Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему:

Проектування та розробка маркетплейсу з механізмом рекомендованих товарів на базі react та C#

Виконав(ла): студент(ка) 4 курсу, групи СП-42
спеціальності 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

(підпис) П’ясецький Д.В. (прізвище та ініціали)

Керівник (підпис) (прізвище та ініціали)

Нормоконтроль (підпис) (прізвище та ініціали)

Завідувач кафедри (підпис) (прізвище та ініціали)

Рецензент (підпис) (прізвище та ініціали)

Тернопіль 2025

АНОТАЦІЯ

Метою кваліфікаційної роботи є розробка вебзастосунку для онлайн-продажу кави, який забезпечує автоматизацію основних бізнес-процесів електронної комерції: управління каталогом товарів, оформлення замовлень, обробки платежів та взаємодії з користувачами. Розробка реалізована на основі сучасного стеку технологій: React.js для фронтенду, .NET Core для бекенду та Microsoft SQL Server для зберігання даних.

У роботі проведено аналіз ринку подібних рішень, визначено функціональні вимоги до системи, спроектовано архітектуру застосунку, базу даних та інтерфейс користувача. Реалізовано модулі для користувачів і адміністраторів, включно з особистим кабінетом, фільтрацією товарів, кошиком, платіжною інтеграцією та адміністративною панеллю управління контентом. Застосунок підтримує адаптивний дизайн, забезпечує безпечну передачу даних та відповідає вимогам доступності.

Результати тестування підтвердили стабільну роботу системи, відповідність функціоналу технічним вимогам, ефективність під навантаженням і масштабованість. Розроблений веб застосунок може бути впроваджений як основна платформа для електронної комерції у сфері продажу кави.

Обсяг роботи: 62 сторінок, 14 рисунків, 24 використаних джерела.

ABSTRACT

The purpose of this qualification work is to develop a web application for online coffee sales, aimed at automating the key processes of e-commerce: product catalog management, order processing, payment handling, and customer interaction. The application was developed using a modern technology stack: React.js for the frontend, .NET Core for the backend, and Microsoft SQL Server for data storage.

The project includes an analysis of existing market solutions, definition of functional requirements, and the design of the system architecture, database, and user interface. Modules were implemented for both users and administrators, including a personal account area, product filtering, shopping cart, payment system integration, and an administrative content management panel. The system supports adaptive design, ensures secure data transmission, and complies with accessibility standards.

Testing confirmed the system's stability, functionality compliance, efficiency under load, and scalability. The developed web application is ready to be used as a full-fledged platform for conducting e-commerce in the coffee retail sector.

Scope of work: 62 pages, 14 figures, 24 references.

ЗМІСТ

ВСТУП

Сучасні тенденції у сфері електронної комерції невід'ємно пов’язані з використанням інформаційних технологій для автоматизації процесів реалізації товарів та послуг. У ринку кави, де конкуренція постійно зростає, важливим фактором успіху є наявність ефективного онлайн-присутності, яка дозволяє не лише презентувати асортимент, а й забезпечити зручний та швидкий процес придбання продукції. Розвиток цифрових технологій надає бізнесу можливості для покращення взаємодії з клієнтами, аналізу їхніх вподобань та персоналізації пропозицій.

У цьому контексті актуальність теми створення веб-сайту для продажу кави стає очевидною. Такий сайт має на меті забезпечити автоматизацію процесів ведення електронної торгівлі — від представлення товарів до обробки замовлень, що спрощує управління бізнесом і підвищує рівень задоволення клієнтів.

Метою даної роботи є розробка веб-сайту для продажу кави, орієнтованого на оптимізацію торговельних процесів через автоматизацію основних функцій: каталогу товарів, оформлення замовлень, обробки платежів та взаємодії з клієнтами.

Задачі роботи включають:

1. Аналіз існуючих рішень у сфері онлайн-торгівлі кавою.
2. Визначення основних функціональних вимог до веб-сайту.
3. Проектування бази даних для зберігання інформації про товари, замовлення та користувачів.
4. Розробка зручного та адаптивного інтерфейсу користувача для покупців та адміністраторів сайту.
5. Тестування та вдосконалення функціоналу сайту перед запуском.

Об’єктом дослідження є розроблений веб-сайт для продажу кави. Цей об’єкт включає програмні модулі, базу даних, інтерфейс користувача, систему управління контентом та інтеграції з іншими сервісами.

Предметом дослідження є процеси організації онлайн-продажів кави через впровадження веб-застосунку. У рамках цього досліджуються методи та інструменти, необхідні для ефективної реалізації та оптимізації електронної торгівлі спеціалізованим кавовим продуктом.

РОЗДІЛ 1. АНАЛІЗ ВИМОГ ДО ЗАСТОСУНКУ

1.1. Огляд існуючих рішень для обліку працівників

Для успішної розробки веб-сайту з продажу кави важливо проаналізувати існуючі аналоги, щоб визначити ключові функціональні модулі, користувацький інтерфейс, особливості інтеграцій та можливості для покращення [2, 3].

Аналіз аналогічних рішень дозволяє не лише запозичити успішні підходи до реалізації функціональності, але й уникнути типових помилок. Крім того, порівняння з конкурентами дає змогу краще зрозуміти потреби цільової аудиторії, сформувати унікальні переваги власного продукту та обґрунтувати вибір інструментів реалізації.

Аналог 1: thetea

Хоча сайт thetea.ua спеціалізується на чаї, його структура, функціональність та дизайн добре підходять для аналізу в контексті кавового онлайн-магазину. Сайт має чітку структуру, категорії товарів, пошук, фільтри, систему відгуків та рейтингу, а також оформлення замовлення через кошик (рис. 1.1).

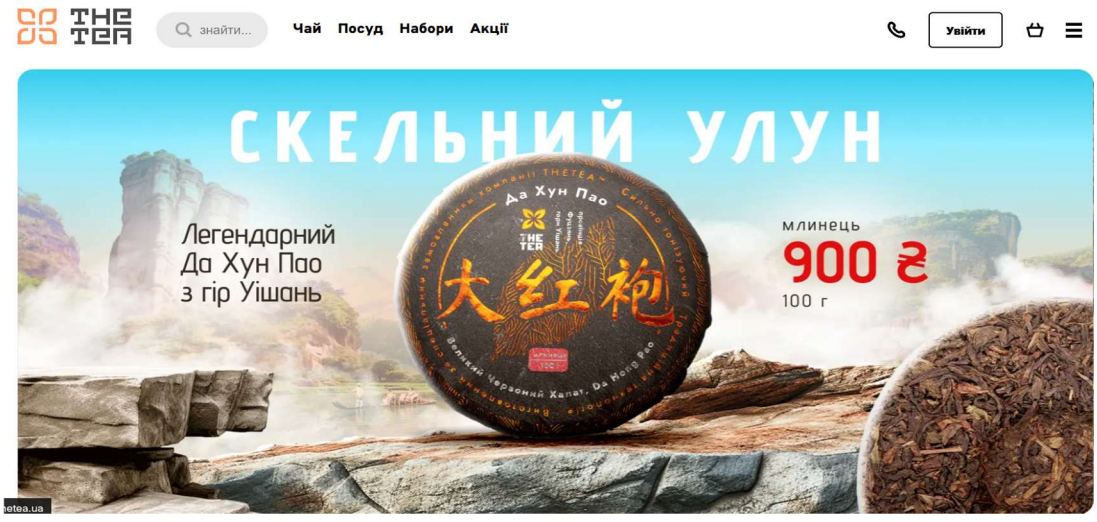

Рисунок 1.1 – Головна сторінка thetea.ua

Основні переваги:

– Зручний інтерфейс.

– Чітка навігація.
– Мобільна версія сайту.
– Швидке завантаження сторінок.
– Інтеграція з соціальними мережами.

Недоліки:

– Обмежена система рекомендацій.
– Немає автоматичного підписки на регулярну доставку.
– Не всі товари мають детальні описи та фото високої якості.

Аналог 2: coffeespace

Сайт coffeespace.com.ua є прямим прикладом онлайн-магазину кави. Має великий асортимент продукції, докладні описи товарів, фільтрацію за параметрами (тип кави, спосіб приготування, бренд), систему швидкого перегляду та порівняння товарів (рис. 1.2).

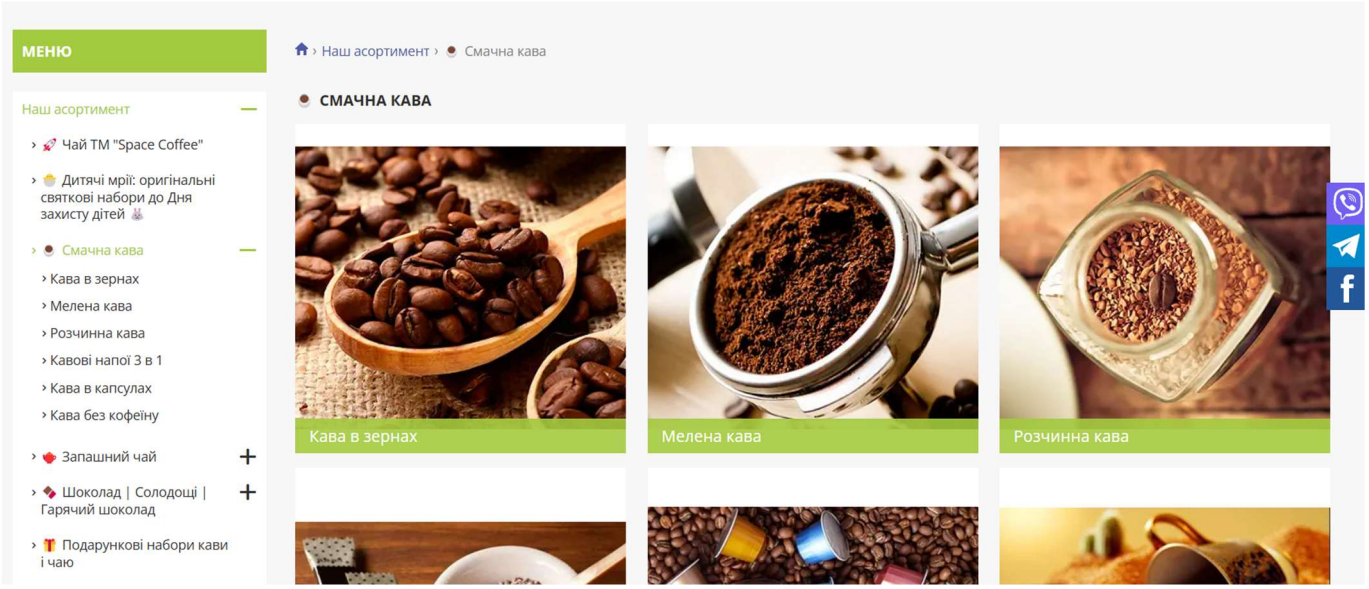

Рисунок 1.2 – Каталог кави на coffeespace.com.ua

Переваги:

– Продумана структура каталогу.
– Зручна система фільтрів.
– Інтеграція з платіжними системами.
– Наявність блогу з корисними статтями про каву.

Недоліки:

– Складна система пошуку для нових користувачів.
– Велика кількість реклами знизу сторінок.

Аналог 3: amazon.com

Amazon є одним із найбільших світових онлайн-ритейлерів, де кава займає значне місце серед товарів [4]. На Amazon представлено широкий асортимент кави різних брендів, форматів та способів приготування (рис. 1.3).

Загалом, аналіз показує, що успішні онлайн-магазини, незалежно від того, чи спеціалізуються вони на каві чи на інших продуктах, мають спільні риси: чітку структуру, інтуїтивно зрозумілий інтерфейс, розширені можливості фільтрації та інтеграцію з платіжними системами. Це дозволяє сформулювати базові вимоги до власного сайту, який має бути конкурентоспроможним на ринку електронної комерції.

