МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Острозька академія»
Економічний факультет
Кафедра інформаційних технологій та аналітики даних

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня бакалавра

на тему: «**Проєктування та розробка Frontend частини веб-сервісу для продажу кави»**

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

Керівник: *старший викладач кафедри ЕММІТ,*
Клебан Юрій Вікторович

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

РОБОТА ДОПУЩЕНА ДО ЗАХИСТУ

Завідувач кафедри інформаційних технологій та аналітики даних
_____ (проф., д.е.н. Кривицька О.Р.)

Протокол № 11 від « 20 » травня 2026 р.

Острог, 2026

АНОТАЦІЯ
кваліфікаційної роботи
на здобуття освітнього ступеня бакалавра

Тема: Проєктування та розробка Frontend частини веб-сервісу для продажу кави

Автор: *Гладуненко Андрій Романович*

Науковий керівник: *Клебан Юрій Вікторович, старший викладач*

Захищена «……»…… 20__ року.

Пояснювальна записка до кваліфікаційної роботи: ____*(кількість сторінок роботи) с., ____ (кількість рисунків) рис., _____ (кількість таблиць) табл., _____ (кількість додатків) додатків, _____ (кількість джерел) джерел.*

Ключові слова: *FRONTEND, REACT, ПРОДАЖ КАВИ, ІНТЕРНЕТ-МАГАЗИН, ЛОГІСТИКА, ВЕБЗАСТОСУНОК, АВТОМАТИЗАЦІЯ, UX-СЦЕНАРІЇ, КОНТРОЛЬ ЗАМОВЛЕНЬ*

Короткий зміст праці:

У кваліфікаційній роботі розглянуто процес проєктування та розробки frontend сервісу CoffeeShop для продажу кави через вебзастосунок. Актуальність теми зумовлена необхідністю підвищення ефективності роботи інтернет-магазинів, автоматизації обробки замовлень та зменшення кількості помилок під час взаємодії з клієнтами.

У роботі проаналізовано сучасні підходи до створення вебзастосунків для онлайн-продажу товарів, визначено вимоги до користувацького інтерфейсу та обґрунтовано вибір технологій. Для реалізації frontend частини використано React, Vite, Redux Toolkit та React-Bootstrap. Архітектура застосунку побудована за принципами модульності, масштабованості та розділення відповідальності між компонентами.

Практична частина присвячена реалізації вебзастосунку CoffeeShop. Реалізовано каталог товарів, кошик, оформлення замовлень, перегляд історії замовлень, відстеження доставки та формування рахунків. Передбачено механізми

перевірки коректності даних, контролю залишків товарів і систему попереджень для зменшення кількості помилок.

У застосунку реалізовано mock-режим для тестування без серверної частини, а також виконано розгортання демо-версії на платформі Vercel. У результаті створено frontend сервіс, який дозволяє оптимізувати процес продажу кави, скоротити час виконання типових операцій та підвищити контроль за оформленням замовлень і доставкою.

This qualification work examines the design and development of the CoffeeShop frontend service for coffee sales through a web application. The relevance of the topic is determined by the need to improve the efficiency of online stores, automate order processing, and reduce operational errors.

The study analyzes modern approaches to frontend web application development and justifies the selection of technologies. React, Vite, Redux Toolkit, and React-Bootstrap were used for implementation. The application architecture is based on modularity, scalability, and separation of concerns.

The practical part describes the implementation of the CoffeeShop web application, including product catalog, shopping cart, order placement, order history, delivery tracking, and invoice generation. Validation mechanisms, stock control, and warning systems were implemented to minimize errors.

A mock mode was developed for testing without backend integration, and the application was deployed on Vercel for demonstration purposes. As a result, a frontend service was created that optimizes coffee sales processes, reduces operation time, and improves control over ordering and delivery workflows.

ЗМІСТ

ВСТУП

У сучасному світі цифровізація та автоматизація є ключовими чинниками успіху будь-якого бізнесу. Ритм життя змушує споживачів цінувати свій час, тому попит на швидкі, зручні та інтуїтивно зрозумілі онлайн-сервіси невпинно зростає. Для сфери торгівлі, зокрема у сегменті продажу кави, наявність якісного веб представництва — це вже не просто перевага, а необхідна умова конкурентоспроможності. Ефективна система взаємодії з клієнтом дозволяє не лише автоматизувати процес замовлення, а й значно підвищити рівень лояльності аудиторії.

Проєктування та розробка Frontend-частини веб-сервісу для продажу кави є актуальним завданням, спрямованим на створення сучасного інтерфейсу, який забезпечить безперешкодну комунікацію між покупцем та закладом. Якісний Frontend безпосередньо впливає на конверсію, адже саме через візуальну оболонку та користувацький досвід (UX) клієнт приймає рішення про покупку. Використання передових технологій дозволяє зробити цей процес інтерактивним, швидким та безпечним.

Метою роботи є розробка клієнтської частини веб-сервісу, яка дозволить користувачам зручно переглядати асортимент кавової продукції, оформлювати замовлення в онлайн-режимі, відстежувати доставку та отримувати персоналізовані пропозиції. Особлива увага приділяється швидкодії системи та адаптивності інтерфейсу, щоб забезпечити комфортну роботу як на десктопних, так і на мобільних пристроях.

Основними завданнями, що вирішуються в межах роботи, є:

- аналіз ринку існуючих рішень у сфері онлайн-продажів кави та кавового обладнання;
- проєктування архітектури фронтенд-частини та структури даних;
- реалізація функціоналу оформлення замовлень та особистого кабінету користувача;
- впровадження системи відстеження доставки та перегляду історії покупок;

- забезпечення високого рівня UI/UX через адаптивний дизайн.

Об’єктом дослідження є процес онлайн-продажу кави та взаємодії клієнта з торговим майданчиком через веб-інтерфейс.

Предметом дослідження є методи та інструменти розробки Frontend-частини веб-сервісу на основі бібліотеки ReactJS та супутніх технологій керування станом.

Для реалізації поставленої мети було обрано сучасний стек технологій:

- ReactJS — як основна бібліотека для побудови компонентної архітектури;
- Redux Toolkit — для централізованого керування складним станом додатка (кошик, історія замовлень, статус користувача);
- Vite — як швидкий інструмент збірки проєкту;
- React-Bootstrap та SCSS — для створення гнучкого, адаптивного та привабливого дизайну.

Розробка цього проєкту дозволить закладу автоматизувати обробку замовлень, зменшити навантаження на персонал та надати клієнтам сучасний інструмент для швидких покупок. Практичне значення роботи полягає у створенні готового до експлуатації інтерфейсу, який поєднує в собі естетику, функціональність та високу продуктивність.

РОЗДІЛ 1.ТЕОРЕТИЧНІ АСПЕКТИ ТЕХНОЛОГІЙ ДЛЯ СТВОРЕННЯ ІНТЕРНЕТ-МАГАЗИНУ ПРОДАЖУ КАВИ

1.1. Постановка проблеми, опис базових понять бізнес-процесу

У сучасному цифровому суспільстві електронна комерція (E-commerce) перетворилася з допоміжного інструменту продажів на основний канал взаємодії між бізнесом та споживачем. Сфера реалізації кавової продукції, яка традиційно асоціювалася з фізичними кав'ярнями та магазинами, сьогодні проходить етап активної трансформації. Створення ефективного вебсервісу для продажу кави — це не лише питання представлення товару в мережі, а складний процес інтеграції технологій для забезпечення якісного користувацького досвіду (UX).

Основна проблема, що стоїть перед розробником сучасного Frontend-рішення у цій ніші, полягає у високій конкуренції та високих очікуваннях користувачів щодо швидкодії та зручності інтерфейсу. Користувач, купуючи каву онлайн, очікує на таку ж миттєву реакцію системи, як і при замовленні в закладі. Затримки у завантаженні сторінок, незручна навігація або складний процес оформлення замовлення призводять до відтоку клієнтів. Таким чином, постає завдання розробки високоефективного додатка, який би поєднував естетичну привабливість із технічною досконалістю.

Бізнес-процес онлайн-продажу кави включає кілька базових елементів [1]:

- Навігація та фільтрація (Catalog Management): користувач повинен мати можливість швидко знайти потрібний сорт кави за ключовими параметрами (країна походження, ступінь обсмаження, метод приготування). Це потребує гнучкої архітектури компонентів на фронтенді.
- Управління станом замовлення (State Management): оскільки процес покупки є багатоетапним (вибір товару → кошик → введення даних → підтвердження), критично важливо забезпечити цілісність даних на всіх етапах. Саме тут виникає потреба у використанні централізованих сховищ даних, таких як Redux Toolkit.

- Персоналізація та лояльність: сучасні сервіси повинні пропонувати користувачеві історію його замовлень та персоналізовані пропозиції, що стимулює повторні покупки.
- Адаптивність (Responsive Design): за статистикою, понад 60% замовлень їжі та напоїв здійснюються з мобільних пристроїв. Використання адаптивних фреймворків (наприклад, React-Bootstrap) стає обов'язковим стандартом.

Слід розмежувати поняття «веб-сайт» та «веб-сервіс (SPA)». Веб-сайт — це традиційна багатосторінкова структура, де кожен перехід вимагає перезавантаження сторінки з сервера. Веб-сервіс (Single Page Application, SPA) — це сучасний підхід (реалізований за допомогою ReactJS), де інтерфейс оновлюється динамічно без перезавантаження всієї сторінки. Для продажу кави формат SPA є оптимальним, оскільки він забезпечує плавність роботи, схожу на мобільний додаток, що критично важливо для утримання уваги клієнта.

Ефективність впровадження онлайн-платформ для торгівлі підтверджується глобальними трендами.

1.1.1. Аналіз сучасних підходів до розробки інтерфейсів та обґрунтування вибору бібліотеки ReactJS

У сучасній веб-розробці вибір базового фреймворка або бібліотеки для побудови клієнтської частини (Frontend) є фундаментальним рішенням, що визначає не лише швидкість написання коду, а й масштабованість, продуктивність та подальшу підтримку продукту. Для проєктування сервісу з продажу кави, де важливо забезпечити плавність взаємодії та динамічне оновлення даних (наприклад, актуалізація кошика або зміна статусів доставки), розгляд варіантів зосереджується навколо трьох найбільш популярних технологій: React, Angular та Vue.js.

Angular, що розробляється та підтримується компанією Google, являє собою повноцінний фреймворк, побудований на мові TypeScript. Він пропонує жорстку архітектуру «з коробки», що включає інструменти для маршрутизації, управління формами та HTTP-запитами. Проте для розробки інтернет-магазину кави Angular

може бути надлишковим. Його «важка» структура та складний поріг входження (через обов’язкове знання TypeScript та патернів RxJS) можуть сповільнити процес розробки на початкових етапах. Крім того, Angular використовує реальний DOM (Document Object Model), що при великій кількості динамічних елементів може поступатися у швидкодії рішенням на базі Virtual DOM.

Vue.js позиціонує себе як прогресивний фреймворк, що є «золотою серединою» між React та Angular. Він має дуже низький поріг входження та пропонує зручну систему шаблонів. Проте, попри свою гнучкість, Vue.js має меншу екосистему та меншу кількість готових рішень для великих комерційних проєктів у порівнянні з React. Для бізнес-рішень, де важлива підтримка спільноти та наявність перевірених часом бібліотек (наприклад, для складних UI-компонентів), вибір часто схиляється на користь більш розповсюджених технологій.

Бібліотека ReactJS, створена Meta (Facebook), була обрана для даного проєкту як найбільш оптимальний варіант. На відміну від Angular, React не є повноцінним фреймворком, а фокусується виключно на рівні представлення (View). Це дає розробнику свободу вибору додаткових бібліотек для управління станом (Redux) або маршрутизації.

Основними перевагами React, що є критичними для сервісу з продажу кави, є:

1. Декларативний підхід: розробник описує, як має виглядати інтерфейс у певному стані, а React автоматично оновлює та рендерить потрібні компоненти при зміні даних. Це дозволяє уникнути помилок при маніпуляціях з DOM-деревом.
2. Virtual DOM: це абстракція реального DOM, яка дозволяє React мінімізувати кількість важких операцій у браузері. Коли клієнт додає товар у кошик, React порівнює поточну версію віртуального дерева з попередньою і оновлює лише змінений елемент, що забезпечує миттєву реакцію інтерфейсу.
3. Компонентно-орієнтована архітектура: весь додаток розбивається на дрібні, незалежні модулі (компоненти). Наприклад, картку кави, кнопку

замовлення або рядок пошуку можна розробити один раз і використовувати багаторазово в різних частинах додатку.

4. Екосистема та підтримка: завдяки величезній популярності, React має кращу документацію та бібліотеки (як-от React-Bootstrap), що дозволяє швидко інтегрувати готові елементи дизайну та зосередитися на бізнес-логіці продажу продукції.

Таким чином, використання ReactJS забезпечує необхідну гнучкість для створення сучасного, швидкого та адаптивного веб-сервісу, який відповідає очікуванням користувачів у 2026 році.

1.1.2. Обґрунтування вибору середовища розробки та допоміжного інструментарію

Для реалізації програмного коду Frontend-частини було обрано середовище розробки Visual Studio Code (VS Code) від компанії Microsoft. Незважаючи на наявність потужних платних IDE (наприклад, WebStorm), VS Code залишається стандартом де-факто у сучасній веб-розробці.

Основними аргументами на користь VS Code є:

- Легковажність та швидкість: на відміну від повноцінних IDE, програма споживає менше ресурсів системи, що важливо при одночасному запуску локального сервера розробки, браузера та інструментів дизайну.
- Екосистема розширень: підтримка плагінів для роботи з React, автоматичного форматування коду (Prettier), перевірки синтаксису (ESLint) та підсвічування SCSS-файлів робить процес написання коду ефективнішим.
- Інтеграція з Git: вбудовані інструменти керування версіями дозволяють синхронізувати роботу з GitHub безпосередньо з інтерфейсу редактора.
- Термінал: наявність інтегрованого терміналу спрощує роботу з менеджерами пакетів (NPM/Yarn) та інструментом збірки Vite.

Важливим елементом технологічного стеку є використання Vite як інструменту збірки проєкту. У порівнянні з класичним Create React App, Vite забезпечує майже миттєве "гаряче перезавантаження" (Hot Module Replacement) під час написання коду, що значно підвищує продуктивність розробника.

Для оформлення стилів обрано препроцесор SCSS, оскільки він дозволяє використовувати змінні (для кольорової гами бренду кави), вкладеність та міксини, що робить CSS-код більш організованим та масштабованим порівняно зі стандартним CSS.

1.2. Огляд існуючих рішень та аналіз сайтів-аналогів у сфері онлайн-торгівлі кавовою продукцією

Для розробки успішного Frontend-рішення необхідно провести ґрунтовний аналіз існуючих ринкових пропозицій. Дослідження сайтів-аналогів дозволяє виявити найбільш ефективні патерни проектування користувацького інтерфейсу (UI), проаналізувати зручність взаємодії (UX) та визначити функціональні прогалини, які можна заповнити у власному проєкті. У сегменті продажу преміальної кави та чаю ключовими факторами є візуальна естетика, швидкість доступу до кошика та інформативність карток товарів.

1.2.1. Аналіз веб-ресурсу «The Tea»

Одним із найяскравіших представників нішевої електронної комерції в Україні є ресурс «The Tea». Хоча основна спеціалізація закладу зосереджена на чайній культурі, структура їхнього веб-сервісу є ідеальним прикладом для кавового інтернет-магазину через схожість бізнес-процесів: вибір за характеристиками, опис смакових профілів та акцент на естетиці споживання.

Візуальне виконання та UI: сайт виконаний у стилі мінімалізму з переважанням світлих тонів, що дозволяє акцентувати увагу на професійних фотографіях продукції. Це критично важливо для харчової галузі, де візуальне сприйняття безпосередньо

впливає на бажання здійснити покупку. Використання великих шрифтів та чіткої ієрархії заголовків робить інтерфейс читабельним.

