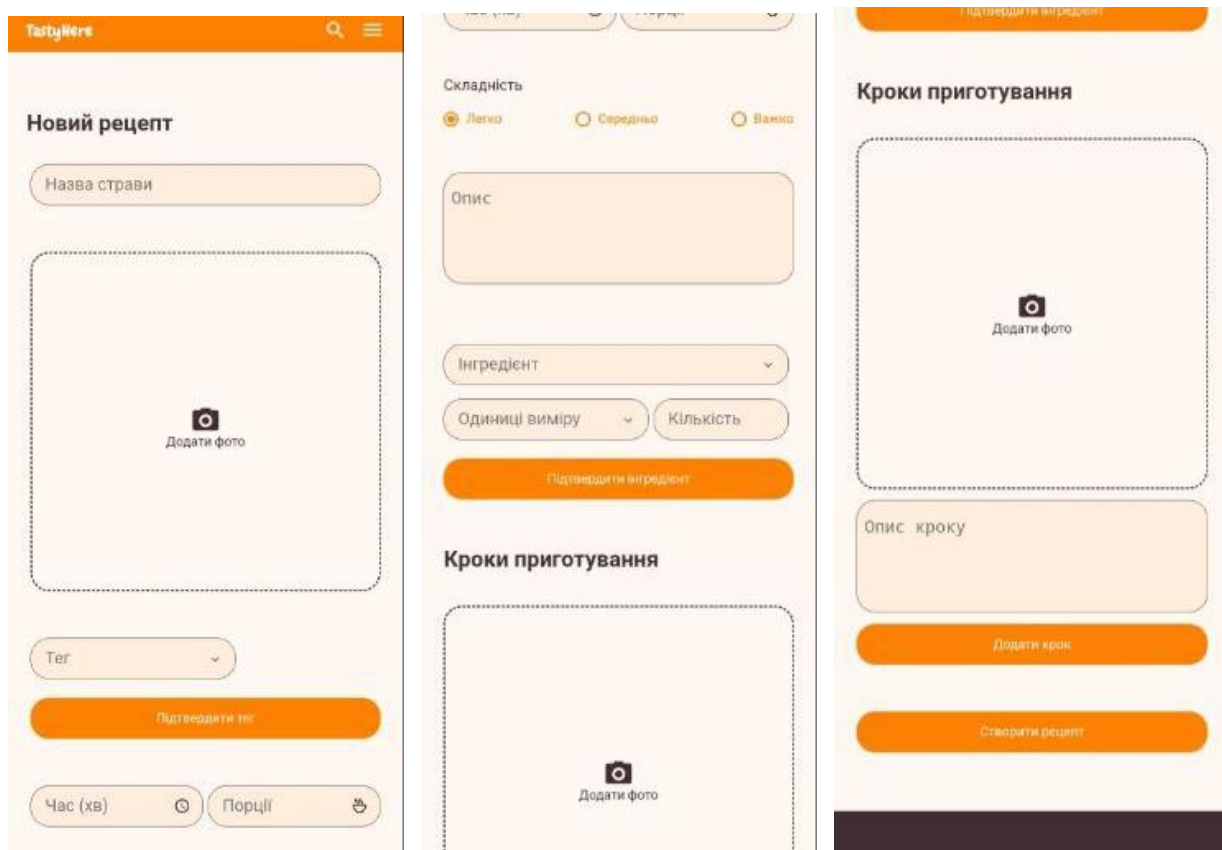


Рис. 3.13-3.15. Форма створення / редагування рецепту.

На сторінках рецепту або відгуку користувач може подати скаргу, натиснувши відповідну опцію в меню (Рис.3.13.). В модальному вікні він обирає причину скарги з попередньо визначеного списку та має змогу залишити додатковий опис (Рис.3.14.).

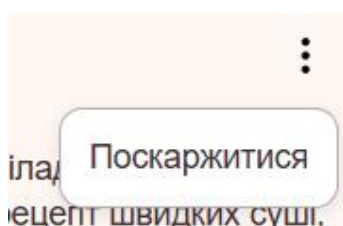


Рис. 3.16. Опція «Поскаржитися».

Нова скарга

Причина скарги ▼

Деталі

Відмінити

Надіслати

Рис. 3.17. Модальне вікно для залишення скарги.

На адміністративній сторінці перегляду скарг модератору надається список усіх поданих звернень із зазначенням автора, дати, об'єкта скарги, причини та опису. Кожен запис супроводжується посиланням на пов'язаний рецепт або відгук. Після розгляду скаргу можна позначити як вирішену. Інтерфейс адаптований до різних розмірів екранів та дозволяє ефективно обробляти великий обсяг звернень.

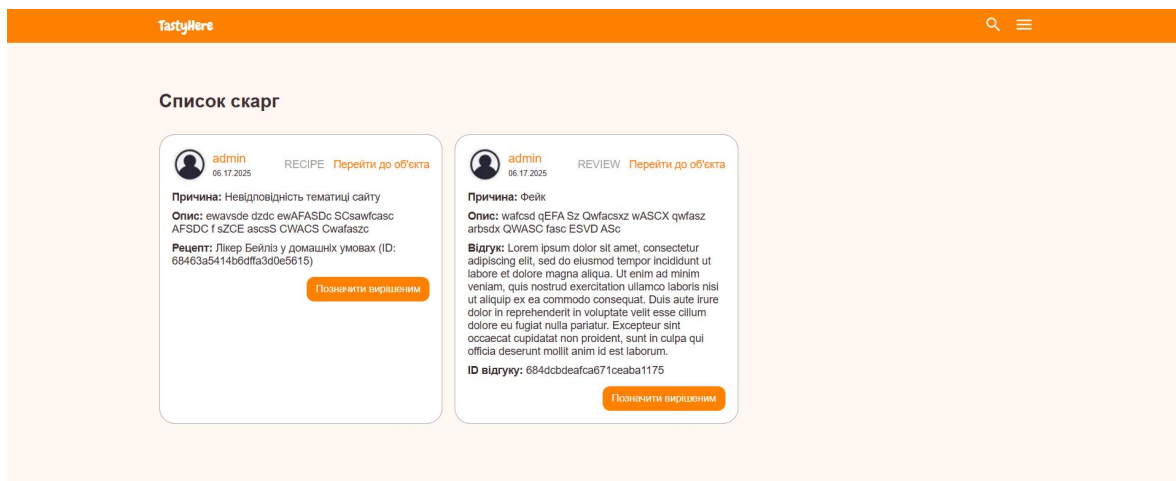


Рис. 3.18. Сторінка перегляду скарг.

Адміністратор платформи має доступ до сторінки управління тегами, де реалізовано функціональність створення, редагування та видалення тегів. Кожен тег має назву та колір, які можна змінювати через інтерактивну форму. Зміни одразу відображаються у загальному списку.