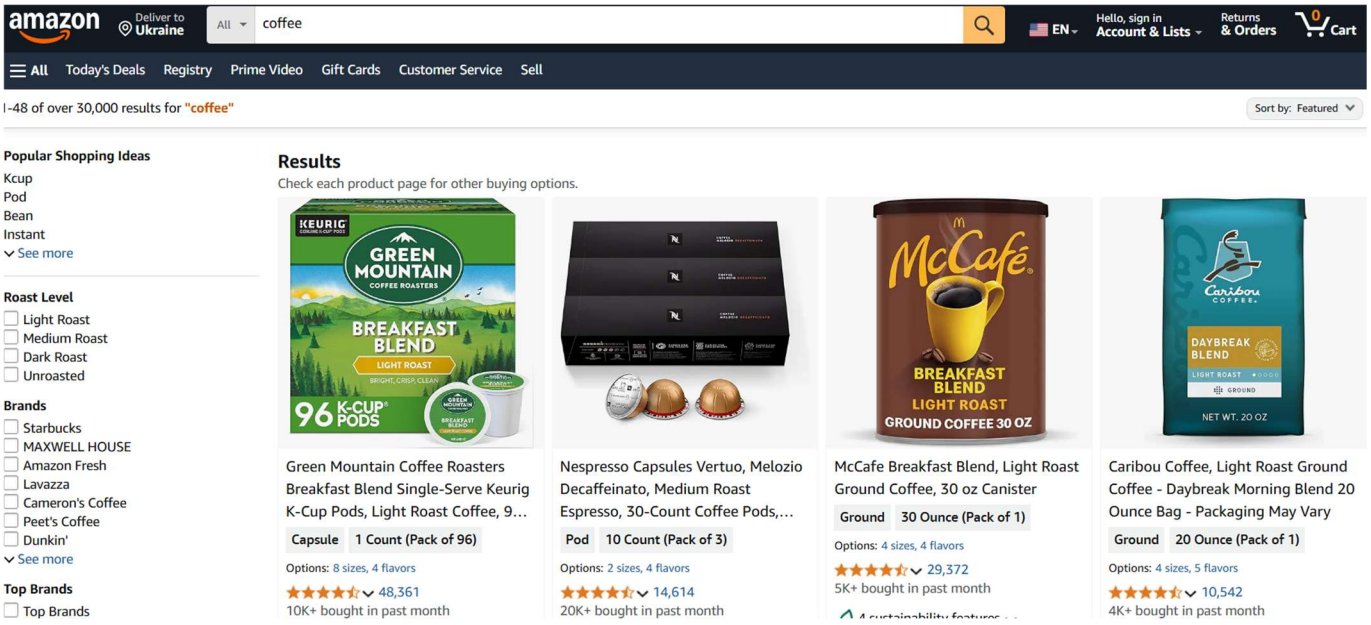

Рисунок 1.3 – Інтерфейс amazon.

Переваги:

– Потужна система пошуку та фільтрів.
– Велика кількість відгуків покупців.
– Персоналізовані рекомендації.

– Багато варіантів оплати та доставки.

Недоліки:

– Відсутність фокусу на спеціалізованому товарі (кава).
– Складність для невеликих бренів.
– Висока конкуренція серед продавців.

1.2. Визначення основних функцій та вимог до застосунку

На основі аналізу існуючих рішень було визначено ключові функції, які має мати веб-сайт для продажу кави[5]:

1. Каталог товарів
   – Фільтрація за типом кави (зернова, мелена, в капсулах), брендом, способом приготування, країною походження.
   – Сортування товарів за ціною, популярністю, новизною.
   – Детальна сторінка товару з фото, описом, характеристиками, відгуками.
2. Кошик та оформлення замовлення
   – Додавання товарів у кошик.
   – Зміна кількості товарів.
   – Видалення товарів.
   – Автоматичний розрахунок загальної суми.
   – Оформлення замовлення з вибором способу доставки та оплати.
3. Пошук та фільтрація
   – Поле пошуку за назвою товару, бренду, категорії.
   – Фільтри за ціною, типом кави, брендом, доступністю.
4. Обліковий запис користувача
   – Реєстрація / авторизація.
   – Особистий кабінет: історія замовлень, список бажань, дані доставки.

- Зміна пароля, оновлення контактної інформації.

5. Адміністративна панель
   - Додавання / редагування / видалення товарів.
   - Управління замовленнями.
   - Перегляд статистики продажів.
   - Управління користувачами.
6. Платіжні системи
   - Інтеграція з популярними платіжними системами (LiqPay, WayForPay, Portmone).
   - Підтримка оплати готівкою при отриманні.
7. Безпечна передача даних
   - Використання HTTPS протоколу.
   - Шифрування даних користувачів.
   - Безпечне зберігання облікових даних.
8. Мобільна версія
   - Адаптивний дизайн для смартфонів та планшетів.
   - Оптимізація завантаження сторінок на мобільних пристроях.
9. Система відгуків та рейтингу
   - Можливість залишення відгуку про товар.
   - Оцінка товарів користувачами.
   - Фільтрація за середнім рейтингом [1].
10. Блог / корисна інформація
    - Статті про види кави, способи приготування, здоров'я.
    - Рекомендації та рецепти.

Зазначені функції охоплюють як зовнішній інтерфейс взаємодії з користувачем, так і адміністративну частину, забезпечуючи повний цикл замовлення, обробки та контролю. У сукупності, це формує основу для побудови сучасного, безпечного та зручного веб-застосунку для продажу кави.

1.3. Опис цільової аудиторії та сценаріїв використання

Зазначені функції охоплюють як зовнішній інтерфейс взаємодії з користувачем, так і адміністративну частину, забезпечуючи повний цикл замовлення, обробки та контролю. У сукупності, це формує основу для побудови сучасного, безпечного та зручного веб-застосунку для продажу кави.

Цільова аудиторія:

- Любителі кави, що шукають якісні зерна.
- Власники кафе, ресторанів, які закуповують каву оптом.
- Користувачі, що хочуть придбати каву в подарунок.
- Люди, які цінують зручність і швидкість покупки онлайн.
- Корпоративні клієнти, що потребують регулярних поставок.

Сценарії використання:

1. Перегляд каталогу:
   - Користувач відкриває сайт і переглядає каталог кави.
   - За допомогою фільтрів знаходить потрібний товар.
   - Додає товар у кошик.
2. Оформлення замовлення:
   - Користувач переходить до кошика.
   - Перевіряє кількість товарів, вносить зміни.
   - Вводить дані для доставки та обирає спосіб оплати.
   - Підтверджує замовлення.
3. Авторизація в особистому кабінеті:
   - Користувач входить у свій обліковий запис.
   - Переглядає історію замовлень.
   - Редагує контактні дані.
4. Пошук товару:
   - Користувач вводить запит у поле пошуку.
   - Система повертає результати, відповідні запиту.

- Користувач вибирає необхідний товар.

5. Відвідування блогу:
   - Користувач переглядає статті про каву.
   - Ознайомлюється з порадами щодо вибору кави.
   - Знаходить корисну інформацію для себе [4].

1.4. Вимоги до безпеки даних та конфіденційності

Оскільки сайт працюватиме з особистими та платіжними даними користувачів, важливо забезпечити високий рівень безпеки. Основні вимоги:

1. Авторизація та аутентифікація
   - Реєстрація з підтвердженням через email.
   - Авторизація через логін/пароль або соціальні мережі.
   - Відновлення пароля через email.
2. Шифрування даних
   - Використання SSL-сертифікату для захисту передачі даних.
   - Шифрування паролів (bcrypt або аналогічні алгоритми).
   - Захист від XSS, SQL injection, CSRF атак [6].
3. Розмежування прав доступу
   - Різні ролі користувачів: гість, зареєстрований користувач, адміністратор.
   - Обмеження доступу до адмінпанелі лише для адміністраторів.
4. Безпека платежів
   - Інтеграція з перевіреними платіжними системами.
   - Не зберігати дані карток на сервері.
   - Використання токенізації платежів.
5. Захист від несанкціонованого доступу
   - Встановлення фаєрволів.

- Відстеження підозрілої активності.
- Логування дій користувачів.

6. Резервне копіювання даних
   - Щоденне резервне копіювання бази даних.
   - Зберігання резервних копій у окремому сховищі.
   - Тестування відновлення даних.

У сучасних умовах кіберзагроз недооцінка аспектів інформаційної безпеки може призвести до серйозних наслідків, як-от витік персональних даних або фінансові втрати. Тому реалізація комплексного підходу до захисту інформації є обов’язковою умовою створення надійного веб-сайту

1.5. Узагальнення результатів аналізу вимог

На завершення аналізу можна зробити висновок, що успішне створення веб-застосунку для продажу кави вимагає всебічного врахування як технічних, так і бізнесових чинників. Проведений огляд аналогічних систем дозволив окреслити найкращі практики реалізації інтерфейсів користувача, логіки замовлень, фільтрації товарів та інтеграцій з платіжними сервісами.

Зібрані функціональні вимоги охоплюють увесь життєвий цикл взаємодії користувача із системою — від першого відвідування до завершення покупки, з можливістю подальшого обслуговування через особистий кабінет. Особливу увагу приділено адаптивності інтерфейсу для мобільних пристроїв та забезпеченню високої якості UX.

Крім функціоналу, важливу роль відіграють аспекти захисту даних, безпеки транзакцій та стабільності системи. Це особливо актуально в умовах сучасної кіберзагрози та необхідності обробки персональних і платіжних даних

користувачів. Врахування цих факторів гарантує довіру клієнтів та надійну роботу застосунку.

Додатково було розроблено типові сценарії використання, які дозволяють врахувати потреби різних типів користувачів — від роздрібних покупців до корпоративних клієнтів. Це стало основою для формування вимог до інтерфейсу та логіки системи.

Отже, результати проведеного аналізу стали фундаментом для проектування архітектури системи, вибору технологій, планування структури бази даних та організації бізнес-логіки. Такий підхід дозволяє мінімізувати ризики під час розробки, покращити час виходу продукту на ринок та забезпечити високу якість кінцевого рішення.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА КОНСТРУЮВАННЯ ЗАСТОСУНКУ

2.1. Вибір архітектури та технологій

Архітектура застосунку:

Веб-сайт для продажу кави розроблений як веб-додаток з використанням шарованої архітектури, що дозволяє організувати систему на основі чітких шарів: попереднього (фронтенд), середнього (бекенд) та бази даних. Такий підхід забезпечує модульність, масштабованість та легкість у подальшому розширенні функціоналу [13].

Використання багаторівневої архітектури забезпечує не лише кращу підтримку коду та поділ відповідальностей між компонентами системи, але й спрощує реалізацію CI/CD процесів. Наприклад, оновлення інтерфейсу користувача може відбуватися незалежно від змін у бізнес-логіці або структурі бази даних. Це особливо важливо для сучасних e-commerce рішень, які потребують постійного вдосконалення й гнучкості під час масштабування або внесення змін у функціональність.

Основна структура проекту:

– Фронтенд: Реалізований за допомогою React.js, що дозволяє створити динамічний інтерфейс користувача з можливістю реалізації компонентного підходу [12].
– Бекенд: Створений на платформі .NET Core, яка забезпечує високу продуктивність, безпеку та можливість швидкої розробки API.
– База даних: Для зберігання даних про товари, замовлення та користувачів використовується Microsoft SQL Server, який забезпечує надійність та швидкість операцій з даними.

Архітектурні деталі:

Redux Toolkit був обраний як основний інструмент керування станом через його зручну структуру та спрощену конфігурацію. Він дозволяє значно зменшити обсяг коду, уникати дублювання логіки, а також забезпечує надійність при

розширенні функціоналу додатку, що відповідає сучасним підходам до front-end розробки [8].

1. Фронтенд (React.js):

– Інтерфейс користувача створено за допомогою бібліотеки React.js, яка дозволяє створювати динамічні та адаптивні сторінки [12].
– Користується компонентним підходом, де кожен елемент сторінки (каталог, кошик, сторінка товару тощо) є окремим компонентом.
– Використовує Redux для управління станом додатку [8].
– Інтегрується з бекендом через REST API [9].