Функціональні особливості та UX:

1. Глибока фільтрація: система дозволяє фільтрувати товар не лише за категоріями, а й за специфічними властивостями (витривалість, ефект, регіон походження). Це реалізовано через динамічні бічні панелі, що є гарним прикладом для реалізації за допомогою компонентів React.
2. Інформаційна насиченість: картка товару містить не лише ціну, а й історію походження, рекомендації щодо заварювання та відгуки.
3. Процес оформлення: кошик реалізований як випадаюче вікно, що дозволяє користувачеві продовжувати покупки без переходу на окрему сторінку, що значно знижує показник відмов.

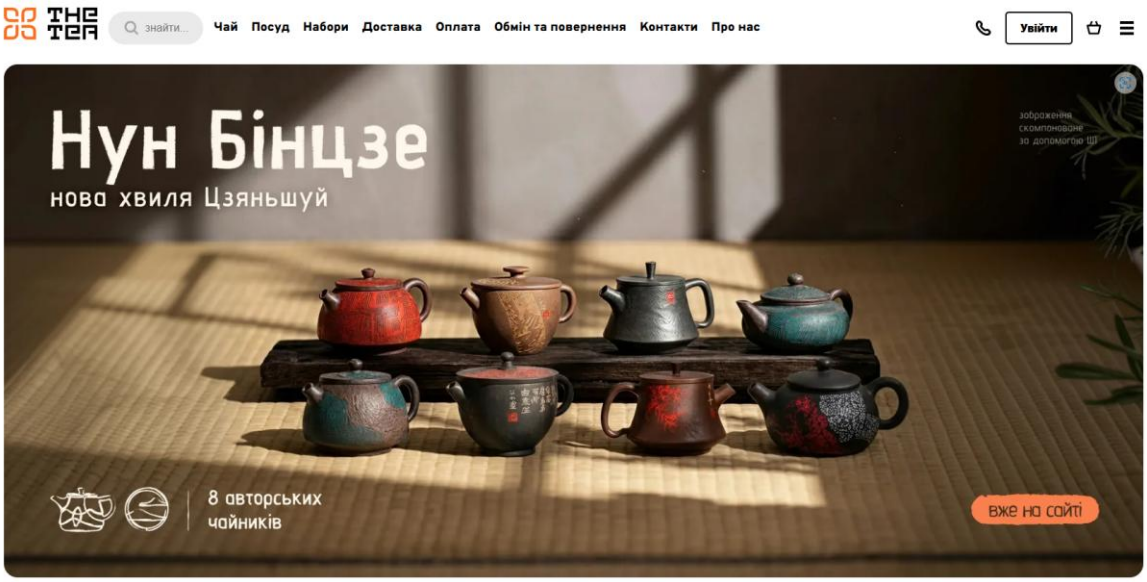


Рис. 1.1. Графічний інтерфейс “TheTea”

Джерело: https://thetea.ua

Недоліки для запозичення: Попри високу якість, сайт подекуди демонструє затримки при переході між категоріями через велику кількість статичного контенту, що завантажується класичним методом. У проєкті «Coffemaker» це буде вирішено завдяки архітектурі Single Page Application (SPA), де оновлюється лише частина екрана.

1.2.2. Аналіз веб-ресурсу «25 Coffee Roasters»

Другим об’єктом дослідження було обрано сервіс «25 Coffee Roasters» — одного з лідерів українського ринку обсмажування кави. Цей аналог є максимально наближеним до теми кваліфікаційної роботи за товарною номенклатурою та цільовою аудиторією.

Технологічні особливості: Сайт орієнтований на професійний підхід. Тут реалізовано інтерактивний помічник («Кавовий гід»), який допомагає клієнту обрати сорт кави на основі його вподобань. З точки зору Frontend-розробки, це складна логіка станів (State Management), яку в нашому проєкті ми реалізуємо за допомогою Redux Toolkit.

Ключові переваги:

1. Система підписки: сайт пропонує регулярну доставку кави, що потребує складного особистого кабінету користувача.
2. Адаптивність: ресурс демонструє високий рівень зручності на мобільних пристроях, що досягається завдяки використанню гнучких сіток (Grid systems).
3. Персоналізація: наявність розділу з історією замовлень та можливістю швидкого повтору покупки.

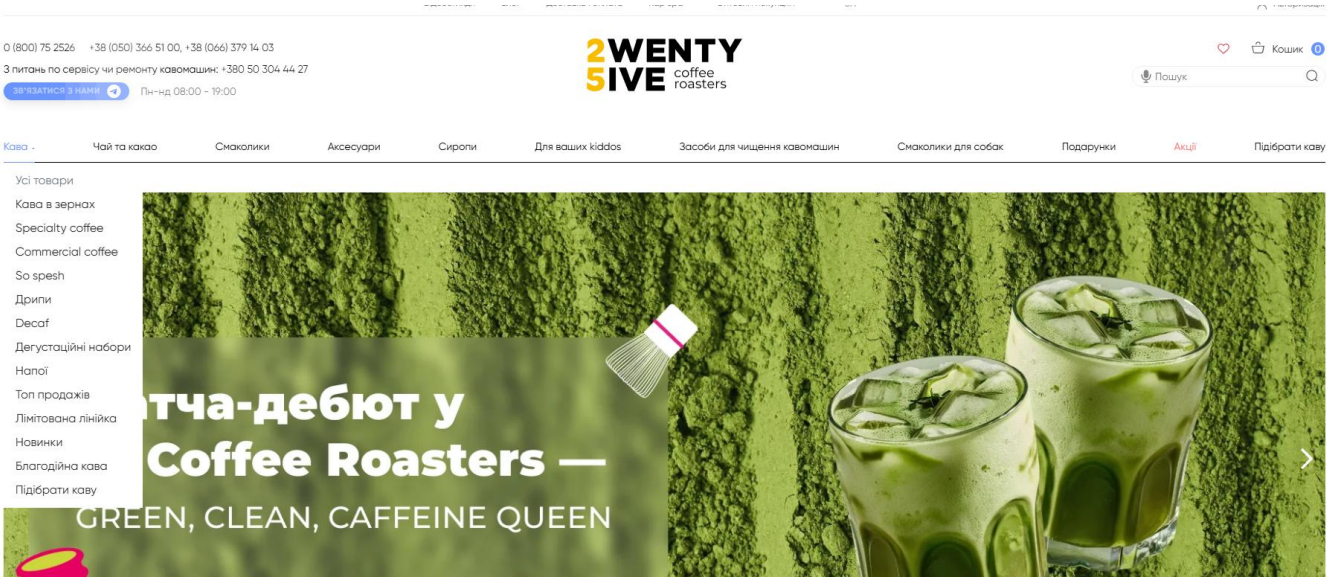


Рис. 1.2. Скріншот сторінки каталогу «25 Coffee Roasters»

Джерело: https://25coffeeroasters.ua

Недоліки для запозичення: незважаючи на потужний функціонал, аналіз сервісу «25 Coffee Roasters» виявив кілька критичних аспектів, які потребують оптимізації

при розробці нашого проєкту «Coffemaker». По-перше, через складну архітектуру та велику кількість інтерактивних елементів (кавовий гід, калькулятори), сайт демонструє значне навантаження на клієнтську частину, що призводить до повільного рендерингу на пристроях із середньою продуктивністю. Це часто є наслідком використання застарілих методів керування станом, які провокують зайві перемальовування (re-renders) всього дерева компонентів.

У нашому вебсервісі цю проблему буде нівельовано завдяки використанню Redux Toolkit та селекторів, що дозволяють оновлювати лише ті частини інтерфейсу, які безпосередньо залежать від змінених даних. По-друге, логіка «Кавового гіда» на аналізованому сайті є досить лінійною і не завжди дозволяє користувачеві швидко повернутися на попередній крок без втрати прогресу. В «Coffemaker» ми застосуємо більш гнучку логіку переходів, реалізовану через глобальний стан додатка, що забезпечить безшовний користувацький досвід та вищу швидкість взаємодії.

1.2.3. Аналіз веб-ресурсу «Coffee Door Brewbar & Roastery»

Третім об’єктом дослідження став веб-ресурс компанії «Coffee Door», яка позиціонує себе як професійна мережа брю-барів та обсмажувальників. Цей аналог є важливим з точки зору дослідження сучасних інтерфейсних рішень для мобільних користувачів.

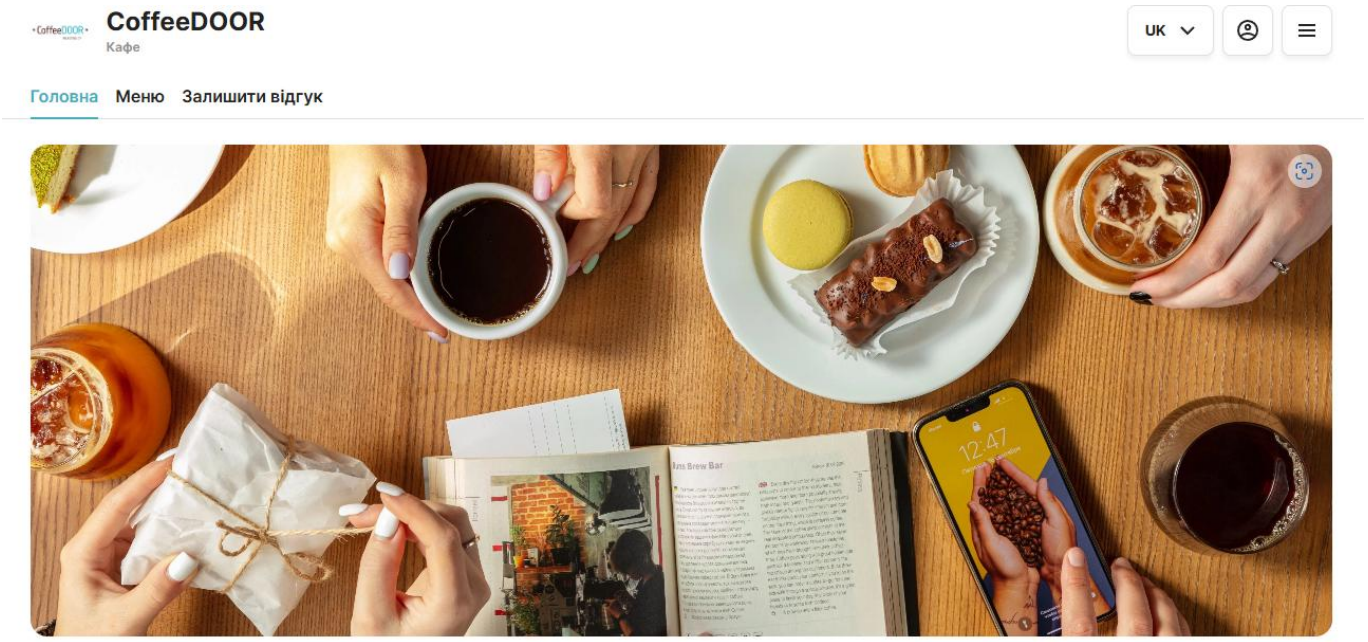
Візуальне виконання та UI: на відміну від світлого мінімалізму попередніх аналогів, «Coffee Door» використовує темну кольорову гаму, що асоціюється з преміальністю та атмосферою класичного кавового закладу. Використання високоякісного відео-контенту на головній сторінці створює ефект присутності, проте це створює додаткове навантаження на Frontend-частину в плані оптимізації швидкості завантаження.

Функціональні особливості та UX:

1. Мікро-взаємодії (Micro-interactions): сайт насичений плавними анімаціями при наведенні на картки товарів та переході між секціями. Для розробника

це сигнал про доцільність використання бібліотек анімацій у React (наприклад, Framer Motion), що ми також врахуємо у проєкті «Replica».

2. Динамічний пошук: реалізовано функцію «живого» пошуку, де результати з’являються миттєво під час введення тексту. Це критично важливий елемент для інтернет-магазинів з великим асортиментом, який вимагає грамотної обробки запитів (debouncing) на стороні клієнта.
3. Особистий кабінет: система лояльності інтегрована безпосередньо в інтерфейс покупок, дозволяючи бачити кількість накопичених бонусів у реальному часі.



Рисунку 1.3 — Скріншот інтерфейсу «Coffee Door»

Джерело: https://coffeedoor-brewbar.choiceqr.com/

Недоліки для запозичення: основним мінусом ресурсу є його «важкість». Через велику кількість скриптів та медіа-файлів первинне завантаження сторінки може тривати занадто довго. У нашому проєкті ми нівелюємо цю проблему за допомогою Vite, який розділяє код на частини (code splitting) та завантажує лише необхідні ресурси.

1.2.5. Висновки за підпунктом 1.2

Аналіз показав, що сучасні кавові сервіси рухаються в бік максимальної персоналізації та спрощення шляху клієнта до покупки. Основною проблемою аналогів залишається певна інертність інтерфейсів через використання застарілих методів збірки проєктів.

Використання ReactJS у поєднанні з Vite дозволить нашому проєкту забезпечити набагато вищу швидкість відгуку інтерфейсу. Реалізація централізованого керування станом через Redux Toolkit дозволить безшовно синхронізувати кошик, історію замовлень та персоналізовані пропозиції, що вигідно виділить наш продукт на фоні конкурентів. Особлива увага буде приділена адаптивності через React-Bootstrap, щоб клієнти могли замовляти каву «на ходу» з максимальною зручністю.

1.3. Постановка задачі

Для якісного управління замовленнями кави, необхідно розробити систему у вигляді вебзастосунку, завдяки якій буде здійснено контроль даних процесів. В першу чергу це оформлення замовлень, перегляд історії замовлень, відстеження доставки та персоналізовані пропозиції для клієнтів. Додатково це адміністрування роботи системи та збір статистичних даних для визначення важливих показників у роботі магазину кави. Оскільки статистичні дані можуть інформувати власника про популярність товарів, ефективність роботи системи та поведінку клієнтів. Проте це додатковий функціонал, який може бути реалізованим за потреби.

Розроблювана система в першу чергу орієнтована на клієнтів магазину, тому в процесі розробки потрібно зосередитися саме на цьому. Та наділити програму всіма необхідними візуальними та функціональними властивостями, які будуть зрозумілими кінцевому користувачеві. Мета розробки такої системи полягає в автоматизації процесу формування замовлень.

Клієнтська частина проєкту є важливою складовою для забезпечення взаємодії між користувачем та магазином кави. Адже саме правильно, а головне якісно створений користувацький інтерфейс є гарантованою запорукою успіху розроблюваної системи. Від зручності та візуальної складової проєкту залежить його

популярність, а разом з цим потреба в даній системі. До клієнтської частини програмного продукту, було висунуто ряд вимог:

1. Реєстрація та авторизація: кінцевий користувач мусить зареєструватися в системі для того, щоб користуватися послугами магазину. На основі обробки персональних даних, які надає користувач в процесі реєстрації, будуть формуватися його замовлення. Якщо користувач не бажає проходити реєстрацію, замовлення можна буде здійснити в анонімному режимі.
2. Доступ до каталогу товарів: як анонімний, так і авторизований користувач має мати можливість переглядати каталог товарів магазину, як в цілях формування замовлень, так і для ознайомлення.
3. Здійснення замовлень: авторизований в системі користувач має мати можливість здійснити замовлення товарів магазину дистанційно. Неавторизований користувач матиме можливість замовити товари лише в межах магазину.
4. Безпека особистих даних користувача: інформація кожного клієнта має бути збережена і захищена від пошкоджень та крадіжки. Для цього програмний продукт має зберігати пароль користувача в зашифрованому вигляді. Всі паролі будуть збережені у базі даних з використанням алгоритму одностороннього шифрування.
5. Зручний користувацький інтерфейс: інтерфейс користувача має бути простим для розуміння та легким у користуванні. Стиль сайту має виглядати сучасним та привабливим.

