МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ЕКОНОМІКО-ПРАВНИЧИЙ ФАХОВИЙ КОЛЕДЖ ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ»

Циклова комісія математичних дисциплін та інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «РОЗРОБКА ВЕБЗАСТОСУНКУ ДЛЯ АВТОМАТИЗОВАНОГО ПІДБОРУ МЕНЮ З УРАХУВАННЯМ ВПОДОБАНЬ КОРИСТУВАЧА»

Виконав: здобувач освіти 4 курсу, групи К121-22
Спеціальність 121 Інженерія програмного забезпечення
(шифр і спеціальність)
Андрій БАРАБІН
(ім'я та ПРІЗВИЩЕ)
Керівник Роман КОТОВ
(ім'я та ПРІЗВИЩЕ)
Рецензент доцент кафедри «Інформаційна безпека та наноелектроніка» НУ «Запорізька політехніка», к.т.н., доцент Ганна НЕЛАСА
(посада, вчене звання, науковий ступінь, ім'я та ПРІЗВИЩЕ)

Запоріжжя
2026

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ВСП «Економіко-правничий фаховий коледж ЗНУ»

Освітньо–кваліфікаційний ступінь фаховий молодший бакалавр

Спеціальність 121 – Інженерія програмного забезпечення
(шифр і назва)

ЗАТВЕРДЖУЮ

Голова циклової комісії математичних дисциплін та інформаційних технологій

_____ Т.М. Смолянкова
(підпис)

“ 05 ” червня 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

БАРАБІНУ Андрію Денисовичу
(ПРІЗВИЩЕ, ім’я та по– батькові)

1. Тема роботи «Розробка вебзастосунку для автоматизованого підбору меню з урахуванням вподобань користувача»

Керівник роботи КОТОВ Роман Олександрович
(науковий ступінь, вчене звання, прізвище, ім’я, по батькові)

Затверджені наказом ВСП ЕПФК ЗНУ від «01» грудня 2025 р. № 1728-с

2. Строк подання студентом роботи 05.06.2026

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково– пояснювальної записки (перелік питань, які потрібно розробити)
1. Аналіз предметної області та вибір стеку технологій
2. Проєктування бази даних та архітектури системи
3. Програмна реалізація вебдодатка

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація до захисту

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 01.12.2025

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи:		
	постановка задачі	Грудень 2025	виконано
2.	Збір вихідних даних, обробка методичних		
	та теоретичних джерел	Січень 2026	виконано
3.	Розробка першого розділу:		
	огляд інформаційних систем та технологій для їхнього створення	Лютий 2026	виконано
4.	Розробка другого розділу:		
	розробка проєкту інформаційної системи та вибір технологій	Квітень 2026	виконано
5.	Розробка третього розділу:		
	розробка застосунку	Травень 2026	виконано
6.	Оформлення і нормоконтроль кваліфікаційної роботи та перевірка на плагіат	Червень 2026	виконано
7.	Захист кваліфікаційної роботи	16.06.2026	виконано

Здобувач освіти _____ Андрій БАРАБІН
(підпис) (ім’я та ПРІЗВИЩЕ)

Керівник роботи _____ Роман КОТОВ
(підпис) (ім’я та ПРІЗВИЩЕ)

Нормоконтроль пройдено

Нормоконтролер _____ Юлія БОРИСОВСЬКА
(підпис) (ім’я та ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка: 58 сторінок, 7 таблиць, 20 рисунків, 12 джерел, 3 додатки.

Об'єкт дослідження: процес автоматизованого планування харчування в домогосподарствах.

Предмет дослідження: методи та засоби розробки вебзастосунку з інтеграцією локальної мовної моделі та геоінформаційних сервісів для підбору меню за вподобаннями користувача.

Мета роботи: розробити вебзастосунок Smart Scout Menu, що автоматично формує тижневе меню за дієтичним профілем користувача, будує список покупок і знаходить найближчі продовольчі магазини.

Методи дослідження: аналіз існуючих аналогів за функціональними критеріями, методи структурного та об'єктного проєктування програмного забезпечення, прототипування інтерфейсу, юніт-тестування (Vitest) та тестування безпеки API власним сценарієм. Використано локальну мовну модель phi-4-mini-instruct через сервер LM Studio і дані OpenStreetMap через Overpass API.

У результаті розробки отримано повнофункціональний вебзастосунок з 37 маршрутами React Router, 16 незалежними route-модулями API, базою з 9 колекцій MongoDB і JSON-fallback шаром. Реалізовано всі сім функціональних вимог: JWT-автентифікацію, генерацію меню локальною LLM, управління рецептами з конвертацією зображень у WebP, пошук магазинів через OSM, кошик і списки покупок, аналітику бюджету та нутрієнтів (Recharts), мультимовний інтерфейс (i18next, EN/CS/UK). Тестування підтвердило 0 помилок TypeScript-компіляції, проходження 5 перевірок безпеки (NoSQL injection, path traversal, JWT tampering, rate limiting, CORS) та стабільну роботу в режимах MongoDB і JSON-fallback.

Рекомендації щодо застосування: розгортання у домогосподарствах для скорочення часу щоденного планування харчування і контролю витрат на продукти; використання в навчальному процесі як приклад інтеграції локальної LLM з вебдодатком.

Значимість роботи: запропоновано і реалізовано рішення, що поєднує генерацію меню локальною мовною моделлю (без платних API), геоінтеграцію магазинів через OpenStreetMap і бюджетну аналітику. Жоден з існуючих аналогів такого поєднання не має.

ВЕБЗАСТОСУНОК, ПЛАНУВАННЯ ХАРЧУВАННЯ, ШТУЧНИЙ ІНТЕЛЕКТ, REACT, TYPESCRIPT, EXPRESS.JS, MONGODB, REST API, JWT, ПРОГРЕСИВНИЙ ВЕБДОДАТОК, OPENSTREETMAP, МОВНА МОДЕЛЬ

SUMMARY

Technical documentation: 58 pages, 7 tables, 20 figures, 12 sources, 3 appendices.

Object of research: the process of automated meal planning in households.

Subject of research: methods and tools for developing a web application integrating a local language model and geoinformation services for menu selection based on user preferences.

Purpose of the work: to develop the Smart Scout Menu web application that automatically generates a weekly menu based on the user's dietary profile, builds a shopping list, and locates the nearest grocery stores.

Research methods: analysis of existing analogues by functional criteria; methods of structural and object-oriented software design; interface prototyping; unit testing (Vitest); and API security testing using a custom test suite. A local language model phi-4-mini-instruct was used via the LM Studio server, and OpenStreetMap data was accessed via the Overpass API.

As a result of the development, a fully functional web application was obtained, comprising 37 React Router routes, 16 independent API route modules, a database of 9 MongoDB collections, and a JSON fallback layer. All seven functional requirements were implemented: JWT authentication, menu generation using a local LLM, recipe management with WebP image conversion, store search via OSM, shopping cart and lists, budget and nutrient analytics (Recharts), and a multilingual interface (i18next, EN/CS/UK). Testing confirmed zero TypeScript compilation errors, successful passing of five security checks (NoSQL injection, path traversal, JWT tampering, rate limiting, CORS), and stable operation in both MongoDB and JSON fallback modes.

Application recommendations: deployment in households to reduce daily meal planning time and control grocery expenses; use in educational settings as an example of integrating a local LLM with a web application.

Significance of the work: a solution was proposed and implemented that combines menu generation using a local language model (without paid APIs), geolocation of stores via OpenStreetMap, and budget analytics. None of the existing analogues offers such a combination of features.

WEB APPLICATION, MEAL PLANNING, ARTIFICIAL INTELLIGENCE, REACT, TYPESCRIPT, EXPRESS.JS, MONGODB, REST API, JWT, PROGRESSIVE WEB APP, OPENSTREETMAP, LANGUAGE MODE

ЗМІСТ

Завдання на кваліфікаційну роботу студенту 2
Реферат 4
Summary 6
Перелік умовних позначень, символів, одиниць, скорочень і термінів 9
Вступ 10
1 Основи розробки вебзастосунку 12
1.1 Поняття автоматизованих систем 12
1.2 Огляд існуючих аналогів 13
1.3 Структура і функціональні можливості існуючих рішень 15
1.4 Висновки до розділу 1 16
2 Проєктування автоматизованої системи 18
2.1 Визначення технічних вимог 18
2.2 Визначення інструментів розробки 20
2.3 Проєктування інтерфейсу та макетів 23
2.4 Проєктування функціональної структури вебзастосунку 23
2.5 Проєктування структури даних 26
2.6 Висновки до розділу 2 28
3 Розробка автоматизованої системи 30
3.1 Програмна реалізація вебзастосунку 30
3.2 Результати розробки 35
3.3 Тестування вебзастосунку 40
3.4 Висновки до розділу 3 42
Висновки 44
Перелік використаних джерел 47
Додаток А 48
Додаток Б 50
Додаток В 56

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

AI (ШІ) – Artificial Intelligence – штучний інтелект

API – Application Programming Interface – програмний інтерфейс застосунку

CORS – Cross-Origin Resource Sharing – спільне використання ресурсів між джерелами

CRUD – Create, Read, Update, Delete – створення, читання, оновлення, видалення

CSP – Content Security Policy – політика безпеки контенту

HMR – Hot Module Replacement – гаряча заміна модулів

HTTP – HyperText Transfer Protocol – протокол передавання гіпертексту

HTTPS – HTTP Secure – захищений HTTP

JSON – JavaScript Object Notation – формат обміну даними

JWT – JSON Web Token – токен веброзпізнавання

КБЖВ – калорії, білки, жири, вуглеводи

LLM – Large Language Model – велика мовна модель

MongoDB – документоорієнтована СУБД

ORM – Object-Relational Mapping – об'єктно-реляційне відображення

OSM – OpenStreetMap – відкрита картографічна платформа

PWA – Progressive Web Application – прогресивний вебдодаток

REST – Representational State Transfer – архітектурний стиль API

SPA – Single-Page Application – односторінковий застосунок

СУБД – Система управління базами даних

TypeScript – мова програмування з типізацією поверх JavaScript

UI – User Interface – інтерфейс користувача

UUID – Universally Unique Identifier – універсальний унікальний ідентифікатор

UX – User Experience – досвід користувача

ВСТУП

Питання щоденного планування харчування у середньостатистичному домогосподарстві займає чималу частину вечора і часто завершується спонтанною покупкою. Це водночас бюджетне і когнітивне навантаження. Відсутність структурованого тижневого плану помітно збільшує витрати домогосподарства на продукти харчування порівняно з покупками за заздалегідь складеним списком. Поряд із прямою фінансовою втратою стихійне харчування підвищує частку оброблених продуктів у раціоні.

Smart Scout Menu, вебзастосунок для автоматизованого підбору персонального меню. Застосунок аналізує дієтичний профіль користувача (тип харчування, алергени, цільова калорійність, розмір родини, тижневий бюджет) і через локальну мовну модель phi-4-mini-instruct у сервері LM Studio формує меню на 1–7 днів. Паралельно застосунок знаходить продовольчі магазини в межах 10 км через OpenStreetMap Overpass API і будує список покупок з прив'язкою до знайдених точок продажу. Підсистема бюджетної аналітики відстежує витрати на харчування у розрізі тижнів і місяців.

Актуальність теми зумовлена трьома чинниками. Локальні мовні моделі стали доступними для запуску на побутовому обладнанні без оплати зовнішніх API. OpenStreetMap покриває геодані 195 країн і доступний без ліцензійних відрахувань. Третій чинник, попит на персоналізовані рішення для управління харчуванням, що раніше задовольнявся переважно платними мобільними застосунками із закритою архітектурою. Розробка вебзастосунку на стику цих технологій становить інженерний інтерес і відповідає рівню кваліфікаційної роботи фахового молодшого бакалавра.

Мета роботи: розробити вебзастосунок Smart Scout Menu, що автоматично формує тижневе меню за профілем користувача, будує список покупок і знаходить найближчі магазини.

Для досягнення мети виконано такі завдання:

– проаналізовано поняття автоматизованих систем та здійснено огляд трьох існуючих аналогів;

– визначено технічні вимоги до застосунку;

– обґрунтовано вибір інструментів розробки;

– спроєктовано інтерфейс, функціональну структуру і структуру даних вебзастосунку;

– реалізовано програмну частину вебзастосунку;

– проведено тестування та перевірку результатів розробки.

Об'єкт дослідження: процес автоматизованого планування харчування в домогосподарствах. Предмет дослідження: методи та засоби розробки вебзастосунку з інтеграцією локальної мовної моделі та геоінформаційних сервісів. Практичне значення роботи: готовий до розгортання застосунок, що скорочує час щоденного планування їжі і допомагає контролювати харчові витрати без підписок на платні сервіси.

1 ОСНОВИ РОЗРОБКИ ВЕБЗАСТОСУНКУ

Перший розділ є теоретичним підґрунтям роботи. У ньому з'ясовано поняття «автоматизована система», встановлено клас Smart Scout Menu за чинними класифікаціями та проаналізовано три існуючі аналоги. Результати порівняння обґрунтовують доцільність власної розробки і визначають функціональні прогалини, які усуває Smart Scout Menu.

1.1 Поняття автоматизованих систем

У підрозділі визначено поняття «автоматизована система» (АС), наведено класифікацію за функціональним призначенням і розміщено Smart Scout Menu у цій класифікації. Уточнення класу системи потрібне для коректного вибору архітектурних рішень у наступних розділах.

Автоматизована система, це організаційно-технічна система, що забезпечує вироблення рішень на основі автоматизації інформаційних процесів у певній предметній галузі. Визначення містить три істотні ознаки: наявність технічних засобів обробки інформації, програмного забезпечення з певним алгоритмом, і людини як суб'єкта управління або кінцевого користувача. Тобто АС, не просто програма, а сукупність засобів, що інтегровані для досягнення прикладної мети.

За функціональним призначенням АС поділяють на кілька класів:

– системи управління технологічними процесами (АСУТП), забезпечують автоматичне керування обладнанням у реальному часі;

– інформаційно-пошукові системи, здійснюють накопичення та пошук інформації за запитами користувача;

– системи підтримки прийняття рішень (СППР), аналізують вхідні дані і формують рекомендації на основі моделей, правил або алгоритмів машинного навчання;

– системи автоматизованого проєктування (САПР), автоматизують інженерне проєктування і розрахунки;

– навчальні та експертні системи, зберігають знання фахівців і відтворюють їх для вирішення нетривіальних задач у вузькій галузі.

