

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

Маркетплейс "Rate"

Виконав студент групи ІПЗк-23-1
УЛЬКІН Владислав Вікторович

Керівник роботи:
ГОДЛЕВСЬКИЙ Юрій Олександрович

Рецензент:
МОРОЗОВ Андрій Васильович

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення
Тетяна Вакалюк
«26» лютого 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **УЛЬКІН Владислав Вікторович**

Керівник роботи: **ГОДЛЕВСЬКИЙ Юрій Олександрович**

Тема роботи: **Маркетплейс "Rate",**

затверджена Наказом закладу вищої освіти **№61/с** від **14.02.2025 р.**

Вихідні дані для роботи: Об'єктом дослідження є процеси та інформаційні технології побудови веборієнтованих інформаційних додатків. Предметом дослідження є моделі, методи, алгоритми та засоби побудови веборієнтованого інформаційного додатку з фокусом на актуальний інформаційний контент і зручний інтерфейс користувача.

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

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Годлевський Ю.О.	27.02.2025р.	27.02.2025р.
2	Годлевський Ю.О.	27.02.2025р.	27.02.2025р.
3	Годлевський Ю.О.	27.02.2025р.	27.02.2025р.

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається зі вступу, трьох розділів, висновків, джерел інформації, додатків та програмної розробки. Пояснювальна записка до випускної роботи містить 72 сторінок, 27 ілюстрації, 9 таблиць.

Метою роботи є розробка сайту веборієнтованого додатку з використанням Next.js, React, Antd, MUI, Node.js, Express, а також розробка системи та використання хмарної бази даних MongoDB.

В роботі були встановлені вимоги користувача і обґрунтовано створення концептуальної моделі. Було проведено аналіз аналогів програмних продуктів і вибрано засоби для реалізації.

Розроблена структура бази даних, обрана архітектура та розроблена структура системи. Була описана програмна реалізація вебсайту і наведені результати тестування системи.

КЛЮЧОВІ СЛОВА: NEXTJS, АДМІНІСТРУВАННЯ, БАЗА ДАНИХ, ВЕБСАЙТ, МАРКЕТПЛЕЙС, ТОВАРИ.

					ІПЗ.КР.Б – 121 – 25 – ПЗ				
Змн.	Арк.	№ докум.	Підпис	Дата					
Розроб.		Улькін В.В.			Маркетплейс «Рейт» Пояснювальна записка	Літ.	Арк.	Аркушів	
Керівник		Годлевський Ю.О.					3	70	
Рецензент		Морозов А.В.				Житомирська політехніка, група ІПЗк-23-1			
Н. контр.		Тичина О.П.							
Зав. каф.		Вакалюк Т.А.							

ABSTRACT

The Bachelor's qualification thesis consists of an introduction, three chapters, conclusions, references, and appendices. The explanatory report for the final work contains 72 pages, 27 illustrations and 9 tables.

The aim of the project is to develop a website for a web-oriented application using Next.js, React, Antd, MUI, Node.js, and Express, as well as to develop a system utilizing the cloud Based Database MongoDB.

The work includes identifying user requirements and substantiating the creation of a conceptual model. An analysis of similar software products was carried out, and tools for implementation were selected. The database structure was developed, the architecture was chosen, and the system structure was designed. The software implementation of the website was described, and the results of system testing were provided.

KEYWORDS: NEXTJS, ADMINISTRATION, DATABASE, WEBSITE, MARKETPLACE, PRODUCTS.

		Улькін В.В.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Годлевський Ю.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		4

ЗМІСТ

РЕФЕРАТ	1
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	6
ВСТУП.....	7
1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ ПЛАГУВАННЯ ТА ОБЛІКУ РОБОСОГО ЧАСУ НА ПІДПРИЄМСТВІ.....	8
1.1. Постановка задачі.....	8
1.2. Аналіз аналогів програмного продукту	10
1.3. Вибір архітектури веб-додатку маркетплейсу	13
1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення	15
Висновки до першого розділу.....	18
2. ПРОЕКТУВАННЯ МАРКЕТПЛЕЙСУ «Rate».....	19
2.1. Визначення варіантів використання та об'єктно-орієнтованої структури системи.....	19
2.2. Розробка системи бази даних	27
2.3. Проектування та реалізація алгоритмів роботи системи	31
2.4. Реалізація функціоналу інформаційного додатка	34
Висновки до другого розділу	36
3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З МАРКЕТПЛЕЙСУ «Rate».....	37
3.1. Порядок встановлення та налаштування параметрів системи.....	37
3.2. Структура інтерфейсу та порядок взаємодії з онлайн-сервісом.....	39
3.3. Тестування роботи програмного продукту	46
Висновки до третього розділу	48
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	50
ДОДАТКИ.....	53

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

REST API - Веб-запити.

JWT, OAuth - Токени, авторизація.

DOM - Структура сторінки.

SPA - Односторінковий додаток.

MPA - Багатосторінковий додаток.

БД – База даних.

UI - Інтерфейс користувача.

Маркетплейс – платформа купівлі/продажі товарів.

		Улькін В.В.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Годлевський Ю.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		6

ВСТУП

Актуальність теми полягає у тому, що у сучасному цифровому середовищі веборієнтовані маркетплейси відіграють ключову роль у забезпеченні ефективної взаємодії між продавцями та покупцями. Активний розвиток електронної комерції, зміна споживчих уподобань, зростання популярності онлайн-шопінгу та необхідність централізованого пошуку, порівняння товарів і послуг визначають актуальність створення інноваційних, адаптивних та ефективних платформ.

У цьому контексті вебзастосунок "Rate" виступає як сучасний маркетплейс, що дозволяє користувачам оцінювати, переглядати та порівнювати товари й послуги на основі реальних відгуків, рейтингу та функціоналу.

Особливу значущість проєкт набуває в умовах економічної нестабільності та війни, коли забезпечення доступу до перевірених товарів і послуг, а також підтримка малого бізнесу стає критично важливим. Платформа "Rate" може служити не лише для купівлі, але й для побудови довготривалих партнерських відносин.

Об'єктом дослідження є процеси та інформаційні технології побудови веборієнтованих інформаційних додатків.

Предметом дослідження є моделі, методи, алгоритми та засоби розробки вебзастосунків з фокусом на якісний контент та зручний інтерфейс користувача.

Метою роботи є розробка веборієнтованого маркетплейсу "Rate", який дозволяє користувачам взаємодіяти з товарами та послугами, залишати оцінки, переглядати рейтинги, керувати обліковими записами, а адміністраторам — здійснювати модерацію, редагування і управління контентом платформи.

Завдання дослідження:

1. Дослідити популярні маркетплейси, їх функціональності та досвіду користувачів.
2. Здійснити вибір технологій, оцінити та обґрунтувати використання інструментів розробки

		Улькін В.В.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Годлевський Ю.О.				7
Змн.	Арк.	№ докум.	Підпис	Дата		

1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ ПЛАГУВАННЯ ТА ОБЛІКУ РОБОСОГО ЧАСУ НА ПІДПРИЄМСТВІ

1.1. Постановка задачі

Метою кваліфікаційної роботи бакалавра є створення інноваційного вебсервісу — сучасного маркетплейсу під назвою “Rate”, який має стати ефективним інструментом для взаємодії між продавцями та покупцями у цифровому середовищі. Цей майданчик має забезпечити зручний інтерфейс, надійність процесів, високу швидкість обміну інформацією та підтримку масштабованості, що дозволить у подальшому розширювати функціональність і інтегрувати систему з іншими цифровими сервісами. Розробка “Rate” передбачає створення зручної екосистеми, в якій користувачі зможуть не тільки купувати й продавати товари, а й отримувати актуальну інформацію, обмінюватися досвідом через відгуки та брати участь у формуванні прозорого рейтингу продавців.

У цифрову епоху маркетплейси стали ключовим елементом онлайн-торгівлі, трансформували традиційні форми комерції та об’єднавши величезну кількість користувачів у межах однієї платформи. Такі гіганти, як Amazon, Etsy, OLX, Prom та Rozetka, демонструють, як завдяки продуманому функціоналу можна одночасно задовольнити потреби і продавців, і покупців, створивши зручну цифрову інфраструктуру для продажу, доставки та взаємодії. Вони пропонують користувачам простий спосіб розміщення товарів, швидкий доступ до кращих пропозицій, можливість залишати коментарі й будувати репутацію в цифровому просторі. На цьому фоні платформа “Rate” також прагне реалізувати ці ключові принципи, пропонуючи користувачам інтерфейс із високим рівнем довіри, прозорими механізмами оцінювання та зворотним зв’язком.