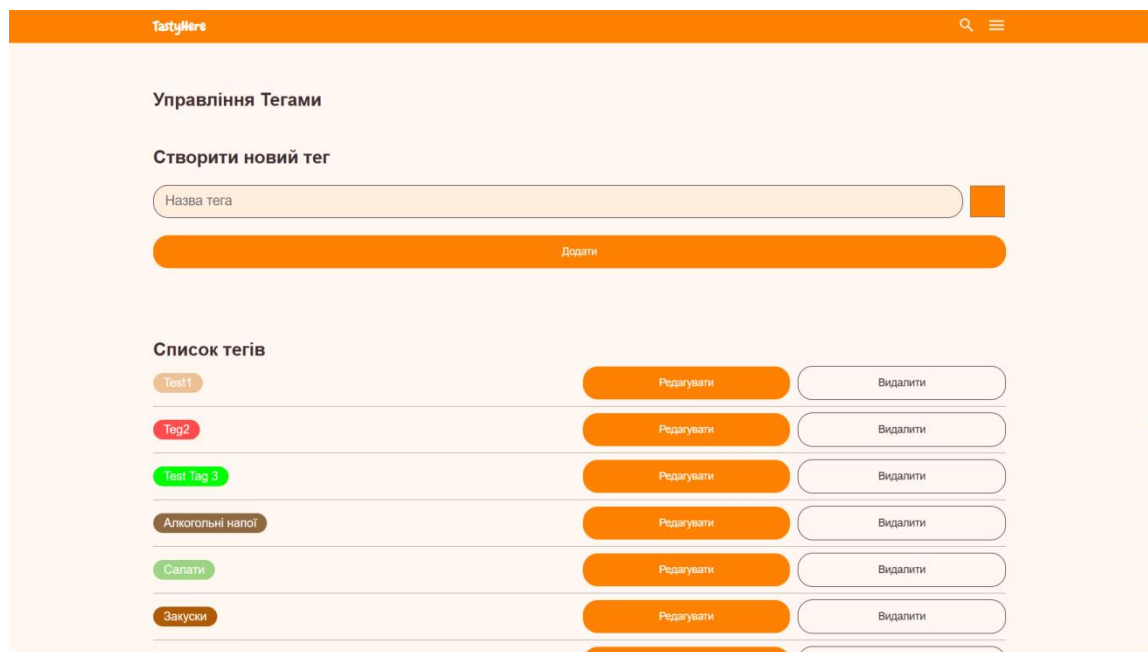


Рис. 3.19. Сторінка управління тегами.

Модератору доступна сторінка з переліком усіх інгредієнтів, які можуть бути позначені як невалідні. Кожен інгредієнт має список пов'язаних продуктів, що також можуть бути відмічені як невалідні. Внесені зміни тимчасово зберігаються на клієнтській стороні та застосовуються лише після підтвердження кнопкою «Зберегти». Передбачена також функція пошуку за назвою.

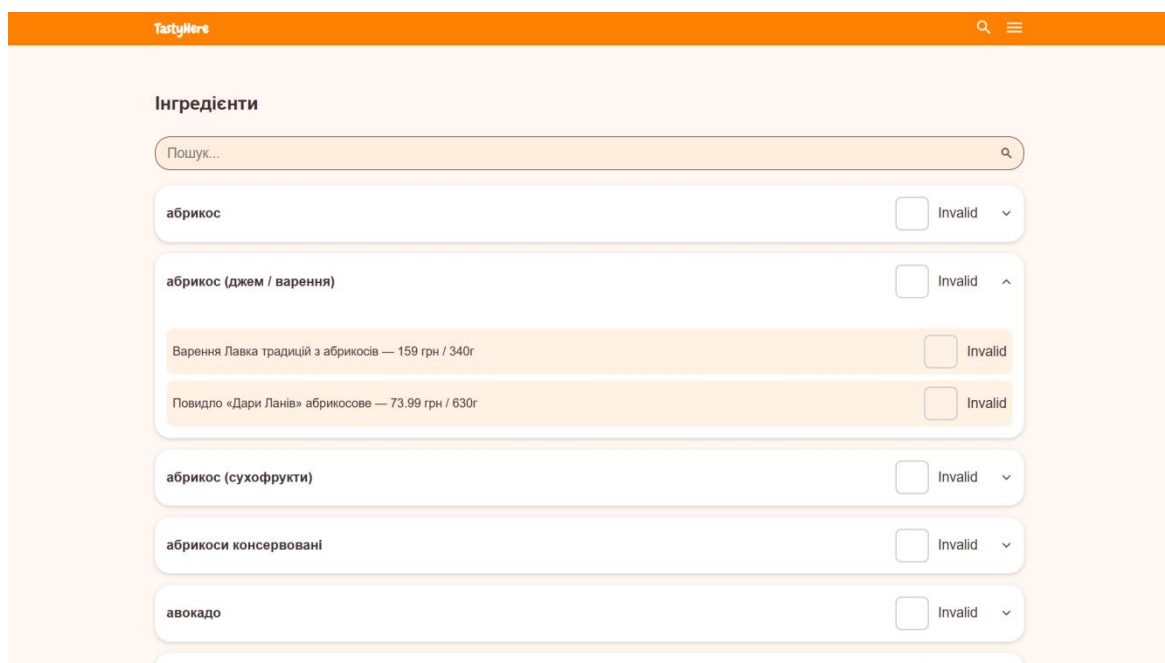


Рис. 3.20. Сторінка управління інгредієнтами.

3.3. Тестування платформи

Для оцінки продуктивності вебзастосунку було проведено тестування за допомогою сервісів GTMetrix[28] та Google PageSpeed Insights[29]. Тестування охоплювало як десктопну, так і мобільну версії головної сторінки.

GTMetrix проаналізував час відображення ключових елементів, блокування інтерфейсу та загальну структуру сторінки. Загальний результат Performance склав 94 %, а Structure — 89 %. Час First Contentful Paint (FCP), який показує, коли користувач бачить перший вміст сторінки, становив 762 мс, тоді як Speed Index — 790 мс, що свідчить про надзвичайно швидке формування візуального вмісту. Показник Largest Contentful Paint (LCP) — 1.5 с, що входить у рекомендовані межі Google[30]. Time to Interactive — 1.1 с, а Total Blocking Time (TBT) — лише 51 мс. Важливо зазначити, що Cumulative Layout Shift (CLS) дорівнює 0, що вказує на відсутність зміщення елементів при завантаженні (Рис. 3.21). Повне завантаження сторінки відбувалося за 1.7 с.