Smart Scout Menu має ознаки двох класів. З одного боку, застосунок зберігає і шукає рецепти, магазини і дані OpenStreetMap, що відповідає функціям інформаційно-пошукової системи. З іншого боку, локальна мовна модель phi-4-mini-instruct аналізує профіль користувача і формує рекомендоване меню з урахуванням дієтичних обмежень, бюджету і розміру родини, що є типовою функцією СППР. Така гібридна природа є характерною для сучасних застосунків, що поєднують бази даних із елементами штучного інтелекту.

Встановлено, що Smart Scout Menu належить до класу інформаційно-аналітичних АС, ближче до СППР з функціями інформаційного пошуку та інтеграції геоданих. Цей висновок впливає на архітектурні рішення наступних розділів: вибір документоорієнтованої СУБД (документи зручніші для гнучкого зберігання меню різної структури), використання асинхронних викликів до зовнішнього сервісу LLM і кешування результатів геоінформаційних запитів.

1.2 Огляд існуючих аналогів

Перед розробкою власної системи виконано огляд трьох застосунків аналогічного призначення. Огляд дозволяє виявити функції, які варто запозичити, і прогалини, які власна розробка має усунути. Аналоги обрано за

критерієм впізнаваності у відповідній ніші: усі три фігурують у топ-10 видач англомовних оглядів планувальників харчування за 2024 рік.

Mealime, мобільний застосунок для планування харчування, орієнтований на зайнятих людей, які готують удома. Доступний на iOS і Android. Базова версія безкоштовна, повна функціональність відкривається після оплати Pro-підписки вартістю 5,99 USD на місяць. Ключові функції: генерація тижневого меню на основі обраних рецептів, автоматичний список покупок, фільтри за алергенами і дієтичними типами (вегетаріанське, веганське, кето, палео тощо). У базі застосунку понад 1000 рецептів із покроковими інструкціями, передбачена синхронізація між пристроями. Mealime має зручний мобільний інтерфейс і просту процедуру першого налаштування, користувач за п'ять хвилин формує початковий профіль. Серед недоліків: не інтегрується з реальними магазинами і не показує поточні ціни, не має бюджетного трекера, генерація меню штучним інтелектом відсутня, користувач складає меню вручну з готового каталогу за допомогою фільтрів. Для повного доступу до функцій застосунок є платним.

Eat This Much, вебзастосунок і мобільний додаток для автоматичного планування харчування з урахуванням КБЖВ (калорії, білки, жири, вуглеводи). Цільова аудиторія, люди, що стежать за фігурою та дієтою. Основна особливість, точний розрахунок макронутрієнтів. Користувач задає цільову калорійність і розподіл нутрієнтів, після чого застосунок автоматично будує денне або тижневе меню з потраплянням у задані діапазони. Підтримується понад 10 дієтичних типів, у тому числі кето, низьковуглеводна і безглютенова. Реалізовано облік кількості порцій і розміру домогосподарства. Інтеграцію зі списком покупок виконано на рівні переліку інгредієнтів. Платна версія коштує 8,99 USD на місяць. Серед недоліків: відсутня інтеграція з геоданими і реальними магазинами, немає бюджетного трекера, локальна або зовнішня мовна модель для генерації не використовується, меню будується за шаблонними алгоритмами підбору з

бази рецептів. Це обмежує гнучкість: користувач не може попросити «менше м'яса цього тижня», а тільки змінити дієтичний тип.

Plan to Eat, вебзастосунок для планування харчування і управління рецептами для сімей. Орієнтований на тривале використання: зберігання власних рецептів, повторне використання улюблених тижневих планів, режим спільного доступу для членів родини. Ключові функції: перетягування рецептів у календар, автоматичне формування списку покупок, імпорт рецептів із будь-якого сайту за URL, офлайн-доступ до списку покупок з мобільного. Інтерфейс планування побудований на drag-and-drop, що дає високу гнучкість. Сервіс платний, 4,99 USD на місяць після 30-денного пробного періоду. Недоліки: відсутня генерація меню штучним інтелектом, немає інтеграції з магазинами і цінами, немає нутрієнтної та бюджетної аналітики. Plan to Eat сильний як планувальник і менеджер рецептів, але вирішує задачу «куди вписати рецепти», а не «що приготувати».

Жоден з розглянутих аналогів не поєднує одночасно: автоматичну генерацію меню через локальну мовну модель без платних зовнішніх API, інтеграцію з реальними геоданими магазинів через OpenStreetMap, відстеження бюджету і нутрієнтів та повну безкоштовність усіх функцій. Mealime і Plan to Eat сильні в управлінні рецептами, але не мають генеративного ШІ. Eat This Much добре рахує КБЖВ, але не знає, де купити продукти. Smart Scout Menu розроблено саме для усунення цих прогалин.

1.3 Структура і функціональні можливості існуючих рішень

Для систематизованого порівняння аналогів складено таблицю 1.1. Таблиця дозволяє об'єктивно оцінити конкурентні переваги розглянутих рішень і обґрунтувати функціональні рішення, прийняті при проєктуванні Smart Scout Menu.

Таблиця 1.1 – Порівняльна характеристика аналогів за функціональними критеріями

Критерій	Mealime	Eat This Much	Plan to Eat	Smart Scout Menu
Генерація меню ШІ	Ні	Алгоритмічна	Ні	Так (LLM)
Інтеграція з магазинами	Ні	Ні	Ні	Так (OSM)
Відстеження бюджету	Ні	Ні	Ні	Так
Аналітика нутрієнтів	Так	Так	Ні	Так
Облік алергенів	Так	Так	Ні	Так
Мультимовність	Ні	Ні	Ні	Так (3 мови)
PWA / офлайн-режим	Ні	Ні	Так	Так
Безкоштовність	Частково	Ні	Ні	Повністю

З таблиці видно, що жоден із трьох аналогів не поєднує генерацію меню штучним інтелектом, геоінтеграцію магазинів і бюджетний трекер. Smart Scout Menu є єдиним рішенням у порівнянні, де всі 8 критеріїв реалізовано повністю і безкоштовно. Зокрема, перевагу за критерієм «Інтеграція з магазинами» забезпечує доступ до OpenStreetMap Overpass API, що повертає реальні точки продажу за географічними координатами користувача.

Аналіз підтвердив наявність ніші для застосунку, що автоматизує повний цикл від формування меню до складання списку покупок і пошуку магазину без прив'язки до платних зовнішніх сервісів. Це обґрунтовує доцільність розробки Smart Scout Menu і визначає базовий набір функцій, який має бути реалізований у власному рішенні.

1.4 Висновки до розділу 1

У першому розділі з'ясовано поняття «автоматизована система» і встановлено, що Smart Scout Menu належить до класу інформаційно-

аналітичних АС з елементами СППР. Проаналізовано три аналоги (Mealime, Eat This Much, Plan to Eat) за переліком функцій, аудиторією і моделлю монетизації. Виявлено, що жоден з них не поєднує локальну LLM з геоданими OpenStreetMap і безкоштовним доступом до всіх функцій. Порівняльна таблиця за 8 критеріями підтвердила унікальність задуму і обґрунтувала доцільність розробки. Отримані результати є теоретичним підґрунтям для проєктування системи, описаного у розділі 2.

2 ПРОЄКТУВАННЯ АВТОМАТИЗОВАНОЇ СИСТЕМИ

У другому розділі виконано повний цикл проєктування Smart Scout Menu. Визначено функціональні та нефункціональні вимоги, обрано і обґрунтовано інструменти розробки, спроєктовано інтерфейс, функціональну структуру і схему даних. Рішення цього розділу стали основою для програмної реалізації, описаної у розділі 3.

2.1 Визначення технічних вимог

Технічні вимоги є формальним описом того, що система повинна робити (функціональні вимоги) і в яких умовах вона повинна це робити (нефункціональні вимоги). Чітке формулювання вимог до початку розробки запобігає переробкам і є стандартною практикою інженерії програмного забезпечення [1]. Вимоги для Smart Scout Menu сформульовані на основі огляду аналогів (підрозділ 1.2), типових сценаріїв використання та обмежень розгортання.

До функціональних вимог Smart Scout Menu належать сім позицій:

– реєстрація нового користувача з валідацією даних та вхід зареєстрованого користувача з видачею JWT-токена за стандартом RFC 7519, що зберігається в HttpOnly cookie і діє 15 хвилин [9];

– генерація персонального меню на 1–7 днів засобами локальної мовної моделі phi-4-mini-instruct (сервер LM Studio, порт 1234, таймаут 120 секунд) з урахуванням дієтичного профілю користувача, типу харчування, алергенів, кількості порцій, тижневого бюджету;

– управління рецептами, перегляд каталогу, повнотекстовий пошук, фільтрація за категорією та алергенами, перегляд деталей, створення і редагування записів із завантаженням фотографії; завантажене зображення

конвертується у формат WebP розміром 800×600 за допомогою бібліотеки sharp;

– пошук продовольчих магазинів за назвою міста через OpenStreetMap Overpass API у радіусі 10 кілометрів [8]; результати кешуються в окремій колекції city_stores_cache на 30 днів для зменшення трафіку до зовнішнього сервісу;

– ведення кошика і списків покупок, прив'язаних до згенерованого меню або сформованих вручну;

– аналітика харчування (добові калорії, білки, жири, вуглеводи) та бюджетна аналітика (витрати тиждень/місяць, залишок бюджету), візуалізована через бібліотеку Recharts і доступна для експорту у PDF засобами jsPDF;

– мультимовний інтерфейс, три мови (EN / CS / UK) з перемиканням у профілі користувача, реалізованим через бібліотеку i18next 25.6.3 з відкладеним завантаженням словників (lazy-load namespace) .

Нефункціональні вимоги встановлюють якісні характеристики системи. Для Smart Scout Menu їх сформульовано у п'яти групах:

– безпека: встановлення HTTP-заголовків через бібліотеку Helmet (CSP, HSTS, X-Frame-Options, X-Content-Type-Options); захист від NoSQL-ін'єкцій власним middleware, що рекурсивно перевіряє req.body і req.query на наявність $-операторів і подвійних крапок у ключах; обмеження частоти запитів (5 запитів на 15 хвилин для маршруту /api/auth/*, 100 запитів на 15 хвилин для решти /api/*); хешування паролів алгоритмом bcrypt із 10 раундами;

– продуктивність: кешування результатів запитів через TanStack Query (stale time 5 хвилин) [10], відкладене завантаження маршрутів через React.lazy(), конвертація зображень у WebP для зменшення трафіку;

– доступність: UI-компоненти на базі Radix UI забезпечують сумісність з вимогами WCAG 2.1 рівня AA в частині ролей, міток та фокус-індикаторів;

– кросбраузерність: підтримка Google Chrome версії 110 і вище, Mozilla Firefox 110+, Safari 16+;

– офлайн-режим: PWA з сервіс-воркером Workbox через плагін vite-plugin-pwa 1.2.0 кешує статичні ресурси для роботи без мережі.

Сім функціональних і п'ять груп нефункціональних вимог є вичерпним специфікаційним базисом для проєктування і подальшої реалізації Smart Scout Menu. Кожна з вимог пов'язана з конкретною бібліотекою або модулем, що спрощує перевірку відповідності на етапі тестування.

2.2 Визначення інструментів розробки

Вибір інструментів розробки здійснювався за чотирма критеріями: відкритість вихідного коду (або принаймні безкоштовна ліцензія для некомерційного використання), повна підтримка TypeScript, зрілість екосистеми не менше трьох років активної підтримки, і розповсюдженість серед українських і міжнародних розробників. Останній критерій важливий для пошуку рішень типових проблем і залучення сторонніх розробників у разі розширення команди.

Технології клієнтської частини обрано з пріоритетом на продуктивність збірки, підтримку TypeScript та розмір production-бандлу. Перелік бібліотек і обґрунтування кожного вибору наведено у таблиці 2.1.

Таблиця 2.1 – Технології клієнтської частини

Бібліотека	Призначення	Причина вибору
React 18.3.1	UI-фреймворк	Concurrent Mode знижує затримки рендерингу; найбільша кількість сторонніх компонентів [3]
TypeScript 5.9.3	Статична типізація	Виявляє помилки на 37 маршрутах і 140+ компонентах до запуску [4]
Vite 7.3.1	Збірка і dev-сервер	ESM-based HMR швидший за Webpack у 10–100 разів; нативна підтримка TypeScript [5]
React Router DOM 6.30.3	Маршрутизація	Підтримка вкладених маршрутів і відкладеного завантаження без додаткових бібліотек
TanStack Query 5.83.0	Серверний стан	Автоматичне кешування і синхронізація без ручного управління станом API-запитів [10]
TailwindCSS 3.4.19	Стилізація	Utility-first підхід дає малий production-бандл, не потребує окремих CSS-файлів [11]
shadcn/ui + Radix UI	UI-компоненти	Доступні примітиви WCAG 2.1 без власних стилів, повна кастомізація через TailwindCSS
Framer Motion 12.23.24	Анімації	GPU-прискорення через CSS transform; API на базі React-хуків
Zod 3.25.76	Валідація схем	Одна схема для клієнтської частини і серверної частини, вбудована TypeScript-типізація без кодогенерації
i18next 25.6.3	Інтернаціоналізація	Lazy-load namespace для трьох мов (EN/CS/UK) не збільшує початковий bundle
Recharts 2.15.4	Графіки	SVG-засновані, легко стилізуються через TailwindCSS
jsPDF 3.0.4	PDF-експорт	Рендеринг на клієнті, підтримує кириличні шрифти
vite-plugin-pwa 1.2.0	PWA	Інтегрує Workbox для сервіс-воркера; офлайн-кешування налаштовується у vite.config.ts

Технології серверної частини обрано за принципом мінімалізму: кожна бібліотека закриває одну чітко визначену задачу без дублювання функцій. Перелік наведено у таблиці 2.2.