Особливу увагу в реалізації проєкту буде приділено досвіду таких платформ, як Etsy, де малі підприємці, митці та хендмейд-виробники отримали змогу продавати власну продукцію по всьому світу завдяки зручним інструментам створення онлайн-вітрини.

		Улькін В.В.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Годлевський Ю.О.				8
Змн.	Арк.	№ докум.	Підпис	Дата		

Ідея відкритого оцінювання, прозорості угод та довіри між учасниками системи закладена й у концепцію платформи “Rate”. Назва сервісу недаремно апелює до поняття оцінки — адже саме механізм рейтингу відіграє ключову роль у формуванні довіри між незнайомими користувачами.

Платформа складатиметься з декількох логічних модулів, кожен з яких відповідає за окремі аспекти функціонування системи. Один із головних модулів буде присвячений створенню товарів і послуг: продавці матимуть можливість створювати власні вітрини з детальним описом товарів, завантаженням зображень, встановленням цін і розподілом товарів за категоріями. Інший важливий модуль — система оформлення замовлень і здійснення оплат, яка дозволить покупцям зручно і безпечно замовляти товари, обирати типи доставки та оплачувати покупки в онлайн-режимі. Не менш значущим буде модуль зворотного зв’язку, де користувачі зможуть залишати свої відгуки, оцінки та коментарі після здійснення покупки, що сприятиме зростанню рівня довіри до продавців і формуванню якісного ринку. Також буде реалізовано адміністративну панель, яка дозволить модераторам керувати контентом, відстежувати активність, проводити аналіз даних та організовувати категорії товарів.

На виході планується створення повноцінного вебсайту, де кожен користувач матиме змогу зареєструватися в ролі продавця або покупця, додавати товари, переглядати пропозиції інших користувачів, оформлювати замовлення, оплачувати їх та залишати відгуки. У межах підвищення інноваційності платформи також розглядається можливість часткової інтеграції з технологіями Web 3.0 — зокрема, додавання функціональності криптовалютних розрахунків, а також впровадження смарт-контрактів, які дозволяють укладати угоди з підвищеним рівнем автоматизації, прозорості та юридичної захищеності.

Підсумком цієї кваліфікаційної роботи стане розроблений та протестований прототип маркетплейсу “Rate” — зручної, сучасної та масштабованої платформи для онлайн-торгівлі, яка відповідатиме сучасним вимогам електронної комерції.

		Улькін В.В.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Годлевський Ю.О.				9
Змн.	Арк.	№ докум.	Підпис	Дата		

1.2. Аналіз аналогів програмного продукту

Сьогодні існує багато популярних платформ для продажу товарів в інтернеті. Серед них можна виділити такі міжнародні сервіси, як Amazon, eBay та Etsy. Усі ці платформи надають продавцям можливість завантажувати свої товари, встановлювати ціни та умови доставки, а також спілкуватися з покупцями.

Наприклад, платформа Etsy відома своїм зручним і привабливим дизайном, що підкреслює креативність та унікальність товарів, а також зрозумілим інтерфейсом, який забезпечує комфортне користування. Функції сервісу включають можливість перегляду та покупки товарів, керування кошиком і оформленням замовлення, а також зв'язок із продавцями та службою підтримки для вирішення питань.

Мобільні додатки цих платформ забезпечують легкий доступ з будь-якого пристрою, дозволяючи користувачам здійснювати покупки та керувати своїми замовленнями у будь-який час і в будь-якому місці. Інтеграція із соціальними мережами спрощує комунікацію, підвищує популярність сервісів та дозволяє користувачам ділитися своїми покупками та враженнями з друзями.

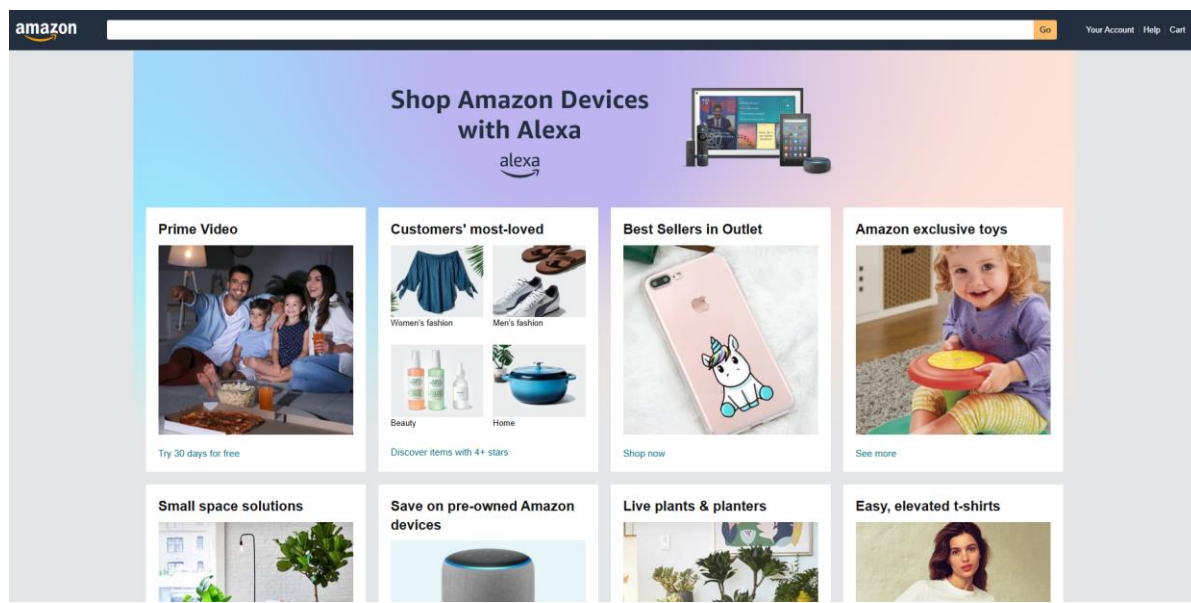


Рисунок 1.1 Вигляд головної сторінки веб-додатку Amazon

Додаток Amazon має простий та інтуїтивно зрозумілий інтерфейс, який забезпечує зручність використання та швидкий доступ до асортименту товарів. Функціонал включає

		Улькін В.В.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Годлевський Ю.О.				10
Змн.	Арк.	№ докум.	Підпис	Дата		

профілі продавців, рейтинги та відгуки покупців, що допомагає користувачам приймати обґрунтовані рішення щодо вибору товарів, а також контактну інформацію для зв'язку з продавцями.

Мобільна версія додатка ідеально сумісна з різними гаджетами і забезпечує легкий доступ з будь-якого пристрою. Інтеграція з соціальними мережами спрощує реєстрацію та підвищує залученість користувачів, дозволяючи їм ділитися своїми покупками. Додаток використовує сучасні технології безпеки для захисту особистих даних і фінансових транзакцій.

Крім того, завдяки адаптивному дизайну та оптимізованій продуктивності, мобільна версія працює швидко навіть за нестабільного інтернет-з'єднання, що робить її зручною у використанні в будь-яких умовах. Постійне оновлення й підтримка дозволяють швидко реагувати на потреби користувачів, покращувати функціональність та впроваджувати нові можливості відповідно до сучасних тенденцій у сфері електронної комерції.