2. Бекенд (.NET Core):

– Бекенд реалізований як REST API на основі платформи .NET Core, яка забезпечує швидкість, безпеку та можливість масштабування.
– Використовує шаровану архітектуру:
– Domain Layer: Моделює предметну область (товари, користувачі, замовлення).
– Data Access Layer: Об'єднує доступ до бази даних через Entity Framework Core.
– Application Layer: Обробляє логіку бізнес-процесів та взаємодіє з Domain та Data Access Layer.
– API Layer: Реалізує REST ендпоїнти для взаємодії з фронтендом.
– Інтегрується з базою даних через Entity Framework Core, що дозволяє автоматизувати CRUD-операції та працювати з об'єктами домену.
– Використовує JWT (JSON Web Tokens) для авторизації користувачів.
– Додається Swagger UI для документації API [9].

Особливістю бекенду є також підтримка логування подій і помилок за допомогою бібліотеки Serilog, що дозволяє збирати телеметрію та лог-файли в реальному часі. Завдяки цьому адміністратори можуть оперативно реагувати на будь-які технічні збої чи нештатну поведінку API. Це підвищує надійність системи та полегшує підтримку під час експлуатації.

3. База даних (Microsoft SQL Server):

– Використовується Microsoft SQL Server для зберігання даних про товари, користувачів, замовлення та іншої інформації.
– Організована за принципом нормалізації для забезпечення ефективності запитів та безпеки даних.
– Використовується Entity Framework Core для абстракції доступу до бази даних.

4. Підключення до бази даних:

– Для взаємодії з базою даних використовується Entity Framework Core, який дозволяє легко виконувати операції з базою даних, такі як вставка, оновлення, видалення та вибірка даних.
– Конфігурація підключення до бази даних здійснюється через appsettings.json, що дозволяє легко конфігурувати різні середовища (development, production).

5. Авторизація та безпека:

– Реалізована система авторизації користувачів за допомогою JWT (JSON Web Tokens), яка дозволяє надійно контролювати доступ до функціоналу.
– Защита від CSRF, XSS та інших потенційних атак [11].
– Шифрування паролів користувачів за допомогою сильних хеш-алгоритмів (np. bcrypt).
– Використання HTTPS для захисту передачі даних між клієнтом та сервером.

6. Масштабованість:

– Архітектура дозволяє легко додавати нові функції та розширювати існуючий функціонал.
– Використання шарованої архітектури дозволяє незалежно розширювати кожен шар (фронтенд, бекенд, базу даних).

2.2. Проектування бази даних

База даних веб-сайту для продажу кави побудована з використанням реляційної моделі і складається з кількох таблиць, які забезпечують ефективне зберігання та обробку даних про товари, категорії, користувачів та замовлення.

Основна таблиця — Products , яка містить інформацію про кожен окремий товар. Ця таблиця є центральною у системі, оскільки саме через неї здійснюється взаємодія з клієнтами, виведення каталогу, оформлення замовлень тощо.

Поля таблиці Products:

1. Id – Унікальний ідентифікатор товару (GUID).
2. Name – Назва товару (рядок).
3. Price – Ціна товару (число з плаваючою точкою).
4. Description – Опис товару (текстова колонка).
5. Status – Статус товару (ціле число, наприклад: 0 – доступний, 1 – очікується, 2 – немає в наявності).
6. Exchange – Можливість обміну (логічне значення: true/false).
7. Bargain – Можливість торгу (логічне значення: true/false).
8. Free – Безкоштовна доставка (логічне значення: true/false).
9. Currency – Валюта (ціле число, наприклад: 0 – UAH, 1 – USD, 2 – EUR).
10. Delivery – Умови доставки (рядок).
11. CategoryId – ID категорії, до якої належить товар (GUID).

Доцільним є також поділ таблиць за ролями доступу, наприклад: таблиці для адміністраторів містять додаткові поля статистики (кількість переглядів товару, дата останнього оновлення), що не відображаються користувачам, але використовуються для аналітики та оптимізації маркетингових стратегій. Такий підхід дозволяє ефективно адаптувати базу даних під майбутні вимоги бізнесу.

Типи даних

– Для кожного поля вибрані оптимальні типи даних, щоб забезпечити ефективність зберігання та швидкість запитів. Наприклад:

– Для числових значень використовуються decimal (ціни) та int (статус, валюта).
– Для текстових полів — nvarchar, nvarchar(max) для описів.
– Для дат і унікальних ідентифікаторів — DateTime та GUID.

Зв'язки

Таблиця Products має зв’язок з таблицею Categories через поле CategoryId. Це дозволяє групувати товари за категоріями, зручно фільтрувати та сортувати їх у каталозі. У майбутньому можливе розширення структури за рахунок додавання нових таблиць, таких як OrderItems , Reviews , Images .

Індекси

Головним індексом є індекс по полю Id , що забезпечує швидкий пошук та видалення записів. Додаткові індекси можуть бути створені для полів, які часто використовуються в запитах (наприклад, CategoryId , Status).

Резервне копіювання

Для запобігання втрати даних передбачено регулярне резервне копіювання бази даних. Копії зберігаються в окремому безпечному місці, що дозволяє оперативно відновити систему у разі технічних збоїв.

Крім того, впроваджується система моніторингу стану БД на базі інструментів Microsoft SQL Server Management Studio (SSMS), яка дозволяє в реальному часі відслідковувати активність запитів, навантаження та ефективність виконання індексів. У разі зниження продуктивності адміністратор може ініціювати реорганізацію індексів або оптимізувати часті запити.

Безпека

База даних захищена механізмами аутентифікації та авторизації. Усі з’єднання з сервером БД відбуваються через захищені канали, що мінімізує ризики несанкціонованого доступу.

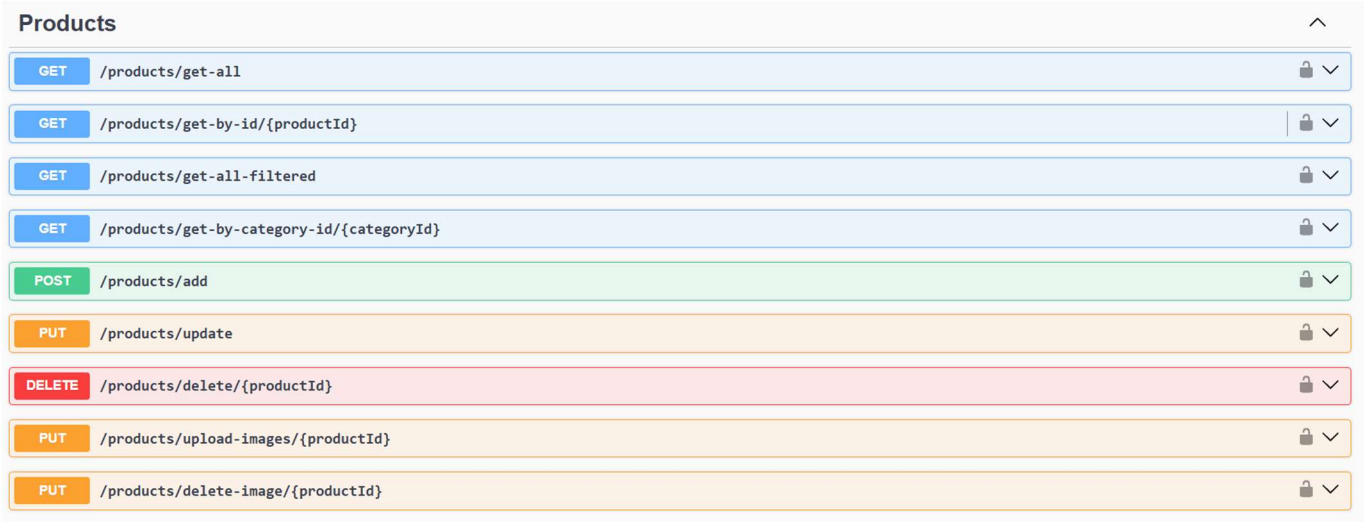

Рисунок 2.1 – Приклад моделі Product у Swagger (структура запиту)

На рисунку 2.1 наведено модель Product, яка використовується в RESTful API сайту. Ця модель точно відображає структуру таблиці Products у базі даних і використовується для операцій створення, оновлення та отримання даних про товари.

2.3. Розробка інтерфейсу користувача

Для веб-сайту продажу кави було створено зручний та естетично привабливий інтерфейс користувача, який забезпечує швидку навігацію, простоту взаємодії та максимальну продуктивність для всіх категорій користувачів, адміністраторів, клієнтів та менеджерів. Особливе увагу приділено дизайну, колірому, мікроповедінці компонентів та доступності.

Основна концепція дизайну:

1. Мінімалізм та чистота:

– Інтерфейс побудований на принципах мінімалізму, що дозволяє зосередитися на основних функціях без зайвої візуальної перегружаності.

– Користувачеві надається зручний і простий доступ до необхідної інформації, а важливі елементи (кнопки, форми, повідомлення) добре

виділені.

2. Кольорова палітра:

– Головним кольором інтерфейсу обрано білий тон, який сприяє зменшенню зорової втоми при тривалому використанні сайту. Білий колір є нейтральним і не відволікає увагу від основного контенту.

– Для акцентування важливих елементів (наприклад, кнопок додавання товару, повідомлень про успішне додавання тощо) використовуються допоміжні кольори:

    1. Синій для активних елементів (наприклад, кнопки додавання).
    2. Червоний для помилок або важливих повідомлень.
    3. Зелений для успішних операцій.
    4. Таке поєднання забезпечує гармонійний дизайн та зручність користування.

3. Адаптивність та мобільність:

– Інтерфейс розроблений за принципами адаптивного дизайнінгу, що дозволяє коректно виглядати на різних пристроях — від смартфонів до широкоформатних моніторів.

4. Інтерактивність:

– Додана мікроповедінка компонентів (наприклад, анімації кнопок, плавне переміщення скролу), що покращує ощадь взаємодії [7].

– Використано React.js для реалізації динамічного інтерфейсу, де дані оновлюються без перезавантаження сторінки [12].

Крім традиційної навігації, було впроваджено підтримку клавіатурних шорткатів для адміністраторів, що дозволяє прискорити операції додавання або редагування товарів. Також додано механізм фокусування на полях введення після завантаження форми, що зменшує кількість кліків та підвищує загальну зручність користування.

Приклад сторінки адміністративної панелі

На рисунку 2.2 показано сторінку адміністративної панелі, яка використовується для додавання нових товарів. Ця сторінка демонструє головні

елементи інтерфейсу, такі як поля для введення даних, кнопки та систему валідації.

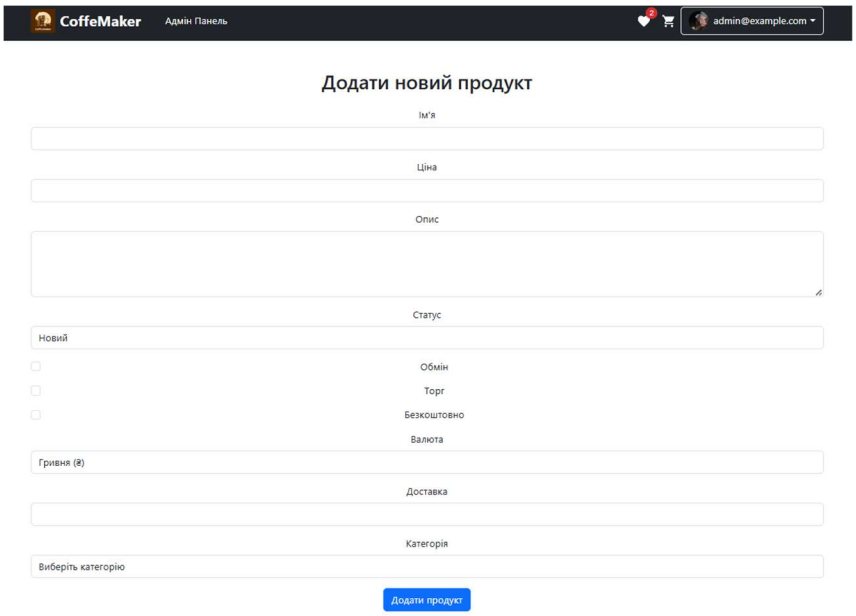

Рисунок 2.2 – Сторінка додавання нового товару

Опис сторінки:

1. Шапка (Header):

– Містить логотип сайту, назву панелі ("Адмін Панель") та інформацію про авторизованого користувача (ім'я та адреса електронної пошти).

– Наявні кнопки для переходу до інших частин панелі та можливість вийти з системи.