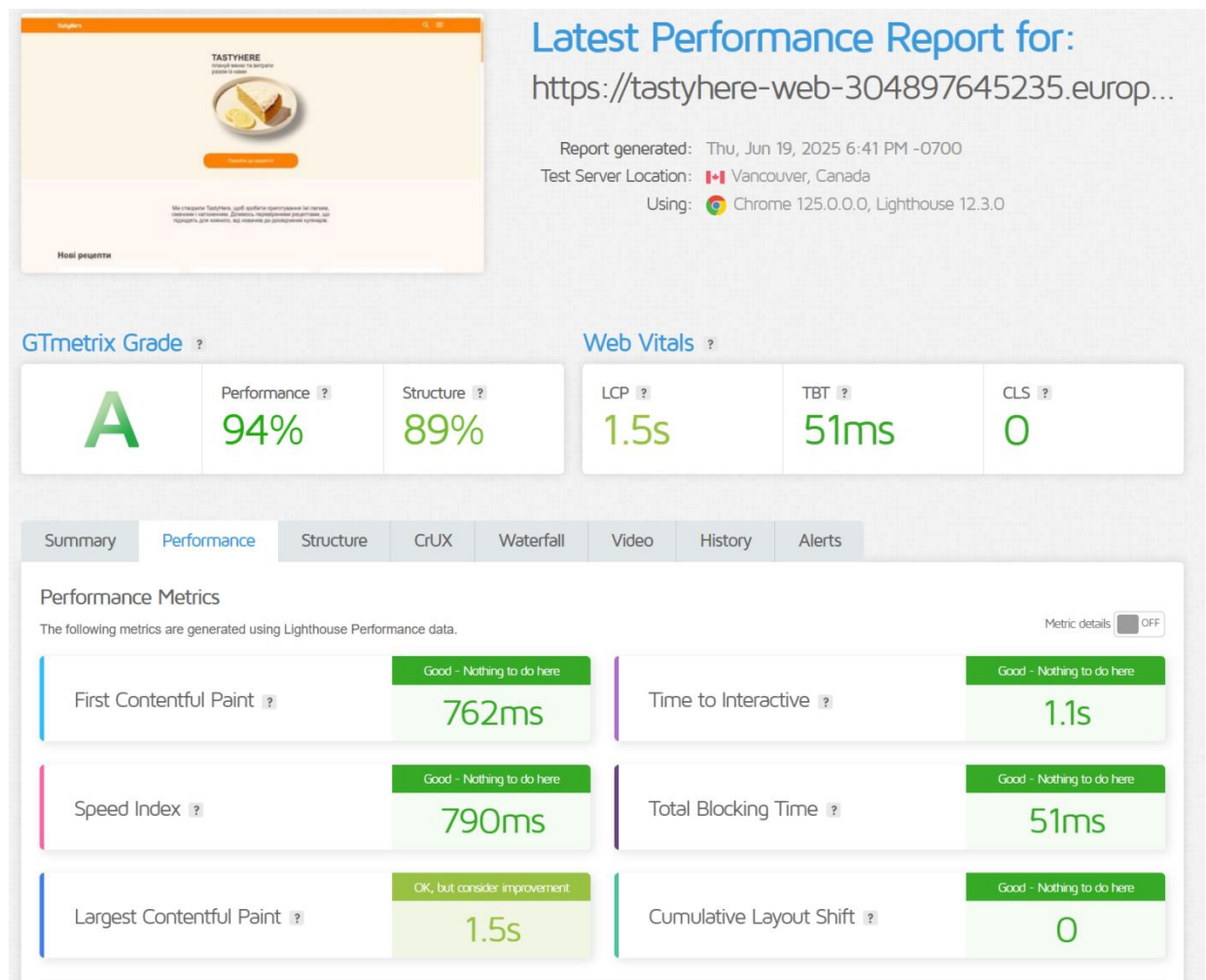


Рис. 3.21. Результати тестування продуктивності платформи у GTMetrix

Додатковий аналіз проведено за допомогою Google PageSpeed Insights. Десктопна версія платформи отримала оцінку ефективності 94/100. First Contentful Paint відбувся через 0.7 с, LCP — 1.3 с, а Speed Index склав 0.8 с. Загальний час блокування — 140 мс, CLS — 0. Ці показники свідчать про швидке завантаження та стабільну поведінку сторінки (Рис. 3.22).

Звіт від 20 черв. 2025 р., 04:53:43

<https://tastyhere-web-304897645235.europe-west3.run.app/>

Аналізувати



Мобільний



Комп'ютер



Дізнайтеся, як відбувається взаємодія в реальних користувачів



Немає даних



Виявляйте проблеми з продуктивністю

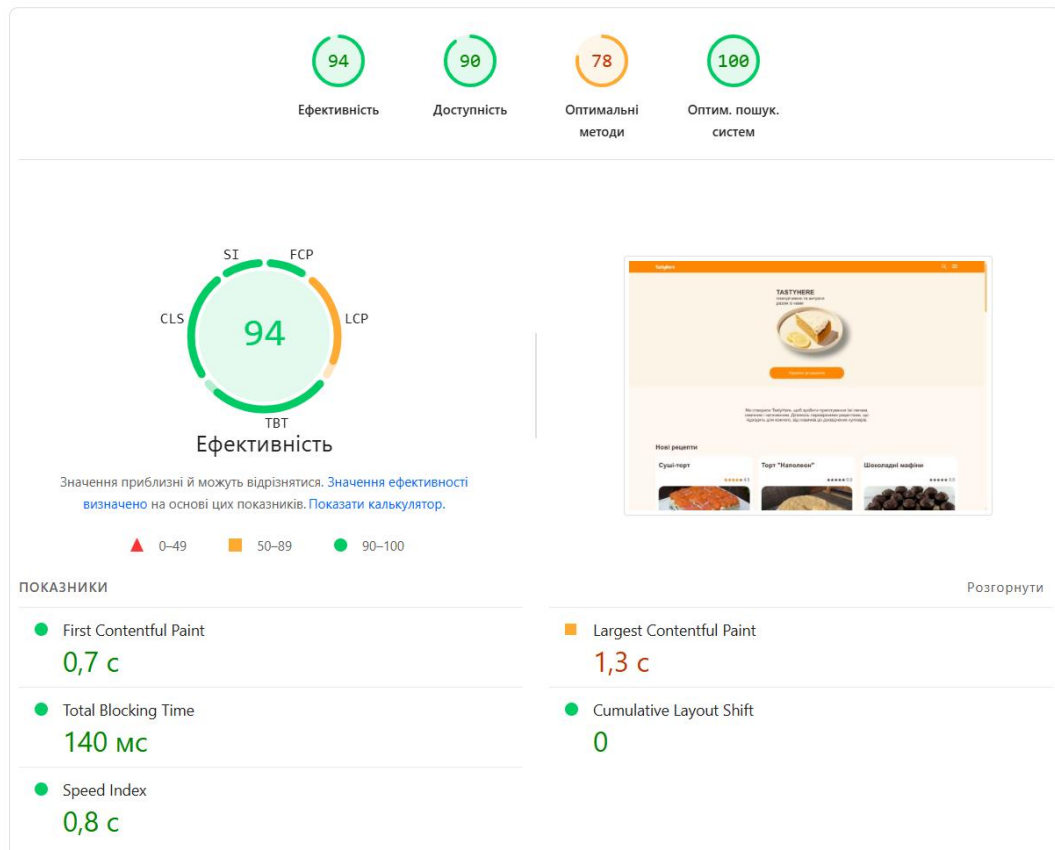


Рис. 3.22. Аналіз десктопної версії платформи у PageSpeed Insights

У мобільній версії спостерігалось зниження продуктивності — оцінка ефективності становила 76/100. Час появи першого вмісту (FCP) дорівнював 2.6 с, тоді як LCP становив 5.2 с. Speed Index — 3.6 с. Проте Total Blocking Time був надзвичайно низьким — лише 10 мс, а CLS залишився рівним 0 (рис. 3.23). Це вказує на хорошу оптимізацію JavaScript, попри необхідність додаткової оптимізації візуального контенту для мобільних пристроїв.