Клієнтська частина надає інтерфейс для користувачів, проте обробка запитів та власне робота з даними виконується на серверній частині проєкту. Тому також важливо розробити та організувати правильну роботу серверної частини проєкту. Використати всі необхідні технології та засоби розробки, для забезпечення стабільної та безвідмовної роботи сервера. Тому, також було висунуто вимоги до серверної частини проєкту:

1. Реалізація правильної архітектури: для того, щоб забезпечити якісну роботу застосунку, рекомендується використовувати розділену архітектуру з

чітким розділенням логіки серверної частини від клієнтської. Правильне архітектурне рішення дозволить легко розширювати систему в майбутньому. Це означає, що можна додавати нові модулі та функціонал без потреби в переписуванні значної частини коду.

2. Комунікація з клієнтською частиною: для взаємодії клієнтської та серверної частини потрібно використати один з найпоширеніших протоколів передачі даних, а саме протокол передачі гіпертексту (HTTP). Клієнтська частина може відправляти різні типи запитів HTTP до серверної частини, такі як GET, POST, PUT, DELETE.
3. Зберігання даних: потрібно обрати технологію для зберігання даних, щоб вся необхідна для користувача інформація була доступною в реальному часі. Для цього потрібно реалізувати базу даних, спроєктувати її архітектуру. Доцільно використати SQL або NoSQL бази даних.
4. Безпека: використання технологій авторизації та автентифікації користувача, для забезпечення безпеки даних та конфіденційності. Доцільно використати технологію JWT (JSON Web Tokens) для безпечного обміну інформацією між клієнтською та серверною частинами проєкту.
5. Управління помилками: необхідно реалізувати механізми для обробки помилок та повідомлень про помилки на серверній стороні. Використати зручність інструментів, таких як middleware або фільтри, для перехоплення та обробки помилок.

В процесі роботи, ми здійснили аналіз предметного середовища. Проаналізували популярні фреймворки для розробки веб-застосунків, виділили особливості та переваги кожного з них. Навели детальний опис особливостей використання даних фреймворків, та прийняли рішення використати для розробки системи управління замовленнями магазину кави технологію ReactJS. Адже враховуючи переваги, які надає ця технологія, системі буде гарантовано швидкодію, та зручність у використанні. Технологія ReactJS є мультиплатформною і гарантує стабільну роботу в незалежності від операційної системи пристрою на якому буде запущено наш веб-застосунок.

Ми встановили вимоги до клієнтської та серверної частини проєкту, виконання всіх вимог гарантує якісний результат. Система отримає всі функціональні особливості, які необхідні для стабільності та швидкодії програмного продукту. При цьому всі частини проєкту будуть незалежними та при необхідності можуть бути заміненими на інші.

РОЗДІЛ 2. ПРИКЛАДНЕ ЗАСТОСУВАННЯ FRONTEND-ТЕХНОЛОГІЙ ДЛЯ СТВОРЕННЯ ІНТЕРНЕТ-МАГАЗИНУ КАВИ

2.1. Аналіз процесів продажу кави

Використання інформаційних технологій у сфері продажу кави є важливим кроком до автоматизації та покращення взаємодії з клієнтами. Сучасні вебзастосунки дозволяють значно спростити процес замовлення, забезпечити прозорість логістики та підвищити рівень обслуговування. У нашому проєкті ми розглядаємо процеси, які відбуваються під час оформлення замовлення, обробки даних клієнтів, доставки товарів та взаємодії з користувачами.

2.1.1. Аналіз процесів оформлення замовлень

Системний аналіз процесів оформлення замовлення є фундаментальним етапом у життєвому циклі розробки програмного забезпечення для сегмента e-commerce, зокрема у сфері автоматизації продажу кавової продукції. Процес проектування логіки взаємодії користувача з інтерфейсом потребує глибокого вивчення поведінкових факторів та алгоритмізації кожного кроку, що здійснює клієнт. Це необхідно для створення безбар’єрного середовища, яке мінімізує когнітивне навантаження на користувача та забезпечує високу конверсію відвідувачів у покупців.

Центральним суб’єктом системи є кінцевий споживач (клієнт), чия взаємодія із вебзастосунком має базуватися на принципах інтуїтивності та функціональної повноти. В умовах сучасного ринку, де швидкість прийняття рішення є критичною, програмний продукт повинен не просто надавати перелік товарів, а виступати інтелектуальним посередником між складом продукції та потребою клієнта. Для досягнення цієї мети інтерфейс має бути адаптивним, забезпечуючи коректне відображення на різних типах пристроїв (Responsive Design), та швидкодіючим, щоб

процес вибору кави, додавання позицій до віртуального кошика та введення персональних даних проходив без технічних затримок.

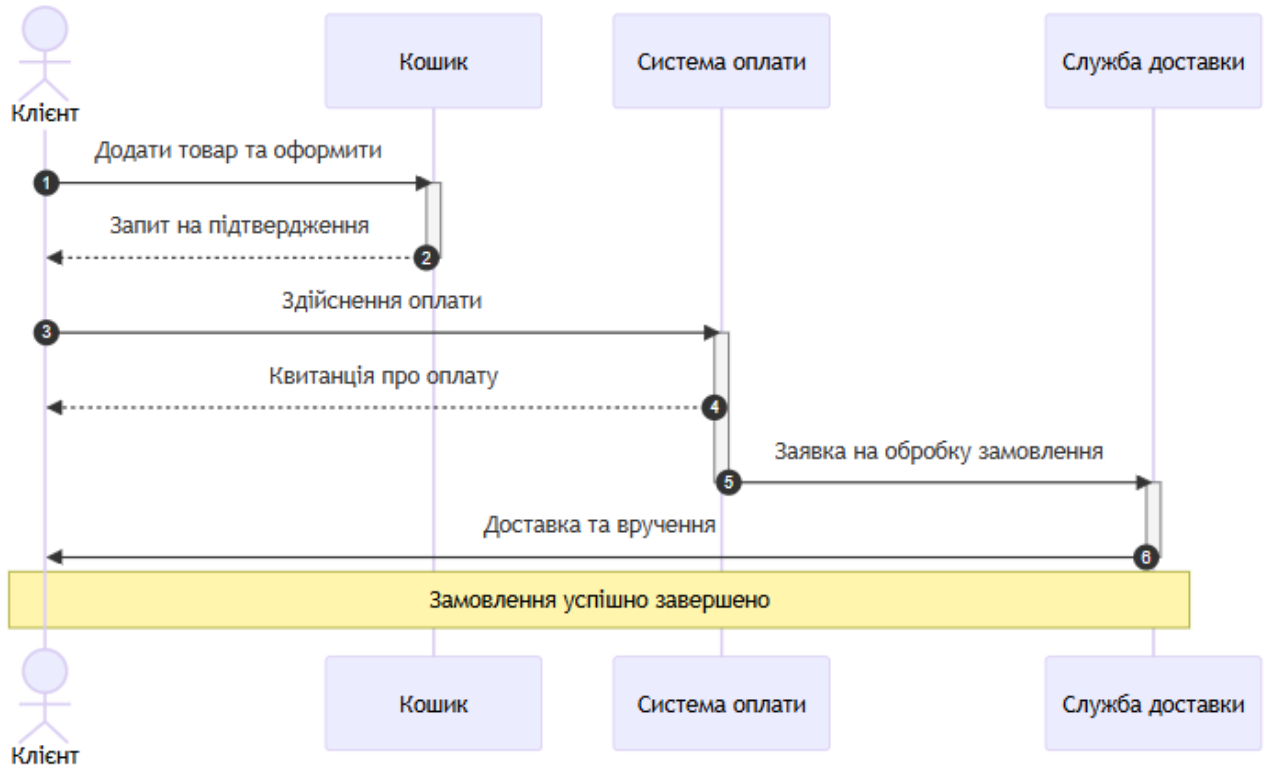


Рис. 2.1. Uml діаграма процесу оформлення замовлення.

Джерело: створено AI на основі промпту автора

Детальна декомпозиція процесу оформлення замовлення дозволяє виділити низку критично важливих стадій, кожна з яких потребує програмної валідації та обробки виключень:

1. Формування пре-чеку та робота з кошиком. На цьому етапі клієнт здійснює селекцію товарів з інтерактивного каталогу. Система має забезпечувати динамічне оновлення стану кошика без перезавантаження сторінки. Окрім кількісних характеристик, важливо враховувати якісні параметри (наприклад, ступінь помелу кави або об’єм пакування), що безпосередньо впливають на фінальну вартість.
2. Верифікація складських залишків. Автоматизований аналіз наявності товарів у режимі реального часу є запорукою уникнення конфліктних ситуацій. У випадку відсутності певної позиції, алгоритм системи повинен реалізувати механізм пропонування альтернативних варіантів (Cross-selling

та Up-selling методики), що базуються на попередніх уподобаннях користувача або популярності схожих сортів кави.

3. Логістичний блок та збір персональних даних. Введення інформації для доставки є найбільш вразливим етапом з точки зору відмови користувача від покупки. Тому система має підтримувати інтеграцію з API поштових сервісів для автоматичного вибору відділень або валідації адрес. Це дозволяє скоротити час на заповнення форм та знизити ймовірність помилкового введення даних.
4. Фінансові транзакції та безпека. Вибір способу оплати (готівковий або безготівковий розрахунок) потребує інтеграції з надійними платіжними шлюзами. Важливо забезпечити шифрування даних та підтримку протоколів безпеки, щоб клієнт відчував захищеність своїх фінансових активів.
5. Фіналізація та зворотний зв’язок. Після підтвердження замовлення система ініціює ряд фонових процесів: генерацію електронного чека, зміну статусу товарів у базі даних, надсилання нотифікацій (SMS, Email або Push-повідомлення) та передачу сформованого логістичного листа до служби збирання замовлень.

Окремим аспектом, що підвищує рівень лояльності клієнтів, є функціональна гнучкість системи. Це передбачає можливість ретроспективного редагування замовлення. Клієнт повинен мати змогу повернутися на будь-який з попередніх кроків для корекції номенклатури товарів, зміни платіжних реквізитів або уточнення адресата без втрати вже введеної інформації. Такий підхід базується на принципах толерантності до помилок (Error Tolerance) та дозволяє значно оптимізувати користувацький досвід, роблячи процес придбання кави максимально комфортним та технологічно досконалим.

2.1.2. Організація та структура інформаційного масиву про клієнтів

У сучасних умовах цифровізації ритейлу формування цілісного та структурованого профілю клієнта є стратегічним завданням для будь-якої інформаційної системи. В межах розробленого застосунку для реалізації кавової продукції, збір та систематизація персональних даних виступає не лише як механізм ідентифікації, а й як фундамент для побудови довгострокових лояльних відносин між брендом та споживачем. Якісне наповнення бази даних дозволяє трансформувати абстрактного відвідувача у конкретний об’єкт маркетингового аналізу, що має унікальний набір характеристик та потреб.

Для забезпечення повноцінного функціонування системи та реалізації високого рівня сервісу, архітектура бази даних передбачає акумуляцію наступних інформаційних блоків:

- Персональна ідентифікація (ПІБ клієнта). Використовується для персоналізації інтерфейсу та офіційного звернення у межах автоматичних розсилок, електронних чеків та систем підтримки. Це формує психологічний аспект довіри до сервісу.
- Верифікований контактний номер телефону. Виступає як первинний ідентифікатор у системі (логін) та ключовий канал для оперативного зв'язку. Номер телефону необхідний для двофакторної автентифікації, надсилання SMS-повідомлень про статус готовності замовлення та координації дій із кур’єрською службою.
- Контактна електронна пошта (E-mail). Окрім функції дублювання фінансової інформації (надсилання інвойсів), електронна пошта є каналом для реалізації стратегії контент-маркетингу: ознайомлення клієнта з новими сортами кави, технічними оновленнями застосунку та наданням детальних звітів про накопичені бонуси.
- Географічна прив’язка (Адреса доставки). Структуровані дані про місцезнаходження клієнта дозволяють автоматизувати логістичні маршрути, розраховувати вартість та час доставки у режимі реального часу,

а також проводити геомаркетингові дослідження для визначення зон найбільшої активності споживачів.

- Ретроспективний аналіз (Історія замовлень). Цей блок даних містить хронологічний реєстр усіх транзакцій, що дозволяє системі розраховувати показник LTV (Lifetime Value) — сукупний прибуток від клієнта за весь час взаємодії. Аналіз історії є базою для функцій «повторного замовлення в один клік».
- Профіль уподобань та споживча поведінка. Сюди належать дані про улюблені сорти кави, типи обсмаження, переважні методи приготування та регулярність покупок. Використання цих даних дозволяє алгоритмам машинного навчання формувати персоналізовані рекомендації, що значно підвищує середній чек.
- Додаткові комунікаційні ідентифікатори (соціальні мережі, месенджери). Інтеграція з профілями в соціальних мережах дозволяє реалізувати омніканальний підхід, де клієнт може отримувати унікальні акційні пропозиції там, де йому зручно, що підвищує загальну залученість у взаємодію із платформою.

2.1.3. Структура та функціональна специфікація даних про замовлення

Процес трансформації споживчого вибору у завершену транзакцію вимагає створення комплексної інформаційної моделі замовлення. У структурі розроблюваної системи об’єкт «Замовлення» виступає центральною сполучною ланкою між клієнтським інтерфейсом, складською базою даних та логістичними модулями. Коректна детермінація параметрів замовлення є критичною для забезпечення цілісності даних (Data Integrity) та мінімізації операційних ризиків під час автоматизованої обробки звернень.

Для забезпечення прозорості бізнес-процесів та реалізації механізмів контролю на кожному етапі життєвого циклу замовлення, система акумулює та оперує наступним переліком атрибутів:

- Унікальний ідентифікатор клієнта (Customer ID). Цей параметр забезпечує реляційний зв’язок між замовленням та персональним профілем користувача. Він дозволяє системі миттєво асоціювати покупку з конкретною особою, підтягуючи дані про її лояльність, персональні знижки та контактну інформацію без необхідності повторного введення.
- Специфікація товарних позицій (Order Items). Це деталізований масив даних, що включає перелік обраних сортів кави, їхнє фасування, кількість одиниць та ціну на момент фіксації замовлення. Даний блок є основою для автоматичного списання товарів із віртуального складу та формування накладної для збірника.
- Агрегована вартість замовлення (Total Amount). Включає не лише суму цін усіх обраних товарів, а й автоматичний розрахунок податків, вартості пакування та логістичних послуг. Фіксація фінальної вартості є юридично важливою для розрахунків із платіжними системами.
- Динамічний статус замовлення (Order Lifecycle Status). Система використовує модель скінченних автоматів для переходу між станами: «Очікується», «В обробці», «Підтверджено», «Сформовано», «Передано кур’єру», «Доставлено». Це дозволяє реалізувати функцію моніторингу замовлення в режимі реального часу, що є критичним фактором для користувацького досвіду (UX).
- Текстові коментарі та специфічні вимоги. Поле для неструктурованих даних, де клієнт може зазначити особливі побажання (наприклад, нюанси щодо помелу кави, код від під’їзду або прохання не телефонувати для підтвердження). Це підвищує гнучкість сервісу та його адаптивність до індивідуальних потреб.
- Метод фінансового розрахунку (Payment Gateway Info). Системна інформація про обраний шлюз оплати. Окрім зручності для клієнта, ці дані необхідні для бухгалтерського обліку та автоматичного звіряння дебіторської заборгованості.

- Тайм-менеджмент доставки. Параметризація часу (бажаний інтервал або точний час) дозволяє інтелектуальному модулю логістики оптимізувати маршрути кур’єрів, уникати «пікових» перевантажень та забезпечувати виконання замовлення у межах регламентованого SLA (Service Level Agreement).
- Контрольний номер відстеження (Tracking Number). Унікальний цифровий або буквено-цифровий код, що генерується системою або інтегрується із зовнішніми службами доставки. Він є ключовим інструментом для самостійного контролю замовлення клієнтом, що суттєво знижує навантаження на службу підтримки.