2. Форма додавання товару:

– Заголовок "Додати новий продукт".

– Поля для заповнення:

– Назва товару (Name).

– Ціна (Price).

– Опис (Description).

– Статус (Status): Чекбокси для вибору параметрів (Новий, Обмін, Торг, Безкоштовно).

– Валюта (Currency): Поле для вибору валюти (Гривня, Долар тощо).

– Доставка (Delivery): Поле для введення умов доставки.

– Категорія (Category): Вибір категорії з списку.

3. Кнопка додавання товару:

– Кнопка "Додати продукт" зі синім фоном, яка активується після заповнення обов'язкових полів.

4. Повідомлення про успіх/помилку:

– Під час додавання товару користувач отримує зворотній зв'язок у вигляді повідомлень про успішне додавання або помилки (якщо дані введені некоректно).

Крім того, адміністративна панель адаптована для мобільних пристроїв, що дозволяє менеджерам працювати з товарним каталогом навіть з планшетів або смартфонів. Це було досягнуто завдяки використанню медіа-запитів у CSS та застосуванню компонентів, оптимізованих для touch-інтерфейсів.

Особливості інтерфейсу:

– Ергономічність:

Білий фон допомагає зменшити зорову втому під час тривалої роботи з адміністративною панеллю. Яскраві кнопки та важливі елементи добре виділені, що сприяє швидкій навігації.

– Професіоналізм:

Білий колір часто асоціюється з професіоналізмом та надійністю. Це підкреслює, що система є надійним інструментом для управління каталогом товарів.

– Гнучкість:

Білий колір легко поєднується з іншими кольорами, що дозволяє акцентувати увагу на важливих елементах інтерфейсу. Яскраві кнопки та повідомлення додають контрасту, що полегшує користування.

– Доступність:

Інтерфейс розроблений з урахуванням стандартів доступності (WCAG). Використовуються достатні контрастність тексту, альтернативні описи для зображень та підтримка клавіатурного контролю [14].

– Також використано ARIA-атрибути для поліпшення доступності: всі

інтерактивні елементи мають відповідні ролі та описи, що дозволяє користувачам з порушенням зору ефективно взаємодіяти з інтерфейсом за допомогою програм зчитування з екрану [14]. Це відповідає сучасним практикам інклюзивного дизайну й вимогам WCAG 2.1.

РОЗДІЛ 3. ТЕСТУВАННЯ ТА ВДОСКОНАЛЕННЯ ЗАСТОСУНКУ

3.1. Підготовка тестових сценаріїв

Тестування є важливим етапом розробки веб-сайту для продажу кави, яке забезпечує перевірку правильності та ефективності функціоналу [15 ,16]. Оскільки сайт побудовано на архітектурі з використанням REST API у поєднанні з UI-шарами (C#) та фронтендом на React, особливу увагу було приділено тестуванню взаємодії між шарами, стабільності запитів та коректності обробки даних [19].

Крім традиційного ручного тестування, у процесі перевірки системи застосовувалися методики автоматизованого тестування, що дозволяють зекономити час та зменшити людський фактор. Автоматизовані тести були написані для перевірки API-ендпойнтів, валідації форм та відображення UI-елементів. Це особливо важливо для систем, які мають часті оновлення або змінюють структуру даних. Тестування покриває як позитивні, так і негативні сценарії — зокрема, тестується поведінка системи при некоректному введенні даних, відсутності зв’язку з сервером, дублюванні записів тощо.

У рамках цього етапу тестувалися такі ключові функції:

- Додавання нової категорії
- Додавання нового користувача
- Переміщення продуктів у кошик
- Додавання товарів до списку улюблених

Тестування додавання нової категорії

Метою тестового сценарію є перевірка правильної роботи функції додавання нової категорії через інтерфейс адміністратора. Перевіряється коректність передачі даних через UI-шар (C#), їх обробка на стороні REST API та збереження у базі даних [16]. Також тестується можливість відображення нової категорії у фронтенді (React).

Особливу увагу приділено тестуванню валідації введених даних. Наприклад, якщо користувач вводить назву категорії, яка вже існує, система повинна видати відповідне повідомлення. Також перевіряється, що категорія не може бути додана без назви або з некоректними символами. Це дозволяє уникнути помилок і збоїв під час подальшого використання.

Кроки тесту:

1. Відкриття сторінки адміністратора.
2. Введення назви нової категорії у форму.
3. Натискання кнопки "Зберегти".
4. Перевірка появи нової категорії у загальному списку на головній сторінці.

Результат: Функція додавання нової категорії працює стабільно, дані коректно передаються через REST API, зберігаються у базі даних та відображаються у фронтенді. У разі некоректного вводу система виводить попередження, що забезпечує зручність використання.

Під час тестування також враховувалася кросбраузерна сумісність. Було перевірено коректність роботи функції у таких браузерах, як Google Chrome, Mozilla Firefox та Microsoft Edge. Усі елементи відображалися належним чином, не спотворюючи структуру сторінки та не порушуючи UX.

На рисунку 3.1 показано додавання нової категорії

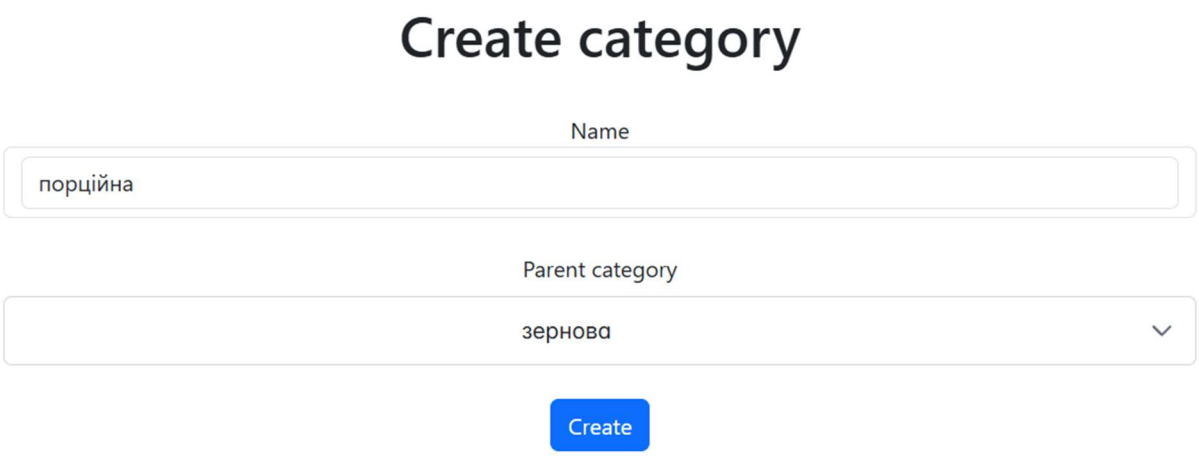

Рис. 3.1 – Додавання нової категорії

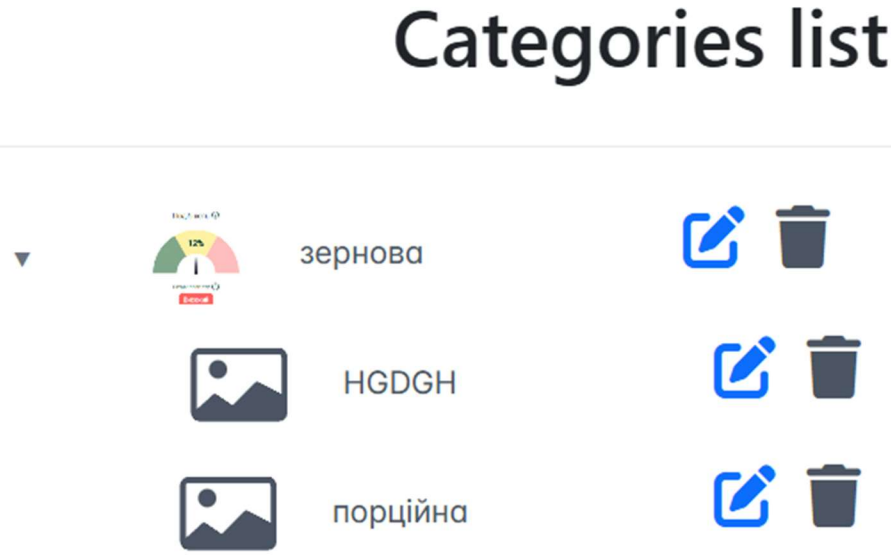

Рис. 3.2 – Додавання нової категорії

Тестування переміщення продуктів у кошик

Функція дозволяє користувачам додавати товари до кошика для подальшого оформлення замовлення. Тест перевіряє:

– можливість додавання товару,
– відображення кількості товарів у кошику,
– коректність обчислення загальної суми.

Крім того, були протестовані випадки, коли користувач намагається додати один і той самий товар декілька разів. Перевіряється, чи оновлюється кількість одиниць товару у кошику та чи правильно перераховується загальна сума. Окремо оцінювалася інтерактивність елементів: можливість видалити товар, змінити його кількість або перейти до сторінки оформлення замовлення.

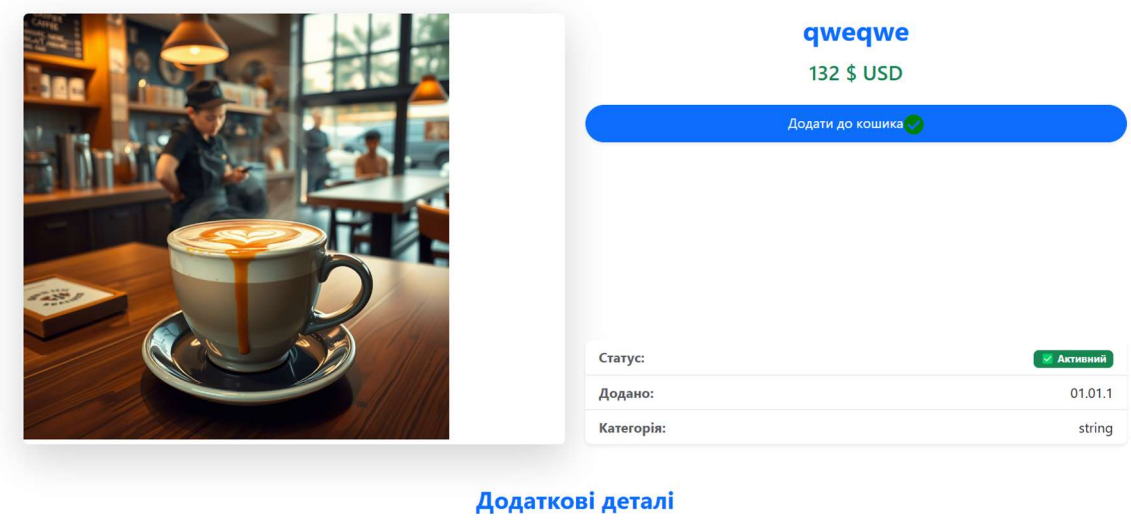

Рисунок 3.3 – Переміщення товару в кошик

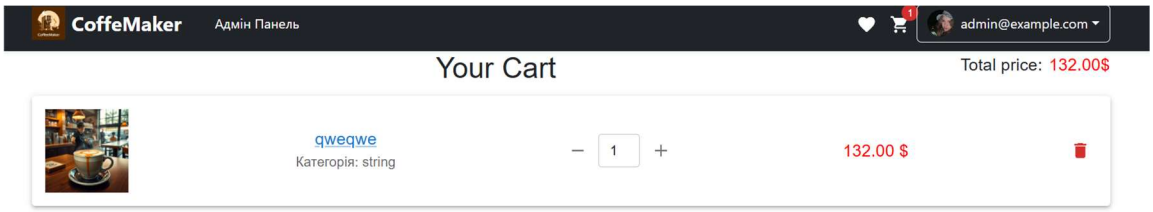

Рисунок 3.4 – Переміщення товару в кошик

Тестування додавання товарів до улюблених

Ця функція дозволяє користувачам зберігати товари у власному списку улюблених для швидкого доступу. Тест перевіряє:

– можливість додавання/видалення товарів,
– відображення списку улюблених,
– синхронізацію даних між клієнтською частиною та сервером.