<https://tastyhere-web-304897645235.europe-west3.run.app/>

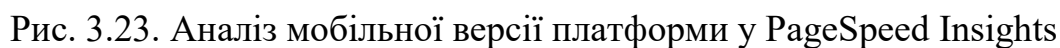
 Мобільний

☐ Комп'ютер

 Дізнайтеся, як відбувається взаємодія в реальних користувачів

ⓘ Немає даних

 Виявляйте проблеми з продуктивністю



Загалом результати тестування підтверджують високу швидкодію та стабільну візуальну поведінку платформи. Особливо варто відзначити нульовий показник CLS у всіх сценаріях, що свідчить про правильну реалізацію адаптивного та доступного інтерфейсу. Певні аспекти, такі як LCP у мобільній версії, можуть бути покращені шляхом додаткової оптимізації графіки та ресурсів.

Висновки до третього розділу

У даному розділі було детально розглянуто реалізацію платформи. Спочатку було описано загальну архітектуру розгортання проєкту, включаючи як вебзастосунок, так і скрапер. Було проаналізовано основні компоненти системи, їхню структуру, взаємозв'язки та особливості взаємодії між ними. Такий підхід дозволяє оцінити технічну складність реалізації та гнучкість платформи з точки зору масштабованості та підтримки.

Далі було представлено інтерфейс користувача. Було детально описано сценарії взаємодії з додатком для різних ролей: звичайних користувачів, зареєстрованих авторів, модераторів і адміністраторів. Наведено приклади навігації, пошуку рецептів, додавання контенту, управління скаргами, тегами та інгредієнтами. Завдяки чіткій структурі інтерфейсу та продуманій організації доступу кожен користувач може інтуїтивно взаємодіяти з системою відповідно до своїх прав і обов'язків.

На завершення було проведено тестування продуктивності додатку за допомогою сервісів GTMetrix та Google PageSpeed Insights. За результатами тестування встановлено, що платформа демонструє високу швидкодію, особливо на десктопних пристроях, із мінімальним часом блокування та нульовими зміщеннями інтерфейсу. Це підтверджує відповідність реалізації сучасним вебстандартам і дозволяє користувачам комфортно взаємодіяти з додатком навіть за умов обмежених ресурсів пристрою.

ВИСНОВКИ

У межах виконання кваліфікаційної роботи бакалавра було спроектовано та реалізовано сервіс формування меню та рецептів страв. У ході роботи проведено аналіз предметної області, який охоплював сучасні підходи до побудови онлайн-сервісів з рецептами, зокрема їхній функціонал, способи взаємодії користувачів і засоби персоналізації вмісту. На основі аналізу сформульовано ключові вимоги до функціоналу та зручності використання платформи.

В рамках проєктування системи розроблено архітектуру вебзастосунку, визначено основні програмні модулі, а також обрано відповідні технології: Next.js для клієнтської та серверної логіки, MongoDB для зберігання даних, а також NextAuth.js для реалізації автентифікації та контролю доступу. Архітектура була реалізована з урахуванням масштабованості, модульності та зручності подальшої підтримки.

Платформа підтримує декілька ролей користувачів, включаючи відвідувачів, зареєстрованих авторів, модераторів і адміністраторів. Реалізовано механізм створення рецептів, їх перегляду, редагування, залишення відгуків, а також пошуку з використанням системи фільтрів, тегів, сортування та інтерфейсних компонентів, що адаптуються до розміру екрана.

Окремо реалізовано функціонал управління вмістом з боку адміністрації — управління тегами, модерація скарг, а також перегляд і редагування списку інгредієнтів та їхніх відповідностей до торгових продуктів. Це дозволяє підтримувати якість контенту та зменшити ризик появи недостовірної інформації.

Проведено докладний опис інтерфейсу, що охоплює основні сценарії використання платформи: пошук рецептів, взаємодію з результатами, перегляд сторінки рецепту, створення власного рецепту, перегляд відгуків, подання скарги тощо. Інтерфейс побудований з урахуванням принципів зручності, швидкої навігації та естетичної привабливості.

Після завершення розробки було проведено тестування продуктивності платформи з використанням сервісів GTMetrix та Google PageSpeed Insights, результати яких підтвердили високий рівень оптимізації клієнтської частини. Зокрема, було досягнуто оцінки Performance 94 % у GTMetrix, а також 94/100 у PageSpeed для десктопної версії. Це підтверджує швидке завантаження контенту, відсутність візуальних зрушень та ефективну обробку запитів.

Таким чином, реалізований вебзастосунок є повноцінним програмним продуктом, що дозволяє зручно працювати з рецептами, має функціональні інтерфейси для кінцевих користувачів та адміністрації, відповідає сучасним вимогам до продуктивності та гнучкості, і є придатною для подальшого розвитку та масштабування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Cookpad [Електронний ресурс] — Режим доступу до ресурсу:
<https://cookpad.com/ua>
2. Allrecipes [Електронний ресурс] — Режим доступу до ресурсу:
<https://www.allrecipes.com/>
3. ЯБпоїла [Електронний ресурс] — Режим доступу до ресурсу:
<https://ua.yabpoela.net/>
4. Де Дешевше [Електронний ресурс] — Режим доступу до ресурсу:
<https://de-deshevshe.com.ua/>
5. Bass, L. Software Architecture in Practice. 3rd ed. Boston : Addison-Wesley, 2012. 640 с.
6. Fowler, M. Patterns of Enterprise Application Architecture. Boston : Addison-Wesley, 2003. 528 с.
7. Newman, S. Building Microservices: Designing Fine-Grained Systems. Sebastopol : O'Reilly Media, 2015. 270 с.
8. Fowler, M. Microservices. Martin Fowler's Bliki. 2014, 25 березня [Електронний ресурс] — Режим доступу до ресурсу:
<https://martinfowler.com/articles/microservices.html>
9. Dragoni, N., Giallorenzo, F., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., & Safina, L. Microservices: Yesterday, Today, and Tomorrow. У Advances in Computers. Vol. 104: Software Development, Design and Verification. Academic Press, 2017. С. 195–236.
10. Kurose, J. F., & Ross, K. W. Computer Networking: A Top-Down Approach. 7th ed. Boston : Pearson, 2017. 912 с.
11. Kimball, R., & Ross, M. The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling. 3rd ed. Indianapolis : Wiley, 2013. 552 с.
12. Inmon, W. H. Building the Data Warehouse. 3rd ed. New York : John Wiley & Sons, 2002. 464 с.
13. Martin, R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design / R. C. Martin. — Addison-Wesley, 2017. — 416 с.