Інтеграція цих даних у єдиний реєстр дозволяє досягти високого рівня автоматизації управління замовленнями (Order Management System). Наявність детальної інформації про кожен етап проходження замовлення мінімізує вплив людського фактора та виключає можливість втрати інформації. Наприклад, наявність історії статусів та коментарів стає незамінною у випадку виникнення спірних ситуацій, дозволяючи провести ретроспективний аналіз дій як системи, так і персоналу.

Крім того, накопичена інформація про замовлення стає цінним ресурсом для прогнозної аналітики. Аналізуючи часові проміжки між замовленнями, середню вартість чека та популярні комбінації товарів, система може генерувати автоматичні звіти для менеджменту, що сприяє прийняттю обґрунтованих рішень щодо розширення асортименту, впровадження сезонних акцій або оптимізації логістичних витрат. Таким чином, інформаційна модель замовлення є не просто технічним описом покупки, а стратегічним інструментом забезпечення життєздатності та масштабованості вебзастосунку.

2.1.5. Логістика та доставка

Логістика є невід'ємною частиною процесу продажу кави. Для забезпечення ефективної доставки необхідно враховувати такі аспекти:

1. Оптимізація маршрутів доставки
2. Відстеження статусу замовлень у реальному часі
3. Інтеграція з логістичними службами
4. Забезпечення зворотного зв'язку з клієнтами
5. Автоматизація процесу пакування замовлень

Система повинна автоматично розраховувати оптимальні маршрути доставки на основі адрес клієнтів, а також надавати клієнтам можливість відстежувати статус їхніх замовлень у режимі реального часу. Крім того, інтеграція з логістичними службами дозволить автоматизувати процес передачі замовлень на доставку та зменшити кількість помилок. Зворотний зв'язок з клієнтами, наприклад, через SMS або електронну пошту, допоможе підвищити рівень довіри та задоволеності клієнтів. Автоматизація пакування замовлень дозволить зменшити час обробки замовлень та підвищити ефективність роботи складу.

2.1.6. Висновки до розділу

У результаті аналізу процесів продажу кави було визначено основні вимоги до системи, які забезпечать її ефективну роботу. Реалізація цих вимог дозволить створити зручний та безпечний вебзастосунок для продажу кави, який відповідатиме потребам клієнтів та забезпечить автоматизацію основних бізнес-процесів. Крім того, система сприятиме підвищенню рівня обслуговування, зменшенню кількості помилок та оптимізації логістичних процесів. Впровадження таких рішень дозволить не лише покращити взаємодію з клієнтами, але й підвищити конкурентоспроможність компанії на ринку.

2.2. Розробка архітектури та вибір технологій

Для створення ефективної системи продажу кави необхідно ретельно спроєктувати архітектуру та обрати відповідний стек технологій. Це дозволить забезпечити високу продуктивність, масштабованість та зручність використання системи. У цьому розділі ми розглянемо основні аспекти архітектурного рішення та обґрунтуємо вибір технологій для реалізації проєкту.

Ось розширений варіант розділу 2.2, оформлений у науковому стилі з використанням професійної термінології. Текст значно збільшено в об’ємі за рахунок деталізації архітектурних принципів та обґрунтування вибору кожної технології.

2.2.1. Архітектурна модель та системне проектування

Архітектура програмного забезпечення є фундаментальною концепцією, що визначає структурну організацію системи, її компоненти, їхні зовнішні властивості та взаємозв'язки між ними. Для реалізації вебзастосунку з продажу кави було обрано багаторівневу архітектуру (N-tier architecture), яка базується на принципі розділення відповідальності (Separation of Concerns). Такий підхід дозволяє ізолювати логіку представлення від бізнес-логіки та механізмів збереження даних, що суттєво полегшує масштабування, тестування та подальшу підтримку продукту.

Детальний аналіз обраної моделі дозволяє виділити наступні логічні рівні:

- Рівень презентації (Presentation Layer). Даний рівень є зовнішнім інтерфейсом системи, з яким безпосередньо взаємодіє кінцевий користувач. Його основне завдання — візуалізація даних, отриманих від сервера, та збір вхідної інформації (запитів) від клієнта. Використання сучасних фронтенд-технологій дозволяє реалізувати цей рівень як Single Page Application (SPA), що забезпечує високу швидкість відгуку та відсутність потреби у повному перезавантаженні сторінок при кожній дії.
- Рівень бізнес-логіки (Business Logic Layer / Service Layer). Це ядро системи, де зосереджені основні алгоритми обробки інформації. Тут реалізуються правила валідації замовлень, розрахунок вартості з урахуванням дисконтних

програм, управління правами доступу та координація транзакцій. Рівень бізнес-логіки виступає посередником, що інтерпретує команди користувача у конкретні операції над даними.

- Рівень доступу до даних (Data Access Layer). Відповідає за персистентність (постійне зберігання) інформації. Цей рівень інкапсулює логіку взаємодії із системою управління базами даних (СУБД). Вибір між реляційною моделлю (SQL) для структурованих замовлень та нереляційною (NoSQL) для гнучких профілів клієнтів дозволяє оптимізувати швидкість запитів та забезпечити надійність зберігання критично важливої інформації.

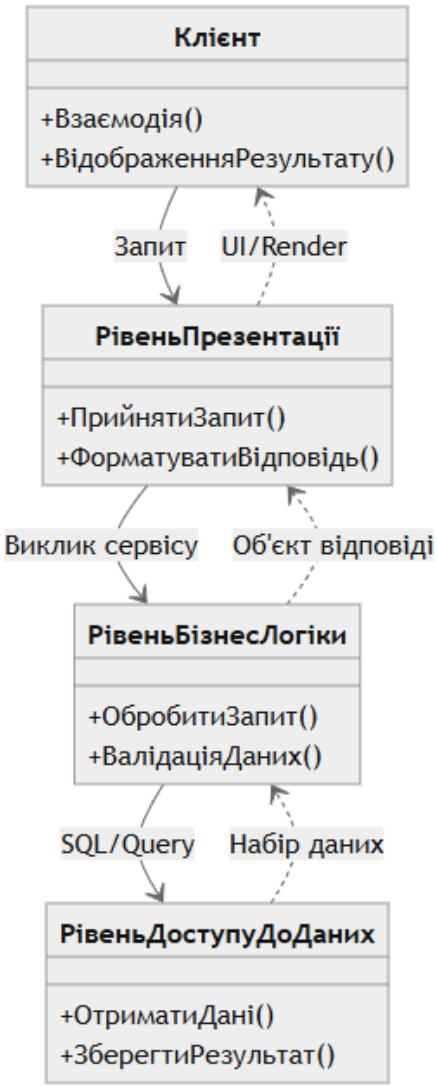


Рис. 2.2. UML Діаграма архітектурних рівнів.

Джерело: створено AI на основі промпту автора

2.2.2. Технологічний стек та обґрунтування вибору інструментарію

Для реалізації технічного рішення було обрано стек технологій, що відповідає сучасним стандартам індустрії за критеріями продуктивності, безпеки та швидкості розробки. Кожен інструмент було інтегровано в екосистему проєкту на основі його функціональних переваг:

1. Бібліотека ReactJS. Обрана як основний інструмент для побудови користувацьких інтерфейсів завдяки декларативному підходу та використанню технології Virtual DOM. Це дозволяє мінімізувати кількість прямих маніпуляцій з деревом документів (DOM), що критично важливо для плавності анімацій та швидкодії інтерфейсу при великій кількості товарних позицій.
2. Державний менеджмент Redux Toolkit (RTK). Для великих застосунків управління станом (State Management) стає складним завданням. RTK забезпечує єдине «джерело істини» для всіх даних у застосунку, спрощує логіку оновлення інформації про кошик та профіль користувача, а також надає потужні інструменти для відлагодження коду.
3. Інструмент збірки Vite. На відміну від застарілих рішень, Vite використовує нативні ES-модулі браузера, що забезпечує миттєвий запуск сервера розробки (Hot Module Replacement). Це суттєво оптимізує процес написання коду та фінальну оптимізацію (bundling) проєкту перед деплоєм.
4. Методологія стилізації SCSS. Використання препроцесора дозволяє впровадити принципи модульності у CSS-код. Завдяки змінним, міксинам та вкладеним правилам створюється гнучка дизайн-система, де зміна одного параметра (наприклад, корпоративного кольору) автоматично оновлює весь інтерфейс.
5. Фреймворк React-Bootstrap. Виступає базою для створення адаптивного дизайну. Використання готових протестованих компонентів дозволяє зосередитися на бізнес-логіці, гарантуючи коректне відображення інтерфейсу на мобільних пристроях, планшетах та десктопах.

6. Середовище виконання Node.js. Обрано для серверної частини через його подієво-орієнтовану архітектуру та неблокуючий ввід/вивід (Non-blocking I/O). Це дозволяє серверу обробляти велику кількість одночасних запитів від клієнтів із мінімальними витратами ресурсів.
7. Безпека та JWT (JSON Web Tokens). Для забезпечення захищеного доступу реалізовано механізм автентифікації на основі токенів. JWT дозволяє передавати інформацію про користувача у зашифрованому вигляді, що виключає потребу зберігати сесії на сервері (Stateless architecture) та підвищує загальну стійкість системи до атак.

2.2.3. Стратегія інтеграції компонентів та забезпечення системної цілісності

Інтеграція компонентів у межах розробленого веб-застосунку є комплексним процесом об'єднання програмних модулів у єдину, злагоджену систему. Цей етап передбачає не лише налагодження технічного зв’язку між фронтенд- та бекенд-частинами, а й забезпечення семантичної узгодженості даних, що циркулюють у системі. Основним завданням інтеграційного процесу є створення безвідмовної інфраструктури, де кожен компонент функціонує як частина цілісного організму, мінімізуючи ризики виникнення системних конфліктів.

Для реалізації високоефективної інтеграції було застосовано наступні методологічні підходи:

- Проектування та впровадження прикладного програмного інтерфейсу (API). В основі взаємодії між клієнтською частиною (ReactJS) та сервером (Node.js) покладено архітектурний стиль REST (Representational State Transfer). Це дозволяє стандартизувати запити за допомогою протоколу HTTP (GET, POST, PUT, DELETE) та забезпечити обмін даними у форматі JSON. Такий підхід гарантує слабку пов’язаність (loose coupling) компонентів, що дає змогу модифікувати серверну логіку без необхідності докорінної перебудови інтерфейсу користувача.

- Принципи модульності та компонентно-орієнтованого проектування. Розробка системи базується на принципі високої внутрішньої згуртованості (high cohesion). Кожен функціональний елемент — від модуля кошика до системи обробки платежів — проектується як автономна одиниця із чітко визначеним інтерфейсом взаємодії. Це не лише спрощує процес паралельної розробки, а й закладає фундамент для рефакторингу та повторного використання коду в інших модулях або майбутніх версіях системи.
- Верифікація та валідація через багаторівневе тестування. Для гарантування стабільності інтегрованої системи впроваджено стратегію автоматизованого тестування. Вона включає Unit-тести (перевірка окремих функцій та компонентів), інтеграційні тести (контроль коректності взаємодії між модулями) та End-to-End (E2E) тести, які імітують реальний шлях клієнта від вибору кави до фінальної оплати. Такий підхід дозволяє виявляти регресійні помилки на ранніх етапах розробки та забезпечує відповідність коду критеріям якості.

2.2.4. Висновки до розділу

Підсумовуючи результати проведеного дослідження та проектування, викладені у другому розділі, можна зробити наступні узагальнюючі висновки:

1. Проведено детальний аналіз процесів оформлення замовлень, що дозволило декомпозувати складну взаємодію користувача із системою на послідовність логічних етапів. Це заклало підґрунтя для проектування інтуїтивно зрозумілого інтерфейсу, що мінімізує помилки користувача та оптимізує шлях до здійснення покупки.
2. Визначено структуру інформаційних масивів щодо клієнтів та замовлень. Впровадження розгалуженої системи збору даних дозволяє реалізувати персоналізований підхід, забезпечити високу швидкість обробки

транзакцій та створити базу для подальшої бізнес-аналітики й прогнозування попиту.

3. Обґрунтовано вибір багаторівневої архітектури, яка забезпечує необхідну масштабованість та гнучкість вебзастосунку. Розподіл системи на рівень презентації, бізнес-логіки та доступу до даних дозволяє ефективно керувати складністю проекту та забезпечує його стійкість до високих навантажень.
4. Сформовано сучасний технологічний стек (ReactJS, Node.js, Redux Toolkit, JWT), який є оптимальним для створення високопродуктивних веб-рішень. Обрані інструменти дозволяють реалізувати складну функціональність при збереженні високої швидкості розробки та забезпеченні належного рівня безпеки персональних даних.
5. Запропонована стратегія інтеграції та тестування компонентів гарантує технічну цілісність системи. Використання стандартизованих API та модульного підходу робить систему адаптивною до майбутніх змін ринкових умов або технічних вимог.

Таким чином, сформована теоретична та архітектурна база є достатньою для переходу до етапу практичної реалізації програмного продукту, гарантуючи його відповідність сучасним вимогам до якості, безпеки та зручності використання у сфері електронної комерції.

2.3. Забезпечення безпеки даних та комплексні алгоритмічні рішення у системі

У сучасну епоху стрімкого розвитку інформаційних технологій питання кібербезпеки та захисту персональних даних перетворюється з додаткової опції на фундаментальну вимогу до будь-якого програмного продукту. Для вебзастосунку, орієнтованого на сферу електронної комерції та продаж кавової продукції, захист конфіденційної інформації клієнтів є критичним, оскільки система оперує не лише технічними даними, а й персональними ідентифікаторами, фінансовими

вподобаннями та логістичними адресами. Реалізація багаторівневої системи захисту дозволяє мінімізувати ризики несанкціонованого доступу, витоку інформації та компрометації облікових записів користувачів. У цьому контексті особлива увага приділяється інтеграції сучасних криптографічних стандартів, використанню безпечних протоколів передачі даних та впровадженню алгоритмів, що забезпечують цілісність та автентичність інформації на кожному етапі її життєвого циклу.

Одним із наріжних каменів архітектури безпеки нашого проєкту є використання технології JSON Web Tokens (JWT) для реалізації механізмів автентифікації та авторизації. На відміну від традиційних сесій, що зберігаються на сервері, JWT дозволяє створити безстанну (stateless) систему, де вся необхідна інформація про права доступу користувача інкапсульована в самому токені. Процес починається з моменту успішної перевірки облікових даних користувача на стороні сервера, після чого генерується унікальний цифровий підпис. Цей токен передається клієнту та зберігається у безпечному сховищі браузера. Під час кожного наступного запиту до захищених ресурсів системи фронтенд-частина автоматично додає цей токен до заголовка Authorization, що дозволяє серверу миттєво валідувати особу користувача без повторного звернення до бази даних. Візуалізація цього процесу наведена нижче.