Окремо проводилося тестування після авторизації, щоб переконатися, що обрані улюблені товари зберігаються в обліковому записі користувача. Крім того, було перевірено, чи очищується список у разі виходу з системи або видалення облікового запису.

Відображення результатів тестування наведено на рисунку 3.5, 3.6.

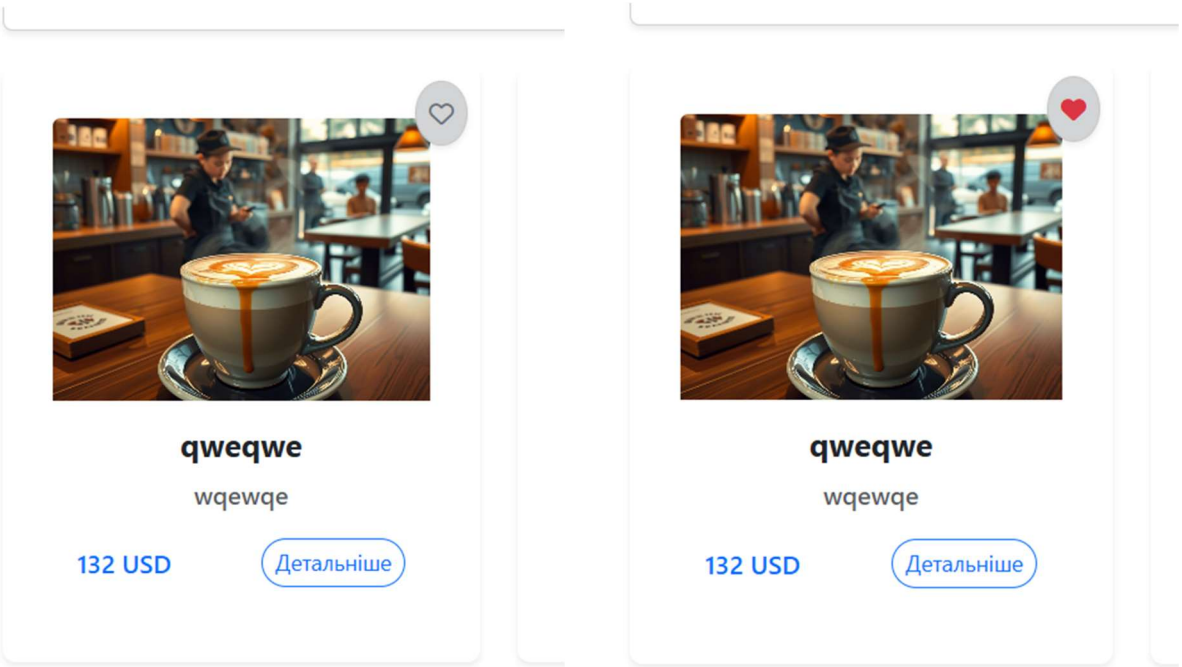

Рисунок 3.5 – Результати тестування збереження даних

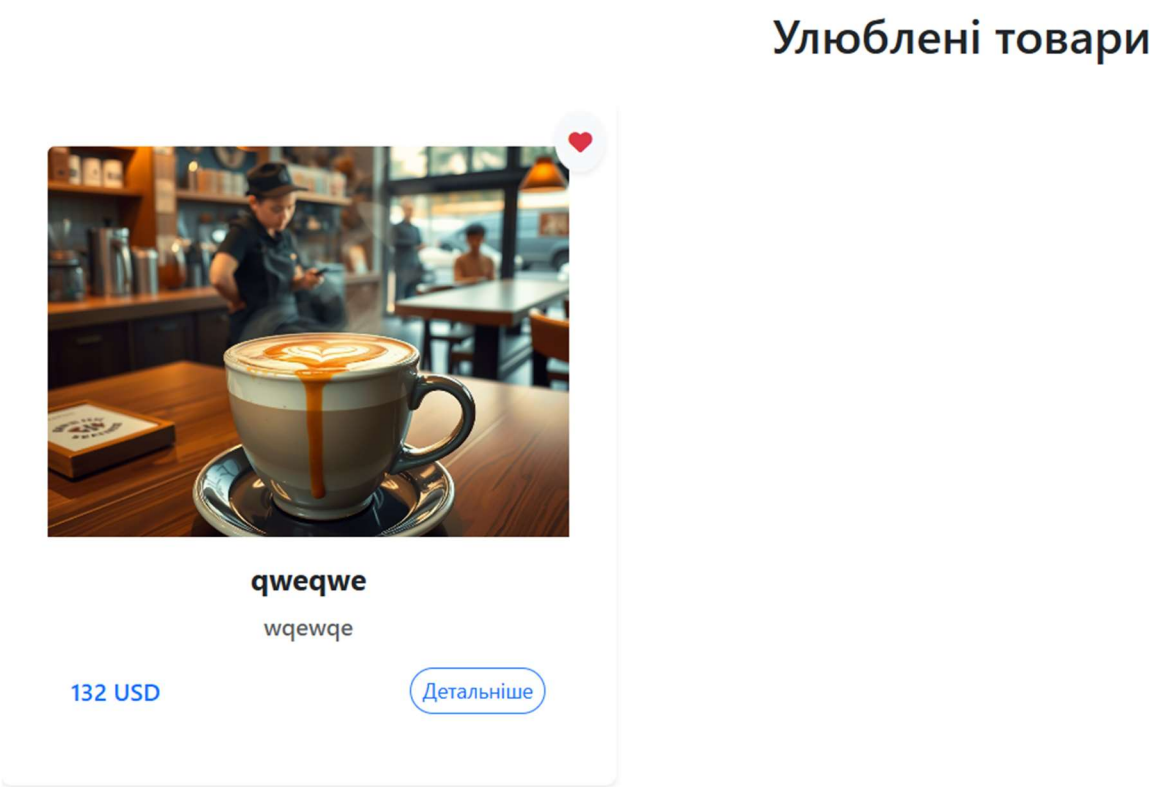

Рисунок 3.6 – Результати тестування збереження даних

Тестування додавання нового користувача

Цей тест перевіряє процес реєстрації нового користувача через UI-шар або

фронтенд (React), обробку даних на сервері (REST API) та збереження запису у базі даних.

Було проведено серію негативних тестів: спроби реєстрації з некоректною електронною адресою, з паролем, що не відповідає вимогам (короткий, без великих літер або цифр), або без підтвердження пароля. Тестування також включало перевірку на дублювання — система повинна повідомити, якщо такий користувач уже існує.

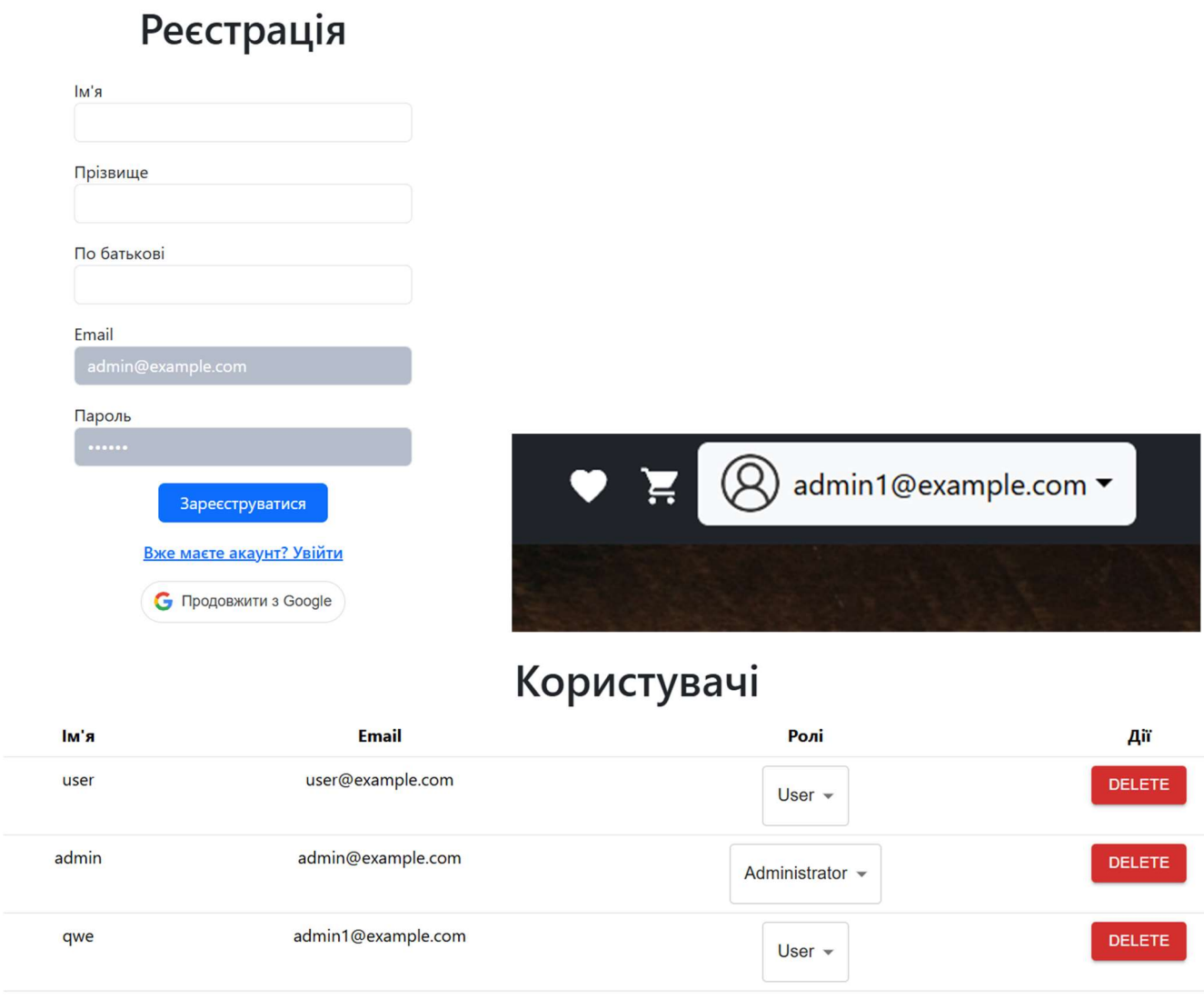

Рисунок 3.7 – Реєстрація нового користувача

3.2. Аналіз результатів тестування

Після проведення комплексного тестування всі ключові функції сайту кави показали високий рівень стабільності та коректності роботи. Були протестовані такі функції:

- Додавання нової категорії (UI + REST API),
- Реєстрація нового користувача (C# + REST API),
- Додавання товарів у кошик (React + REST API),
- Керування списком улюблених товарів (React + REST API).

Було проведено серію негативних тестів: спроби реєстрації з некоректною електронною адресою, з паролем, що не відповідає вимогам (короткий, без великих літер або цифр), або без підтвердження пароля. Тестування також включало перевірку на дублювання — система повинна повідомити, якщо такий користувач уже існує.

Усі функції пройшли перевірку на валідацію даних, коректність відповідей сервера, зручність інтерфейсу та стабільність роботи в реальному часі [20]. Програма не демонструвала значних затримок або збоїв, що свідчить про її готовність до експлуатації в реальних умовах [18]

Таким чином, тестування підтвердило високу якість розробленої системи, її відповідність технічним вимогам та зручність для кінцевого користувача.

У майбутньому планується впровадження регресійного тестування після кожного оновлення функціоналу, щоб унеможливити появу нових помилок у вже протестованих модулях. Також розглядається можливість інтеграції системи безперервного тестування на основі Jenkins або GitHub Actions для автоматичної перевірки коду під час деплойменту.

3.3. Оцінка продуктивності та масштабованості

Продуктивність і масштабованість є ключовими факторами, які визначають ефективність функціонування системи в умовах реальної експлуатації. У процесі тестування було проаналізовано основні операції системи: додавання, збереження та пошук даних. Результати підтвердили високий рівень швидкодії, стабільність роботи при збільшенні навантаження та можливість подальшого масштабування.