14. Next.js Documentation [Електронний ресурс] — Режим доступу до ресурсу: <https://nextjs.org/docs>
15. TypeScript Documentation [Електронний ресурс] — Режим доступу до ресурсу: <https://www.typescriptlang.org/docs/>
16. Python Documentation [Електронний ресурс] — Режим доступу до ресурсу: <https://docs.python.org/>
17. Twist, L. LLMs Love Python: A Study of LLMs' Bias for Programming Languages and Libraries [Електронний ресурс] / L. Twist, J. M. Zhang, M. Harman, D. Syme, J. Noppen, D. Nauck. — 2025. — Режим доступу до ресурсу: <https://arxiv.org/abs/2503.17181>
18. MongoDB Documentation [Електронний ресурс] — Режим доступу до ресурсу: <https://www.mongodb.com/docs/>
19. Docker Documentation [Електронний ресурс] — Режим доступу до ресурсу: <https://docs.docker.com/>
20. Fowler, M. Continuous Integration. Martin Fowler's Bliki. 2006, 31 серпня [Електронний ресурс] — Режим доступу до ресурсу: <https://martinfowler.com/articles/continuousIntegration.html>
21. Google Cloud Documentation [Електронний ресурс] — Режим доступу до ресурсу: <https://cloud.google.com/docs>
22. Сільпо [Електронний ресурс] — Режим доступу до ресурсу: <https://silpo.ua/>
23. Montgomery, D. C. Applied Statistics and Probability for Engineers / D. C. Montgomery, G. C. Runger. — 7th ed. — Wiley, 2018. — 800 с.
24. NextAuth.js Documentation [Електронний ресурс] — Режим доступу до ресурсу: <https://next-auth.js.org/>
25. Formidable Documentation [Електронний ресурс] — Режим доступу до ресурсу: <https://www.formidablejs.org/docs>
26. OWASP Top 10. OWASP Foundation [Електронний ресурс]. — Режим доступу до ресурсу: <https://owasp.org/www-project-top-10/>

27. The Twelve-Factor App [Электронный ресурс] — Режим доступа до ресурсу: <https://12factor.net/>
28. GTmetrix [Электронный ресурс] — Режим доступа до ресурсу: <https://gtmetrix.com/>
29. Google PageSpeed Insights [Электронный ресурс] — Режим доступа до ресурсу: <https://pagespeed.web.dev/>
30. Core Web Vitals [Электронный ресурс] — Режим доступа до ресурсу: <https://developers.google.com/search/docs/appearance/core-web-vitals>

ДОДАТКИ

					ПЗ.КР.Б-121-25-ПЗ	Арк.
						69

```

def fetch_and_store_products(self):
    logger.info(" початок скрапінгу silpo...")
    cats = self.fetch_categories_tree()
    edible_slugs = self.build_edible_slugs(cats)[0]
    logger.info(f"Знайдено {len(edible_slugs)} істивних
категорій.")

    v1_url = "https://sf-ecom-
api.silpo.ua/v1/uk/branches/00000000-0000-0000-0000-
000000000000/products"
    for slug in edible_slugs:
        logger.info(f" Категорія {slug}")
        filters = self.fetch_filters(slug)

        def make_urls(filter_code=None, filter_val=None):
            params = {"limit": 100, "category": slug,
"includeChildCategories": "false"}
            if filter_code:
                params[f"filters[{filter_code}]"] = filter_val
            total = self._get(f"{v1_url}?{urlencode(params,
quote_via=quote_plus)}").json().get("total", 0)
            pages = math.ceil(total / 100)
            return [
                f"{v1_url}?{urlencode(**params, 'offset': p*100},
quote_via=quote_plus)}"
                for p in range(pages)
            ]

        for f in filters:
            code = f.get("code")
            if code in self.EXCLUDED_FILTERS or not
f.get("values"):
                continue
            for val in f["values"]:
                title = val.get("info", {}).get("title", "").strip()
                if not title or val.get("products", {}).get("total",
0) == 0:
                    continue

```

```

        urls = make_urls(code, val["code"])
        self._process_urls(urls, {code: title})

    if not any(code not in self.EXCLUDED_FILTERS for code in
[f["code"] for f in filters]):
        urls = make_urls()
        self._process_urls(urls, {})

```

Додаток Б

```

def determine_name(raw: dict) -> str | None:
    slug = raw.get('sectionSlug', '')
    det = raw.get('details', {})

    # М'ясо
    if slug in CATEGORY_GROUPS['meat']:
        tip = det.get('backoffice', {}).get('tipsteika',
        '').lower()
        return f"Яловичий стейк {tip}" if tip and tip != 'інші'
    else 'Яловичий стейк'

    # Горіхи та сухофрукти
    if slug in CATEGORY_GROUPS['nuts']:
        return det.get('subspecies') or det.get('maincomposition')
    if slug in CATEGORY_GROUPS['dry_fruits']:
        sub = det.get('subspecies')
        comp = det.get('maincomposition')
        return f"{sub} ({comp})" if sub and comp else sub or comp

    # Фрукти/овочі
    if slug in get_child_categories('frukty-ovochi-4788'):
        comp = next((det.get(key) for key in ('subspecies',
        'maincomposition', 'typeofproduct') if det.get(key)), None)
        if comp:
            return transliterate(slug) + ' мікс' if comp.lower()
            == 'мікс' else comp

    # Яйця
    if slug in CATEGORY_GROUPS['eggs']:
        return 'яйце куряче' if 'kuriachi' in slug else 'яйце
        перепелине'

    # Вина
    if slug in CATEGORY_GROUPS['wines']:
        color = det.get('color', '').replace('ий', 'е')
        taste = det.get('taste') or det.get('tastees',
        '').rstrip('і')
        return f"Вино {color} {taste}" if color and taste else
        None

    # Ковбаси
    if slug in get_child_categories('kovbasni-vyroby-i-m-iasni-
        delikatesy-4731'):
        return next((det.get(key) for key in ('maincomposition',
        'subspecies', 'typeofproduct') if det.get(key)), '')