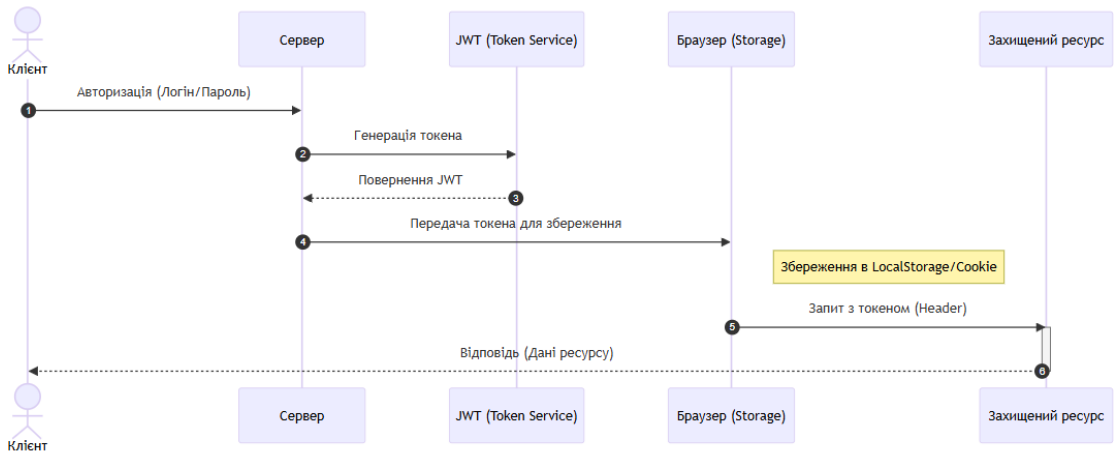


Рис. 2.3. Алгоритм автентифікації користувача за допомогою технології JWT

Джерело: створено AI на основі промпту автора

Окрім контролю доступу, першочерговим завданням є надійне зберігання паролів користувачів. Зберігання паролів у відкритому вигляді є неприпустимим згідно з будь-якими стандартами безпеки. Тому в нашій системі інтегровано алгоритм одностороннього криптографічного хешування SHA-256 (Secure Hash Algorithm 256-bit). Суть цього методу полягає у перетворенні пароля будь-якої довжини у фіксований рядок символів, який неможливо реверсувати. Навіть у малоймовірному випадку отримання зловмисником доступу до бази даних, він побачить лише набір хеш-сум, які не дозволять йому дізнатися оригінальні паролі. Процедура обробки та звірки пароля відображена на наступній схемі.

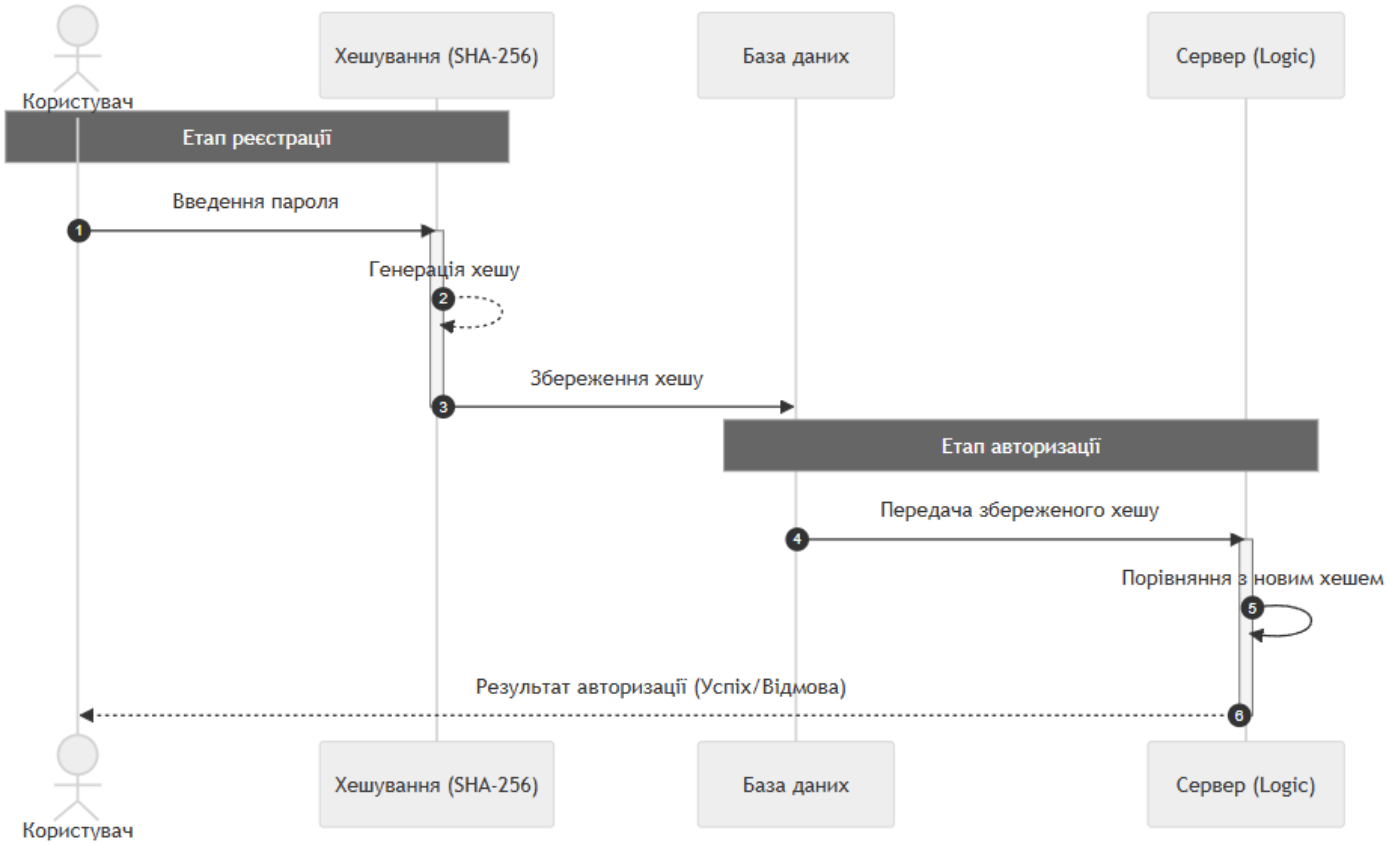


Рис. 2.4. Модель процесу хешування та верифікації пароля користувача

Джерело: створено AI на основі промпту автора

Безпека фронтенд-частини застосунку реалізується через комплексний підхід, який починається з обов'язкового використання протоколу HTTPS. Це забезпечує надійне шифрування всього трафіку між клієнтом і сервером, унеможливлюючи перехоплення даних за допомогою атак типу «Man-in-the-Middle». Додатково на стороні клієнта впроваджено жорстку валідацію та санітизацію всіх вхідних даних.

Це є необхідним заходом для запобігання поширеним вразливостям, таким як SQL-ін'єкції або міжсайтовий скриптинг (XSS). Система автоматично перевіряє відповідність форматів полів та блокує будь-які спроби впровадження шкідливого коду. Також реалізована рольова модель доступу (RBAC), яка гарантує, що звичайний клієнт ніколи не отримає доступу до адміністративної панелі або чутливих аналітичних даних системи.

Процес інтеграції цих алгоритмічних рішень у загальну архітектуру здійснюється через стандартизовані API-запити. Фронтенд виступає активним учасником циклу безпеки, ініціюючи запити на авторизацію та коректно обробляючи відповіді сервера. Коли користувач вводить свої дані, вони у зашифрованому вигляді надсилаються на сервер, де проходять процедуру верифікації. У разі позитивного результату система генерує JWT-токен, який стає «цифровим ключем» для всіх подальших операцій. Загальна логіка інтеграції безпекових компонентів представлена на рисунку.

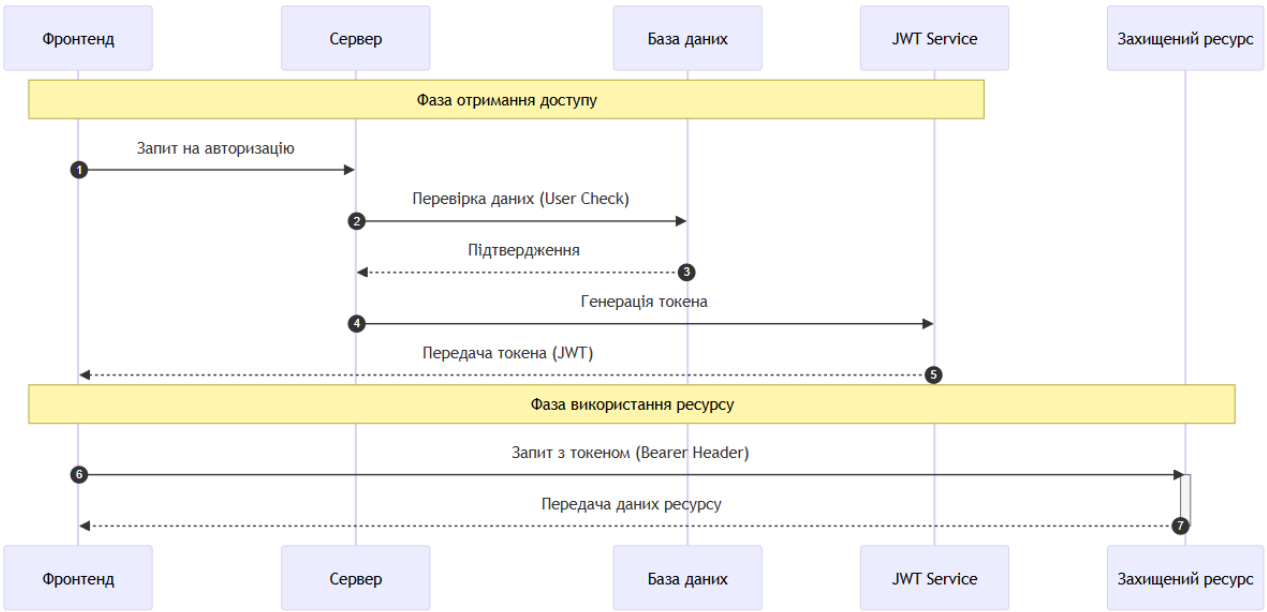


Рис. 2.5. Схема інтеграції алгоритмів безпеки у фронтенд-частину системи

Джерело: створено AI на основі промпту автора

Завершуючи огляд архітектури безпеки, слід підкреслити, що обрана комбінація технологій JWT, SHA-256 та HTTPS у поєднанні з ретельною валідацією на фронтенді формує стійку до зовнішніх загроз екосистему. Використання цих підходів не лише відповідає сучасним галузевим стандартам розробки програмного забезпечення, а й створює фундамент довіри між споживачем та сервісом. Клієнт може бути впевнений, що його персональні дані та фінансова історія надійно захищені від будь-яких форм кіберзлочинності.

Висновки до розділу

У межах другого розділу було проведено всебічне теоретичне дослідження та проектування архітектурно-технологічного базису системи. Аналіз процесів оформлення замовлень та деталізація інформаційних структур щодо клієнтів і товарних позицій дозволили сформувати чітку модель даних, що відповідає реальним потребам бізнесу у сфері електронної комерції. Запропонована багаторівнева архітектура забезпечує необхідну гнучкість та розділення відповідальності між компонентами, що є ключовим для подальшого масштабування проекту. Особлива увага була приділена питанням безпеки: впровадження протоколів JWT для автентифікації та алгоритмів SHA-256 для захисту паролів у поєднанні з HTTPS-шифруванням створює захищене середовище для обробки персональних даних. Таким чином, результати цього розділу складають завершену теоретичну та інженерну основу, яка дозволяє перейти до практичної реалізації програмного продукту з повною відповідністю встановленим технічним і безпековим вимогам.

РОЗДІЛ 3. ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ФРОНТЕНД ЧАСТИНИ САЙТУ ДЛЯ КАВИ

3.1. Обґрунтування вибору засобів розробки та середовища проектування

Ефективність розробки сучасних вебзастосунків, зокрема Single Page Applications (SPA), значною мірою залежить від обраного інтегрованого середовища розробки (IDE). Для реалізації фронтенд-частини проєкту «Coffeemaker» було обрано Visual Studio Code (VS Code) — кросплатформний редактор коду від компанії Microsoft, який де-факто став стандартом в індустрії веброзробки завдяки своїй легковажності, швидкодії та надзвичайній гнучкості налаштувань.

Вибір VS Code як основного інструменту проектування зумовлений наступними функціональними особливостями, що критично впливають на якість фінального програмного продукту:

- Інтелектуальна підтримка сучасних екосистем. VS Code забезпечує нативну підтримку мов JavaScript та TypeScript, що є базовими для обраного стеку технологій. Технологія IntelliSense реалізує не просто підсвічування синтаксису, а інтелектуальне автодоповнення коду на основі типів змінних, визначень функцій та імпортованих модулів. Це суттєво мінімізує кількість синтаксичних помилок на етапі написання коду.
- Інтеграція з системами контролю версій (Git). Вбудований термінал та графічний інтерфейс для роботи з Git дозволяють здійснювати фіксацію змін (commit), створення гілок та синхронізацію з віддаленими репозиторіями (наприклад, на GitHub) безпосередньо в робочому вікні. Це оптимізує робочий процес та забезпечує збереження історії розробки.
- Розширюваність та екосистема плагінів. Потужна система розширень дозволяє адаптувати середовище під специфічні потреби React-розробки. Зокрема, було інтегровано розширення для підтримки ES7+ React/Redux/React-Native snippets, що прискорює створення компонентної структури застосунку.

- Інструментарій для налагодження (Debugging). Вбудований дебагер дозволяє встановлювати точки зупинки (breakpoints), інспектувати стан змінних та відстежувати стек викликів безпосередньо в середовищі розробки, що значно пришвидшує пошук та усунення логічних помилок у бізнес-логіці замовлень.
- Оптимізація командної взаємодії. Завдяки використанню конфігураційних файлів, середовище дозволяє синхронізувати правила розробки між різними учасниками проєкту, гарантуючи ідентичне форматування коду незалежно від локальних налаштувань окремих розробників.

Узагальнюючи, Visual Studio Code виступає не просто як текстовий редактор, а як повноцінний центр керування розробкою, який інтегрує в собі термінал, засоби налагодження, систему контролю версій та інструменти статичного аналізу коду, що є критично важливим для створення стабільного фронтенд-рішення для сайту «Coffeemaker».

3.2. Специфікація вимог до програмного та технічного забезпечення

Реалізація сучасного вебзастосунку потребує чітко визначеного програмного стеку та відповідних апаратних потужностей для забезпечення стабільного процесу збірки, тестування та розгортання проєкту.

3.2.1. Програмне забезпечення та інструментарій розробки

Для створення клієнтської частини системи було впроваджено наступний комплекс програмних засобів:

- Бібліотека React (v18+). Обрана як основна декларативна бібліотека для побудови компонентно-орієнтованого інтерфейсу. Використання механізму Virtual DOM дозволяє досягти високої продуктивності при рендерингу складних списків товарів та динамічних фільтрів кавової продукції.

- Середовище збірки Vite. Використовується як сучасна альтернатива Webpack. Завдяки використанню нативних ES-модулів, Vite забезпечує миттєвий запуск сервера розробки та надшвидку гарячу заміну модулів (HMR), що підвищує ефективність роботи розробника в рази.
- Інструменти статичного аналізу (ESLint та Prettier). ESLint забезпечує дотримання стандартів написання коду та виявлення потенційно небезпечних конструкцій до моменту запуску застосунку. Prettier автоматизує форматування коду згідно з заданим стилем, що забезпечує чистоту та читабельність архітектури проєкту.
- Мови програмування JavaScript (ES6+) та TypeScript. Використання TypeScript дозволяє впровадити сувору типізацію даних про замовлення та клієнтів, що на етапі компіляції виключає цілий клас помилок, пов'язаних із невідповідністю типів даних.
- Node.js (LTS версія) та NPM/Yarn. Використовуються як середовище виконання для інструментів розробки та менеджер пакетів для керування залежностями проєкту.

3.2.2. Технічне забезпечення та апаратні вимоги

З огляду на використання ресурсномістких інструментів (Vite, Docker, складні графічні препроцесори), для забезпечення безперебійної роботи середовища розробки встановлено наступні мінімальні та рекомендовані вимоги до апаратної частини:

1. Центральний процесор (CPU). Процесор архітектури Intel Core i5 (8-го покоління або новіше) або еквівалент від AMD (Ryzen 5). Багатопотоковість є необхідною для швидкої паралельної компіляції модулів та роботи фонових процесів аналізу коду.
2. Оперативна пам’ять (RAM). Об’єм не менше 16 ГБ. Це зумовлено високим споживанням пам’яті браузерами при відлагодженні, роботою локального сервера розробки та одночасним функціонуванням IDE з великою кількістю активних плагінів.