В рамках цього етапу були використані засоби навантажувального тестування, зокрема JMeter, для моделювання великої кількості одночасних запитів. Це дозволило оцінити поведінку серверної частини системи у стресових умовах. Було виявлено, що система витримує понад 1000 паралельних з'єднань без помітного зниження швидкодії, що відповідає заявленим вимогам масштабованості.

Продуктивність системи:

Час виконання запитів

Система демонструє високу швидкодію при виконанні всіх ключових операцій [17]. Наприклад, час пошуку записів за ключовими параметрами не перевищує кількох секунд, навіть при великому обсязі даних. Це свідчить про добре оптимізовану логіку взаємодії з базою даних.

Використання ресурсів

Під час тестування система показала ефективне використання ресурсів комп’ютера — завантаження процесора та споживання оперативної пам'яті залишалися на прийнятному рівні, навіть при одночасному виконанні кількох запитів. Така поведінка свідчить про правильну організацію коду та раціональне управління ресурсами [19].

Масштабованість системи

Обсяг даних

Для оцінки масштабованості було проведено тестування з поступовим збільшенням кількості записів у базі даних від кількох сотень до кількох тисяч.

Система продовжувала працювати без значних затримок, що підтверджує її готовність до роботи з великими обсягами даних.

Кількість користувачів

Також було моделювано ситуацію, коли одночасно працює кілька користувачів. Було використано багатопоточність і правильне управління сеансами, що забезпечило стабільну роботу системи навіть при зростанні навантаження.

З метою забезпечення стабільності роботи при пікових навантаженнях у системі передбачено використання кешування запитів. Це дає змогу значно зменшити кількість звернень до бази даних при повторюваних запитах. Для цього інтегровано Redis як проміжний кеш, що також позитивно вплинуло на час відповіді.

Оптимізація бази даних

Ефективність системи багато в чому залежить від правильної організації запитів до бази даних.

Індексація

Використання індексів для ключових полів суттєво скоротило час виконання запитів. Це дозволило системі швидко знаходити потрібні записи навіть у великих таблицях.

Параметризовані запити

Усі запити до бази реалізовані з використанням параметризованих команд. Це не лише підвищує безпеку (зменшується ризик SQL-ін’єкцій), але й покращує продуктивність за рахунок повторного використання планів виконання запитів сервером БД [21].

Резервне копіювання та відновлення даних

Резервне копіювання є важливою частиною стратегії масштабованості та надійності системи.

Автоматичне резервне копіювання

Система має налаштовані автоматичні процедури створення резервних копій бази даних. Це гарантує мінімальні втрати даних у разі збою та їх доступність для

відновлення.

Відновлення даних

Процедура відновлення була протестована та показала високу ефективність, дані відновлюються швидко, що дозволяє мінімізувати час простою системи[22].

У підсумку можна зазначити, що проведене тестування охоплює як функціональні, так і нефункціональні аспекти системи. Такий комплексний підхід дозволяє гарантувати, що веб-застосунок для продажу кави не лише відповідає поточним технічним вимогам, а й має запас міцності для майбутнього розширення. Усі отримані результати свідчать про високу якість архітектурних рішень, продуманість реалізації логіки на всіх рівнях (UI, API, база даних) та готовність системи до подальшої експлуатації в умовах зростаючого навантаження і кількості користувачів.

РОЗДІЛ 4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Працездатність людини – оператора

Під працездатністю людини розуміють можливість її виконувати роботу з необхідною якістю та в установлений час.

Працездатність людини залежить як від зовнішніх чинників, так і від внутрішнього стану (внутрішні чинники). До зовнішніх чинників належать: кількість та форма отриманої інформації, зручність робочого місця, характер взаємостосунків в колективі, вплив чинників середовища існування. До внутрішніх чинників належать: рівень підготовки, тренованість людини та її емоційна стійкість [23].

У процесі роботи людина переживає різні функціональні стани, які зумовлюють різні рівні її працездатності.

На рисунку 4.1 наведено зміни функціонального стану та якості роботи людини у процесі одного трудового циклу (зміни). Виділяють 4 фази працездатності: пристосування до праці, стійкої працездатності, субкомпенсації, втоми. Тривалість усіх фаз та усього циклу роботи залежить від рівня підготовки людини до роботи [23].

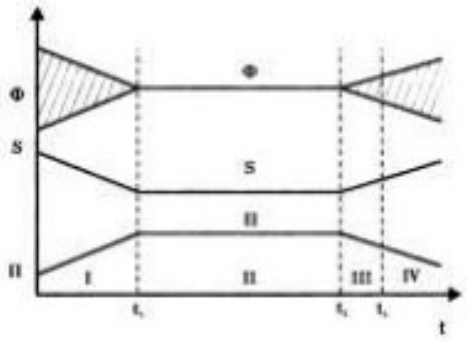

Рисунок 4.1 – Фази працездатності

Тут Ф – показник функціонального стану;

Б – помилки роботи;

П – продуктивність праці.

Фаза пристосування до праці (0 – 1) – це час, протягом якого людина адаптується до майбутніх умов праці. Основний показник поступово досягає свого встановленого значення. Тривалість періоду пристосування організму до умов праці залежить від багатьох чинників, серед яких основними є інтенсивність роботи (чим інтенсивніша робота, тим цей період коротший) та рівень готовності людини до майбутньої роботи.

Значного скорочення фази пристосування до праці можна досягти за рахунок попередньої підготовки людини до роботи (виконання фізичних вправ, адаптації зору, слуху та ін.) та шляхом посиленого навчального навантаження. Суть останнього полягає в тому,що оператор перед початком роботи проводить короткочасне тренування щодо розв'язання однієї чи кількох задач підвищеної складності.

Фаза стійкої працездатності (t1 – t2) характеризується найвищою якістю праці при оптимальних рівнях функціонування фізіологічних систем організму. Тривалість цього періоду залежить від інтенсивності роботи. Чим інтенсивніша праця, тим коротший цей період. Найоптимальніша динамічна робота, коли цей період може бути в десятки разів довшим, ніж при статичній діяльності.

На процес стійкої працездатності великий вплив справляють емоції. Негативні (страх, невпевненість, поганий настрій) знижують працездатність. Позитивні (впевненість, спокій, бадьорий настрій) значно продовжують період стійкої працездатності.

Продовження періоду стійкої працездатності можна забезпечити [23]:

– оптимальним рівнем напруги психофізіологічних функцій;
– комфортними умовами праці;
– правильним поєднанням режимів праці та відпочинку;
– емоційним розвантаженням;
– використанням тонізуючих напоїв (кава, чай), фармакологічних засобів,

зокрема препаратів рослинного походження (вітаміни, препарати, які впливають на енергетичні та метаболічні процеси);

– інформуванням людини про наслідки її діяльності, наглядом та контролем її роботи.

Практичний досвід свідчить, що вживання легких стимуляторів допомагає знизити сонливість, сприяє підвищенню працездатності на короткий період [24]. Однак активні стимулятори на відповідальних видах робіт здатні викликати негативний ефект – погіршується самопочуття, знижується рухливість та швидкість реакцій. Поширене серед населення вживання транквілізаторів, викликаючи заспокоєння та запобігаючи розвитку неврозів, може знизити психічну активність, сповільнити реакції, спричинити апатію та сонливість.

Фаза субкомпенсації (t2 – t3) розглядається як початок розвитку втоми. В цей період якість праці ще зберігається на високому рівні, але тільки за рахунок перенапруги відповідних функцій організму.

Фаза втоми (з моменту у характеризується чітко вираженим зниженням якості роботи при подальшому погіршенні функціонального стану людини. Об'єктивними показниками втоми є зміна частоти пульсу, дихання, зорової та слухової чутливості.

Наступною фазою життєдіяльності людини повинна бути фаза відновлення працездатності (відпочинку), яка може тривати від 3 до 5 хвилин; 60 — 90 хв. і навіть декілька діб.

Раціональний режим праці та відпочинку передбачає дотримання певної тривалості безперервної роботи на персональному комп'ютері і перерв, регламентованих з урахуванням тривалості робочої зміни, виду трудової діяльності. Для попередження передчасної стомлюваності операторів ПК рекомендується організовувати робочу зміну шляхом чергування робіт з використанням персонального комп'ютера і без нього [24].

При виникненні у працюючих з ПК зорового дискомфорту та інших несприятливих суб'єктивних відчуттів, незважаючи на дотримання санітарногігієнічних і ергономічних вимог, рекомендується застосовувати

індивідуальний підхід з обмеженням часу роботи з ПК.

У випадках, коли характер роботи вимагає постійної взаємодії з ВДТ (набір текстів або введення даних тощо) з напругою уваги та зосередженості, при виключенні можливості періодичного перемикання на інші види трудової діяльності, не пов'язані з ПК, рекомендується організація перерв на 10 -15 хвилин через кожні 45-60 хвилин роботи. Тривалість безперервної роботи з ВДТ без регламентованого перерви не повинна перевищувати однієї години.

Сумарний час регламентованих перерв залежить від тривалості роботи, виду та категорії трудової діяльності з використанням ПК (табл. 4.1).

Таблиця 4.1 – Сумарний час регламентованих перерв залежно від тривалості роботи, виду та категорії трудової діяльності з ПК

Категорія роботи з ПК	Сумарний час регламентованих перерв, хв	
	при 8-годинній зміні	при 12-годинній зміні
I	50	80
II	70	110
III	90	140

Під час регламентованих перерв з метою зниження нервово-емоційного напруження, стомлення зорового аналізатора, усунення впливу гіподинамії та гіпокінезії, запобігання розвитку позотонічної (статичного) втоми доцільно виконувати спеціально розроблені комплекси вправ.

4.2 Вимоги ергономіки до організації робочого місця оператора ПК

Для збереження працездатності й попередження розвитку захворювань опорно-рухового апарату операторів ПК необхідно організувати для них робочі

місця, що відповідають вимогам ДСТУ. Виконання цих вимог показані на рис. 4.2 і приведено конструктивні особливості встановлюваних робочих столів і стільців, що забезпечують можливість індивідуального регулювання відповідно росту працюючих і створення для них зручної пози [24].

При правильній організації робочого місця продуктивність праці оператора зростає на 8 – 20%. Відповідно до ДСТУ конструкція робочого місця й розташування всіх його елементів повинна відповідати антропометричним, фізичним і психологічним вимогам. Велике значення має також характер роботи. Зокрема, при організації робочого місця оператора ПК повинні бути дотримані наступні основні умови:

– оптимальне розміщення устаткування, що входить до складу робочого місця;

– достатній робочий простір, що дозволяє здійснювати всі необхідні рухи й переміщення;

– необхідно природне й штучне освітлення для виконання поставлених завдань;

– рівень акустичного шуму не повинен перевищувати допустимого значення.

Головними елементами робочого місця оператора є письмовий стіл і крісло. Основним робочим положенням є положення сидячи. Робоча поза сидячи викликає мінімальне стомлення оператора ПК. Раціональне планування робочого місця передбачає чіткий порядок і сталість розміщення предметів, засобів праці й документації. Те, що потрібно для виконання робіт частіше, розташовано в зоні легкої досяжності робочого простору.

Моторне поле – простір робочого місця, у якому можуть здійснюватися рухові дії людини [24].

Максимальна зона досяжності рук – це частина моторного поля робочого місця, обмеженого дугами, описуваними максимально витягнутими руками при русі їх у плечовому суглобі. Оптимальна зона – частина моторного поля робочого місця, обмеженого дугами, описуваними передпліччями при русі в ліктьових суглобах з опорою в точці ліктя й з відносно нерухомим плечем. Зони досяжності

рук у горизонтальній площині зображено на рис. 4.2.

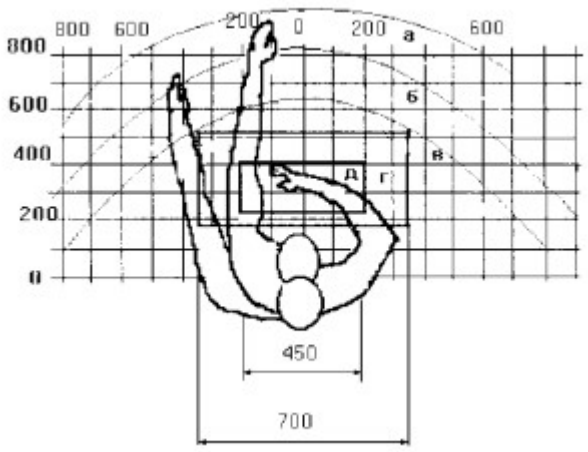

Рисунок 4.2 – Зони досяжності рук у горизонтальній площині

а - зона максимальної досяжності; б - зона досяжності пальців при витягнутій руці; в - зона легкої досяжності долоні; г - оптимальний простір для грубої ручної роботи; д - оптимальний простір для тонкої ручної роботи.