```

```

# Сири
if slug in CATEGORY_GROUPS['chees']:
    return transliterate(slug)

if slug in get_child_categories('sry-1468'):
    type_of_prod = det.get('typeofproduct')
    sub = det.get('subspecies')
    if sub == "сирний" or not sub:
        return None
    return f"{type_of_prod or 'Сир'} {sub}"

# Хліб та випічка
if slug in get_child_categories("khlіb-ta-vypichka-5121"):
    return transliterate(slug)

# Бакалія та консерви
if slug in get_child_categories("bakaliia-i-konservy-4870"):
    comp = det.get('maincomposition')
    if not comp or comp.lower() in BLACKLIST_NAMES['bakaliia']:
        return None
    if comp.lower() == 'бульйони':
        return 'бульйон'
    if slug == 'inshi-olii-4907' and not ('олія' in
comp.lower() or 'олій' in comp.lower()):
        return f"{comp} (олія)"
    if slug == 'otset-4908' and not ('оцет' in comp.lower()):
        return f"{comp} (оцет)"
    if slug == 'varennia-ta-dzhemy-4921' and not (
'джем' in comp.lower().split() or 'варення' in
comp.lower().split()):
        return f"{comp} (джем / варення)"
    if slug == 'm-iasna-4911' and len(comp.split()) == 1:
        return f"{comp} (консерви)"
    return comp

# Соуси та спеції
if slug in CATEGORY_GROUPS['spices'] or slug in
CATEGORY_GROUPS['souses']:
    sub = det.get('subspecies')
    if slug in CATEGORY_GROUPS['spices']:
        transliterated = transliterate(slug).replace('сіл',
'сіль')
    else:
        transliterated =
transliterate(slug).rstrip('ийиююі').replace("заправк",
'заправка')

    if sub:
        if sub in BLACKLIST_NAMES['souses_and_spices']:
            return None
        if transliterated not in sub.lower():
            return f"{transliterated} {sub}"
        else:
            return sub
    return transliterated

# Все для випічки
if slug in CATEGORY_GROUPS['bakery']:

```

```

mc = det.get('maincomposition')
at = det.get('addtaste')
if mc and at:
    return f"{mc} ({at})"

sub = det.get('subspecies')
if sub in BLACKLIST_NAMES['souses_and_spices']:
    return None
if sub:
    return sub

if mc:
    return mc

return transliterate(slug)

# Напої
if slug in CATEGORY_GROUPS['drink']:
    record = det
    taste = record.get('taste', '').lower()
    base = transliterate(slug)
    if taste and taste not in ('без добавок', 'з добавками'):
        if any(group in base for group in ('соки', 'нектари',
        'соковмісні', 'морси')):
            return f"{base} ({taste})"
        return base

    return base

# Загальний випадок
comp = next((det.get(key) for key in ('maincomposition',
'subspecies', 'typeofproduct') if det.get(key)), '')
return comp if comp.lower() not in ('інші', 'асорті', '')
else None

```

Додаток В

```

UNIT_MAP = {
    'г': ('grams', 1), 'кг': ('grams', 1000), 'мл':
    ('milliliters', 1), 'л': ('milliliters', 1000),
    'дм': ('milliliters', 1000), 'літр': ('milliliters', 1000),
    'шт': ('pieces', 1), 'уп': ('pieces', 1),
    'пучок': ('grams', 50), 'пляшка': ('milliliters', 1000),
    'бух': ('grams', 500), 'м': ('centimeters', 100)
}
UNIT_REGEX = re.compile(r'(?P<value>[\d.,]+)\s*(?P<unit>[а-яА-
я]+)')

def parse_ratio(ratio: str) -> tuple[float, str] | None:
    match = UNIT_REGEX.search(ratio)
    if not match:
        return None
    raw_value = float(match.group('value').replace(',', '.'))
    unit_key = match.group('unit')
    if unit_key not in UNIT_MAP:
        return None

```

```
unit_label, factor = UNIT_MAP[unit_key]
return raw_value * factor, unit_label
```

Додаток Г

```
def compute_prices(price_list: list[float]) -> dict[str, float]:
    arr = np.array(price_list)
    q1, q3 = np.percentile(arr, [25, 75])
    iqr = q3 - q1
    inliers = arr[(arr >= q1 - 1.5 * iqr) & (arr <= q3 + 1.5 *
iqr)]
    avg = float(np.mean(inliers)) if len(inliers) > 0 else
float(np.mean(arr))
    return {'min': float(np.min(arr)), 'max': float(np.max(arr)),
'avg': avg}
```

Додаток Д

```
export function withRoles<P extends object>(
  handler: GetServerSideProps<P>,
  options: WithRolesOptions = {}
): GetServerSideProps<P & { user: UserInSession }> {
  return async (
    ctx: GetServerSidePropsContext
  ): Promise<
    GetServerSidePropsResult<P & { user: UserInSession }>
  > => {
    const session = await getSession(ctx.req, ctx.res,
authOptions)
    if (!session) {
      return {
        redirect: { destination: '/auth/login', permanent: false }
      }
    }
    const userRole = session.user.role as Role
    if (
      options.roles &&
      options.roles.length > 0 &&
      !options.roles.includes(userRole)
    ) {
      return { notFound: true }
    }

    const result = await handler(ctx)

    if ('redirect' in result || 'notFound' in result) {
      return result as any
    }
    const originalProps = result.props
    return {
      props: {
        ...originalProps,
        user: {
          email: session.user.email!,
          nickname: session.user.name ?? null,

```

```

        id: session.user.id!,
        role: session.user.role as Role
    }
  }
}
}
}

```

Додаток Е

```

import type { NextApiRequest, NextApiResponse } from 'next'
import dbConnect from '@common/lib/mongodb'
import Recipe from '@common/models/Recipe'
import { getServerSession } from 'next-auth/next'
import { authOptions } from '@pages/api/auth/[...nextauth]'
import formidable, { File as FormidableFile } from 'formidable'
import { uploadFileToGCS, uploadFilesToGCS, deleteFileFromGCS }
from '@common/helpers/fileUpload'
import { Difficulty } from '@types/IRecipe'

export const config = {
  api: { bodyParser: false }
}

type Data = { _id?: string; message?: string }

function extractField(f: any): string {
  return Array.isArray(f) ? f[0] : f
}

export default async function handler(
  req: NextApiRequest,
  res: NextApiResponse<Data>
) {
  await dbConnect()

  const { id } = req.query
  const session = await getServerSession(req, res, authOptions as any)
  if (!session) return res.status(401).json({ message: 'Не авторизовано' })

  const existing = await Recipe.findById(id).exec()
  if (!existing) return res.status(404).json({ message: 'Не знайдено' })
  const isOwner = existing.author.toString() === session.user.id
  const isPrivileged =
    ['moderator', 'admin'].includes(session.user.role!)
  if (!isOwner && !isPrivileged) {
    return res.status(403).json({ message: 'Немає прав' })
  }

  if (req.method !== 'PUT') {
    res.setHeader('Allow', ['PUT'])
    return res.status(405).end(`Метод ${req.method} не дозволений`)
  }

  const form = formidable({ multiples: true, keepExtensions: true })