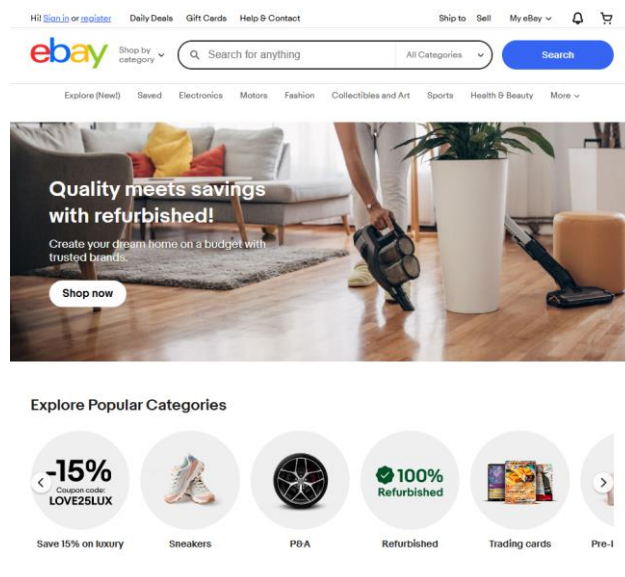


Рисунок 1.2 Вигляд головної сторінки веб-додатку Ebay

Платформа eBay являє собою один із провідних прикладів цифрового торгового майданчика, що забезпечує можливість купівлі та продажу широкого спектру товарів — від антикваріату та колекційних предметів до сучасних електронних пристроїв і

		Улькін В.В.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Годлевський Ю.О.				11
Змн.	Арк.	№ докум.	Підпис	Дата		

елементів одягу. Високий рівень різноманіття пропозицій робить платформу привабливою для різних категорій користувачів.

Крім того, платформа підтримує локальне збереження даних, що дозволяє користувачам зберігати обрані товари у кошику або списках бажань без необхідності постійного входу в акаунт, що значно підвищує зручність та доступність сервісу. Такий підхід робить користувацький досвід більш безперервним і комфортним, особливо для нових або випадкових відвідувачів.

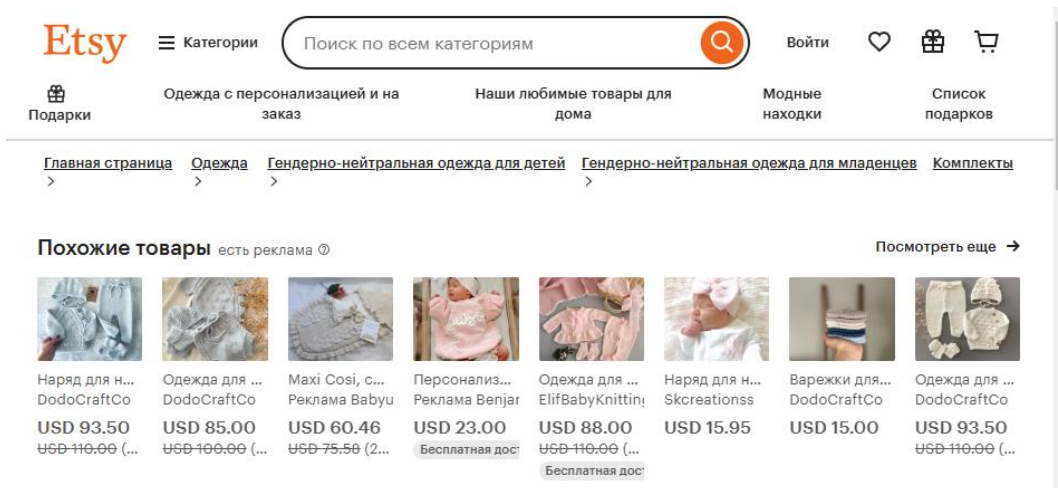


Рисунок 1.3 Вигляд головної сторінки веб-додатку Etsy

Etsy — це одна з провідних міжнародних платформ для продажу унікальних та крафтових товарів. Вона пропонує продавцям інтуїтивно зрозумілий інтерфейс для розміщення своїх товарів, встановлення цін та управління замовленнями. Користувачі можуть переглядати широкий асортимент, включно з прикрасами, одягом, товарами для дому та мистецтвом.

Мобільний додаток Etsy забезпечує легкий доступ до покупок з будь-якого пристрою, дозволяючи швидко знаходити і купувати товари. Функціонал платформи включає можливість перегляду профілів продавців, читання відгуків і оцінок, а також зв'язок із продавцями для уточнення деталей. Інтеграція з соціальними мережами спрощує обмін улюбленими товарами та підвищує довіру користувачів.

Аналізуючи ці платформи, можна виділити такі ключові функції як, купівля та продаж товарів (постачальники завантажують свої продукти, встановлюють ціни та умови продажу, а покупці можуть їх купувати), система рекомендацій та відгуків (деякі

		Улькін В.В.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Годлевський Ю.О.				12
Змн.	Арк.	№ докум.	Підпис	Дата		

сервіси дозволяють формувати персоналізовані пропозиції для покупців), інтеграція з соціальними мережами (спрощує просування товарів та залучення нових клієнтів), вбудовані маркетингові інструменти (реклама, аналітика, акції та розсилки).

Попри великий функціонал, ці сервіси мають деякі недоліки, такі як, високі комісії на продаж, обмежені можливості для кастомізації особистої сторінки продавця, відсутність інтеграції з певними платіжними системами, закритість платформи (не всі функції доступні у безкоштовній версії).

Отже, за проведеним аналізом можна визначити, що додаток «Маркетплейс «Рейт» видається найбільш привабливим завдяки своєму сучасному дизайну, широкому спектру послуг та інтеграції з соціальними мережами, що робить його зручним для продавців і покупців.

1.3. Вибір архітектури веб-додатку маркетплейсу

Під час розробки веб-орієнтованої системи для планування робочого часу надзвичайно важливо обрати таку архітектуру, яка не лише відповідатиме поточним потребам, але й дозволить проєкту безпечно й ефективно масштабуватися в майбутньому. Саме тому було прийнято рішення на користь класичної клієнт-серверної моделі, яка дозволяє розмежувати відповідальність між різними компонентами системи, а також забезпечити гнучкість у процесі розробки, підтримки та розгортання. Такий підхід дозволяє ефективно організувати розробку, підвищити безпеку обміну даними та забезпечити стабільну роботу при зростанні навантаження на систему.

У межах цієї архітектури користувацький інтерфейс або клієнтська частина реалізована за допомогою сучасних вебтехнологій. Основним фреймворком для фронтенду обрано React — потужний інструмент для створення динамічних і масштабованих інтерфейсів. У зв'язці з ним використовуються такі бібліотеки, як Redux та RTK Query, які дозволяють ефективно керувати станом застосунку та здійснювати обмін даними з сервером. Для побудови адаптивного і візуально привабливого інтерфейсу було застосовано бібліотеку компонентів Ant Design, яка пропонує велику

		Улькін В.В.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Годлевський Ю.О.				13
Змн.	Арк.	№ докум.	Підпис	Дата		

кількість готових рішень для реалізації форм, таблиць, повідомлень та інших елементів взаємодії з користувачем [2].

Серверна частина, яка виконує роль бекенду, реалізована на платформі Node.js, що дозволяє створювати високопродуктивні сервери з низьким часом відгуку. Як основний фреймворк використовується Express — гнучке і мінімалістичне рішення, яке дозволяє швидко створювати маршрути, обробляти запити та реалізовувати бізнес-логіку. Для керування сесіями користувачів, автентифікації та авторизації впроваджено NextAuth — сучасну бібліотеку, що підтримує роботу з токенами, соціальними логінами та різними стратегіями ідентифікації. Сервер відповідає за перевірку прав доступу, обробку запитів до бази даних, виконання бізнес-операцій, збереження звітів та статистики [6].

Для зберігання інформації про користувачів, їхні графіки роботи, запити на зміну робочого часу, звіти й інші дані була обрана база даних MongoDB. Це документно-орієнтоване сховище, яке відмінно підходить для вебзастосунків із динамічною структурою даних. MongoDB дозволяє швидко зберігати, шукати та оновлювати документи, гнучко структурувати інформацію і масштабувати систему у випадку збільшення кількості записів [8].

Перевагою обраної архітектури є її висока масштабованість: усі основні частини системи — клієнт, сервер і база даних — можуть розгортатися на різних серверах або в контейнерах, що дозволяє окремо масштабувати їх відповідно до навантаження. Окрім того, така структура підтримує незалежний розвиток компонентів, що дає змогу фронтенд- і бекенд-командам працювати паралельно, не створюючи одне одному перешкод. Безпека системи також врахована — всі дії користувачів проходять через перевірку автентичності, використовуються сесії або JWT токени, що дозволяє захистити дані й забезпечити дотримання актуальних стандартів інформаційної безпеки [15].