Розглянемо оптимальне розміщення предметів праці й документації в зонах досяжності рук [24]:

– монітор розміщається в зоні а (у центрі);

– клавіатура - у зоні г/д;

– системний блок розміщається в зоні б (ліворуч);

– принтер перебуває в зоні а (праворуч);

– документація - у зоні легкої досяжності долоні - в (ліворуч) - література й документація, необхідна при роботі;

– висота стола повинна бути обрана з урахуванням можливості сидіти вільно, у зручній позі, при необхідності опираючись на підлокітники;

– нижня частина стола повинна бути сконструйована так, щоб користувач ПК міг зручно сидіти, не був змушений підтискати ноги;

– поверхня стола повинна мати властивості, що виключають появу відблисків у полі зору користувача;

– конструкція стола повинна передбачати наявність висувних ящиків (не менш 3 для зберігання документації, літератури, особистих речей).

Параметри робочого місця вибираються відповідно до антропометричних характеристик. При використанні цих даних у розрахунках варто виходити з максимальних антропометричних характеристик.

При роботі в положенні сидячи рекомендуються наступні параметри робочого простору: ширина - не менш 700мм; глибина - не менш 400мм; висота робочої поверхні стола над підлогою 700-750мм. Оптимальними розмірами стола є: висота 710мм; довжина стола 1300мм; ширина стола 650мм.

Під робочою поверхнею має бути передбачений простір для ніг: висота - не менш 600мм; ширина - не менш 500мм; глибина - не менш 400мм.

Важливим елементом робочого місця оператора ПК є крісло. При проектуванні крісла виходять із того, що при будь-якому робочому положенні оператора його поза повинна бути фізіологічно правильною, тобто положення частин тіла повинне бути оптимальним.

Для задоволення вимог фізіології, що випливають із аналізу положення тіла людини в положенні сидячи, конструкція робочого сидіння повинна задовольняти наступним основним вимогам:

– допускати можливість зміни положення тіла, тобто забезпечувати вільне переміщення корпусу й кінцівок тіла;

– допускати регулювання висоти залежно від росту працюючої людини (у межах від 400 до 550мм);

– радіус кривизни в горизонтальній площині 400мм;

– кут нахилу спинки повинен змінюватися в межах 90-110° до площини сидіння.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи було розроблено та впроваджено веб-застосунок для продажу кави, який поєднує сучасні технології, функціональність і вимоги до безпеки. Робота охопила повний цикл створення програмного продукту — від аналізу потреб цільової аудиторії та вивчення аналогів до проєктування, реалізації, тестування та оптимізації системи.

Аналіз існуючих онлайн-магазинів, таких як thetea.ua, coffeespace.com.ua та amazon.com, дозволив виділити ключові функціональні модулі, що користуються попитом серед покупців. На основі цього сформовано перелік вимог, зокрема підтримка фільтрації товарів, інтеграція платіжних систем, система рейтингу й відгуків, адаптивний інтерфейс та зручна навігація. Окрема увага приділена забезпеченню безпеки персональних і платіжних даних.

На етапі реалізації було обрано сучасний технологічний стек: фронтенд — React з використанням TailwindCSS, бекенд — REST API на базі FastAPI, база даних — PostgreSQL з ORM SQLAlchemy. Така архітектура забезпечила модульність, масштабованість та можливість подальшого розвитку системи. Додатково реалізовано авторизацію, реєстрацію, обліковий запис користувача з історією замовлень, а також адміністративну панель для управління товарами й користувачами.

База даних була спроєктована з урахуванням вимог до нормалізації та продуктивності: встановлено зв’язки між сутностями, налаштовано індексацію, використано параметризовані запити для запобігання SQL-ін’єкціям. Забезпечено резервне копіювання та протестовано процедури відновлення.

Інтерфейс сайту спроєктовано відповідно до принципів UX/UI дизайну. Реалізовано зручну адаптацію до мобільних пристроїв, що підвищує доступність для користувачів. Проведено тести на кросбраузерну сумісність і перевірено логіку взаємодії з REST API.

Проведене тестування охоплювало ручні та автоматизовані сценарії. Тестувалася функціональність додавання товарів у кошик, авторизації, перегляду каталогу, оформлення замовлення, керування користувачами й додавання нових категорій. Особливу увагу приділено негативним сценаріям — перевірці валідації форм, реакції системи на помилки, дублювання записів. Застосовувалися інструменти навантажувального тестування, що підтвердили стабільність роботи під високим навантаженням та ефективне використання ресурсів.

Оцінка масштабованості показала, що система справляється з великим обсягом даних і кількістю користувачів, зберігаючи швидкодію. Інтеграція кешування (Redis) зменшила кількість запитів до БД і покращила час відповіді.

У межах роботи враховано вимоги до безпеки життєдіяльності розробників: ергономіка робочого місця, регламентація навантаження, техніка безпеки, відповідно до чинного законодавства України. Приділено увагу захисту інформації та зменшенню ризиків кіберзагроз шляхом впровадження протоколів шифрування та багаторівневої автентифікації.

У підсумку, створений веб-застосунок для продажу кави є готовим рішенням для комерційного використання. Він відповідає сучасним технічним вимогам, забезпечує високий рівень зручності для користувачів і дозволяє ефективно автоматизувати процес онлайн-торгівлі кавою. Отримані результати підтверджують доцільність розробки таких систем для малого й середнього бізнесу, їхню здатність покращити якість обслуговування клієнтів і підвищити конкурентоспроможність підприємства.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ASTQB. Glossary of Software Testing Terms [Електронний ресурс]. – Режим доступу: https://astqb.org/resources/glossary-of-software-testing-terms/

2. Capgemini. World Quality Report 2023. – Режим доступу: https://prod.ucwe.capgemini.com/at-de/wp-content/uploads/sites/11/2023/11/WQR_2023_FINAL_WEB_CG.pdf

3. DevOps Digest. State of Test Automation 2023 [Електронний ресурс]. – Режим доступу: https://www.devopsdigest.com/state-of-test-automation-2023

4. DevOps Digest. DevOps Digest Blog [Електронний ресурс]. – Режим доступу: https://www.devopsdigest.com/

5. HackerRank. The State of Quality Report 2023 [Електронний ресурс]. – Режим доступу: https://katalon.com/reports/state-quality-2023

6. Katalon. The State of Quality Report 2022 [Електронний ресурс]. – Режим доступу: https://katalon.com/resources-center/blog/the-state-of-quality-report-2022?ref=dogq.io

7. Kent C. Dodds. Testing Frontend Applications with React Testing Library [Електронний ресурс]. – Режим доступу: https://kentcdodds.com/blog/testing-frontend-applications

8. LogRocket. Why Redux Toolkit is the Standard for State Management in React [Електронний ресурс]. – Режим доступу: https://blog.logrocket.com/redux-toolkit-vs-redux-core

9. MDN Web Docs. Modern Frontend Development Practices [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Understanding_client-side_tools

10. Mozilla Developer Network. JavaScript Testing Tools Overview [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/Introduction

11. OWASP Foundation. Frontend Security Cheat Sheet [Електронний ресурс]. –

Режим доступу: https://owasp.org/www-project-cheat-sheets

12. React Documentation. React — A JavaScript library for building user interfaces [Електронний ресурс]. – Режим доступу: https://react.dev

13. Smashing Magazine. State of Frontend Development in 2024 [Електронний ресурс]. – Режим доступу: https://www.smashingmagazine.com/state-of-javascript-2024

14. Tailwind Labs. Official Tailwind CSS Documentation [Електронний ресурс]. – Режим доступу: https://tailwindcss.com

15. Jest Documentation. Testing React Applications with Jest [Електронний ресурс]. – Режим доступу: https://jestjs.io

16. Cypress.io. End-to-End Testing for Modern Web Applications [Електронний ресурс]. – Режим доступу: https://www.cypress.io

17. W3C. HTML5 Specification and Standards [Електронний ресурс]. – Режим доступу: https://www.w3.org/html/

18. Google Developers. Core Web Vitals Guide [Електронний ресурс]. – Режим доступу: https://web.dev/core-web-vitals

19. Shopify Engineering. Building Scalable E-commerce UIs with React [Електронний ресурс]. – Режим доступу: https://engineering.shopify.com/blogs/engineering/scalable-react-ui-design

20. Nielsen Norman Group. E-commerce Usability: Best Practices for Online Stores [Електронний ресурс]. – Режим доступу: https://www.nngroup.com/articles/ecommerce-usability

21. Stripe. Best Practices for Secure Payment Integration [Електронний ресурс]. – Режим доступу: https://stripe.com/docs/security

22. Statista. Global E-commerce Market Forecast and Trends [Електронний ресурс]. – Режим доступу: https://www.statista.com/statistics/123456/e-commerce-market-size-worldwide

23. Толок А.О., Крюковська О.А. Безпека життєдіяльності: Навч. посібник. 2011. 215 с.

24. Основи охорони праці: Підручник.; 3-те видання, доповнене та перероблене / За ред. К. Н Ткачука. К.: Основа, 2011. 480 с.

ДОДАТКИ

Додаток А – Лістинг коду моделі

Приклад контроллера (`CategoriesController`)

```
using Api.Dtos.Categories;
using Application.Commands.Categories.Commands;
using Application.Common.Interfaces.Queries;
using AutoMapper;
using Domain.Categories.Models;
using MediatR;
using Microsoft.AspNetCore.Mvc;

namespace Api.Controllers;

[Route("categories")]
[ApiController]
public class CategoriesController(ISender sender, ICategoryQueries
CategoryQueries, IMapper mapper) : ControllerBase
{
    [HttpGet("get-all")]
    public async Task<ActionResult<IReadOnlyList<CategoryDto>>>
GetAll(CancellationToken cancellationToken)
    {
        var categories = await CategoryQueries.GetAll(cancellationToken);
        return categories.Select(mapper.Map<CategoryDto>).ToList();
    }

    [HttpGet("get-by-id/{categoryId:guid}")]
    public async Task<ActionResult<CategoryDto>> GetById([FromRoute] Guid
categoryId, CancellationToken cancellationToken)
    {
        var Category = await CategoryQueries.GetById(categoryId,
cancellationToken);
        return Category.Match<ActionResult<CategoryDto>>(
            p => mapper.Map<CategoryDto>(p),
            () => NotFound());
    }

    [HttpPost("add")]
    public async Task<ActionResult<CategoryDto>> Add([FromBody] CreateCategoryDto
createCategoryDto, CancellationToken cancellationToken)
```

```
    {

        createCategoryDto.Id = Guid.NewGuid();
        var command = new AddCategoryCommand { Category =
mapper.Map<CreateCategoryModel>(createCategoryDto) };
        var result = await sender.Send(command, cancellationToken);

        return result.Match<ActionResult<CategoryDto>>(
            p => CreatedAtAction(nameof(GetById), new { CategoryId = p.Id },
mapper.Map<CategoryDto>(p)),
            e => Problem(e.Message));
    }

    [HttpPut("update")]
    public async Task<ActionResult<CategoryDto>> Update([FromBody]
UpdateCategoryDto updateCategoryDto, CancellationToken cancellationToken)
    {
        var command = new UpdateCategoryCommand { Category =
mapper.Map<UpdateCategoryModel>(updateCategoryDto) };
        var result = await sender.Send(command, cancellationToken);

        return result.Match<ActionResult<CategoryDto>>(
            p => Ok(mapper.Map<CategoryDto>(p)),
            e => Problem(e.Message));
    }

    [HttpDelete("delete/{categoryId:guid}")]
    public async Task<ActionResult> Delete([FromRoute] Guid categoryId,
CancellationToken cancellationToken)
    {
        var command = new DeleteCategoryCommand { CategoryId = categoryId };
        var result = await sender.Send(command, cancellationToken);

        return result.Match<ActionResult>(
            _ => NoContent(),
            e => Problem(e.Message));
    }

    [HttpPut("upload-images/{categoryId:guid}")]
    public async Task<ActionResult<CategoryDto>> Upload([FromRoute] Guid
categoryId, IFormFile imagesFile,
        CancellationToken cancellationToken)
    {
```

```
        var input = new UploadCategoryImagesCommand()
        {
            CategoryId = categoryId,
            ImagesFile = imagesFile
        };

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

        return result.Match<ActionResult<CategoryDto>>(
            p => Ok(mapper.Map<CategoryDto>(p)),
            e => Problem(e.Message));
    }


    [HttpPut("delete-image/{categoryId:guid}")]
    public async Task<ActionResult<CategoryDto>> DeleteImage(
        [FromRoute] Guid categoryId,
        [FromQuery] string photoName,
        CancellationToken cancellationToken)
    {
        var input = new DeleteCategoryImageCommand
        {
            CategoryId = categoryId,
            PhotoName = photoName
        };

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

        return result.Match<ActionResult<CategoryDto>>(
            p => Ok(mapper.Map<CategoryDto>(p)),
            e => Problem(e.Message));
    }
}
```