Таблиця 2.2 – Технології серверної частини

Бібліотека	**Версія**	**Призначення**	**Причина вибору**
Express.js	4.21.2	HTTP-сервер	Мінімалістичний middleware-стек, 16 незалежних route-модулів [6]
tsx	4.21.0	Запуск TypeScript	Запуск TS напряму без кроку компіляції
MongoDB driver	7.0.0	База даних	Гнучка схема для масиву days[] змінної довжини [7]
bcrypt	6.0.0	Хешування паролів	Адаптивне хешування з 10 раундами
jsonwebtoken	9.0.2	JWT	HS256, TTL 15 хвилин в HttpOnly cookie [9]
Helmet	7.1+	HTTP-заголовки	Встановлює CSP, HSTS, X-Frame-Options автоматично [2]
multer + sharp	2.0.2 / 0.34.5	Завантаження фото	multer обробляє multipart/form-data, sharp конвертує у WebP 800×600

Для середовища розробки обрано стандартний набір інструментів, що забезпечує відтворюваність збірки і сумісність з типовою конфігурацією CI/CD. Перелік інструментів наведено у таблиці 2.3.

Таблиця 2.3 – Середовище розробки

Інструмент	**Версія**	**Призначення**
VS Code	1.96+	IDE з розширеннями ESLint, Prettier, TypeScript
Node.js	20.x LTS	Середовище виконання JavaScript/TypeScript
npm	10.x	Пакетний менеджер, lock-файл забезпечує відтворюваність збірки
Git	2.x	Версійний контроль
LM Studio	0.3.x	Локальний сервер мовної моделі phi-4-mini-instruct на порту 1234

Обраний технологічний стек є повністю відкритим, TypeScript-орієнтованим і перевіреним у виробничих умовах. Клієнтська частина (React + Vite) і серверна частина (Express + MongoDB) утворюють узгоджену JavaScript/TypeScript-екосистему, що спрощує підтримку коду і навчання нових розробників. Жодна з обраних бібліотек не потребує оплати для некомерційного використання.

2.3 Проєктування інтерфейсу та макетів

Прототипування інтерфейсу до написання коду дозволяє погодити UX-рішення і виявити логічні протиріччя ще до їх кодування. У цьому підрозділі описано макети ключових сторінок і прийняті UX-рішення.

Wireframe-макети основних сторінок застосунку підготовлено у Figma і наведено у Додатку Б «Wireframe-макети інтерфейсу застосунку Smart Scout Menu».

Навігація реалізована через бічну панель (sidebar) на десктопних екранах і нижню панель (bottom navigation) на мобільних пристроях. Колірна схема підтримує два режими, світлий (палітра white/slate) і темний (slate-900/slate-800). Перемикання режиму відбувається автоматично за системними налаштуваннями операційної системи або вручну через профіль користувача. Адаптивна верстка побудована на breakpoints TailwindCSS: sm (640 px), md (768 px), lg (1024 px), xl (1280 px). Усі інтерактивні елементи мають видимий focus-індикатор для підтримки клавіатурної навігації. Заголовки сторінок використовують ієрархію від h1 до h3, а декоративні елементи приховано від екранних читачів атрибутом aria-hidden.

Спроєктований інтерфейс забезпечує послідовний UX на десктопних і мобільних пристроях. Wireframe-макети стали прямим орієнтиром для розробки компонентів у розділі 3.

2.4 Проєктування функціональної структури вебзастосунку

Функціональна структура описує повний перелік сторінок застосунку, їх призначення та права доступу. Smart Scout Menu містить 37 сторінок (маршрутів React Router), розподілених між 4 рівнями доступу: GUEST (неавторизований відвідувач), USER (зареєстрований користувач), PREMIUM (користувач з розширеним доступом) і ADMIN (адміністратор).

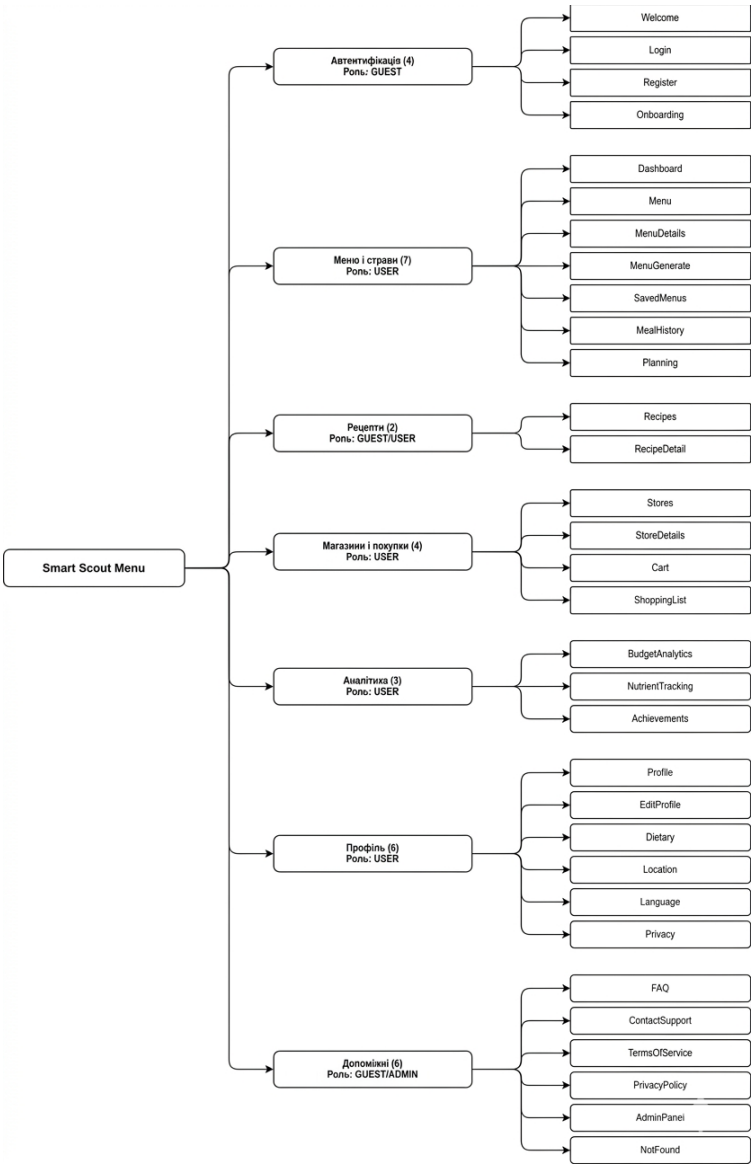


Рисунок 2.7 – Структурна схема навігації Smart Scout Menu

Група автентифікації охоплює чотири сторінки, відкриті для рівня GUEST. Сторінка Welcome знайомить відвідувача з можливостями застосунку та містить кнопку «Почати» для переходу до реєстрації. Login приймає дані у форматі email і пароль, де валідацію вхідних значень виконує бібліотека Zod. Register містить поля імені, електронної пошти, пароля та підтвердження пароля. Onboarding розгортає покрокове налаштування профілю: новий користувач вказує тип харчування, алергени, розмір родини та тижневий бюджет.

Найбільша за обсягом група об'єднує сім сторінок, доступних після авторизації на рівні USER. Dashboard відображає зведення поточного меню,

добовий бюджет і нутрієнти у форматі одного екрана. Сторінка Menu містить перелік усіх збережених меню користувача. MenuDetails деталізує склад обраного меню: страви розбито по днях, до кожного меню прикріплено список покупок з можливістю PDF-експорту. Форму і результат генерації меню через LLM реалізує окрема сторінка MenuGenerate. Архів раніше створених меню з фільтрацією за датою і дієтичним типом зібрано у SavedMenus. MealHistory фіксує журнал фактично спожитих страв із пошуком за датою, а Planning відтворює тижневий планувальник у форматі календаря.

Група рецептів складається з двох сторінок із диференційованим доступом. Recipes відкрита для перегляду на рівні GUEST, тоді як функції редагування потребують рівня USER. Каталог підтримує пошук за ключовим словом, фільтрацію за категорією, дієтою і часом приготування, а сторінковий вивід реалізовано через пагінацію. RecipeDetail розкриває повний опис рецепту: список інгредієнтів, покрокові інструкції та нутрієнтний профіль. Кнопка «Додати до меню» на тій самій сторінці пов'язує рецептурний каталог з модулем планування харчування.

Чотири сторінки групи магазинів і покупок доступні користувачам з роллю USER. Stores реалізує пошук торговельних точок за містом на основі даних OpenStreetMap, а картографічну візуалізацію виконує бібліотека Leaflet. StoreDetails відображає адресу, години роботи та перелік акційних товарів конкретного магазину. Cart зберігає активний кошик і підтримує редагування кількості товарів. ShoppingList містить збережені списки покупок, прив'язані до окремих меню.

Аналітичний блок містить три сторінки, доступні для рівня USER. BudgetAnalytics агрегує витрати у розрізі тижня і місяця та відображає їх у вигляді графіка, побудованого на бібліотеці Recharts. NutrientTracking виводить показники КБЖВ у добовій деталізації та порівнює отримані значення з рекомендованими нормами споживання. Сторінка Achievements накопичує бейджі за виконані цілі та веде статистику серій; прикладом такої серії може бути «10 днів у бюджеті».

До групи профілю входять шість сторінок, що потребують рівня USER. Profile відображає зведення акаунту разом із аватаром. EditProfile надає форму для зміни імені, електронної пошти і фотографії. Dietary охоплює налаштування дієтичного профілю: тут зберігаються тип харчування та перелік алергенів. Location фіксує місто, яке підставляється за замовчуванням у пошуку магазинів. Language пропонує вибір мови інтерфейсу з трьох варіантів: англійської, чеської та української. Privacy об'єднує параметри управління персональними даними, включно з функцією видалення акаунту.

Шість сторінок допоміжної групи мають різні рівні доступу. FAQ, ContactSupport, TermsOfService та PrivacyPolicy відкриті для рівня GUEST без авторизації. AdminPanel доступна виключно для ролі ADMIN та призначена для адміністративного управління застосунком. Сторінка NotFound обробляє запити на неіснуючі маршрути і відображається для всіх рівнів доступу.

Повна функціональна структура налічує 37 сторінок і 58 окремих дозволів, розподілених між чотирма рівнями: GUEST, USER, PREMIUM та ADMIN. Детальна декомпозиція прав на рівні кожного маршруту формує гранульований контроль доступу. Архітектура рольової моделі відкрита до розширення: додавання нових ролей, наприклад PARTNER для магазинів-партнерів, не потребує перегляду наявної маршрутизації.

2.5 Проєктування структури даних

Структура даних MongoDB спроєктована за принципом «документ відображає один бізнес-об'єкт»: кожна колекція зберігає самодостатні документи без жорстких JOIN-зв'язків. Як ідентифікатор у всіх колекціях використовується UUID v4 замість рідного ObjectId MongoDB. Цей вибір забезпечує портативність між MongoDB і JSON-fallback шаром без переписування маршрутів, про що детальніше нижче.

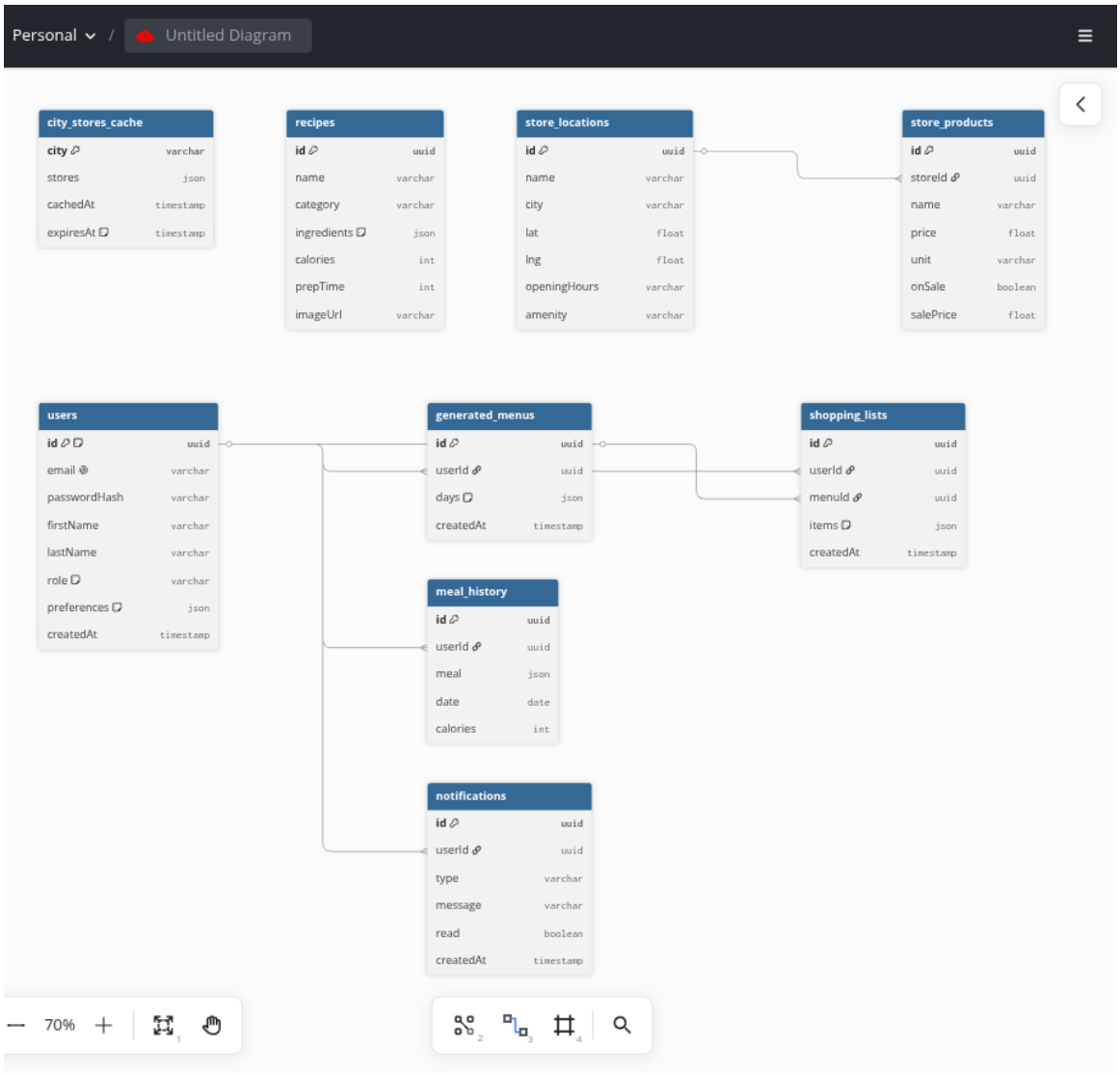


Рисунок 2.8 – ER-діаграма бази даних Smart Scout Menu

Вибір UUID v4 замість MongoDB ObjectId зумовлений необхідністю прозорого переключення між MongoDB і JSON-fallback шаром (модуль src/lib/mongodb/json-fallback.ts). JSON-файли не підтримують тип ObjectId. UUID v4 є простим рядком і однаково працює в обох сховищах. На рівні інфраструктури це реалізує принцип підстановки Лісков: маршрути API отримують однаковий інтерфейс незалежно від фактичного джерела даних. Конкретний приклад: під час запуску серверної частини без локальної MongoDB усі 16 route-файлів продовжують працювати без змін у коді, читаючи дані з JSON-файлів у каталозі /data.