3. Графічна підсистема. Відеокарта з підтримкою стандарту WebGL. Це необхідно для коректного відображення сучасних елементів інтерфейсу, анімацій та можливої візуалізації кавових карт або схем у форматі 3D.
4. Дискова підсистема. Наявність SSD-накопичувача є критичною вимогою для швидкого читання тисяч дрібних файлів у папці *node_modules* та прискорення процесу збірки проєкту.
5. Операційна система. Windows 10/11 або macOS (версії Catalina та новіше). Вибір операційної системи обумовлений сумісністю з термінальними командами та стабільністю роботи Node.js оточення.

Такий збалансований підхід до вибору програмного та технічного забезпечення дозволяє не лише реалізувати поточні завдання проєкту «Coffeemaker», а й створює потенціал для подальшого масштабування системи, додавання нових функціональних модулів та забезпечення високої якості обслуговування кінцевих користувачів.

3.3. Опис програмної реалізації

3.3.1 Вибір та обґрунтування технологічного стека

Для реалізації клієнтської частини веб-сервісу продажу кави було обрано сучасний стек технологій, орієнтований на створення SPA-застосунків (Single Page Application). Основною метою під час вибору технологій було забезпечення високої швидкодії застосунку, масштабованості архітектури, зручності підтримки коду та можливості інтеграції із зовнішніми сервісами.

В якості основної мови програмування було використано JavaScript. Дана мова є стандартом для розробки веб-застосунків і підтримується всіма сучасними браузерами. JavaScript дозволяє створювати динамічні інтерфейси користувача, працювати з HTTP-запитами, обробляти події та взаємодіяти з REST API серверної частини.

Для побудови інтерфейсу користувача було використано бібліотеку React 18. Даний фреймворк забезпечує компонентний підхід до розробки, що дозволяє

повторно використовувати UI-компоненти та спрощує підтримку великого проєкту. React також забезпечує ефективне оновлення DOM через механізм Virtual DOM, що позитивно впливає на продуктивність застосунку.

Для створення та запуску проєкту було використано інструмент Vite. У порівнянні з традиційними збирачами проєктів, Vite забезпечує значно швидший запуск локального середовища розробки та швидке оновлення сторінок під час розробки завдяки технології Hot Module Replacement (HMR). Це дозволило прискорити процес розробки та тестування функціоналу.

Для організації маршрутизації між сторінками застосунку було використано бібліотеку React Router DOM. Вона забезпечує реалізацію навігації між сторінками без перезавантаження браузера та дозволяє реалізувати захищені маршрути для авторизованих користувачів і адміністраторів.

Для управління глобальним станом застосунку було використано Redux Toolkit разом із React Redux. Використання Redux дозволило централізовано зберігати дані користувача, товари, категорії, кошик, фільтри та інші сутності. Redux Toolkit був обраний через спрощений синтаксис та зменшення кількості шаблонного коду у порівнянні з класичним Redux.

Для взаємодії із серверною частиною було використано бібліотеку Axios. Вона забезпечує зручне виконання HTTP-запитів та підтримує interceptor-и для автоматичної обробки помилок авторизації та оновлення JWT-токенів. Також Axios дозволяє централізувати роботу з API через окремий HTTP-клієнт.

Для стилізації інтерфейсу було використано SCSS Modules. Такий підхід дозволяє ізолювати стилі компонентів та уникати конфліктів CSS-класів у великому проєкті. Додатково використовувались бібліотеки Bootstrap, React-Bootstrap, MUI та PrimeReact для швидкого створення адаптивних елементів інтерфейсу.

Для реалізації повідомлень користувача використовувалась бібліотека React Toastify, а для відображення іконок — React Icons, Font Awesome та Lucide React.

У проєкті також використовувались бібліотеки Chart.js та react-chartjs-2 для побудови графіків і статистики в адміністративній панелі.

Для реалізації онлайн-оплати було інтегровано платіжну систему Stripe. Дана інтеграція дозволяє безпечно здійснювати оплату товарів банківськими картками без необхідності зберігання платіжних даних на сервері застосунку.

Для входу через сторонній сервіс було використано Google Identity Services, що дозволяє користувачам авторизовуватись через власний Google-акаунт.

У процесі розробки використовувалась система контролю версій Git. Вона забезпечувала збереження історії змін, можливість роботи з окремими гілками та спрощувала процес розробки й тестування нового функціоналу.

Для тестування REST API використовувався Postman. За допомогою цього інструмента перевірялась коректність роботи HTTP-запитів, авторизації, передачі токенів та взаємодії фронтенду з бекендом.

Для контейнеризації застосунку використовувався Docker. Це дозволило створити ізольоване середовище запуску та спростило процес розгортання проєкту. Для автоматичного запуску контейнерів використовувався файл docker-compose.yml.

Також у проєкті використовувалась система змінних середовища .env, у якій зберігались конфігураційні параметри, такі як URL серверної частини, Stripe publishable key та URL для роботи із зображеннями Cloudinary.

```
export default class HttpClient {
  constructor(configs) {
    this.axiosInstance = axios.create({
      baseURL: configs.baseURL,
      timeout: configs.timeout || 3000,
      headers: {
        'Content-Type': 'application/json',
        Accept: 'application/json',
        ...configs.headers,
        'Access-Control-Allow-Origin': '*',
      },
      ...configs,
    })

    this.axiosInstance.interceptors.response.use(
      response => response,
```

```
      async error => {
        const originalRequest = error.config

        if (error.response?.status === 401 &&
!originalRequest._retry) {
          originalRequest._retry = true
          try {
            const token = await refreshToken(originalRequest,
this.setAuthorizationToken.bind(this))
            originalRequest.headers['Authorization'] = `Bearer
${token}`
            return this.axiosInstance(originalRequest)
          } catch (refreshError) {
            return Promise.reject(refreshError)
          }
        }
        return Promise.reject(error)
      },
    )
  }
```

Лістинг 3.1. Приклад конфігурації Axios HTTP-клієнта з interceptor-ом

Джерело: [створено автором]

3.3.2 Архітектура та структура проєкту

Під час розробки клієнтської частини веб-сервісу для продажу кави було використано модульний підхід до побудови архітектури застосунку. Основною метою такої структури було забезпечення масштабованості, повторного використання компонентів, спрощення підтримки коду та розділення відповідальностей між окремими частинами системи. Проєкт реалізований у вигляді SPA-застосунку (Single Page Application), у якому взаємодія між сторінками виконується без повного перезавантаження браузера. Це дозволяє забезпечити більш швидку роботу інтерфейсу та покращити користувацький досвід.

Уся структура проєкту організована у директорії src, де кожна папка відповідає за окремий функціональний модуль системи. Такий підхід дозволяє ізолювати різні частини застосунку, спрощує навігацію по коду та полегшує подальше масштабування проєкту. Основними директоріями є pages, components, store, utils, routes, hooks, contexts та styles.

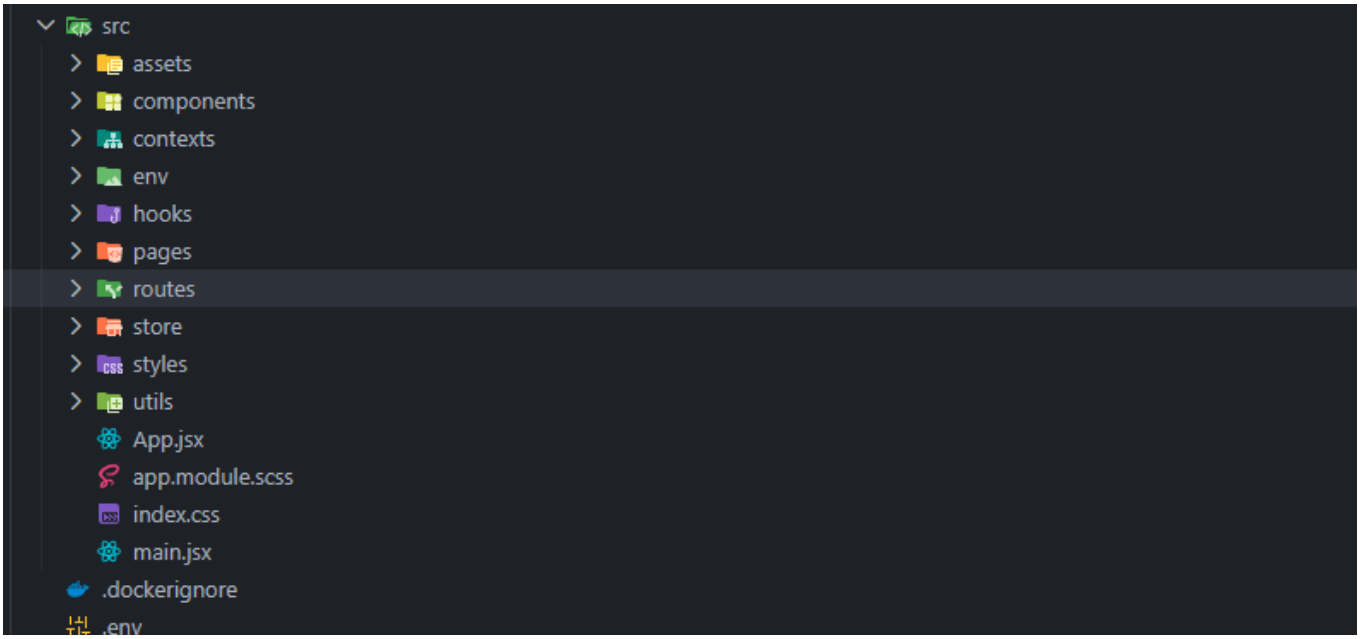


Рис. 3.1. Загальна структура директорій проєкту

Джерело: [створено автором]

Папка pages містить сторінки застосунку. Кожна сторінка є окремим React-компонентом, який відповідає за конкретний функціонал системи. У даній директорії реалізовано головну сторінку, каталог товарів, сторінку детального перегляду товару, кошик, checkout, сторінки авторизації та реєстрації, профіль користувача, історію замовлень, а також адміністративну панель. Такий підхід дозволяє розділити логіку окремих екранів системи та зробити код більш структурованим.

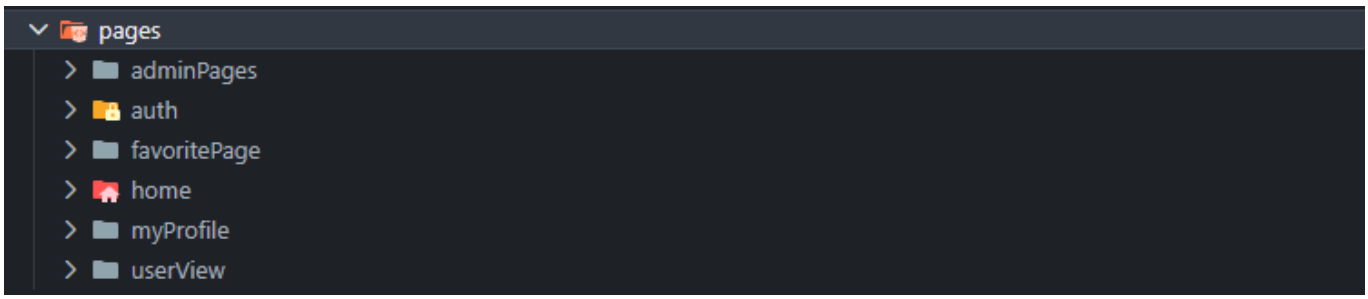


Рис. 3.2. Структура сторінок застосунку

Джерело: [створено автором]

Для повторного використання елементів інтерфейсу у проєкті використовується папка components. У ній розміщені незалежні UI-компоненти, які використовуються на різних сторінках застосунку. До таких компонентів належать navbar, footer, карточки товарів, кнопки, форми, таблиці, модальні вікна, фільтри, пагінація та AI-віджет. Компонентний підхід є однією з головних особливостей React і дозволяє будувати інтерфейс із незалежних перевикористовуваних модулів.

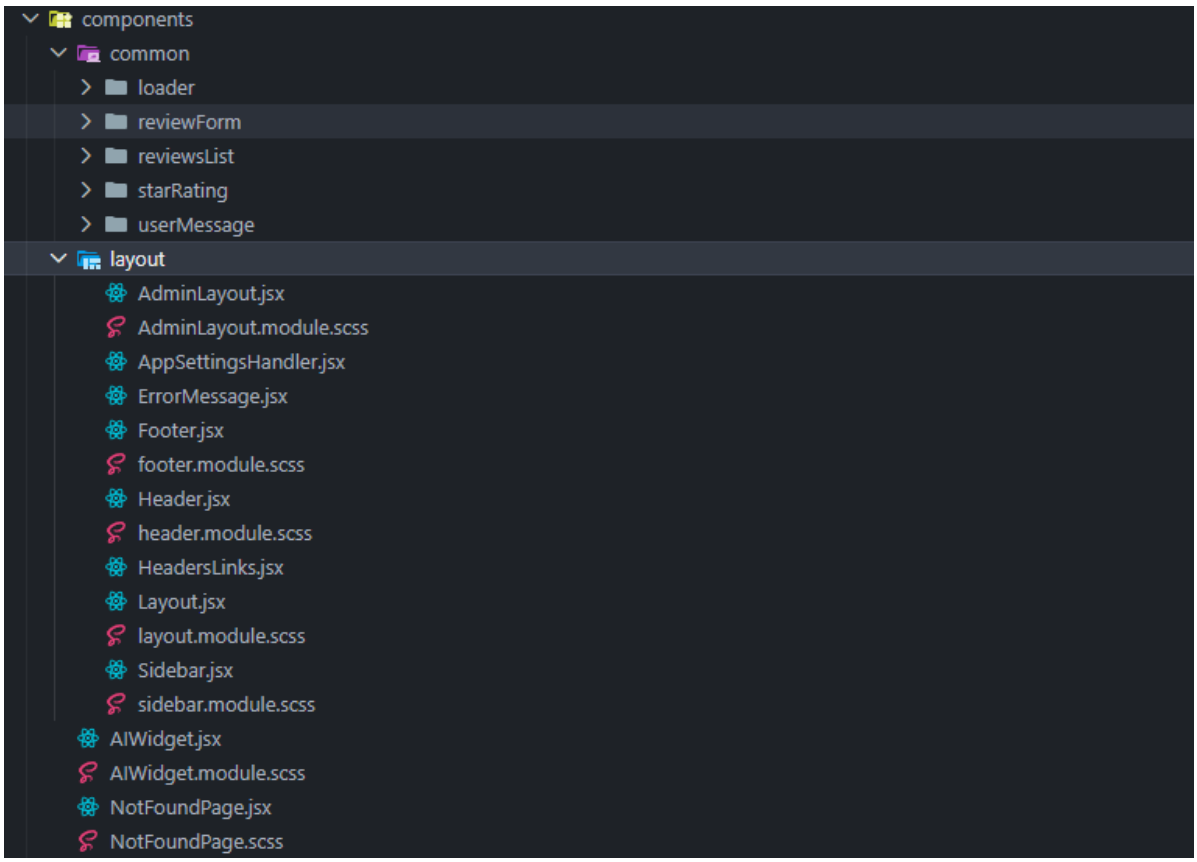


Рис. 3.3. Структура компонентів застосунку

Джерело: [створено автором]

У межах клієнтської частини застосунку можна виділити кілька архітектурних рівнів. Рівень представлення відповідає за відображення інтерфейсу користувача та взаємодію з ним. До цього рівня належать сторінки та UI-компоненти. Рівень бізнес-логіки реалізує фільтрацію товарів, авторизацію, роботу з кошиком, checkout та іншу функціональність системи. Рівень доступу до даних відповідає за HTTP-запити до REST API через Axios. Рівень управління станом забезпечує централізоване зберігання даних у Redux Store.