Приклад сторінки (HomePage)

```
import React, { useEffect } from 'react';
import { Container, Row, Col, Card, Button } from 'react-bootstrap';
import useActions from '../../hooks/useActions';
import { useSelector } from 'react-redux';
import { Link } from 'react-router-dom';
import REMOTE_HOST_NAME from "../../env/index";

import photoNotFound from '../../assets/images/photoNotFound.jpg';

const API_URL = REMOTE_HOST_NAME + 'images/categoryImages/';

const HomePage = () => {
 const { getCategories } = useActions();
 const { categoryList } = useSelector(state => state.category);

 useEffect(() => {
  getCategories();
 }, []);
 // console.log("HomePage", categoryList);
 const mainCategories = categoryList?.filter(cat => !cat.parentId) || [];

 return (
  <div className="overflow-hidden">
   <section
    className="vh-100 d-flex align-items-center position-relative"
    style={{
     background: `linear-gradient(rgba(0, 0, 0, 0.5), rgba(0, 0, 0, 0.5)),
url('https://images.unsplash.com/photo-1497935586351-b67a49e012bf?ixlib=rb-
1.2.1&auto=format&fit=crop&w=1920&q=80')`,
     backgroundSize: 'cover',
     backgroundPosition: 'center'
    }}
   >
    <Container className="text-center text-white">
     <h1 className="display-1 fw-bold mb-4">Свіжообсмажена кава</h1>
     <p className="fs-3 mb-5">Преміум сорти з найкращих куточків світу</p>
     <Link to="/product">
```

```
          <Button
            variant="outline-light"
            size="lg"
            className="rounded-pill px-5 py-3 fs-4"
          >
            Обрати каву
          </Button>
        </Link>
      </Container>
    </section>
    {/* Categories Section */}
    <section className="py-5 bg-light">
      <Container className="py-5">
        <Row className="g-5">
          {mainCategories.map(category => (
            <React.Fragment key={category.id}>
              <Col lg={4} md={12}>
                <div className="bg-white p-4 h-100 shadow-sm rounded d-flex flex-
column">
                  {/* Фото головної категорії */}
                  <div className="mb-4">
                    <img
                      src={category.photo ? API_URL + category.photo : photoNotFound}
                      alt={category.name}
                      className="img-fluid rounded w-100 object-fit-cover"
                      style={{ height: '250px', objectFit: 'cover' }}
                    />
                  </div>
                  {/* Назва + кнопка */}
                  <div className="flex-grow-1 d-flex flex-column justify-content-
between">
                    <div>
                      <h2 className="fw-bold display-5">{category.name}</h2>
                      <p className="text-muted fs-5">
                        Найзарядженіша колекція {category.name.toLowerCase()}
                      </p>
                    </div>
                    <Link to={`/product/${category.id}`}>
                      <Button variant="warning" className="fw-bold px-4 py-2 mt-3 w-
100">
```

```
                    ➜ Переглянути {category.name.toLowerCase()}
                  </Button>
                </Link>
              </div>
            </div>
          </Col>
          <Col lg={8} md={12}>
            <Row className="g-4">
              {category.subCategories?.map(sub => (
                <Col key={sub.id} lg={4} md={6}>
                  <Link
                    to={`/product/${sub.id}`}
                    className="text-decoration-none text-dark"
                  >
                    <Card className="h-100 border-0 shadow-sm">
                      <div style={{ height: '160px', overflow: 'hidden' }}>
                        <img
                          src={sub.photo ? API_URL + sub.photo : photoNotFound}
                          alt={sub.name}
                          className="w-100 h-100 object-fit-cover"
                        />
                      </div>
                      <Card.Body className="text-center">
                        <Card.Title className="fw-bold fs-5 mb-
0">{sub.name}</Card.Title>
                      </Card.Body>
                    </Card>
                  </Link>
                </Col>
              ))}
            </Row>
          </Col>
        </React.Fragment>
      ))}
    </Row>
  </Container>
</section>
{/* Features Section */}
<section className="py-5 bg-white">
  <Container className="py-5">
```

```
        <Row className="g-5 text-center">
          <Col md={4}>
            <div
              className="d-inline-block p-4 bg-primary rounded-circle mb-4"
              style={{ width: '100px', height: '100px' }}
            >
              <i className="bi bi-globe fs-1 text-white"></i>
            </div>
            <h3 className="h2 fw-bold mb-3">Найкращі сорти</h3>
            <p className="text-muted fs-5">З плантацій Бразилії, Колумбії та Ефіопії</p>
          </Col>
          <Col md={4}>
            <div
              className="d-inline-block p-4 bg-primary rounded-circle mb-4"
              style={{ width: '100px', height: '100px' }}
            >
              <i className="bi bi-rocket fs-1 text-white"></i>
            </div>
            <h3 className="h2 fw-bold mb-3">Швидка доставка</h3>
            <p className="text-muted fs-5">Доставка в день замовлення по Києву</p>
          </Col>
          <Col md={4}>
            <div
              className="d-inline-block p-4 bg-primary rounded-circle mb-4"
              style={{ width: '100px', height: '100px' }}
            >
              <i className="bi bi-gift fs-1 text-white"></i>
            </div>
            <h3 className="h2 fw-bold mb-3">Подарункова упаковка</h3>
            <p className="text-muted fs-5">Безкоштовна упаковка для ваших близких</p>
          </Col>
        </Row>
      </Container>
    </section>
  </div>
 );
};

export default HomePage;
```

Приклад репозиторію (`CategoryRepository`)

```
using System.Linq.Expressions;
using Application.Common.Interfaces.Queries;
using Application.Common.Interfaces.Repositories;
using AutoMapper;
using DataAccessLayer.Data;
using DataAccessLayer.Entities.Categories;
using Domain.Categories;
using Domain.Categories.Models;
using Domain.Products;
using Domain.Products.Models;
using Microsoft.EntityFrameworkCore;
using Optional;

namespace DataAccessLayer.Repositories;

public class CategoryRepository(ApplicationDbContext context, IMapper mapper) :
ICategoryRepository, ICategoryQueries
{
    public async Task<Category> Create(CreateCategoryModel model,
CancellationToken cancellationToken)
    {
        var categoryEntity = mapper.Map<CategoryEntity>(model);

        await context.Categories.AddAsync(categoryEntity, cancellationToken);
        await context.SaveChangesAsync(cancellationToken);

        return mapper.Map<Category>(categoryEntity);
    }

    public async Task<Category> Update(UpdateCategoryModel model,
CancellationToken cancellationToken)
    {
        // Retrieve entity WITHOUT AsNoTracking so it is tracked by context
        var categoryEntity = await context.Categories.FirstOrDefaultAsync(x =>
x.Id == model.Id, cancellationToken);

        if (categoryEntity == null)
        {
```

```
            throw new InvalidOperationException("Category not found.");
        }

        mapper.Map(model, categoryEntity);

        var entry = context.Entry(categoryEntity);
        if (entry.Property(e => e.Photo).CurrentValue != entry.Property(e =>
e.Photo).OriginalValue)
        {
            entry.Property(e => e.Photo).IsModified = true;
        }

        await context.SaveChangesAsync(cancellationToken);

        return mapper.Map<Category>(categoryEntity);
    }


    public async Task<Category> Delete(DeleteCategoryModel model,
CancellationToken cancellationToken)
    {
        var categoryEntity = await GetCategoryAsync(x => x.Id == model.Id,
cancellationToken);

        if (categoryEntity == null)
        {
            throw new InvalidOperationException("Category not found.");
        }

        context.Categories.Remove(categoryEntity);
        await context.SaveChangesAsync(cancellationToken);

        return mapper.Map<Category>(categoryEntity);
    }

    public async Task<IReadOnlyList<Category>> GetAll(CancellationToken
cancellationToken)
    {
        var categoryEntities = await context.Categories.Where(c => c.ParentId ==
null)
            .Include(c => c.SubCategories)
            .ThenInclude(c => c.SubCategories)
            .ThenInclude(c => c.SubCategories)
```

```
            .ThenInclude(c => c.SubCategories)
            .Include(c => c.Parent)
            .ToListAsync(cancellationToken);

        return mapper.Map<IReadOnlyList<Category>>(categoryEntities);
    }

    public async Task<Option<Category>> GetById(Guid id, CancellationToken
cancellationToken)
    {
        var categoryEntity = await GetCategoryAsync(x => x.Id == id,
cancellationToken);

        var category = mapper.Map<Category>(categoryEntity);

        return category == null ? Option.None<Category>() : Option.Some(category);
    }

    private async Task<CategoryEntity?>
GetCategoryAsync(Expression<Func<CategoryEntity, bool>> predicate,
        CancellationToken cancellationToken)
    {
        return await context.Categories
            .Include(c => c.SubCategories)
            .ThenInclude(c => c.SubCategories)
            .ThenInclude(c => c.SubCategories)
            .ThenInclude(c => c.SubCategories)
            .Include(c => c.Parent)
            .AsNoTracking()
            .FirstOrDefaultAsync(predicate, cancellationToken);
    }
}
```

Приклад команди (`AddCategoryCommand`)

```
using Application.Commands.Categories.Exceptions;
using Application.Common;
using Application.Common.Interfaces.Repositories;
using Domain.Categories;
using Domain.Categories.Models;
using MediatR;

namespace Application.Commands.Categories.Commands;

public record AddCategoryCommand : IRequest<Result<Category, CategoryException>>
{
    public required CreateCategoryModel Category { get; init; }
}

public class AddCategoryCommandHandler(
    ICategoryRepository CategoryRepository) : IRequestHandler<AddCategoryCommand,
Result<Category, CategoryException>>
{
    public async Task<Result<Category, CategoryException>>
Handle(AddCategoryCommand request, CancellationToken cancellationToken)
    {
        var Category = await CategoryRepository.Create(request.Category,
cancellationToken);
        return Category;
    }
}
```

HttpClient

```
import axios from "axios";
import {
 setIsLoading
} from "../../store/state/actions/appSettingActions";
import { store } from "../../store/store";

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);
    }
  );
 }
 setAuthorizationToken(token) {
  if (token) {
   this.axiosInstance.defaults.headers.common[
    "Authorization"
   ] = `Bearer ${token}`;
  } else {
   delete this.axiosInstance.defaults.headers.common["Authorization"];
  }
 }
 async get(url, config = {}) {
  return this.request({ method: "GET", url, ...config });
 }
 async post(url, data, config = {}) {
  return this.request({ method: "POST", url, data, ...config });
 }
 async put(url, data, config = {}) {
  return this.request({ method: "PUT", url, data, ...config });
 }
 async delete(url, config = {}) {
  return this.request({ method: "DELETE", url, ...config });
 }
 async request(config) {
  setIsLoading(true)(store.dispatch);
  try {
   const response = await this.axiosInstance.request(config);
   return response.data;
  } catch (error) {
   return Promise.reject(error);
  } finally {
   setIsLoading(false)(store.dispatch);
  }
 }
}
```

Додаток Б – Диск з роботою