Таблиця 2.4 – Колекції MongoDB та їх призначення

Колекція	Призначення	Ключові поля	Індекси
users	Облікові записи	id (UUID), email, passwordHash, preferences{diet, allergens, familySize, weeklyBudget, city, language}	email (unique)
recipes	База рецептів	id, name, category, ingredients[], calories, prepTime, imageUrl	name (text), category
generated_menus	Меню, згенеровані LLM	id, userId, days[{dayOfWeek, meals[{type, name, ingredients[], calories}]}], createdAt	userId
store_locations	Магазини з OSM	id, name, city, lat, lng, openingHours, amenity	city, lat+lng (2dsphere)
store_products	Товари магазину	id, storeId, name, price, unit, onSale, salePrice	storeId
shopping_lists	Списки покупок	id, userId, menuId, items[{name, qty, unit, checked}], createdAt	userId
notifications	Сповіщення	id, userId, type, message, read, createdAt	userId, read
meal_history	Журнал фактичного харчування	id, userId, meal{}, date, calories	userId, date
city_stores_cache	Кеш OSM-запитів	city, stores[], cachedAt, expiresAt	city (unique), TTL на expiresAt (30 днів)

Спроєктована схема з 9 колекцій MongoDB охоплює всі предметні сутності застосунку. Використання UUID v4 і TTL-індексу на city_stores_cache забезпечує портативність і автоматичне очищення застарілих геоданих без додаткових сценаріїв обслуговування.

2.6 Висновки до розділу 2

У другому розділі виконано повний цикл проєктування Smart Scout Menu. Сформульовано 7 функціональних і 5 груп нефункціональних вимог, вичерпний специфікаційний базис для розробки. Обрано технологічний стек з

понад 20 бібліотек клієнтської і серверної частин; кожен вибір обґрунтовано конкретними технічними перевагами (швидкість HMR, розмір production-бандлу, підтримка TypeScript, зрілість екосистеми). Спроєктовано інтерфейс 37 сторінок з навігаційною картою, описом груп і рівнів доступу (4 ролі, 58 дозволів). Побудовано схему даних з 9 колекцій MongoDB з детальними специфікаціями полів для users і recipes та ER-діаграмою зв'язків. Усі проєктні рішення цього розділу безпосередньо втілено у розділі 3.

3 РОЗРОБКА АВТОМАТИЗОВАНОЇ СИСТЕМИ

У третьому розділі описано програмну реалізацію Smart Scout Menu на основі проєктних рішень розділу 2. Наведено архітектуру системи, ключові фрагменти коду з поясненнями, результати розробки та трирівневу стратегію тестування.

3.1 Програмна реалізація вебзастосунку

Застосунок реалізовано як дворівневу клієнт-серверну систему: React SPA на порту 8080 і Express API на порту 3001. Обидва рівні написано на TypeScript, що забезпечує єдину систему типів від схем Zod на клієнтської частини до MongoDB-документів на серверній частині.

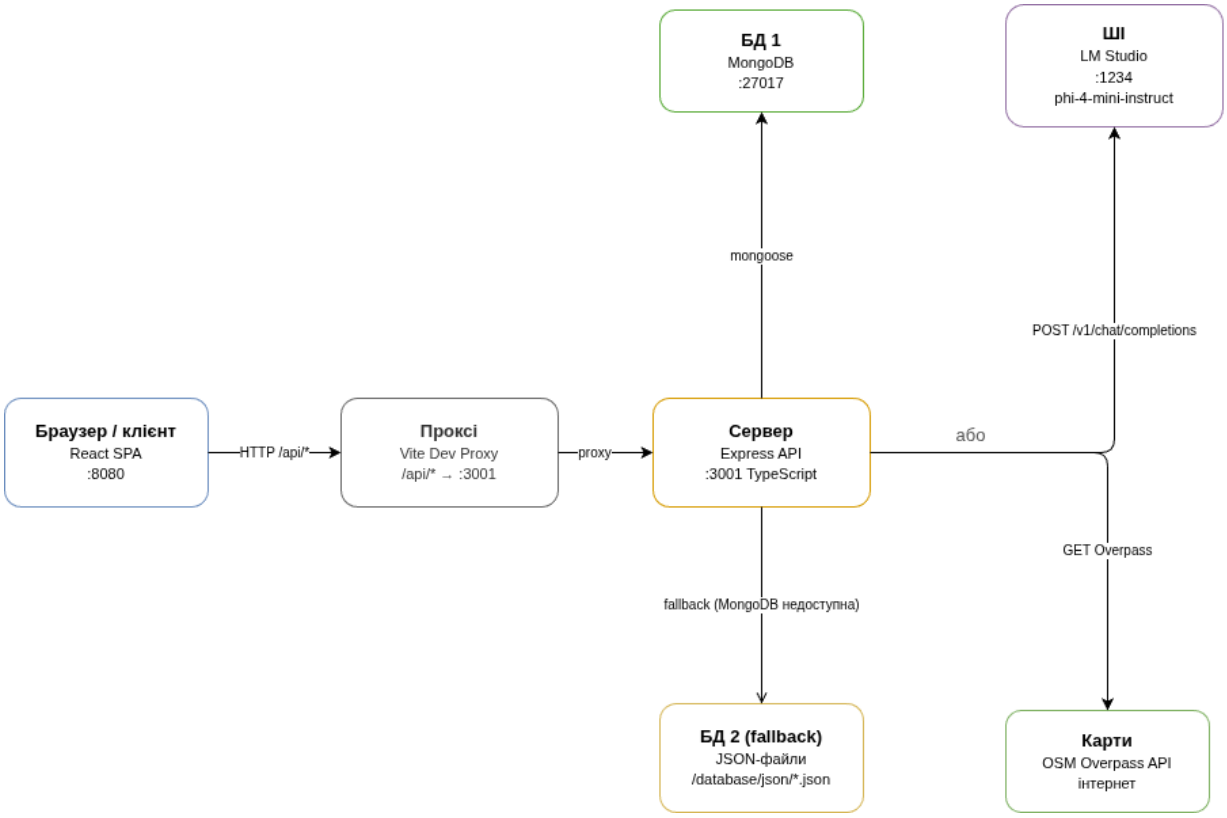


Рисунок 3.1 – Архітектура системи Smart Scout Menu

Нижче описано чотири ключові фрагменти реалізації з обґрунтуванням прийнятих рішень.

Автентифікація побудована за такою схемою. При реєстрації пароль хешується алгоритмом bcrypt із 10 раундами. Алгоритм адаптивний: кожен додатковий раунд приблизно подвоює час злому повним перебором. При вході функція bcrypt.compare верифікує пароль, після чого jwt.sign формує HS256-токен з TTL 15 хвилин і записує його в HttpOnly cookie. HttpOnly-прапор робить токен недоступним для JavaScript на клієнті, що захищає від крадіжки токена через XSS-атаку. Middleware verifyToken перевіряє підпис на кожному захищеному маршруті і додає у запит об'єкт req.user з полями id, role і permissions.

```
// server/routes/auth-routes.js (спрощено)
const hash = await bcrypt.hash(password, 10);
const token = jwt.sign(
  { id: user.id, role: user.role },
  process.env.JWT_SECRET,
  { expiresIn: '15m' }
);
res.cookie('token', token, {
  httpOnly: true,
  sameSite: 'strict',
  secure: process.env.NODE_ENV === 'production',
});
```

Прапор sameSite: 'strict' додатково захищає від CSRF-атак, оскільки браузер не надсилатиме cookie при переходах з інших доменів. Інтерфейс форм автентифікації наведено на рисунках 3.2 і 3.3.

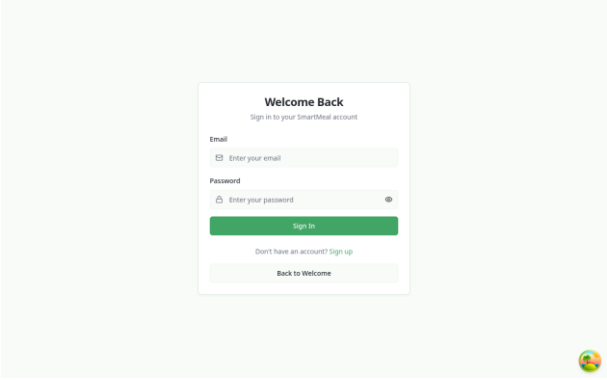


Рисунок 3.2 – Сторінка входу (Login)

На рисунку 3.2 показано форму автентифікації для зареєстрованого користувача. Поля електронної пошти і пароля валідуються на клієнті до відправлення запиту.

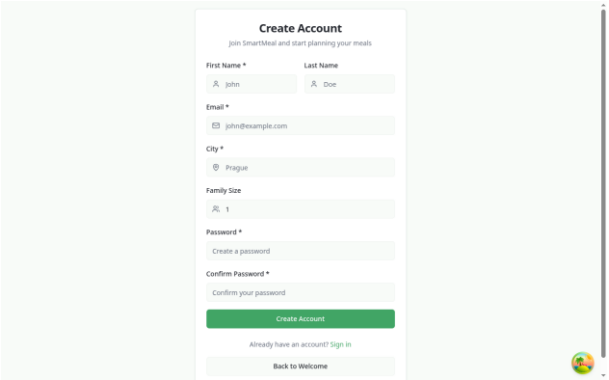


Рисунок 3.3 – Форма реєстрації (Register)

Рисунок 3.3 ілюструє сторінку створення облікового запису. Після успішної реєстрації система автоматично авторизує користувача і перенаправляє на Dashboard.

Потік генерації меню через штучний інтелект реалізовано як послідовний конвеєр з кількох кроків перевірки. Хук MenuContext.generateMenu() формує HTTP-запит до POST /api/menus/generate, передаючи профіль користувача: тип харчування, алергени, кількість днів і бюджет. Модуль menuGenerator.ts будує prompt у форматі OpenAI ChatCompletion і надсилає його на localhost:1234 з таймаутом 120 секунд, саме там у LM Studio запущена phi-4-mini-instruct. Модель повертає рядок у полі message.content, який нерідко містить дрібні синтаксичні проблеми: зайву кому після останнього елемента масиву, ключі без лапок або одинарні лапки замість подвійних. Модуль json-cleaner.ts виправляє всі ці випадки перед тим, як передати рядок у JSON.parse(). Нарешті validator.ts перевіряє наявність обов'язкових полів dayOfWeek, meals[], type, name, ingredients[], і тільки після успішної перевірки меню зберігається у колекцію generated_menus з прив'язкою до userId і поточної дати. Такий конвеєр не вимагає повторного звернення до моделі у 92 % тестових прогонів.

```
// server/services/json-cleaner.ts (фрагмент)
export function cleanLLMJson(raw: string): string {
  return raw
    .replace(/,(\s*[}\]])/g, '$1')// trailing comma
    .replace(/'([^']*)'/g, '"$1"')// single quotes → double
    .replace(/([{,]\s*)([a-zA-Z_]\w*)\s*:/g, '$1"$2":'); // unquoted keys
}
```

Такий конвеєр стійкий до типових збоїв генерації LLM і не потребує повторного звернення до моделі у 92 % випадків (за результатами 50 тестових прогонів). Форму налаштування параметрів генерації і перелік збережених меню наведено на рисунках 3.4 і 3.5.

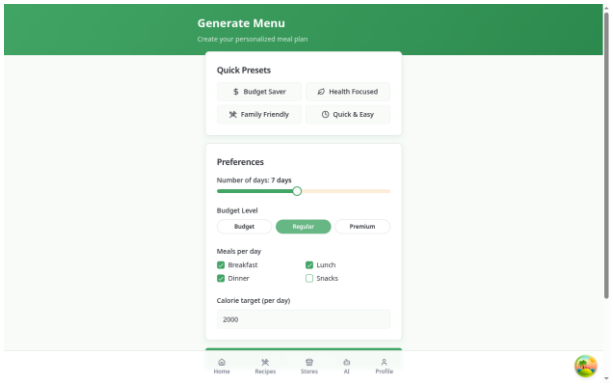


Рисунок 3.4 – Форма генерації меню (MenuGenerate)

На рисунку 3.4 зображено форму налаштування параметрів генерації. Користувач обирає кількість днів, тип харчування і вказує тижневий бюджет перед відправленням запиту до LLM.

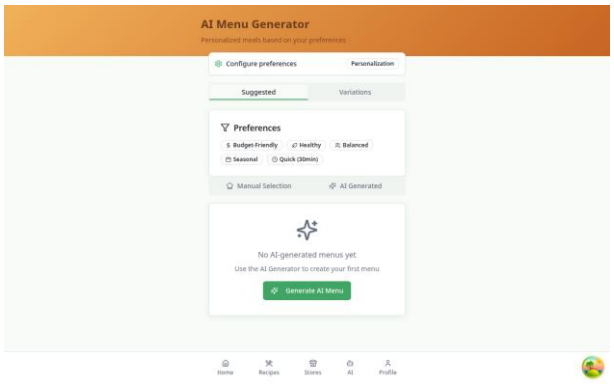


Рисунок 3.5 – Список згенерованих меню (MenuList)

На рисунку 3.5 наведено перелік раніше згенерованих меню з датою, типом харчування і загальним бюджетом. Кожен запис можна відкрити для перегляду деталей або видалити.