Під час проєктування було розглянуто й інші архітектурні підходи, серед яких — монолітна архітектура та мікросервісна. Монолітна система є більш простою у впровадженні, однак має значні обмеження в масштабуванні та складність у супроводі при розростанні функціоналу. Мікросервісна ж архітектура забезпечує високу гнучкість і дозволяє кожен модуль обслуговувати окремо, але вимагає складної інфраструктури,

		Улькін В.В.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Годлевський Ю.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		14

додаткових ресурсів і складніших підходів до моніторингу, безпеки та налагодження взаємодії між сервісами.

Беручи до уваги технічні потреби та очікувані масштаби проєкту, класична клієнт-серверна архітектура виявилася найбільш доцільним вибором. Вона поєднує в собі переваги структурованості, гнучкості й надійності, що дозволяє створити ефективну, сучасну систему для планування робочого часу з перспективою подальшого вдосконалення й розширення функціональності..

1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Вебдодаток створено за клієнт-серверною архітектурною моделлю, що дозволяє чітко розмежувати функціональність між фронтендом і бекендом.

Бекенд реалізовано за допомогою Express.js, який виступає як обробник HTTP-запитів та відповідає за маршрутизацію, логіку взаємодії з базою даних та обробку помилок. Кожен API-ендпоінт структуровано у вигляді окремого маршруту, який делегує логіку відповідному контролеру, що у свою чергу викликає потрібні сервіси або моделі для взаємодії з базою даних. Завдяки цьому легко відслідковувати хід виконання запиту, логіку легко тестувати та змінювати [7].

Для забезпечення зручної типізації використовуються інтерфейси TypeScript, які описують структуру об'єктів, моделей та відповідей сервера. Це дозволяє уникнути помилок типів та підвищити надійність під час розробки. Для взаємодії з MongoDB застосовується ORM-бібліотека Mongoose, яка дозволяє створювати гнучкі схеми для колекцій, додавати валідацію, хуки, індексацію та інші необхідні функціональні можливості. Всі моделі винесені в окремі файли для зручності та централізованого доступу [13].

Обробка помилок реалізована за допомогою централізованого middleware, який перехоплює всі помилки, що виникають у процесі обробки запитів, і повертає відповідну структуровану відповідь. Це дозволяє забезпечити єдиний підхід до логування та дебагу. Аутентифікація та авторизація реалізовані через NextAuth, з підтримкою сесій, токенів

		Улькін В.В.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Годлевський Ю.О.				15
Змн.	Арк.	№ докум.	Підпис	Дата		

та ролей користувачів, що дозволяє будувати розмежування доступу до захищених маршрутів та ресурсів.

Зберігання та обробка даних реалізовані за допомогою MongoDB — документоорієнтованої NoSQL бази даних, яка ідеально підходить для зберігання гнучких та динамічних структур даних. Для зручної взаємодії з базою даних використовується бібліотека Mongoose, яка дозволяє створювати моделі, валідувати схеми та працювати з колекціями MongoDB у вигляді об'єктів [9].

У системі реалізовано автентифікацію та авторизацію користувачів за допомогою бібліотеки NextAuth, яка забезпечує інтеграцію з різними провайдерами, такими як Google, GitHub або кастомна база даних. Цей підхід дозволяє захистити особисті дані користувачів, обмежити доступ до приватних маршрутів та забезпечити належний рівень безпеки.

Фронтенд побудовано з використанням Next.js — сучасного фреймворку, який дозволяє використовувати переваги як клієнтського, так і серверного рендерингу. В основі лежить React, що забезпечує компонентний підхід до побудови інтерфейсу. Кожен екран, сторінка, форма чи віджет реалізований як окремий реюзабельний компонент, що сприяє чистоті коду та полегшує тестування [1].

Для управління станом використовується Redux Toolkit, який дозволяє ефективно зберігати, модифікувати та передавати дані між компонентами. Завдяки RTK Query реалізовано зручну взаємодію з серверним API — підключення, кешування, рефетчинг та обробка помилок [3]. Усі запити винесено в окремі сервіси, що дозволяє централізовано керувати логікою доступу до даних.

В якості UI-бібліотек використовуються Ant Design та Material UI, що дає змогу реалізувати адаптивний і сучасний інтерфейс з багатим вибором готових компонентів: таблиць, модальних вікон, селекторів, вкладок, форм, кнопок тощо. Темізація інтерфейсу дозволяє легко адаптувати зовнішній вигляд системи до вимог бренду або користувача. Компоненти стилізовані відповідно до корпоративного дизайну, з урахуванням UX/UI принципів [4].

		Улькін В.В.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Годлевський Ю.О.				16
Змн.	Арк.	№ докум.	Підпис	Дата		

Фронтенд-додаток реалізовано на основі Next.js — потужного фреймворку, побудованого поверх React, який підтримує серверний рендеринг (SSR), статичну генерацію (SSG) та гібридні режими рендерингу. Завдяки використанню SSR та SSG забезпечується швидке завантаження сторінок, покращення індексації пошуковими системами та загальна оптимізація продуктивності.

Для управління глобальним станом додатку використовується Redux Toolkit, зокрема його сучасна надбудова RTK Query, яка надає зручний механізм взаємодії з API, кешування відповідей, автоматичне оновлення даних та обробку помилок.

Візуальна частина додатку розроблена із застосуванням двох потужних UI-бібліотек — Ant Design та Material UI (MUI). Вони забезпечують велику кількість готових адаптивних компонентів, таких як таблиці, модальні вікна, повідомлення, форми та інші інтерактивні елементи інтерфейсу [14].

Завдяки ним створено інтуїтивно зрозумілий, адаптивний та візуально привабливий користувацький інтерфейс. Обидві бібліотеки підтримують кастомізацію тем, що дозволяє легко налаштовувати зовнішній вигляд застосунку відповідно до побажань замовника або корпоративного стилю.

Крім того, у процесі розробки було враховано масштабованість проєкту, що забезпечує легку інтеграцію нових модулів, компонентів та сервісів у майбутньому. Інфраструктура проєкту побудована таким чином, щоб зменшити залежність між модулями, підвищити повторне використання коду та забезпечити високий рівень підтримуваності. Завдяки використанню TypeScript як на клієнтській, так і на серверній частині, вдалося досягти єдиного підходу до типізації, що значно спрощує процес розробки та тестування.

У результаті, обраний технологічний стек (Next.js, React, Redux Toolkit, RTK Query, Ant Design, Material UI, Express, MongoDB, Mongoose, TypeScript, NextAuth) дозволив реалізувати ефективний, безпечний, масштабований та сучасний вебдодаток з продуманим інтерфейсом та стабільною роботою як на стороні клієнта, так і на стороні сервера [20].

		Улькін В.В.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Годлевський Ю.О.				17
Змн.	Арк.	№ докум.	Підпис	Дата		

Висновки до першого розділу

Проведено аналіз предметної області, в результаті якого були визначені основні вимоги до платформи для планування робочого часу. Основною задачею системи є створення зручного інтерфейсу для ефективного планування та контролю за робочим часом, з урахуванням індивідуальних потреб користувачів.

У процесі дослідження існуючих рішень та платформ, що пропонують подібні функціонали, були проаналізовані різні підходи до реалізації планування робочого часу, а також можливості інтеграції з іншими системами та сервісами. Це дозволило визначити основні функціональні можливості, які повинна мати розроблена система, такі як планування завдань, керування графіками, статистика та інтеграція з календарними сервісами.

На основі цього дослідження були сформульовані вимоги до функціональності та дизайну майбутнього веб-додатку, що дозволяє оптимізувати робочі процеси та забезпечити зручність для користувачів.

На основі визначених вимог та критеріїв було обрано набір технологій для реалізації веб-додатку, що включає фронтенд (React, Redux), бекенд (Node.js, Express), систему управління даними (MongoDB), а також засоби автентифікації та авторизації (NextAuth). Це дозволяє забезпечити високий рівень продуктивності, безпеки та зручності користування платформою.