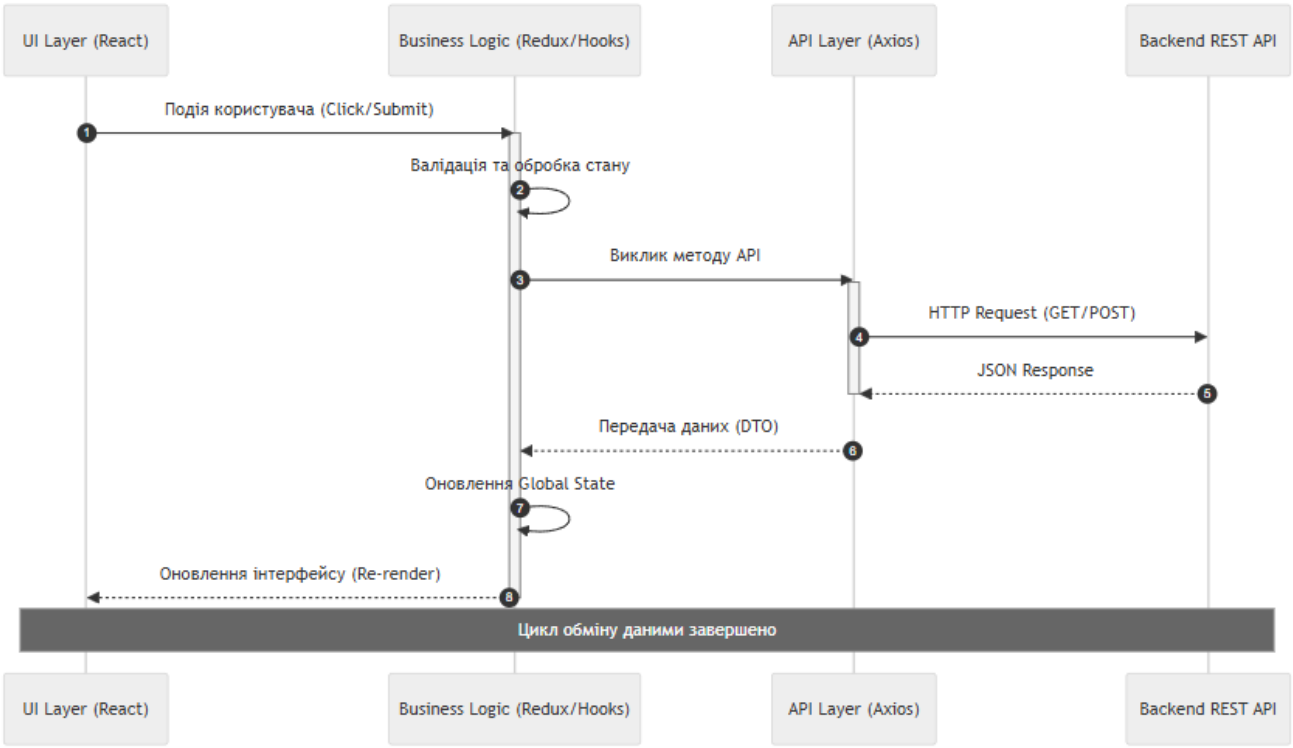


Рис. 3..4. UML-діаграма архітектурних рівнів застосунку

Джерело: створено AI на основі промпту автора

3.3.3 Реалізація підсистеми безпеки та авторизації

Однією з найважливіших частин веб-сервісу є підсистема безпеки та авторизації користувачів. У розробленому застосунку реалізовано механізм реєстрації, автентифікації та контролю доступу на основі JWT-токенів (JSON Web Token). Такий підхід дозволяє забезпечити захист користувацьких даних, контроль доступу до внутрішніх сторінок системи та безпечну взаємодію між клієнтською та серверною частинами застосунку.

Під час реєстрації користувач вводить свої дані у форму створення облікового запису. На клієнтській стороні виконується первинна перевірка коректності введених значень, зокрема перевірка електронної пошти, довжини пароля та обов'язкових полів. Після успішної валідації дані передаються на серверну частину через HTTP-запит.

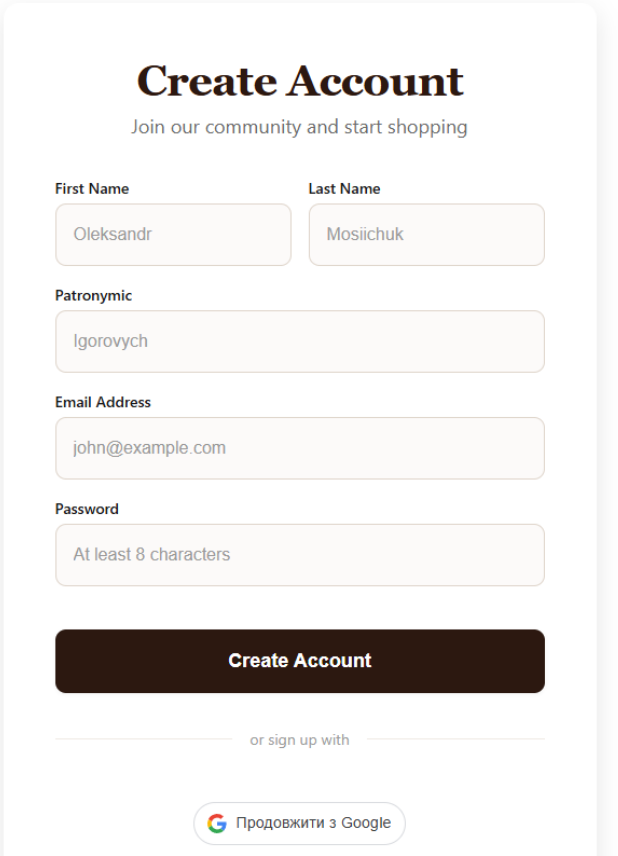


Рис. 3.5. Форма реєстрації користувача

Джерело: [створено автором]

Після успішної реєстрації користувач може виконати вхід до системи. Під час авторизації користувач вводить електронну пошту та пароль, після чого дані надсилаються на сервер для перевірки. Сервер знаходить користувача у базі даних та порівнює введений пароль із збереженим хешем за допомогою функції `bcrypt.compare`.

Якщо перевірка проходить успішно, сервер генерує JWT-токен, який містить основну інформацію про користувача, зокрема його ідентифікатор та роль.

Згенерований токен передається клієнтській частині та використовується для подальшої авторизації запитів.

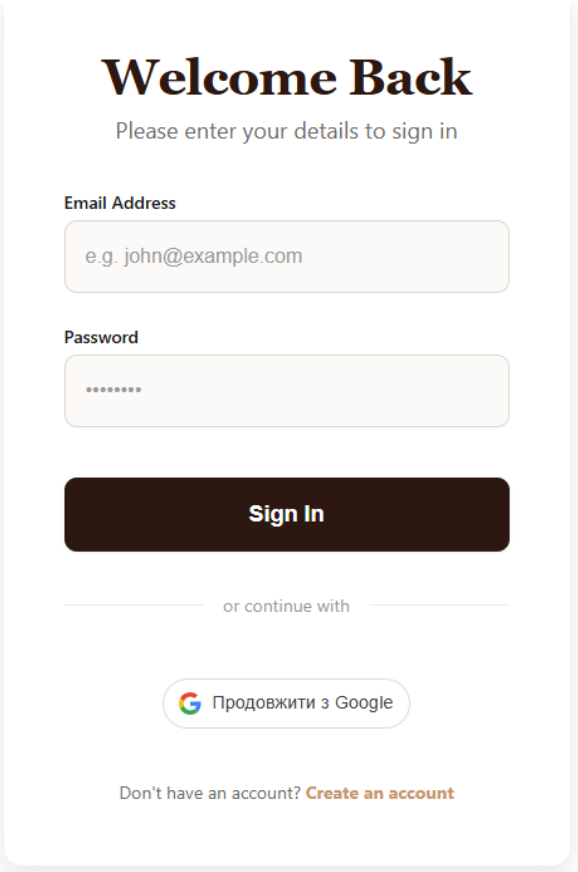


Рис. 3.6. Форма авторизації користувача

Джерело: [створено автором]

JWT-токен дозволяє серверу визначати авторизованого користувача без необхідності постійного збереження сесії на серверній стороні. Це забезпечує масштабованість системи та спрощує взаємодію між фронтендом і бекендом.

Після успішного входу токен зберігається на клієнтській стороні та автоматично додається до HTTP-запитів через Axios interceptor. Завдяки цьому користувач може взаємодіяти із захищеними ресурсами системи без повторної авторизації.

Крім стандартної авторизації через email і пароль, у системі реалізовано вхід через Google. Для цього використовується Google Identity Services. Після успішної автентифікації через Google клієнтська частина отримує спеціальний токен, який передається на сервер для створення або авторизації користувача.

Такий підхід дозволяє спростити процес входу в систему та покращити користувацький досвід.

Таким чином, у проєкті реалізовано повноцінну підсистему безпеки та авторизації користувачів із використанням сучасних механізмів автентифікації. Використання JWT-токенів, bcrypt-хешування, refresh token та role-based access control дозволило забезпечити безпечну взаємодію користувачів із системою та захистити внутрішні ресурси веб-сервісу.

3.3.4. Керівництво користувача

Однією з основних частин веб-сервісу є інтерфейс користувача, оскільки саме через нього користувач взаємодіє із системою. Під час розробки клієнтської частини застосунку основна увага приділялась зручності використання, швидкості взаємодії, адаптивності інтерфейсу та зрозумілому розташуванню функціональних елементів. Інтерфейс реалізовано у вигляді SPA-застосунку на базі React, що дозволяє виконувати навігацію між сторінками без повного перезавантаження браузера.

Для стилізації інтерфейсу використовувались SCSS Modules, Bootstrap, React-Bootstrap, MUI та PrimeReact. Поєднання цих технологій дозволило створити адаптивний сучасний інтерфейс із підтримкою різних розмірів екранів. Додатково використовувались бібліотеки React Icons, Font Awesome та Lucide React для відображення іконок та покращення візуального сприйняття елементів інтерфейсу.

Головна сторінка застосунку виконує роль основної точки входу для користувача. На ній розміщується hero-блок із основною інформацією про сервіс, кнопками переходу до каталогу товарів та категоріями продукції. Також на головній сторінці реалізовано секції з популярними товарами та інформаційними блоками.

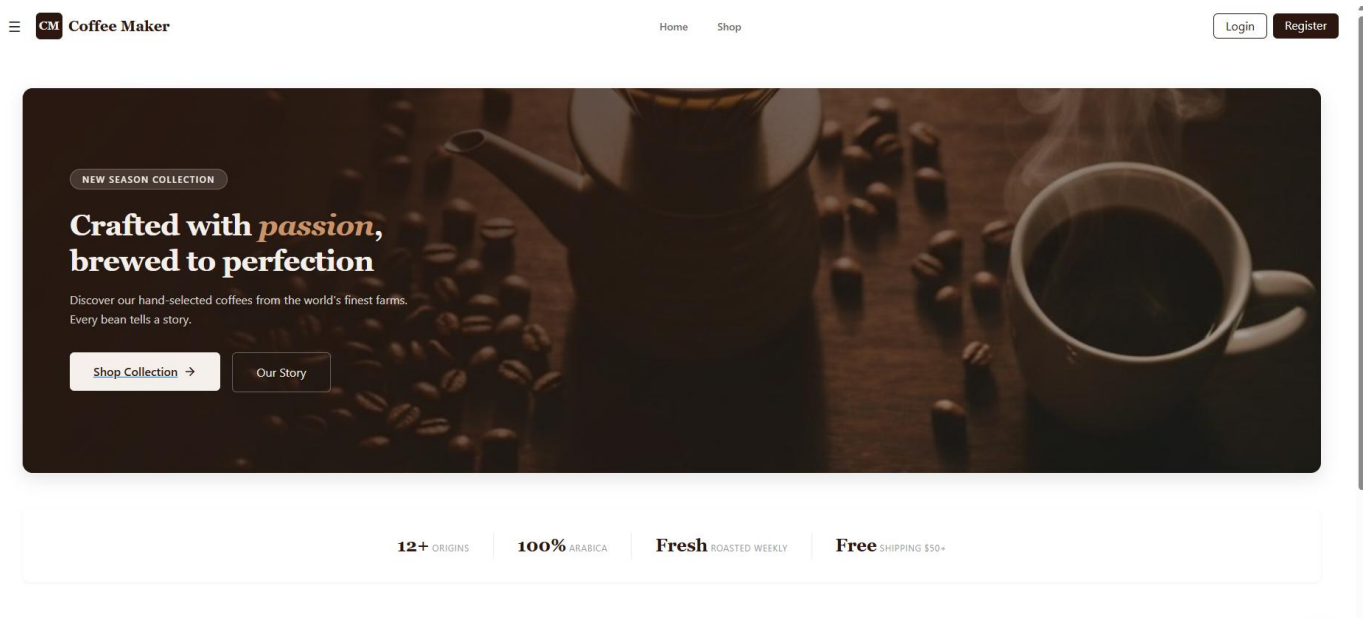


Рис. 3.7. Головна сторінка веб-сервісу

Джерело: [створено автором]

У верхній частині застосунку розташований navigation bar, який забезпечує навігацію між основними сторінками системи. Navbar містить посилання на каталог товарів, кошик, сторінку авторизації, профіль користувача та адміністративну панель. Для авторизованих користувачів також відображається інформація про профіль та кількість товарів у кошику.

Навігація реалізована за допомогою react-router-dom, що дозволяє швидко перемикатись між сторінками без повторного завантаження сторінки браузера.

Однією з ключових сторінок системи є каталог товарів. На цій сторінці користувач може переглядати список товарів, виконувати пошук, сортування та фільтрацію за категоріями. Для оптимізації взаємодії з великою кількістю товарів реалізовано пагінацію.

Список товарів формується динамічно на основі даних, отриманих із серверної частини через REST API. Кожен товар відображається у вигляді окремої карточки з назвою, ціною, рейтингом та кнопками взаємодії.

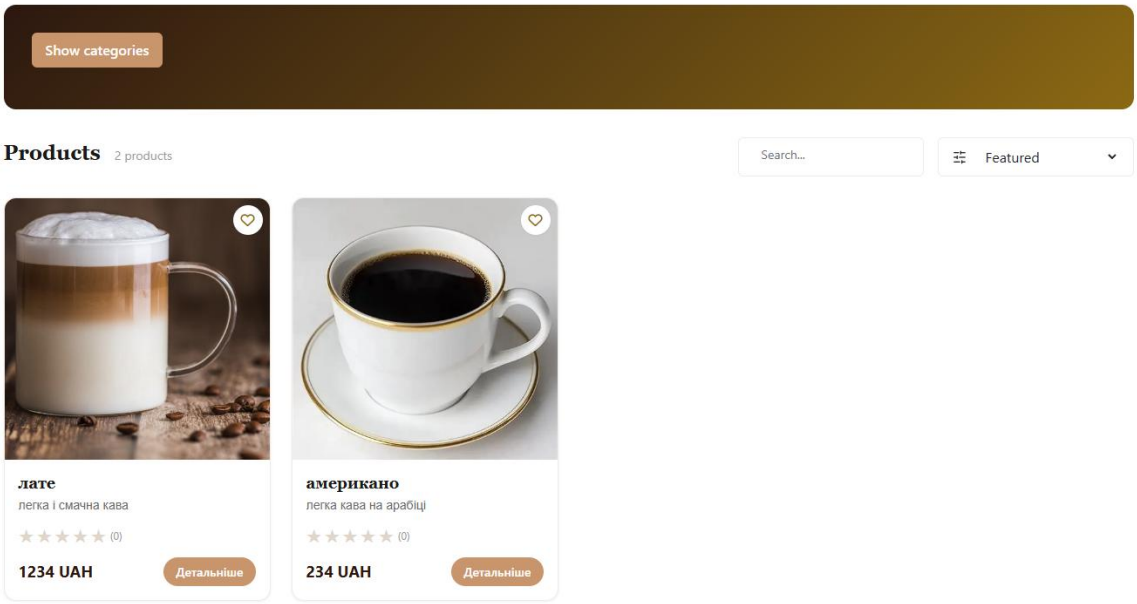


Рисунок 3.8. Сторінка каталогу товарів

Джерело: [створено автором]

Таким чином, інтерфейс користувача веб-сервісу побудований із використанням сучасних UI/UX-підходів та забезпечує зручну взаємодію користувача із системою. Використання React, компонентної архітектури, адаптивного дизайну та інтеграції із зовнішніми сервісами дозволило створити сучасний SPA-застосунок із повноцінним функціоналом інтернет-магазину.