```

```

form.parse(req, async (err, fields, files) => {
  if (err) {
    console.error(err)
    return res.status(500).json({ message: 'Помилка парсингу форми' })
  }
  try {
    let newMain = existing.mainPhoto
    const rawMain = files.mainPhoto as FormidableFile | FormidableFile[] | undefined
    if (rawMain) {
      const file = Array.isArray(rawMain) ? rawMain[0] : rawMain
      newMain = await uploadFileToGCS(file, 'main-')
      if (existing.mainPhoto) {
        await deleteFileFromGCS(existing.mainPhoto)
      }
    }
    const ingArr = JSON.parse(extractField(fields.ingredients || '[]'))
    const ingredients = ingArr.map((i: any) => ({
      ingredient: i.ingredient,
      unit: i.unit,
      quantity: Number(i.quantity),
    }))
    const tags = JSON.parse(extractField(fields.tags || '[]'))

    const oldStepPhotos = existing.steps.map(s => s.photo).filter(Boolean) as string[]

    const stepsMeta = JSON.parse(extractField(fields.steps || '[]'))
    const rawStepFiles = files.stepPhotos as FormidableFile | FormidableFile[] | undefined
    const stepFiles = rawStepFiles
      ? (Array.isArray(rawStepFiles) ? rawStepFiles : [rawStepFiles])
      : []

    const newBlobNames = await uploadFilesToGCS(stepFiles, 'step-')

    const newStepPhotos: string[] = []
    const steps = stepsMeta.map((s: any, idx: number) => {
      const photo = newBlobNames[idx] || s.photoFilename || null
      if (photo) newStepPhotos.push(photo)
      return {
        number: s.number,
        description: s.description,
        photo,
      }
    })

    for (const old of oldStepPhotos) {
      if (!newStepPhotos.includes(old)) {
        await deleteFileFromGCS(old)
      }
    }
  }
})

```

```

const title = extractField(fields.title)
const description = extractField(fields.description)
const difficulty = extractField(fields.difficulty) as
Difficulty
const prepTime = Number(extractField(fields.prepTime))
const servings = Number(extractField(fields.servings))
if (!title || !difficulty || !ingredients.length
|| !steps.length || !prepTime || !servings) {
    return res.status(400).json({ message: 'Неповні дані' })
}

const updated = await Recipe.findByIdAndUpdate(
    id,
    {
        title,
        description,
        difficulty,
        prepTime,
        servings,
        mainPhoto: newMain,
        tags,
        ingredients,
        steps,
    },
    { new: true }
).exec()

return res.status(200).json({ _id:
updated!._id.toString() })
} catch (e: any) {
    console.error(e)
    return res.status(500).json({ message: e.message ||
'Помилка сервера' })
}
}
}

```

Додаток Ж

```

export const attachPrice: PipelineStage = {
    $addFields: {
        price: {
            avg: {
                $sum: {
                    $map: {
                        input: '$ingredients',
                        as: 'ing',
                        in: {
                            $multiply: [
                                '$$ing.quantity',
                                {
                                    $let: {
                                        vars: {
                                            det: {
                                                $arrayElemAt: [
                                                    {
                                                        $filter: {
                                                            input: '$ingredientDetails',
                                                            as: 'detail',

```

```
cond: { $eq: ['$detail._id', '$$ing.ingredient'] },
    },
    ],
    },
    in: {
      $getField: {
        field: {
          $concat: ['avg_', '$$ing.unit']
        },
        input: '$$det.prices'
      }
    }
  },
],
},
},
min: {
  $sum: {
    $map: {
      input: '$ingredients',
      as: 'ing',
      in: {
        $multiply: [
          '$$ing.quantity',
          {
            $let: {
              vars: {
                det: {
                  $arrayElemAt: [
                    '$$det.details',
                    '$_id'
                  ],
                  $filter: {
                    input: '$ingredientDetails',
                    as: 'detail',
                    cond: { $eq: ['$detail._id', '$$ing.ingredient'] },
                  }
                }
              },
              in: {
                $getField: {
                  field: {
                    $concat: ['min_', '$$ing.unit']
                  },
                  input: '$$det.prices'
                }
              }
            }
          ]
        }
      }
    }
  }
}
```

					ИПЗ.КР.Б-121-25-ПЗ	Арк.
						78

```

    },
    },
    },
    },
    max: {
      $sum: {
        $map: {
          input: '$ingredients',
          as: 'ing',
          in: {
            $multiply: [
              '$$ing.quantity',
              {
                $let: {
                  vars: {
                    det: {
                      $arrayElemAt: [
                        {
                          $filter: {
                            input: '$ingredientDetails',
                            as: 'detail',
                            cond: { $eq: ['$$detail._id',
                              '$$ing.ingredient'] },
                        },
                        0,
                      ],
                    },
                  },
                  in: {
                    $getField: {
                      field: {
                        $concat: ['max_', '$$ing.unit']
                      },
                      input: '$$det.prices'
                    }
                  }
                }
              },
            ],
          },
        },
      },
    },
  },
},
};

```

Додаток И

```

import React, {FC, useEffect, useState} from 'react';
import {useRouter} from "next/navigation";
import DifficultySelector from
"@/components/RecipecreateForm/DifficultySelector";
import {IOption} from "@/types/IOption";
import TextInput from "@/components/UI/TextInput";
import styles from "../RecipecreateForm.module.css";
import {MdOutlineAccessTime} from "react-icons/md";

```

```

import {BiBowlingHot} from "react-icons/bi";
import TextInputLong from "@components/UI/TextInputLong";
import SubmitButton from "@components/UI/SubmitButton";
import IngredientAddSection from
"@components/RecipecreateForm/IngredientAddSection";
import TagAddSection from
"@components/RecipecreateForm/TagAddSection";
import ImageLoader from "@components/UI/ImageLoader";
import StepAddSection from
"@components/RecipecreateForm/StepAddSection";
import {convertToBaseUnit, convertToBaseUnitName} from
"@common/helpers/unitsConvert";
import FuncButton from "@components/UI/FuncButton";
import {Difficulty, IRecipeDto} from "@types/IRecipe";
import {IngredientSelection, IRecipeIngredientCreatedDto,
IRecipeStepCreatedDto} from "@types/ICreatedDto";
import {UnitsFull} from "@types/Ingredient";

interface Props {
  mode?: 'create' | 'edit'
  initialData?: IRecipeDto
}

const RecipecreateForm: FC<Props> = ({mode = 'create',
initialData}) => {
  const router = useRouter()

  const [recipe, setRecipe] = useState({
    title: initialData?.title ?? '',
    description: initialData?.description ?? '',
    difficulty: initialData?.difficulty ?? 'easy' as Difficulty,
    prepTime: initialData?.prepTime ?? 0,
    servings: initialData?.servings ?? 0,
  });

  const [numberValues, setNumberValues] = useState({
    prepTime: initialData?.prepTime.toString() ?? '',
    servings: initialData?.servings.toString() ?? '',
  })

  const [errors, setErrors] = useState({
    title: '',
    description: '',
    prepTime: '',
    servings: '',
    mainPhoto: '',
    ingredients: '',
    steps: '',
    tags: '',
  });

  const [mainPhotoFile, setMainPhotoFile] = useState<File |
null>(null);
  const [mainPhotoPreview, setMainPhotoPreview] =
useState<string>(initialData?.mainPhoto || '')
  const [selectedIngredientOptions, setSelectedIngredientOptions]
= useState<IngredientSelection[]>(
  initialData?.ingredients.map(item => ({