		Улькін В.В.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Годлевський Ю.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		18

2. ПРОЕКТУВАННЯ МАРКЕТПЛЕЙСУ «Rate»

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

Маркетплейс "Rate" надає користувачам широкий спектр можливостей для зручної та безпечної взаємодії з платформою. Навіть неавторизовані відвідувачі можуть ознайомитися з основною інформацією на головній сторінці, переглядати доступні товари та послуги, вивчати рейтинги, відгуки інших користувачів, а також читати статті у блозі, що допомагає їм краще орієнтуватися у функціоналі сервісу та приймати обґрунтовані рішення перед покупкою.

Після простої реєстрації користувач отримує повний доступ до функціоналу платформи: може створювати власні оголошення, продавати товари чи послуги, купувати обрані позиції, залишати оцінки та коментарі, а також вести особистий кабінет для управління своїм обліковим записом. Таким чином, "Rate" виступає універсальним інструментом як для покупців, так і для продавців, забезпечуючи комфортний і надійний торговий процес.

Платформа також пропонує систему рейтингу та репутації, яка стимулює чесну поведінку учасників ринку та сприяє формуванню довіри між продавцями і покупцями. Завдяки цьому користувачі можуть швидко знаходити надійних партнерів для співпраці та уникати недобросовісних учасників.

Крім того, платформа реалізує збереження частини даних у localStorage для підвищення зручності користувача. Наприклад, додані в кошик товари, обрані фільтри, історія переглядів або чернетки оголошень зберігаються локально, що дозволяє повертатися до взаємодії з маркетплейсом у будь-який момент без втрати прогресу.

У таблиці нижче описується короткий опис функціоналу який доступний для користувача:

		Улькін В.В.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Годлевський Ю.О.				19
Змн.	Арк.	№ докум.	Підпис	Дата		

Функціонал користувача на платформі "Rate"

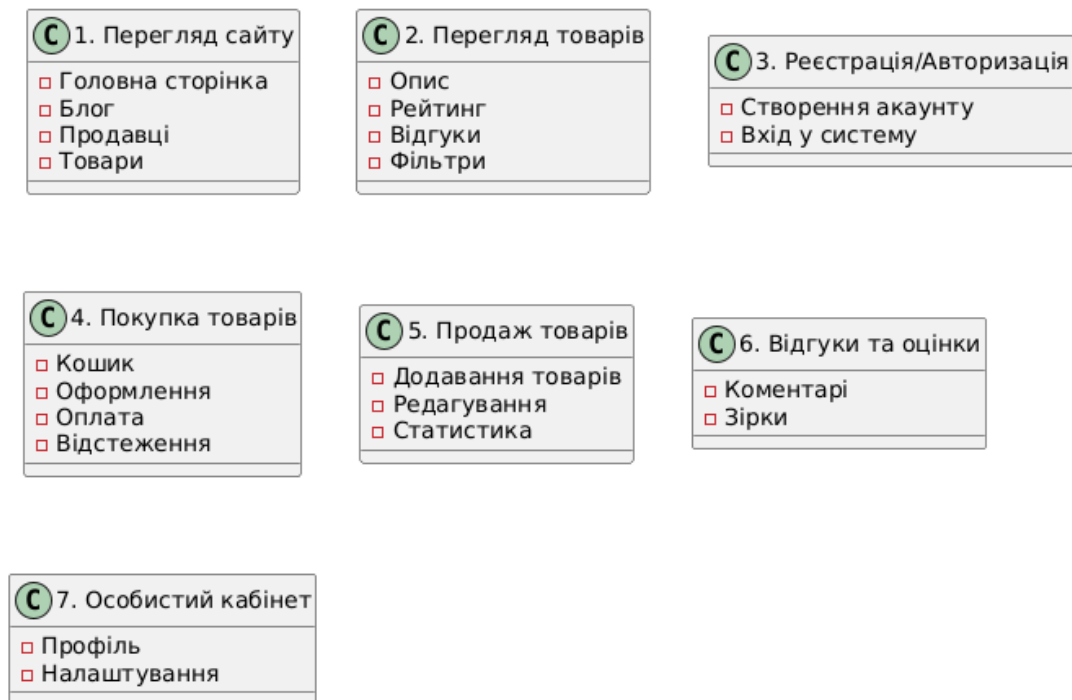


Рисунок 2.1 - Діаграма використання для користувача

Після авторизації адміністратора відкривається доступ до всього керуючого функціоналу платформи. Адміністратор може контролювати дії користувачів, модерувати товари та відгуки, аналізувати активність на сайті через розділ статистики, а також налаштовувати структуру каталогу і контенту. Крім цього, він має можливість обробляти скарги, запускати акції або створювати промокоди для залучення нових клієнтів. Завдяки цьому, адміністративна панель надає повний контроль над усіма аспектами роботи маркетплейсу "Rate", забезпечуючи стабільність, безпеку та якість сервісу.

Інтерфейс адміністратора працює з використанням захищених сесій та перевірки ролей користувачів, що обмежує доступ до критичних функцій лише уповноваженим особам. Також можливе збереження локальних налаштувань або фільтрів у localStorage, що підвищує ефективність щоденної роботи адміністратора, дозволяючи швидко повертатися до потрібних даних без повторного введення параметрів.

У таблиці нижче описується короткий опис функціоналу який доступний для адміністратора:

		Улькін В.В.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Годлевський Ю.О.				20
Змн.	Арк.	№ докум.	Підпис	Дата		

Функціонал адміністратора на платформі "Rate"



Рисунок 2.2 - Діаграма використання для адміністратора

Функціональні вимоги:

Користувачі повинні мати можливість проходити просту реєстрацію на платформі або авторизуватись для отримання повного доступу до функціоналу: створення оголошень, купівля, продаж, управління профілем.

Платформа повинна дозволяти створення, редагування, перегляд та видалення оголошень про товари або послуги. Система повинна забезпечувати конфіденційність персональних даних користувачів та безпечну обробку фінансової інформації.

Уся інформація повинна коректно та автоматично оновлюватися відповідно до дій користувачів (додавання нових оголошень, змін, коментарів тощо).

Нефункціональні вимоги:

Сприйняття інтерфейсу має бути інтуїтивно зрозумілим, адаптивним до різних пристроїв, а час на його освоєння не повинен перевищувати 7 хвилин. Відповідь системи на стандартні запити має становити не більше 0.370 мс, на складні — до 0.640 мс. Надійність системи повинна працювати безперервно не менше 90 годин, з максимально допустимим рівнем помилок — 1 на 2000 запитів.

Масштабованість, додавання нового функціоналу не повинно викликати критичних збоїв у роботі системи.

Аналіз функціональних вимог дав змогу виділити основні елементи, які формуватимуть ядро системи. На базі цього планується розробка архітектури маркетплейсу "Rate", яка забезпечить ефективну взаємодію між модулями та безперебійну роботу всіх компонентів. Докладніша структура взаємодії між модулями буде описана у відповідних розділах технічної документації.



Рисунок 2.3 - Візуалізація функціональних та нефункціональних вимог

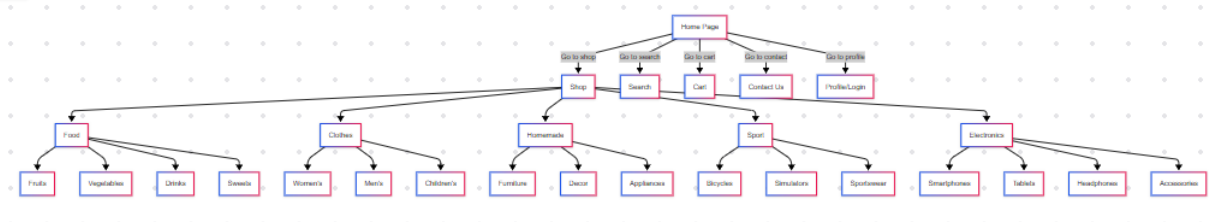


Рисунок 2.4 - Опис діаграми компонентів

Опис функціональних і нефункціональних компонентів:

Файл app/layout.tsx відповідає за загальне компонування сторінок застосунку. У ньому розміщуються спільні елементи інтерфейсу — такі як заголовок (<Header />), футер (<Footer />) та основний вміст (<main />). Також тут підключаються глобальні стилі, які впливають на вигляд усіх сторінок сайту.