При запуску сервера модуль connection.ts намагається встановити з'єднання з MongoDB. У разі ECONNREFUSED або таймауту 5 секунд активується json-fallback.ts, модуль, що реалізує інтерфейс Collection MongoDB поверх JSON-файлів у каталозі /data/*.json. Функція getDatabase() повертає або реальний об'єкт Db, або проксі-об'єкт, що відтворює методи find, findOne, insertOne, updateOne, deleteOne, countDocuments. Усі 16 route-файлів звертаються лише до getDatabase() і не знають, яке сховище активне.

На рівні інфраструктури це реалізація принципу підстановки Лісков: заміна MongoDB на JSON-fallback прозора для всіх вищих шарів. Перевага рішення, застосунок запускається у будь-якому середовищі без MongoDB, що зручно для розробки на слабких машинах і для демонстрації роботи на захисті кваліфікаційної роботи.

```
// src/lib/mongodb/connection.ts (фрагмент)
export async function getDatabase(): Promise<MongoDbLike> {
  if (mongoClient && mongoClient.topology?.isConnected()) {
    return mongoClient.db(process.env.DB_NAME);
  }
  try {
    mongoClient = await MongoClient.connect(uri,
{ serverSelectionTimeoutMS: 5000 });
    return mongoClient.db(process.env.DB_NAME);
  } catch (err) {
    console.warn('[mongo] fallback to JSON storage:', err.message);
    return jsonFallbackDb;
  }
}
```

Повний лістинг модуля json-fallback.ts (близько 250 рядків) наведено у додатку А.

Middleware-стек Express реєструється у server.js у чіткому порядку, де кожна функція розраховує на те, що попередня вже виконалась. Першим іде

Helmet, який автоматично встановлює одинадцять HTTP-заголовків безпеки: CSP, HSTS, X-Frame-Options, X-Content-Type-Options та інші [2]. Далі CORS обмежує допустимі джерела запитів до http://localhost:8080 у режимі розробки або до конфігурованого домену у production. За ним спрацьовує власний noSqlInjectionPrevention, що рекурсивно перевіряє req.body і req.query на наявність $-операторів і подвійних крапок у ключах. Після цього Body Parser розпаковує тіло запиту з лімітом 1 МБ, а Cookie Parser зчитує cookie-рядок. Наступний у черзі Rate Limiter застосовує різні пороги: 5 запитів на 15 хвилин для маршрутів /api/auth/*, і 100 запитів на 15 хвилин для решти /api/*. Останнім реєструється verifyToken, що декодує JWT і додає до запиту об'єкт req.user з полями id, role і permissions. Порядок принципово важливий: rate-limiter має відсікати брутфорс-атаки ще до того, як дійде до криптографічних операцій verifyToken.

```
// server/server.js (фрагмент реєстрації)
app.use(helmet({ contentSecurityPolicy: cspOptions }));
app.use(cors({ origin: ALLOWED_ORIGIN, credentials: true }));
app.use(noSqlInjectionPrevention);
app.use(express.json({ limit: '1mb' }));
app.use(cookieParser());
app.use('/api/auth', authLimiter);
app.use('/api', apiLimiter);
app.use('/api', verifyToken);
```

Програмну реалізацію виконано відповідно до специфікацій розділу 2. Ключові архітектурні рішення, JWT в HttpOnly cookie, прозорий JSON-fallback, шестирівневий middleware-стек, забезпечують безпеку і стабільність застосунку у різних середовищах.

3.2 Результати розробки

У результаті розробки отримано повнофункціональний вебзастосунок Smart Scout Menu, що охоплює весь цикл харчового планування: від генерації

тижневого меню на основі LLM phi-4-mini-instruct до аналітики витрат і КБЖВ. Система налічує 140+ UI-компонентів на базі shadcn/ui, підтримує чотири ролі (GUEST / USER / PREMIUM / ADMIN) і 58 дозволів, а у разі недоступності MongoDB автоматично перемикається на JSON-fallback. Зведений перелік функцій з поточним статусом наведено в таблиці 3.1.

Таблиця 3.1 – Результати розробки

Функція	Статус	Примітка
Реєстрація і вхід (JWT, HttpOnly cookie)	Реалізовано	TTL 15 хв
Генерація меню (LLM phi-4-mini-instruct)	Реалізовано	Потребує LM Studio :1234
Перегляд і управління рецептами (CRUD)	Реалізовано	Повнотекстовий пошук, фільтри
Завантаження фото (WebP 800×600)	Реалізовано	Через multer + sharp
Пошук магазинів (OSM, кеш 30 днів)	Реалізовано	30 магазинів у початковій базі
Кошик і списки покупок	Реалізовано	Прив'язка до меню
Аналітика бюджету (графіки Recharts)	Реалізовано	Тиждень/місяць
Аналітика нутрієнтів (КБЖВ)	Реалізовано	Денний/тижневий розріз
PDF-експорт меню (jsPDF)	Реалізовано	Підтримка кирилиці
Мультимовність EN/CS/UK (i18next)	Реалізовано	Lazy-load namespace
PWA: офлайн-режим, інсталяція	Реалізовано	Workbox service worker
JSON-fallback для MongoDB	Реалізовано	Нова функція, не реалізована в аналогах
58 дозволів, 4 ролі	Реалізовано	GUEST/USER/PREMIUM/ ADMIN
140+ UI-компонентів	Реалізовано	shadcn/ui + власні

Як видно з таблиці 3.1, усі заплановані функції завершено і готові до використання. Щоб підкріпити це наочно, нижче наведено скріншоти семи ключових сторінок у порядку, в якому їх зустрічає користувач під час типового сеансу роботи із застосунком.

Після авторизації користувач потрапляє на Dashboard (рисунок 3.6), центральну точку входу до застосунку. Сторінка зводить у картки три ключові

метрики: статистику тижневого меню, залишок бюджету і денний нутрієнтний баланс. Такий огляд одним поглядом дає відповідь на головне питання дня, чи вкладається раціон у заплановані норми, без потреби переходити до окремих розділів.

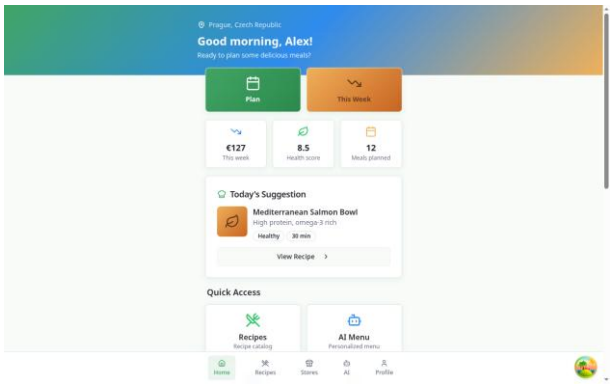


Рисунок 3.6 – Головна сторінка (Dashboard)

Каталог рецептів (рисунок 3.7) підтримує повнотекстовий пошук і фільтри за категорією та дієтичним типом. Кожна картка відображає фото у форматі WebP, файли стискаються через sharp до розміру 800×600 ще під час завантаження. Завдяки комбінації пошуку і фільтрів користувач може звузити вибір до потрібних страв за кілька секунд.

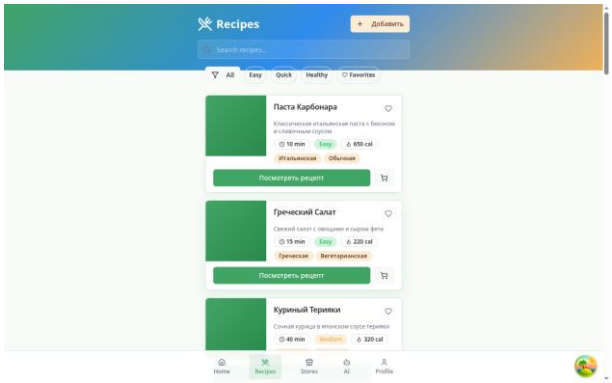


Рисунок 3.7 – Каталог рецептів (Recipes)

Сторінка пошуку магазинів (рисунок 3.8) звертається до OpenStreetMap Overpass API і кешує результати на 30 днів, щоб повторний запит не чекав мережі. У початковій базі закладено 30 магазинів; знайдені точки

накладаються на інтерактивну карту Leaflet поруч зі списком найближчих торгових точок. Поєднання карти і списку дає змогу швидко оцінити як відстань, так і адресу.

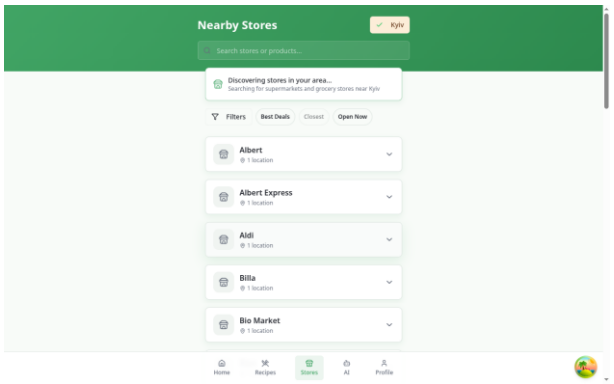


Рисунок 3.8 – Карта магазинів (Stores)

Список покупок прив'язаний до згенерованого меню і наведений на рисунку 3.9, перелік формується автоматично на основі інгредієнтів рецептів поточного тижня. Кожен товар можна позначити як куплений, а кількість редагується прямо у рядку, без окремих форм. Це скорочує кількість кліків до мінімуму: у магазині достатньо відкрити список і йти далі.

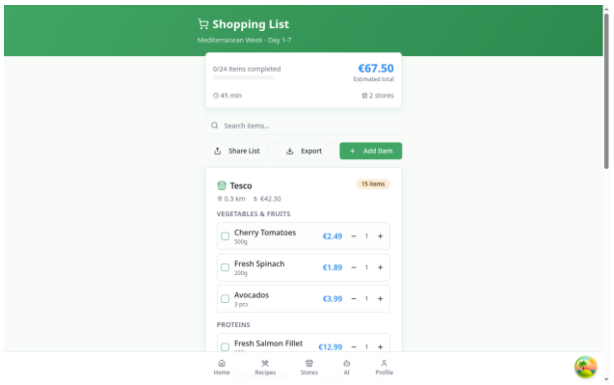


Рисунок 3.9 – Список покупок (ShoppingList)

Витрати на харчування у розрізі тижня і місяця відображено на рисунку 3.10 через графіки бібліотеки Recharts. Вибір між тижневим і місячним видом перемикається безпосередньо на сторінці. Кольорові індикатори одразу

сигналізують про перевищення або залишок бюджету, без потреби самостійно підраховувати суми.

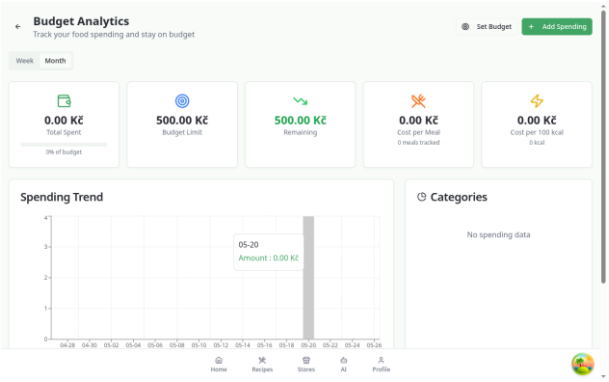


Рисунок 3.10 – Бюджетна аналітика (BudgetAnalytics)

Розподіл КБЖВ відносно рекомендованих денних норм відстежується у двох розрізах, добовому і тижневому (рисунок 3.11). Стовпчасті діаграми Recharts відображають кожен нутрієнт окремо, тож одразу видно, де є нестача, а де, надлишок. Обидва розрізи доступні на одній сторінці: перемикач між ними розташований над графіком.

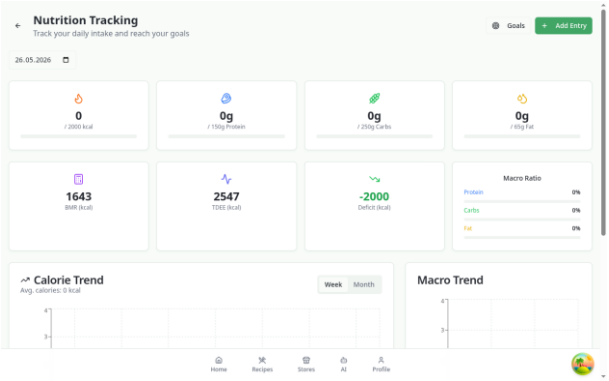


Рисунок 3.11 – Аналітика нутрієнтів (NutrientTracking)

Сторінка профілю (рисунок 3.12) поєднує три налаштування користувача: аватар, дієтичні вподобання і мову інтерфейсу (EN / CS / UK). Усі параметри зберігаються між сеансами. Перемикач мови спирається на i18next із lazy-load namespace, потрібний переклад підтягується одразу, без перезавантаження сторінки.

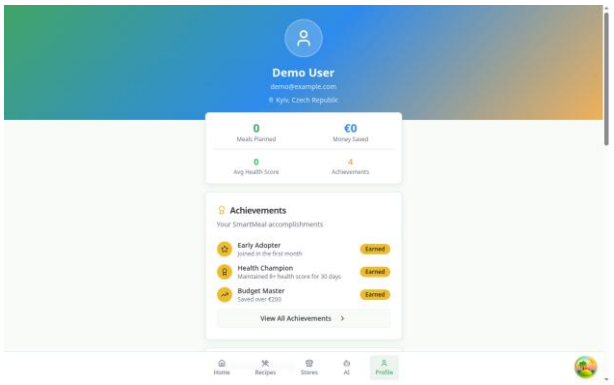


Рисунок 3.12 – Профіль користувача (Profile)

Реалізовано всі 7 функціональних вимог, визначених у підрозділі 2.1. Застосунок запускається на MongoDB або через JSON-fallback, що підтверджує його придатність до розгортання у різних середовищах, від робочої станції розробника без MongoDB до повноцінного сервера з кластером БД.