```

```

        ingredient: { _id: item.ingredient._id, name:
item.ingredient.name },
        unit: item.unit,
        unitCount: item.quantity
    } as IngredientSelection)) || []
    )
    const [selectedTagOptions, setSelectedTagOptions] =
useState<IOption[]>(initialData?.tags || [])
    const [stepList, setStepList] =
useState<IRecipeStepCreatedDto[]>(initialData?.steps.map(item =>
({
    number: item.number,
    description: item.description,
    photoUrl: item.photo
  } as IRecipeStepCreatedDto)) || []);

    useEffect(() => {
        if (mode === 'edit' && initialData) {
            setRecipe({
                title: initialData.title,
                description: initialData.description,
                difficulty: initialData.difficulty,
                prepTime: initialData.prepTime,
                servings: initialData.servings,
            })
            setSelectedTagOptions(initialData.tags)
            setSelectedIngredientOptions(initialData.ingredients.map(i
=> ({
                ingredient: { _id: i.ingredient._id, name:
i.ingredient.name },
                unit: { _id: i.unit, name: i.unit },
                unitCount: i.quantity,
            })))
            setStepList(initialData.steps.map(s => ({
                number: s.number,
                description: s.description,
                photoUrl: s.photo,
            })))
            setMainPhotoPreview(initialData.mainPhoto)
        }
    }, [mode, initialData])

    const handleTitleChange = (v: string) => setRecipe(r => ({...r,
title: v}));
    const handleDescriptionChange = (v: string) => setRecipe(r =>
({...r, description: v}));
    const handleDifficultyChange = (v: Difficulty) => setRecipe(r
=> ({...r, difficulty: v}));
    const handleTimeChange = (v: string) => {
        setNumberValues(prevState => ({...prevState, prepTime: v}))
        setRecipe(r => ({...r, prepTime: parseInt(v) || 0}))
    };
    const handleServingsChange = (v: string) => {
        setNumberValues(prevState => ({...prevState, servings: v}))
        setRecipe(r => ({...r, servings: parseInt(v) || 1}))
    };

    const handleSubmit = async (e: React.FormEvent) => {

```

```

e.preventDefault();

const newErrors = {
  title: recipe.title ? '' : 'Введіть назву.',
  description: recipe.description ? '' : 'Введіть опис.',
  prepTime: recipe.prepTime > 0 ? '' : 'Введіть час
приготування.',
  servings: recipe.servings > 0 ? '' : 'Введіть кількість
порцій.',
  mainPhoto: mainPhotoFile || mainPhotoPreview ? '' :
'Оберіть головне фото.',
  ingredients: selectedIngredientOptions.length ? '' :
'Оберіть інгредієнти.',
  tags: selectedTagOptions.length ? '' : 'Оберіть теги.',
  steps: stepList.length ? '' : 'Додайте кроки приготування.',
};
setErrors(newErrors);

if (Object.values(newErrors).some(msg => msg)) {
  return;
}

const formData = new FormData();
if(mainPhotoFile) formData.append('mainPhoto', mainPhotoFile);
formData.append('title', recipe.title);
formData.append('description', recipe.description);
formData.append('difficulty', recipe.difficulty);
formData.append('prepTime', recipe.prepTime.toString());
formData.append('servings', recipe.servings.toString());
formData.append('tags',
JSON.stringify(selectedTagOptions.map(t => t._id)));
formData.append('ingredients',
JSON.stringify(selectedIngredientOptions.map(item => ({
  unit: convertToBaseUnitName(item.unit._id as UnitsFull),
  ingredient: item.ingredient._id,
  quantity: convertToBaseUnit(item.unitCount, item.unit._id
as UnitsFull)
} as IRecipeIngredientCreatedDto))));

stepList.forEach((step, idx) => {
  if (step.photoFile) {
    formData.append('stepPhotos', step.photoFile, `step${idx +
1}_${step.photoFile.name}`);
  }
});
formData.append('steps', JSON.stringify(
stepList.map((s, idx) => ({
  number: s.number,
  description: s.description,
  photoFilename: s.photoFile ? `step${idx +
1}_${s.photoFile.name}` : null,
})))
));

const url = mode === 'edit' ?
`/api/recipes/${initialData!._id}/edit` : '/api/recipes/create'
const method = mode === 'edit' ? 'PUT' : 'POST'

```

```

const res = await fetch(url, {
  method,
  body: formData,
} as RequestInit);

if (res.ok) {
  const {_id} = await res.json();
  router.push(`/recipes/${_id}`);
} else {
  const err = await res.json();
  alert('Помилка: ' + (err.message || res.statusText));
}
};

return (
  <>
    <form onSubmit={handleSubmit} className={styles.form}>
      <section>
        <TextInput
          type="text"
          value={recipe.title}
          placeholder="Назва страви"
          onChangeHandler={handleTitleChange}
          errorMessage={errors.title}
        />
      </section>
      <section>
        <ImageLoader
          previewUrl={mainPhotoPreview}
          onChange={file => setMainPhotoFile(file)}
        />
        {errors.mainPhoto && <p className={'error-
message'}>{errors.mainPhoto}</p>}
      </section>
      <section>
        <TagAddSection
          values={selectedTagOptions}
          setValues={setSelectedTagOptions}
        />
        {errors.tags && <p className={'error-
message'}>{errors.tags}</p>}
      </section>
      <section className={styles.row}>
        <div>
          <TextInput errorMessage={errors.prepTime}
            type={"number"} value={numberValues.prepTime} placeholder={"Час
(хв)"}
            onChangeHandler={handleTimeChange}>
            <div><MdOutlineAccessTime/></div>
          </TextInput>
        </div>
        <div>
          <TextInput errorMessage={errors.servings}
            type={"number"} value={numberValues.servings}
            placeholder={"Порції"}
            onChangeHandler={handleServingsChange}>
            <div><BiBowlHot/></div>
          </TextInput>
        </div>
      </section>
    </form>
  </>
);

```

```

    </div>
  </section>
  <section>
    <label>Складність</label>
    <DifficultySelector
      value={recipe.difficulty}
      onChange={handleDifficultyChange}
    />
  </section>
  <section>
    <TextInputLong
      errorMessage={errors.description}
      value={recipe.description}
      placeholder="Опис"
      onChangeHandler={handleDescriptionChange}
    />
  </section>
  <section>
    <IngredientAddSection
      values={selectedIngredientOptions}
      setValues={setSelectedIngredientOptions}
    />
    {errors.ingredients && <p className='error-
message'>{errors.ingredients}</p>}}
  </section>
  <section>
    <StepAddSection stepList={stepList}
    setStepList={setStepList}/>
    {errors.steps && <p className='error-
message'>{errors.steps}</p>}}
  </section>
  <section>
    {Object.values(errors).some(msg => msg) && <p
className='error-message'>Введено не всі дані</p>}}
    <SubmitButton>{mode == 'edit' ? 'Оновити' : 'Створити'}
рецепт</SubmitButton>
    {
      mode == 'edit' && !!initialData &&
      <FuncButton
        variant='cancel'>
      </FuncButton>
      Скасувати
    }
  </section>
</form>
</>
);
};

export default RecipeCreateForm;

```