Файл `app/page.tsx` реалізує головну сторінку платформи. Вона містить банер з акційною інформацією, перелік основних категорій товарів, блок із популярними позиціями, а також розділ з акційними або рекомендованими пропозиціями.

Сторінка магазину реалізована у файлі `app/shop/page.tsx`. Тут відображаються всі доступні товари. Передбачено можливість фільтрування за різними критеріями (ціна, категорії, рейтинг), а також сортування (за популярністю, новизною тощо). Це дає змогу користувачу швидко знайти потрібний товар серед великої кількості позицій.

Файл `app/product/[id]/page.tsx` відповідає за детальний перегляд конкретного товару. Користувач може переглянути зображення, ознайомитися з описом, дізнатися ціну, прочитати відгуки інших покупців і натиснути кнопку для додавання товару до кошика або його миттєвого придбання.

Сторінка кошика (`app/cart/page.tsx`) надає користувачу змогу переглядати вибрані товари, змінювати їхню кількість або видаляти з кошика. Також тут реалізовано оформлення замовлення — з введенням даних про доставку та оплату.

Файл `app/profile/page.tsx` реалізує особистий кабінет користувача. У цьому розділі зберігається інформація про акаунт, історія замовлень, а для продавців також передбачено керування власними оголошеннями або товарами.

Адміністративна панель (`app/admin/page.tsx`) доступна лише користувачам з правами адміністратора. Тут реалізовано функціонал для управління базовими сутностями — користувачами, категоріями, товарами. Це дозволяє модераторам контролювати наповнення платформи та дотримання правил.

Файл `components/Header.tsx` містить верхнє меню сайту, яке включає логотип, головну навігацію по розділах, поле для пошуку, а також кнопки для входу, перегляду профілю й переходу до кошика.

Компонент `Footer.tsx` формує нижній колонтитул, у якому розміщується загальна інформація про платформу, контактні дані та корисні посилання — наприклад, умови використання, підтримка, соціальні мережі.

		Улькін В.В.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Годлевський Ю.О.				23
Змн.	Арк.	№ докум.	Підпис	Дата		

Компонент `ProductCard.tsx` використовується для відображення товарів у списках — таких як вітрина, категорії або результати пошуку. Він показує основні дані: зображення, назву, ціну й рейтинг.

Компонент `CategoryList.tsx` відображає список категорій товарів (наприклад, їжа, одяг, електроніка). Це спрощує навігацію та дає змогу швидко перемкнутися на потрібний розділ магазину.

Компонент `FilterSidebar.tsx` відповідає за фільтри. Він розміщується у вигляді бічної панелі й дозволяє налаштувати перегляд товарів за ціною, рейтингом, категоріями тощо.

Файл `components/Breadcrumbs.tsx` реалізує "хлібні крихти" — елемент, що допомагає користувачу орієнтуватися в структурі сайту та швидко повертатися на попередні сторінки.

Основні функції:

Серед основних функціональних можливостей платформи можна виокремити наступні: реєстрація нового користувача шляхом передачі необхідних персональних даних (`registerUser(data)`), авторизація за допомогою облікових даних, зокрема електронної пошти та пароля (`loginUser(credentials)`), а також вихід із системи (`logoutUser()`).

Платформа підтримує створення нових оголошень про товари (`createListing(productData)`), редагування вже існуючих записів (`updateListing(listingId, newData)`) та їх видалення (`deleteListing(listingId)`).

Користувачі можуть переглядати перелік усіх доступних товарів (`getAllListings()`), а також отримувати детальну інформацію про конкретну позицію (`getListingById(id)`). Окрім цього, реалізована можливість оформлення покупки обраного товару (`buyProduct(productId, buyerData)`) та завантаження інформації з особистого кабінету користувача (`getUserDashboard(userId)`), що забезпечує індивідуальний підхід до управління обліковим записом.

		Улькін В.В.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Годлевський Ю.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		24

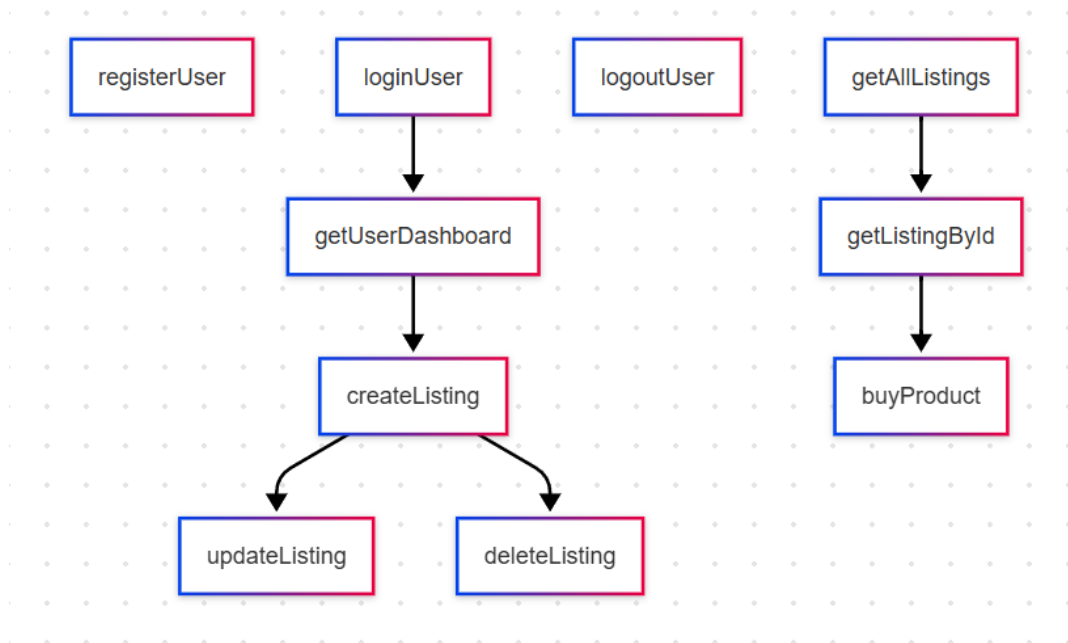


Рисунок 2.5 – Діаграма основних функцій

До функціональних можливостей системи також належать інструменти пошуку та навігації, які забезпечують зручну взаємодію користувача з контентом платформи. Зокрема, реалізовано пошук товарів за назвою з підтримкою автозаповнення (searchByName(name)), що дозволяє оперативно знаходити потрібні позиції.

Фільтрація за категоріями (filterByCategory(category)) дає змогу користувачам звужити результати пошуку відповідно до обраної тематики. Також передбачена можливість сортування списку товарів за різними критеріями (sortBy(criteria)), зокрема за ціною або рейтингом. Це покращує зручність навігації по каталогу та дозволяє швидко знаходити найбільш релевантні пропозиції. Завдяки комбінації фільтрації й сортування користувачі можуть ефективно керувати переглядом товарів і приймати рішення на основі своїх уподобань.

Окрему увагу приділено роботі зі списком обраних товарів: користувач може переглядати свої улюблені позиції (getFavorites(userId)), додавати нові товари до списку (addToFavorites(productId)) та видаляти їх за потреби (removeFromFavorites(productId)), що сприяє персоналізації користувацького досвіду.



Рисунок 2.6 – Діаграма функцій пошуку та навігації

Система підтримує функціонал, спрямований на забезпечення ефективної комунікації між користувачами, а також на залучення зворотного зв'язку щодо товарів. Зокрема, реалізовано можливість надсилання приватних повідомлень між користувачами платформи (`sendMessage(toUserId, text)`), що сприяє оперативному обговоренню умов покупки або обміну додатковою інформацією.