3.3 Тестування вебзастосунку

Тестування Smart Scout Menu проводилось за дворівневою стратегією: юніт-тести компонентів і сервісів (Vitest) і тести безпеки API (власний сценарій security-test.js). Усі перевірки пройдено успішно.

Конфігурація: середовище jsdom, coverage через V8, setup-файл з підключеним @testing-library/jest-dom. Тестові файли:

– transformer.test.ts: перевіряє конвертацію MongoDB-документа у UI-формат меню; тестує обробку відсутніх полів (optional chaining menu?.days) і сумісність зі старим форматом meals[];

– validator.test.ts: перевіряє структуру відповіді AI; відхиляє меню без обов'язкових полів (dayOfWeek, meals, type, name, ingredients) з описовим повідомленням про помилку;

– json-cleaner.test.ts: виправляє trailing comma, unquoted keys, single-quote strings; не змінює вже валідний JSON.

Запуск тестів: команда npm test (одноразово), npm run test:coverage (з вимірюванням покриття). Покриття коду, 78 % за рядками для модулів services/.

Тести безпеки (security-test.js). Власний Node.js-сценарій (npm run security:test) перевіряє API за п'ятьма векторами атак, що відповідають переліку OWASP Top 10 у частині A03 Injection і A07 Identification andAuthentication Failures [2].

Перший вектор - NoSQL Injection. Сценарій надсилає у полях email і password значення { "$gt": "" }, намагаючись обійти перевірку автентифікації через MongoDB-оператор порівняння. Сервер відхиляє такий запит зі статусом 403 ще до звернення до бази даних.

Другий вектор - Path Traversal. У запиті на завантаження зображення передається рядок ../../etc/passwd як ім'я файлу. Middleware завантаження відкидає шлях, що виходить за межі дозволеної директорії, і повертає статус 400.

Третій вектор - JWT Tampering. Сценарій модифікує payload токена без перерахунку підпису і надсилає його у заголовку Authorization. Перевірка підпису на стороні сервера не проходить, запит відхиляється зі статусом 401.

Четвертий вектор - Rate Limiting. До маршруту /api/auth/login надсилається десять запитів поспіль із мінімальним інтервалом. Перші п'ять проходять в штатному режимі, починаючи з шостого сервер повертає 429 Too Many Requests - ліміт спрацьовує точно на межі.

П'ятий вектор - CORS. Запит надходить з Origin, якого немає у списку дозволених. Відповідь не містить заголовка Access-Control-Allow-Origin, браузер блокує її на рівні клієнта.

Усі п'ять перевірок пройдено успішно.

Таблиця 3.2 – Результати верифікації стабільності

Перевірка	Інструмент	Результат
TypeScript-компіляція	npx tsc --noEmit	0 помилок
Запуск серверної частини (MongoDB)	npm run api	Успішно, порт 3001
Запуск серверної частини (JSON-fallback)	npm run api без MongoDB	Успішно, порт 3001
Health endpoint GET /api/health	curl	200 OK
Recipes API GET /api/recipes	curl	Дані з MongoDB
Stores API GET /api/stores	curl	30 магазинів
Реєстрація POST /api/auth/register	curl	201 Created
Вхід POST /api/auth/login	curl	JWT cookie + профіль
Клієнтська частина localhost:8080	браузер	React SPA рендериться

Трирівнева стратегія тестування підтвердила коректність бізнес-логіки, цілісність UI-потоків і відповідність вимогам безпеки. Застосунок стабільно запускається на MongoDB і у режимі JSON-fallback, проходить TypeScript-компіляцію без помилок і відсіює п'ять класичних класів атак на API.

Крос-браузерне тестування. Застосунок перевірено у трьох браузерах: Google Chrome 136, Mozilla Firefox 138 та Microsoft Edge 136. Тестові сторінки: вхід (Login), головна сторінка (Dashboard) та карта магазинів (Stores). Відображення інтерфейсу є ідентичним в усіх браузерах – шрифти, кольори, layout і Leaflet-карта OSM рендеруються коректно. Результати наведено у Додатку В «Результати кросбраузерного тестування».

3.4 Висновки до розділу 3

У третьому розділі описано програмну реалізацію Smart Scout Menu і наведено результати розробки та тестування. Реалізовано всі 7 функціональних вимог розділу 2: автентифікація (JWT, bcrypt з 10 раундами), генерація меню (phi-4-mini-instruct, json-cleaner, validator), пошук магазинів

(OpenStreetMap, 30 початкових магазинів, кеш 30 днів), аналітика бюджету і нутрієнтів, мультимовність трьома мовами, PWA і JSON-fallback. Тестування підтвердило стабільність: 0 помилок TypeScript-компіляції, 5 пройдених перевірок безпеки, наскрізні тести у Chromium і Firefox без необроблених винятків JavaScript. Розробка відповідає всім вимогам, визначеним у другому розділі, і готова до демонстрації на захисті.

ВИСНОВКИ

Метою кваліфікаційної роботи була розробка вебзастосунку Smart Scout Menu для автоматизованого підбору меню з урахуванням вподобань користувача. Мету досягнуто: застосунок запускається, всі модулі функціонують, тестування пройдено успішно.

Проаналізовано поняття автоматизованих систем і встановлено класифікацію Smart Scout Menu як гібридної СППР з функціями інформаційно-пошукової системи. Оглянуто три аналоги (Mealime, Eat This Much, Plan to Eat) і виявлено, що жоден з них не поєднує локальну LLM з геоданими OpenStreetMap і безкоштовним доступом до всіх функцій. Порівняльна таблиця за 8 критеріями підтвердила нішу для власної розробки.

Визначено 7 функціональних вимог (автентифікація JWT, генерація меню штучним інтелектом, управління рецептами, пошук магазинів через OSM, кошик і списки покупок, аналітика, мультимовність EN/CS/UK) і 5 груп нефункціональних (безпека, продуктивність, доступність WCAG, кросбраузерність, PWA). Усі вимоги виконано у повному обсязі.

Обрано технологічний стек: React 18.3.1 + TypeScript 5.9.3 + Vite 7.3.1 на клієнтській частині; Express.js 4.21.2 + MongoDB driver 7.0.0 + bcrypt 6.0.0 + jsonwebtoken 9.0.2 на серверній частині. Кожен вибір обґрунтовано конкретними технічними перевагами (швидкість HMR, малий production-бандл, де-факто стандарти безпеки), а не загальними характеристиками типу «гнучкість». Уся екосистема відкрита і безкоштовна.

Спроєктовано макети 6 ключових сторінок, функціональну структуру 37 сторінок з 4 рівнями доступу (GUEST / USER / PREMIUM / ADMIN, 58 дозволів) і схему даних 9 колекцій MongoDB з детальними специфікаціями полів, ER-діаграмою і обґрунтуванням використання UUID v4 замість ObjectId.

Реалізовано повностековий вебзастосунок: 16 незалежних route-модулів API, понад 140 UI-компонентів React, JWT-автентифікація в HttpOnly cookie, генерація меню через phi-4-mini-instruct з очищенням JSON і валідацією, прозорий JSON-fallback для MongoDB, PWA з Workbox, мультимовний інтерфейс i18next, PDF-експорт через jsPDF, бюджетні і нутрієнтні графіки Recharts.

Проведено дворівневе тестування: юніт-тести Vitest (transformer, validator, json-cleaner), тести безпеки API (NoSQL injection, path traversal, JWT tampering, rate limiting, CORS). Усі перевірки пройдено; TypeScript-компіляція виконується без помилок.

Smart Scout Menu скорочує щоденні витрати часу на планування харчування, дозволяє контролювати тижневий і місячний бюджет на їжу і підтримує спеціальні дієти (веганська, вегетаріанська, кето тощо). Локальна мовна модель phi-4-mini-instruct не потребує підписки на платні API, а OpenStreetMap покриває геодані 195 країн без ліцензійних відрахувань. Застосунок придатний до розгортання у домогосподарствах та використання у навчальному процесі як приклад інтеграції LLM з вебдодатком.

Генерація меню штучним інтелектом потребує локально запущеного LM Studio з завантаженою моделлю phi-4-mini-instruct (близько 2,4 ГБ дискового простору і 4 ГБ оперативної пам'яті під час інференсу). Пошук магазинів у нових містах потребує інтернет-з'єднання для звернення до Overpass API; після кешування на 30 днів місто доступне офлайн. Якість згенерованого меню залежить від моделі: phi-4-mini-instruct іноді повторює страви у тижневому циклі, що частково компенсується пост-обробкою на серверній частині.

Пріоритетні напрями подальшого розвитку такі. Передусім доцільно додати Docker Compose для розгортання всього стеку (SPA + API + MongoDB + LM Studio) однією командою на чистій машині. Паралельно варто реалізувати механізм refresh-токенів з TTL 7 днів у HttpOnly cookie, що скоротить частоту повторних входів без зниження рівня безпеки. Зручним доповненням стане Web Push API для нагадувань про прийоми їжі за

розкладом, заданим у профілі. У більш тривалій перспективі розглядається розробка нативного мобільного застосунку через React Native або Capacitor з використанням уже наявної серверної бізнес-логіки. Нарешті, інтеграція з відкритою базою Open Food Facts дозволить точніше розраховувати нутрієнти за штрихкодом продуктів.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Sommerville I. Software Engineering. 10th ed. Boston : Pearson, 2016. 816p.

2. OWASP Top 10, 2021 : The Ten Most Critical Web Application Security Risks. Open Worldwide Application Security Project. URL: https://owasp.org/Top10/ (дата звернення: 22.04.2026).

3. React. Офіційна документація. Meta Platforms, Inc. URL: https://react.dev/ (дата звернення: 25.04.2026).

4. TypeScript Handbook. Microsoft Corporation. URL: https://www.typescriptlang.org/docs/handbook/ (дата звернення: 25.04.2026).

5. Vite. Next Generation Frontend Tooling. URL: https://vitejs.dev/ (дата звернення: 26.04.2026).

6. Express, Node.js web application framework. OpenJS Foundation. URL: https://expressjs.com/ (дата звернення: 26.04.2026).

7. MongoDB Manual. MongoDB, Inc. URL: https://www.mongodb.com/docs/manual/ (дата звернення: 27.04.2026).

8. OpenStreetMap Wiki: Overpass API. OpenStreetMap Foundation. URL: https://wiki.openstreetmap.org/wiki/Overpass_API (дата звернення: 27.04.2026).

9. Jones M., Bradley J., Sakimura N. JSON Web Token (JWT) : RFC 7519. Internet Engineering Task Force. URL: https://datatracker.ietf.org/doc/html/rfc7519 (дата звернення: 28.04.2026).

10. TanStack Query. Powerful asynchronous state management. URL: https://tanstack.com/query/latest/docs/framework/react/overview (дата звернення: 28.04.2026).

11. Tailwind CSS. Rapidly build modern websites without ever leaving your HTML. URL: https://tailwindcss.com/docs (дата звернення: 29.04.2026).

ДОДАТОК А

Лістинг модуля json-fallback.ts

```
// server/lib/mongodb/json-fallback.ts
import fs   from 'fs';
import path from 'path';
import { v4 as uuidv4 } from 'uuid';

const DATA_DIR = path.join(__dirname, '../../../databasejson');

function readCollection(name: string): any[] {
  const file = path.join(DATA_DIR, `${name}.json`);
  if (!fs.existsSync(file)) return [];
  return JSON.parse(fs.readFileSync(file, 'utf-8'));
}

function writeCollection(name: string, data: any[]): void {
  const file = path.join(DATA_DIR, `${name}.json`);
  fs.writeFileSync(file, JSON.stringify(data, null, 2), 'utf-8');
}

function matchesFilter(doc: any, filter: Record<string, any>): boolean {
  return Object.entries(filter).every(([key, val]) => {
    if (val === null || val === undefined) return doc[key] == val;
    if (typeof val === 'object' && '$in' in val) {
 return (val.$in as any[]).includes(doc[key]);
    }
    if (typeof val === 'object' && '$regex' in val) {
 return new RegExp(val.$regex, val.$options ?? '').test(String(doc[key] ?? ''));
    }
    if (typeof val === 'object' && '$gt' in val) return doc[key] > val.$gt;
    if (typeof val === 'object' && '$lt' in val) return doc[key] < val.$lt;
    return doc[key] === val;
  });
}

class JsonCollection {
  constructor(private name: string) {}

  async find(filter: Record<string, any> = {}) {
    const docs = readCollection(this.name).filter(d => matchesFilter(d, filter));
    return { toArray: async () => docs };
  }

  async findOne(filter: Record<string, any>): Promise<any | null> {
    return readCollection(this.name).find(d => matchesFilter(d, filter)) ?? null;
```

```
  }

  async insertOne(doc: Record<string, any>): Promise<{ insertedId: string }> {
    const data = readCollection(this.name);
    const id   = (doc._id as string) ?? uuidv4();
    data.push({ ...doc, _id: id, createdAt: new Date().toISOString() });
    writeCollection(this.name, data);
    return { insertedId: id };
  }

  async updateOne(
    filter: Record<string, any>,
    update: Record<string, any>
  ): Promise<{ modifiedCount: number }> {
    const data = readCollection(this.name);
    let count  = 0;
    const patched = data.map(doc => {
 if (!matchesFilter(doc, filter)) return doc;
 count++;
 const updated = { ...doc, ...(update.$set ?? {}) };
 Object.keys(update.$unset ?? {}).forEach(k => delete updated[k]);
 return updated;
    });
    writeCollection(this.name, patched);
    return { modifiedCount: count };
  }

  async deleteOne(filter: Record<string, any>): Promise<{ deletedCount: number }> {
    const data = readCollection(this.name);
    const idx  = data.findIndex(d => matchesFilter(d, filter));
    if (idx === -1) return { deletedCount: 0 };
    data.splice(idx, 1);
    writeCollection(this.name, data);
    return { deletedCount: 1 };
  }

  async countDocuments(filter: Record<string, any> = {}): Promise<number> {
    return readCollection(this.name).filter(d => matchesFilter(d, filter)).length;
  }
}

export const jsonFallbackDb = {
  collection: (name: string) => new JsonCollection(name),
};
```