3.3.5 Реалізація основної бізнес-логіки та роботи з даними

Основна бізнес-логіка веб-сервісу для продажу кави реалізує ключові сценарії взаємодії користувача із системою: перегляд товарів, фільтрацію та пошук, роботу з кошиком, оформлення замовлення, обробку платежів, а також керування даними в адміністративній панелі. Вся логіка побудована на основі взаємодії фронтенд-застосунку з REST API бекенд-сервісу через Axios.

Дані у системі передаються за стандартною схемою клієнт–сервер. Клієнтська частина ініціює HTTP-запит, сервер обробляє його, виконує бізнес-логіку та повертає відповідь у форматі JSON. Далі ці дані потрапляють у Redux Store або локальний стан компонентів і відображаються користувачу.

Однією з ключових бізнес-операцій є робота з каталогом товарів. Після завантаження сторінки каталогу виконується запит до серверної частини для отримання списку продуктів. Далі користувач може застосовувати фільтри, сортування та пошук. Всі ці операції виконуються як на клієнті (UI-фільтрація), так і на сервері (запити з параметрами).

Пошук товарів реалізовано через глобальний search context та API-запити. Користувач вводить текст у пошукове поле, після чого система виконує запит до бекенду з параметром пошуку. Сервер повертає відфільтрований список товарів, який відображається у каталозі.

Сортування товарів виконується за такими параметрами:

- ціна (зростання / спадання);
- рейтинг;
- новизна.

Фільтрація здійснюється за категоріями товарів, що дозволяє користувачу швидко знаходити потрібну продукцію.

```
getAllFiltered: async (filters) => {
    const params = new URLSearchParams();
    if (filters.minPrice) params.append('minPrice',
filters.minPrice);
    if (filters.maxPrice) params.append('maxPrice',
filters.maxPrice);
    if (filters.currency) params.append('currency',
filters.currency);
    if (filters.status) params.append('status', filters.status);
    if (filters.searchQuery) params.append('searchQuery',
filters.searchQuery);
    if (filters.sort) params.append('sort', filters.sort);
    if (filters.page) params.append('page', filters.page);

    const response = await axios.get(`${API_URL}/get-all-
filtered?${params.toString()}`, getAuthHeaders());
    return response.data;
  },
```

Лістинг 3.2. Приклад запиту на отримання товарів з фільтрами

Джерело: [створено автором]

Однією з найважливіших частин бізнес-логіки є реалізація кошика користувача. Кошик працює на основі Redux Toolkit і дозволяє додавати, видаляти та змінювати кількість товарів. При кожній зміні стану кошика автоматично перераховується загальна сума замовлення.

Додавання товару до кошика виконується через dispatch Redux action, який передає інформацію про продукт у глобальний state.

Уся бізнес-логіка системи побудована таким чином, щоб забезпечити чітке розділення відповідальностей: клієнтська частина відповідає за взаємодію з користувачем та формування запитів, а серверна частина — за обробку даних, збереження інформації та виконання бізнес-правил.

Завдяки такій архітектурі забезпечується масштабованість системи, можливість розширення функціоналу та стабільна робота всіх ключових сценаріїв веб-сервісу.

Висновки до розділу 3

У третьому розділі кваліфікаційної роботи було розглянуто програмне та технічне забезпечення веб-сервісу для продажу кави, а також описано його практичну реалізацію. На основі аналізу вимог до системи були обрані сучасні інструменти та технології, які забезпечують стабільну роботу застосунку, його масштабованість та зручність подальшого розвитку.

Було обґрунтовано вибір технологічного стека, до якого входять React для побудови інтерфейсу користувача, Vite як інструмент збірки, Redux Toolkit для управління станом застосунку, Axios для взаємодії з серверною частиною, а також додаткові бібліотеки для реалізації UI, анімацій, форм, графіків та інтеграцій із зовнішніми сервісами. Окрему увагу приділено використанню Git для контролю версій та Docker для контейнеризації застосунку.

У межах розділу також було проаналізовано архітектуру програмного забезпечення. Проєкт реалізовано як SPA-застосунок із модульною структурою, що

включає розділення на сторінки, компоненти, сервіси, стан застосунку та утиліти. Такий підхід забезпечує чітке розділення відповідальностей між частинами системи та спрощує її підтримку і масштабування. Використання багаторівневої архітектури дозволило організувати взаємодію між рівнем представлення, бізнес-логікою та рівнем доступу до даних.

Окремо було описано реалізацію підсистеми безпеки та авторизації користувачів. У системі застосовано механізм JWT-токенів, хешування паролів та role-based access control, що дозволяє забезпечити захист даних користувачів та обмежити доступ до адміністративних функцій. Додатково реалізовано механізм оновлення токенів та інтеграцію з Google Identity Services для спрощеної авторизації.

У розділі також детально розглянуто реалізацію інтерфейсу користувача. Було створено адаптивний та зручний UI із використанням сучасних бібліотек компонентів. Реалізовано основні сторінки системи: головну сторінку, каталог товарів, сторінку товару, кошик, оформлення замовлення, авторизацію, профіль користувача, історію замовлень та адміністративну панель. Інтерфейс забезпечує інтуїтивну взаємодію користувача із системою та підтримує роботу на різних пристроях.

Також було описано реалізацію основної бізнес-логіки системи, яка включає обробку товарів, роботу з кошиком, оформлення замовлень, інтеграцію зі Stripe для онлайн-оплати, управління відгуками та реалізацію AI-віджета рекомендацій. Усі ці процеси побудовані на основі взаємодії фронтенду з REST API бекендом та використання глобального стану для синхронізації даних.

Завершально було розглянуто підсистему тестування та обробки помилок, яка забезпечує коректну реакцію системи на некоректні дії користувача, валідацію форм та відображення повідомлень про помилки.

Отже, програмна реалізація веб-сервісу для продажу кави відповідає сучасним вимогам до SPA-застосунків, забезпечує високий рівень функціональності, безпеки та зручності використання, а також має продуману архітектуру, що дозволяє у майбутньому легко розширювати та масштабувати систему.

ВИСНОВКИ

У межах виконання кваліфікаційної роботи було розроблено та реалізовано повноцінний веб-сервіс для продажу кави у форматі сучасного SPA-застосунку. Основною метою проєкту було створення функціональної e-commerce системи, яка забезпечує повний цикл взаємодії користувача з інтернет-магазином: від перегляду каталогу товарів до оформлення замовлення, здійснення онлайн-оплати та подальшого перегляду історії покупок. Окремо також було реалізовано адміністративну частину системи для керування товарами, користувачами та категоріями.

У першому розділі роботи було проведено детальний аналіз предметної області електронної комерції. Розглянуто основні процеси функціонування інтернет-магазину, зокрема роботу з каталогом товарів, системою замовлень, кошиком користувача та механізмами авторизації. Було проаналізовано роль різних типів користувачів у системі, таких як звичайний користувач, адміністратор та менеджер, а також їхні функціональні можливості. Окрему увагу приділено аналізу існуючих аналогів веб-магазинів, що дозволило визначити основні тенденції розвитку e-commerce систем та сформувати вимоги до власного проєкту. На основі цього було сформульовано постановку задачі, яка включає розробку зручного, швидкого та безпечного веб-застосунку з підтримкою повного циклу покупки товару.

У другому розділі було виконано аналіз інформаційного забезпечення системи, зокрема визначено вхідні та вихідні дані, які обробляються у межах веб-сервісу. Вхідними даними є інформація про користувачів, товари, категорії, відгуки та замовлення, тоді як вихідними є сформовані сторінки інтерфейсу, результати пошуку, сформовані замовлення та аналітичні дані. Також було виконано проєктування структури бази даних, яка забезпечує збереження та зв’язок між основними сутностями системи. База даних включає таблиці користувачів, товарів, категорій, кошика, замовлень та відгуків, що дозволяє реалізувати повноцінну модель взаємодії між компонентами системи. Окремо було розглянуто математичне та алгоритмічне забезпечення, яке включає алгоритми фільтрації та сортування товарів, обробку

кошика користувача, розрахунок загальної вартості замовлення, а також логіку формування рекомендацій на основі даних про товари та відгуки.

У третьому розділі було розглянуто програмну реалізацію та технічне забезпечення системи. Обґрунтовано вибір сучасного технологічного стеку, який включає React для побудови інтерфейсу користувача, Vite для швидкої збірки проєкту, Redux Toolkit для управління глобальним станом, Axios для взаємодії з серверною частиною, а також низку додаткових бібліотек для реалізації UI-компонентів, форм, графіків та інтеграцій. Було описано архітектуру застосунку, яка побудована за модульним принципом та відповідає SPA-підходу, що забезпечує високу швидкість роботи та зручність користування.

Окрему увагу приділено реалізації підсистеми безпеки та авторизації користувачів. У системі використовується механізм JWT-токенів, хешування паролів та role-based access control, що дозволяє забезпечити захист даних та обмежити доступ до адміністративних функцій. Також реалізовано механізм оновлення токенів, що підвищує зручність використання системи без необхідності повторної авторизації.

Було детально розглянуто розробку інтерфейсу користувача, який включає основні сторінки системи: головну сторінку, каталог товарів, сторінку товару, кошик, оформлення замовлення, авторизацію, профіль користувача, історію замовлень та адміністративну панель. Інтерфейс реалізовано з урахуванням сучасних UI/UX підходів, що забезпечує інтуїтивну взаємодію користувача із системою та підтримку різних пристроїв завдяки адаптивному дизайну.

Також було описано реалізацію основної бізнес-логіки системи, яка включає обробку товарів, роботу з кошиком, оформлення замовлень, інтеграцію зі Stripe для онлайн-оплати, управління відгуками та реалізацію AI-віджета рекомендацій. Усі ці процеси базуються на взаємодії клієнтської частини з REST API сервером та використанні централізованого управління станом через Redux.

Окремо розглянуто підсистему тестування та обробки помилок, яка забезпечує стабільну роботу системи навіть у випадку некоректних дій користувача. Реалізовано валідацію форм, обробку HTTP-помилок та відображення повідомлень користувачу у зрозумілому вигляді.

У процесі виконання роботи було підтверджено доцільність використання обраних технологій та архітектурних рішень. Розроблена система демонструє високу продуктивність, масштабованість та зручність у використанні. Вона повністю реалізує основні сценарії роботи інтернет-магазину та може бути легко розширена в майбутньому шляхом додавання нових функціональних модулів, інтеграцій або сервісів.

Отже, у результаті виконання кваліфікаційної роботи було створено сучасний, функціональний та структурований веб-застосунок для продажу кави, який відповідає вимогам до сучасних e-commerce систем, забезпечує повний цикл взаємодії користувача із сервісом та має потенціал для подальшого розвитку і масштабування у реальних умовах експлуатації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. React Documentation. – URL: https://react.dev/ (дата звернення: 11.05.2026).
2. Redux Toolkit Documentation. – URL: https://redux-toolkit.js.org/ (дата звернення: 11.05.2026).
3. Redux Official Documentation. – URL: https://redux.js.org/ (дата звернення: 11.05.2026).
4. Vite Documentation. – URL: https://vite.dev/ (дата звернення: 11.05.2026).
5. MDN Web Docs: HTML. – URL: https://developer.mozilla.org/en-US/docs/Web/HTML (дата звернення: 11.05.2026).
6. MDN Web Docs: CSS. – URL: https://developer.mozilla.org/en-US/docs/Web/CSS (дата звернення: 11.05.2026).
7. MDN Web Docs: JavaScript. – URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата звернення: 11.05.2026).
8. SASS Documentation. – URL: https://sass-lang.com/documentation (дата звернення: 11.05.2026).
9. Axios GitHub Repository. – URL: https://github.com/axios/axios (дата звернення: 11.05.2026).
10. React Router Documentation. – URL: https://reactrouter.com/ (дата звернення: 11.05.2026).
11. Stripe Documentation. – URL: https://stripe.com/docs (дата звернення: 11.05.2026).
12. Google Identity Services. – URL: https://developers.google.com/identity (дата звернення: 11.05.2026).
13. JWT Introduction (RFC 7519). – URL: https://www.rfc-editor.org/rfc/rfc7519 (дата звернення: 11.05.2026).
14. OWASP Top 10 Web Security Risks. – URL: https://owasp.org/www-project-top-ten/ (дата звернення: 11.05.2026).
15. Docker Documentation. – URL: https://docs.docker.com/ (дата звернення: 11.05.2026).
16. Git Documentation. – URL: https://git-scm.com/doc (дата звернення: 11.05.2026).

17. Node.js Documentation. – URL: https://nodejs.org/en/docs (дата звернення: 11.05.2026).
18. npm Documentation. – URL: https://docs.npmjs.com/ (дата звернення: 11.05.2026).
19. Bootstrap Documentation. – URL: https://getbootstrap.com/docs/ (дата звернення: 11.05.2026).
20. React Bootstrap Documentation. – URL: https://react-bootstrap.github.io/ (дата звернення: 11.05.2026).
21. Material UI Documentation. – URL: https://mui.com/ (дата звернення: 11.05.2026).
22. PrimeReact Documentation. – URL: https://primereact.org/ (дата звернення: 11.05.2026).
23. React Icons. – URL: https://react-icons.github.io/react-icons/ (дата звернення: 11.05.2026).
24. Font Awesome. – URL: https://fontawesome.com/ (дата звернення: 11.05.2026).
25. Lucide Icons. – URL: https://lucide.dev/ (дата звернення: 11.05.2026).
26. Chart.js Documentation. – URL: https://www.chartjs.org/docs/latest/ (дата звернення: 11.05.2026).
27. React Chart.js 2. – URL: https://react-chartjs-2.js.org/ (дата звернення: 11.05.2026).
28. Formik Documentation. – URL: https://formik.org/ (дата звернення: 11.05.2026).
29. Yup Validation Library. – URL: https://github.com/jquense/yup (дата звернення: 11.05.2026).
30. Cloudinary Documentation. – URL: https://cloudinary.com/documentation (дата звернення: 11.05.2026).
31. REST API Architectural Style (Roy Fielding Dissertation). – URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm (дата звернення: 11.05.2026).
32. Web Storage API (MDN). – URL: https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API (дата звернення: 11.05.2026).
33. Fetch API (MDN). – URL: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API (дата звернення: 11.05.2026).

34. JavaScript ES6 Features. – URL: https://www.w3schools.com/js/js_es6.asp (дата звернення: 11.05.2026).
35. JSON Web Tokens Introduction. – URL: https://jwt.io/introduction (дата звернення: 11.05.2026).
36. bcrypt npm package. – URL: https://www.npmjs.com/package/bcrypt (дата звернення: 11.05.2026).
37. Express.js Documentation. – URL: https://expressjs.com/ (дата звернення: 11.05.2026).
38. React State Management Patterns. – URL: https://react.dev/learn/managing-state (дата звернення: 11.05.2026).
39. Web Performance Best Practices. – URL: https://web.dev/performance/ (дата звернення: 11.05.2026).
40. SPA (Single Page Application) Concept. – URL: https://developer.mozilla.org/en-US/docs/Glossary/SPA (дата звернення: 11.05.2026).