Для збереження контексту взаємодії передбачено завантаження історії листування (`getMessages(conversationId)`), що дозволяє користувачам переглядати попередні повідомлення. Крім того, функція додавання коментарів до товарів (`addComment(productId, comment)`)

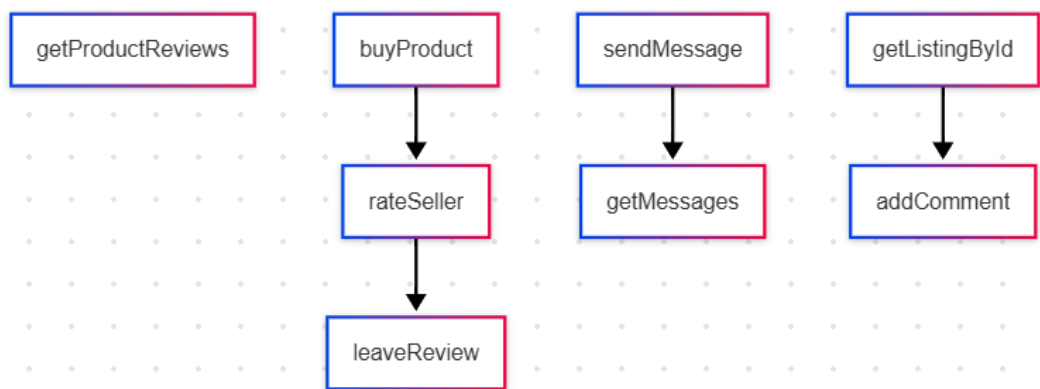


Рисунок 2.7 – Діаграма функцій комунікації та відгуків

Адміністративні та технічні функції платформи включають отримання статистики системи (`getSystemStats()`), що дає змогу відстежувати кількість товарів і користувачів. Модерація оголошень (`moderateListing(listingId)`) забезпечує контроль якості контенту.

Користувачі можуть надсилати скарги на товари (reportProduct(productId)), а адміністрація — розсилати сповіщення (sendNotification(userId, message)). Також реалізовано перемикання теми інтерфейсу між світлою і темною (switchTheme(theme)), що покращує зручність використання.

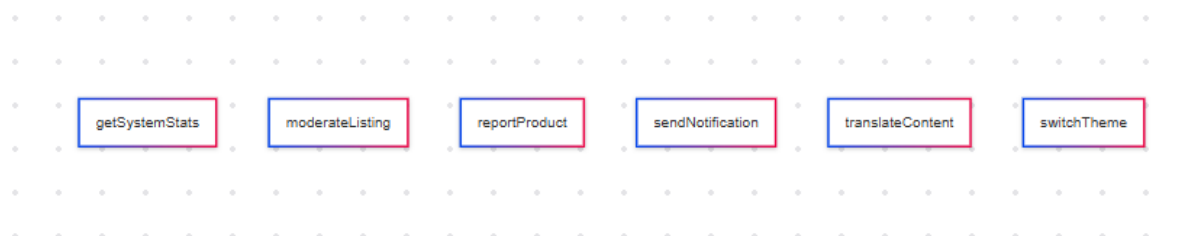


Рисунок 2.8 – Діаграма функцій адміністративні та технічні

2.2. Розробка системи бази даних

Для збереження та подальшого використання даних було розроблено колекції бази даних для кожної зі змінюваних сутностей, таких як:

goods – колекція, в якій зберігається інформація про товари, що публікуються користувачами (назва, категорія, ціна, зображення, теги, рейтинг, автор тощо)

users – колекція, в якій зберігається інформація про зареєстрованих користувачів (ім’я, роль, пароль)

comments – колекція, в якій зберігаються коментарі користувачів до товарів (текст, автор, дата створення)

favorites – колекція, яка містить обрані товари користувачів (userId, перелік товарів)

categories – колекція з переліком категорій товарів (назва, slug)

ratings – колекція, яка зберігає оцінки, що виставляються користувачами для товарів (userId, товар, рейтинг).

Кожна з цих колекцій відіграє ключову роль у структурі даних застосунку та забезпечує гнучкість при реалізації CRUD-операцій. Такий підхід дозволяє легко масштабувати систему, організовувати зв’язки між сутностями та підтримувати узгодженість даних у межах усієї платформи.

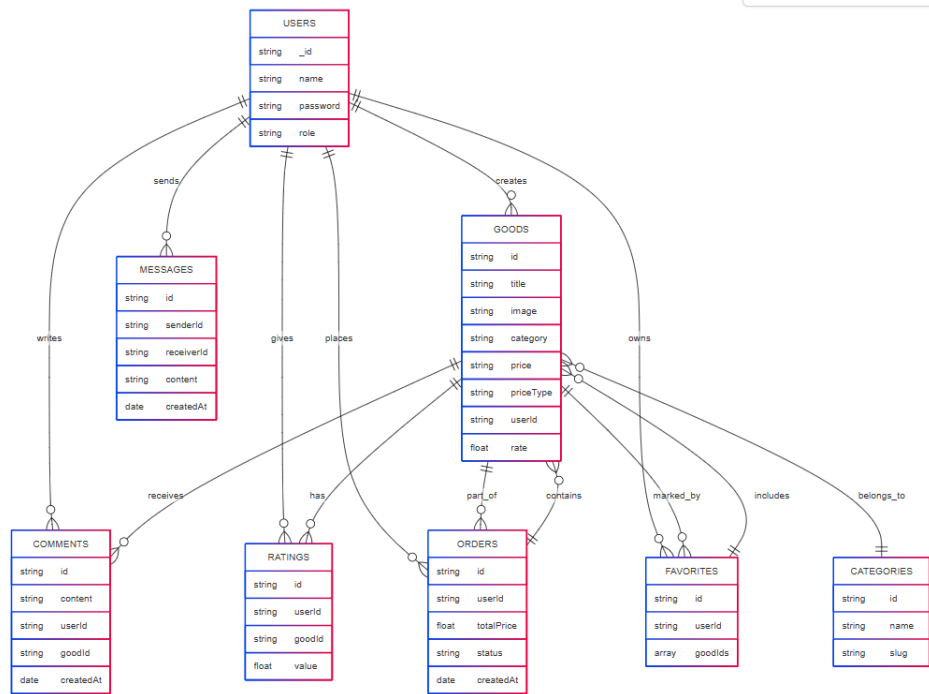


Рисунок 2.9 – Діаграма для опису БД

Нижче буде наведено опис кожної з таблиць бази даних (див. табл. 2.1 – 2.8)

Таблиця users

Таблиця 2.1

Поле	Тип	Призначення
_id	UniqueIdentifier	Унікальний ідентифікатор
name	VARCHAR(255)	Ім'я користувача
password	VARCHAR(255)	Пароль користувача
role	VARCHAR(100)	Роль користувача

Таблиця goods

Таблиця 2.2

Поле	Тип	Призначення
_id	UniqueIdentifier	Унікальний ідентифікатор
title	VARCHAR(255)	Назва товару
image	TEXT	Шлях до зображення

category	VARCHAR(255)	Категорія
price	FLOAT	Ціна
userId	UniqueIdentifier	ID користувача
rate	FLOAT	Рейтинг товару

Таблиця comments

Таблиця 2.3

Поле	Тип	Призначення
_id	UniqueIdentifier	Унікальний ідентифікатор
content	TEXT	Вміст коментаря
userId	UniqueIdentifier	ID користувача
goodId	UniqueIdentifier	ID товару
createdAt	DATE	Дата створення

Таблиця orders

Таблиця 2.4

Поле	Тип	Призначення
_id	UniqueIdentifier	Унікальний ідентифікатор
userId	UniqueIdentifier	ID користувача
totalPrice	FLOAT	Загальна ціна
status	VARCHAR(100)	Статус замовлення
createdAt	DATE	Дата створення

Таблиця favorites

Таблиця 2.5

Поле	Тип	Призначення
_id	UniqueIdentifier	Унікальний ідентифікатор
userId	UniqueIdentifier	ID користувача
goodIds	ARRAY	Список ID товарів

Таблиця categories

Таблиця 2.6

Поле	Тип	Призначення
_id	UniqueIdentifier	Унікальний ідентифікатор
name	VARCHAR(255)	Назва категорії
slug	VARCHAR(255)	Слаг категорії

Таблиця ratings

Таблиця 2.7

Поле	Тип	Призначення
_id	UniqueIdentifier	Унікальний ідентифікатор
userId	UniqueIdentifier	ID користувача
goodId	UniqueIdentifier	ID товару
value	FLOAT	Оцінка

Таблиця messages

Таблиця 2.8

Поле	Тип	Призначення
_id	UniqueIdentifier	Унікальний ідентифікатор
senderId	UniqueIdentifier	Відправник
receiverId	UniqueIdentifier	Одержувач

content	TEXT	Повідомлення
createdAt	DATE	Дата надсилання

Після проведення всебічного аналізу та ретельного проєктування бази даних, що охоплює всі ключові аспекти функціонування маркетплейсу, кожен користувач — незалежно від своєї ролі (адміністратор, продавець чи покупець) — отримає змогу повноцінно реалізувати свої потреби у межах платформи.