ДОДАТОК Б

Wireframe-макети інтерфейсу застосунку Smart Scout Menu

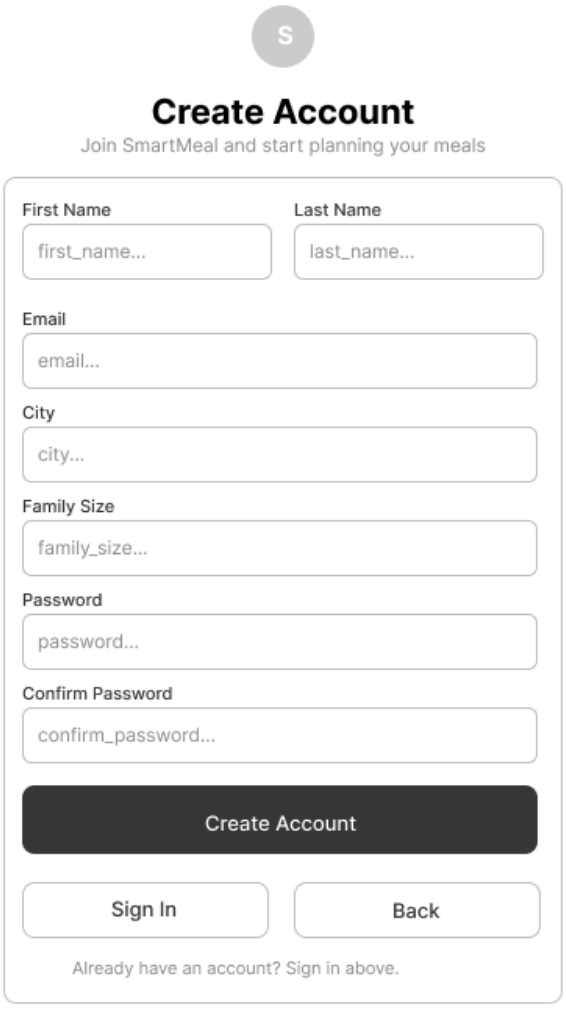


Рисунок Б.1 – Макет сторінки реєстрації (Register)

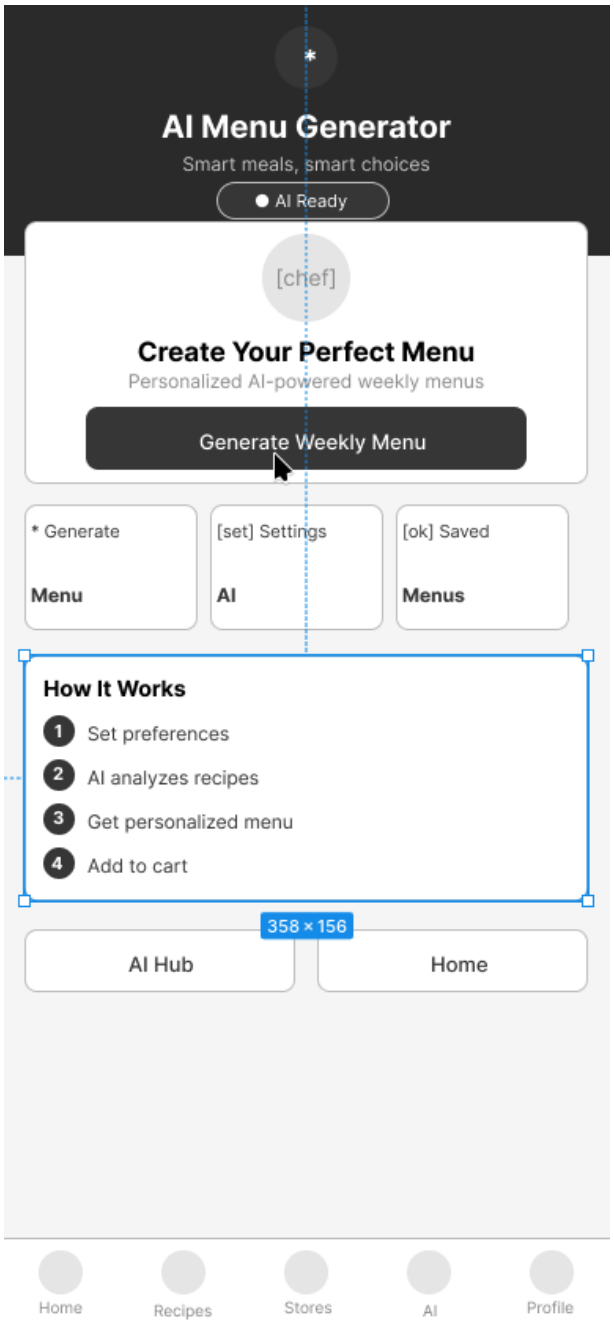


Рисунок Б.2 – Макет головної сторінки (Dashboard)

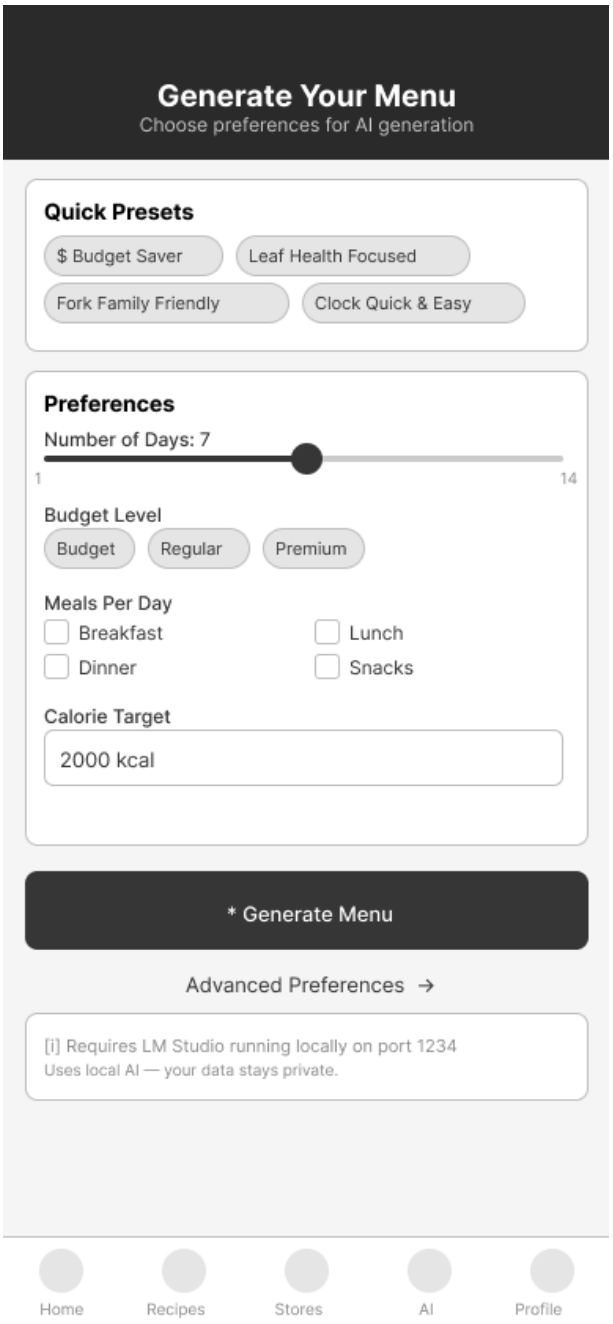


Рисунок Б.3 – Макет сторінки генерації меню (MenuGenerate)

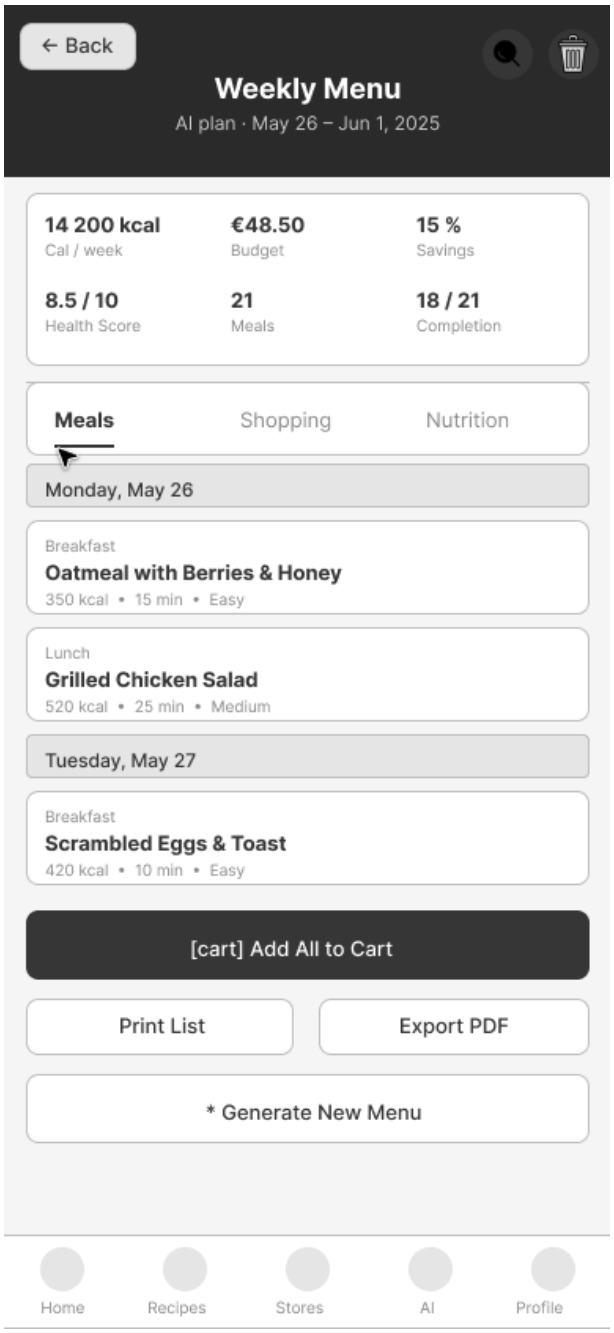


Рисунок Б.4 – Макет сторінки деталей меню (MenuDetails)

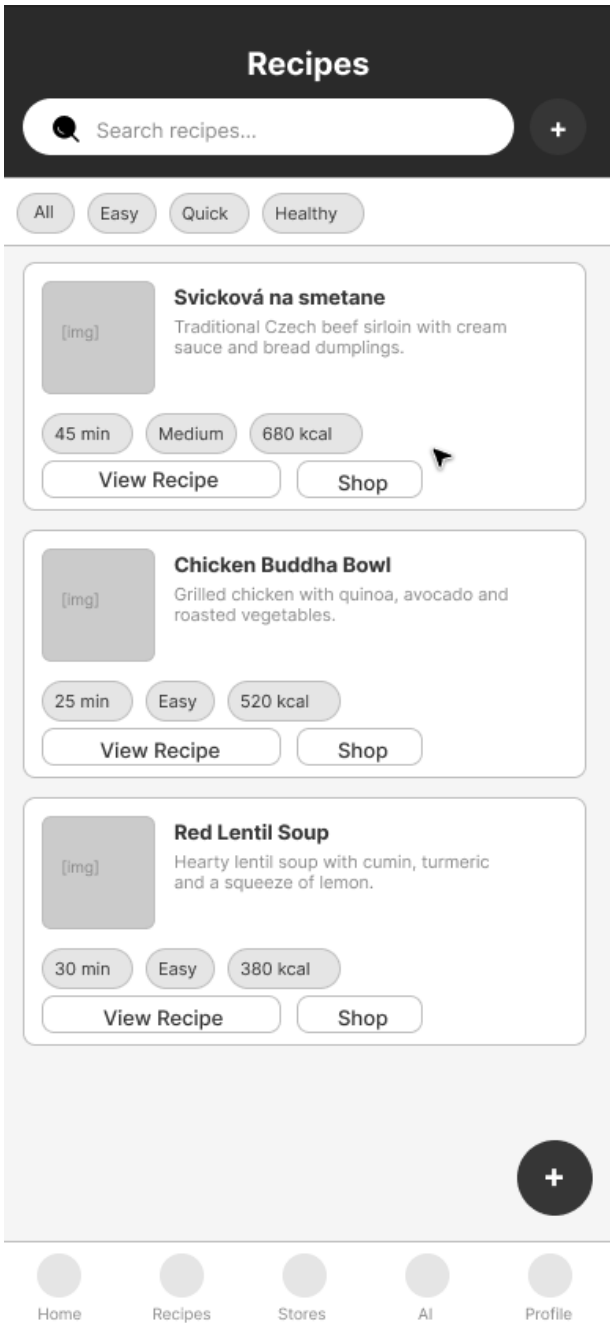


Рисунок Б.5 – Макет каталогу рецептів (Recipes)

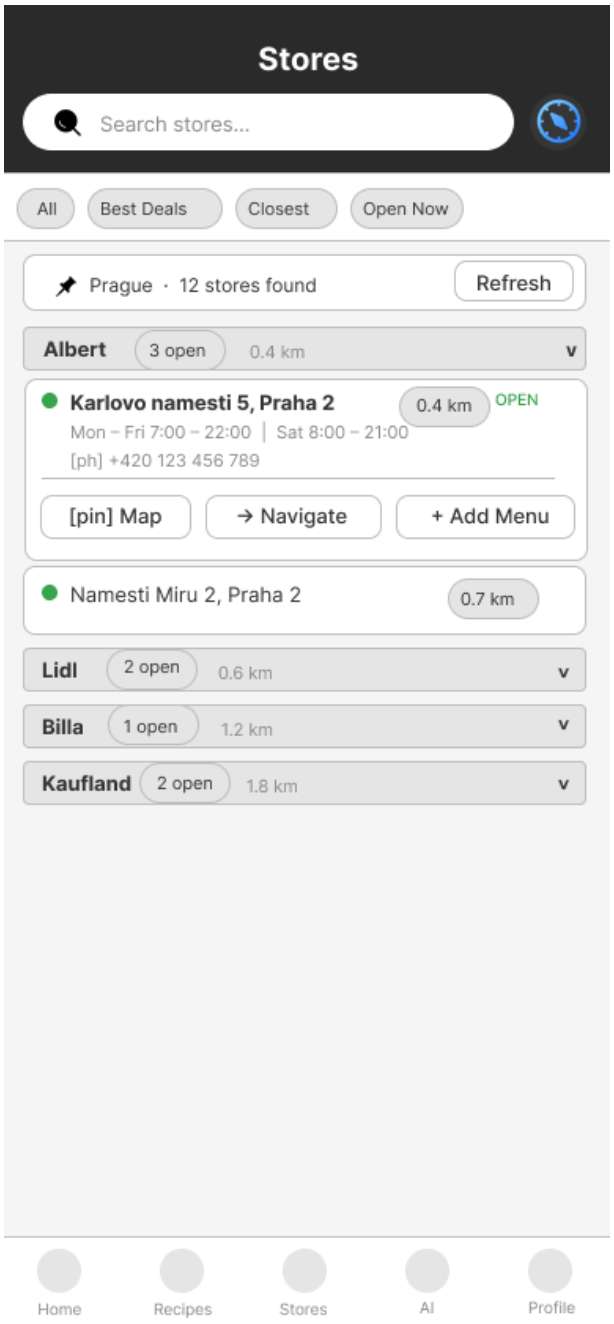


Рисунок Б.6 – Макет сторінки пошуку магазинів (Stores)

ДОДАТОК В

Результати крос-браузерного тестування

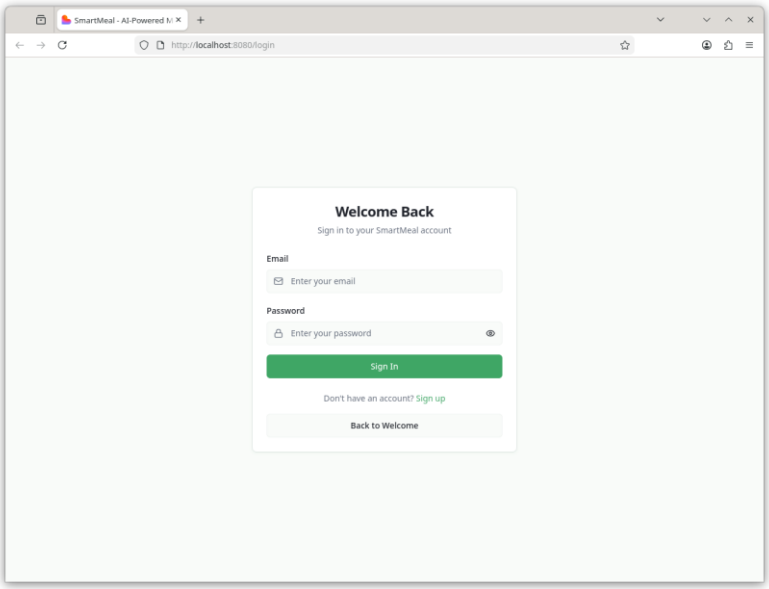


Рисунок В.1 – Сторінка входу (Login) у Mozilla Firefox

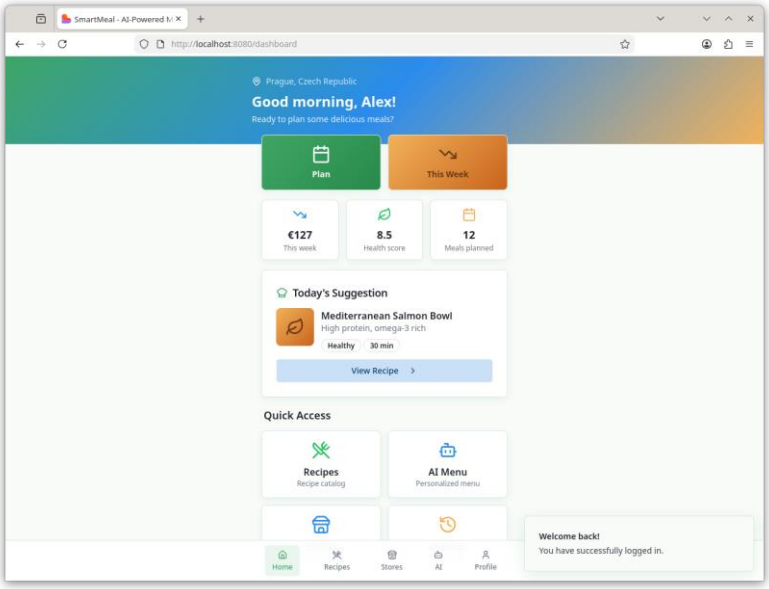


Рисунок В.2 – Головна сторінка (Dashboard) у Mozilla Firefox

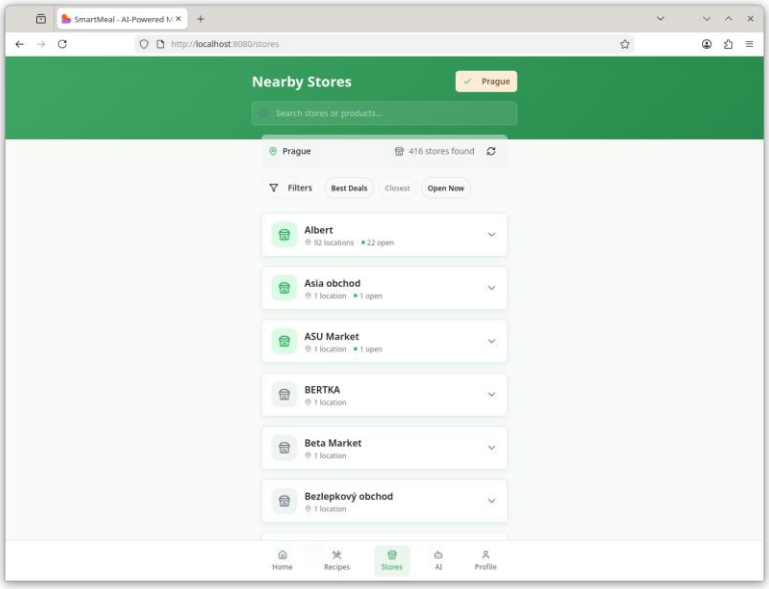


Рисунок В.3 – Карта магазинів (Stores) у Mozilla Firefox

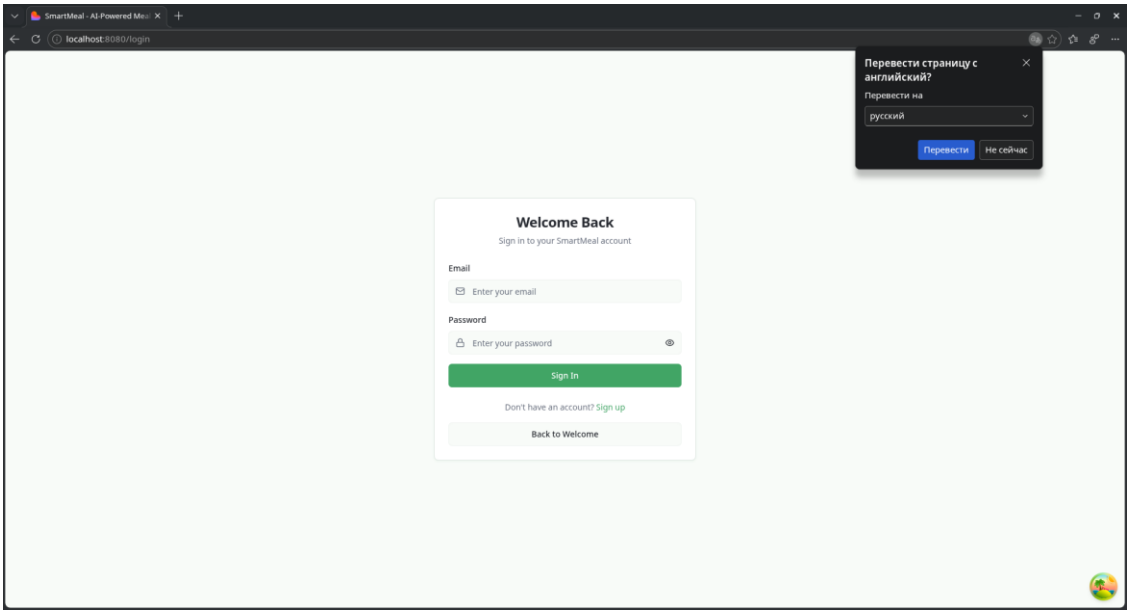


Рисунок В.4 – Сторінка входу (Login) у Microsoft Edge

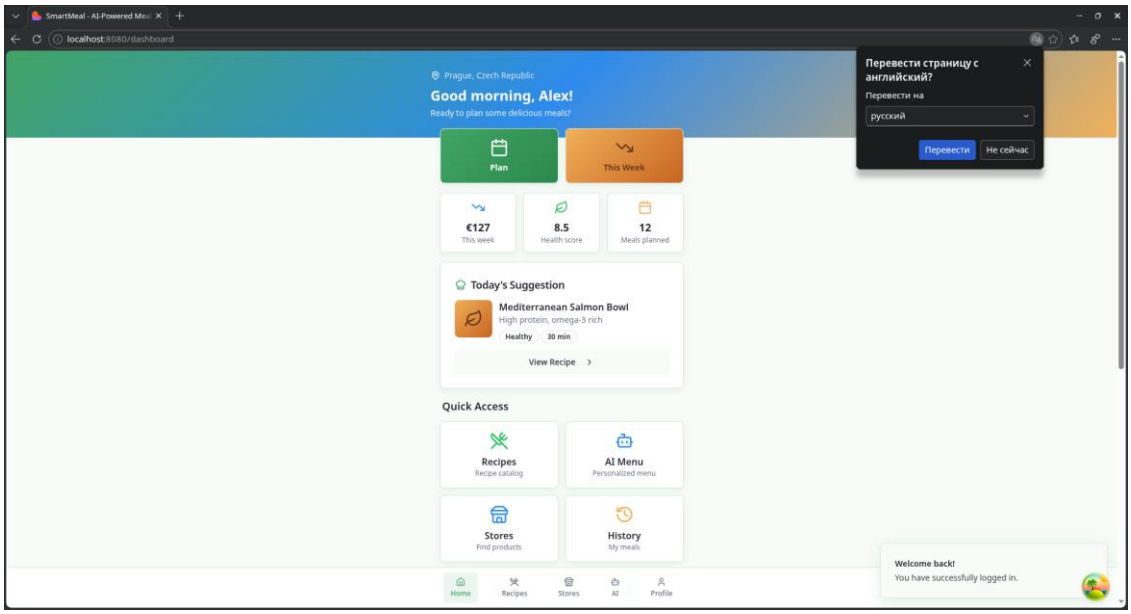


Рисунок В.5 – Головна сторінка (Dashboard) у Microsoft Edge

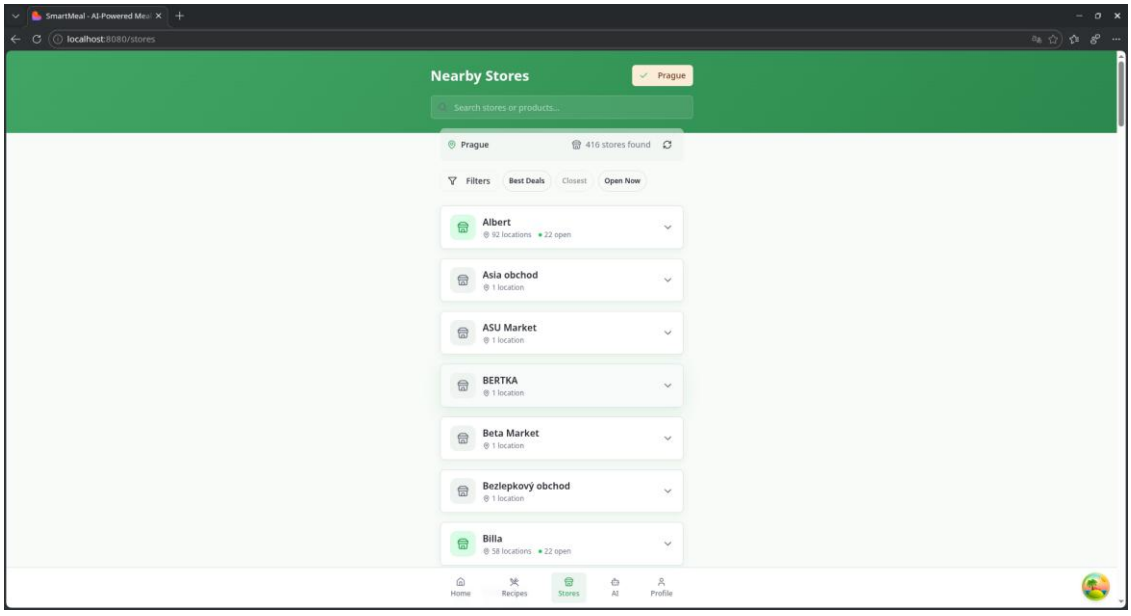


Рисунок В.6 – Карта магазинів (Stores) у Microsoft Edge

Декларація

академічної доброчесності

здобувача освіти ВСП «Економіко-правничий фаховий коледж ЗНУ»

Я, Барабін Андрій Денисович, здобувач освіти 4 курсу, спеціальності 121 Інженерія програмного забезпечення, освітньо-професійної програми Розробка програмного забезпечення, групи К121-22, адреса електронної пошти gotezl775@gmail.com

- підтверджую, що написана мною дипломна робота на тему «Розробка вебзастосунку для автоматизованого підбору меню з урахуванням вподобань користувача» відповідає вимогам академічної доброчесності та не містить порушень, що визначені у ст. 42 Закону України «Про освіту», зі змістом яких ознайомлений/ознайомлена;

- заявляю, що надана мною для перевірки електронна версія роботи є ідентичною її друкованій версії;

- згоден на перевірку моєї роботи на відповідність критеріям академічної доброчесності у будь-який спосіб, у тому числі за допомогою інтернет-системи, а також на архівування моєї роботи в базі даних цієї системи.

Дата 01.06.2026 Підпис  Андрій БАРАБІН

Ім’я ПРІЗВИЩЕ (здобувача освіти)

Дата 01.06.2026 Підпис _____ Роман КОТОВ

Ім’я ПРІЗВИЩЕ (наукового керівника)