Зокрема, користувачі зможуть з легкістю взаємодіяти з товарами, переглядати їх, залишати коментарі, здійснювати покупки або продажі, керувати власними профілями, бачити історію активності, а також швидко отримувати доступ до актуальної інформації, що автоматично оновлюється в режимі реального часу.

Таким чином, проєктована база даних виступає фундаментом, який забезпечує стабільну та ефективну роботу всієї інформаційної системи маркетплейсу.

Крім того, завдяки правильному структуруванню сутностей і зв'язків між ними, база даних дозволяє гнучко реалізовувати нові функціональні можливості без суттєвих змін у загальній архітектурі. Це особливо важливо для майбутнього масштабування системи, оновлення логіки бізнес-процесів або інтеграції з додатковими сервісами.

2.3. Проєктування та реалізація алгоритмів роботи системи

Алгоритм роботи адміністратора передбачає автентифікацію в системі з відповідною роллю. Після входу він отримує доступ до управління даними користувачів: перегляд, редагування та видалення облікових записів. Адміністратор може переглядати всі товари, додані користувачами, а також здійснювати їх модерацію, зокрема публікацію або деактивацію. Крім того, передбачена можливість перегляду та модерації коментарів, ведення блогу (створення, редагування, видалення статей), а також перегляду статистичних даних системи, включаючи кількість товарів, користувачів і категорій.



Рисунок 2.10 - Активність адміністратора

Розглянемо алгоритм роботи продавця:

Продавець проходить автентифікацію в обліковому записі з відповідною роллю, додає нові товари до маркетплейсу, редагує та видаляє існуючі, переглядає особистий кабінет з інформацією про замовлення та рейтинги, спілкується з покупцями через коментарі і має можливість переглядати статистику своїх товарів, зокрема кількість переглядів, оцінки та продажі. Завдяки інтеграції клієнтської частини з localStorage, продавець може зберігати чернетки ще не опублікованих товарів або налаштувань оголошення навіть без негайного збереження в базі даних. Це дає змогу повертатися до редагування пізніше без втрати введених даних.

Крім того, асинхронна взаємодія з API дозволяє оновлювати статистику в реальному часі, що допомагає продавцю оперативно реагувати на зміну попиту та коригувати асортимент. Такий підхід покращує зручність користування системою та сприяє активнішій участі продавців у платформі.



Рисунок 2.11 - Активність продавця

Розглянемо алгоритм роботи покупця:

Покупець проходить реєстрацію або автентифікацію в обліковому записі з відповідною роллю, здійснює пошук і перегляд товарів за категоріями та тегами, додає обрані товари до списку вподобаного або кошика, оформлює замовлення, залишає відгуки та оцінки після покупки, а також має доступ до перегляду історії своїх замовлень у особистому кабінеті. Збереження даних кошика та вподобаного здійснюється через localStorage, що дозволяє покупцю зберігати вибрані товари навіть без авторизації та продовжити покупку пізніше, не втративши вже обрані позиції.

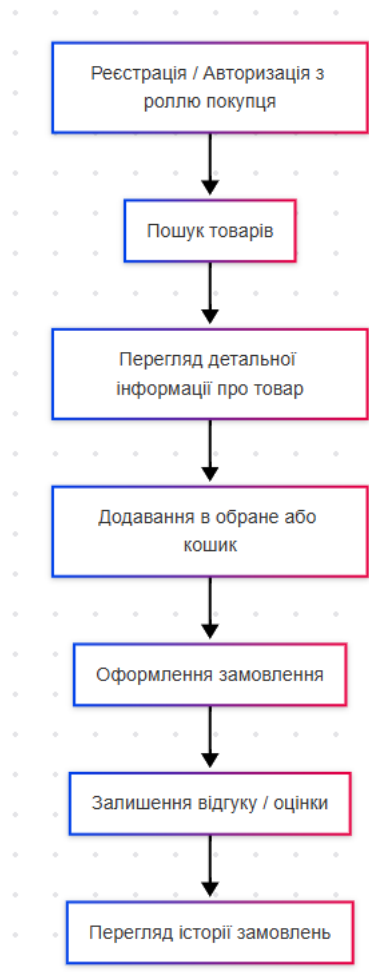


Рисунок 2.12 - Активність покупця

2.4. Реалізація функціоналу інформаційного додатка

Для створення онлайн-маркетплейсу було обрано сучасний і перевірений стек технологій, який забезпечує надійну, масштабовану й ефективну реалізацію усіх необхідних функцій. Основою для фронтенд-частини став фреймворк Next.js, що поєднує можливості серверного рендерингу, статичної генерації та гнучкої маршрутизації. Завдяки Next.js платформа отримує високу продуктивність, покращене індексування в пошукових системах та зручну організацію сторінок. Це особливо важливо для електронної комерції, де швидкість завантаження і SEO мають безпосередній вплив на кількість відвідувачів і конверсію.

React, який використовується як основна бібліотека для побудови

інтерфейсу, забезпечує створення компонентної структури застосунку. Це дозволяє зручно управляти станом, оновлювати лише ті частини сторінки, які змінюються, та легко підтримувати й масштабувати код у майбутньому. Крім того, React має велику екосистему, що дозволяє використовувати сторонні рішення для форм, валідації, маршрутизації, анімацій тощо, значно пришвидшуючи розробку.

З боку серверної логіки було обрано Node.js, який надає змогу реалізовувати backend на тій самій мові програмування, що й frontend — JavaScript. Це забезпечує єдність стеку, спрощує комунікацію між частинами проєкту та дозволяє команді працювати ефективніше, без потреби перемикання між різними мовами програмування. На основі Node.js реалізовано сервер за допомогою Express.js — простого, але гнучкого фреймворку, який дозволяє легко керувати маршрутами, підключати middleware, працювати з аутентифікацією, валідацією, сесіями та іншими ключовими аспектами серверного додатку. Express дає змогу швидко інтегрувати зовнішні сервіси, реалізовувати RESTful API та здійснювати взаємодію з базою даних [6].

Як сховище даних було обрано MongoDB — документо-орієнтовану базу даних, яка відмінно підходить для зберігання гнучких і динамічних структур, таких як товари з різними атрибутами, користувачі з профілями, відгуки, рейтинги тощо. На відміну від реляційних баз даних, MongoDB дозволяє швидко змінювати структуру документів, не вимагаючи складної міграції або попереднього визначення схем, що є важливою перевагою в умовах постійної еволюції проєкту.

Інтерфейс користувача реалізується з використанням двох популярних бібліотек — Ant Design (AntD) та Material UI (MUI). Обидві ці бібліотеки надають великий набір готових компонентів, таких як кнопки, модальні вікна, форми, таблиці, що відповідають сучасним стандартам дизайну. Вони також підтримують адаптивність, що дозволяє платформі коректно відображатися на різних пристроях — від настільних ПК до мобільних телефонів. Використання цих бібліотек значно прискорює процес розробки, дозволяє зосередитися на логіці замість дрібних деталей верстки, а також забезпечує естетично привабливий вигляд інтерфейсу без зайвих зусиль.

		Улькін В.В.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Годлевський Ю.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		35

Таким чином, вибір цього стеку технологій обумовлений не лише зручністю розробки, а й високими вимогами до продуктивності, безпеки, масштабованості та сучасного користувацького досвіду. Усі обрані інструменти добре інтегруються між собою, утворюючи цілісну екосистему, що дозволяє створити повнофункціональний маркетплейс із широкими можливостями для розширення, оптимізації та підтримки в майбутньому.

Висновки до другого розділу

Функціонал онлайн-маркетплейсу товарів повністю відповідає поставленому завданню. У другому розділі було продемонстровано проектування бази даних, яка складається з восьми основних колекцій. Також було показано ключові взаємодії користувачів платформи з її функціоналом та окремими модулями. У пункті 2.4 детально розглянуто реалізацію функціоналу, контролерів та моделей, які формують основну структуру вебдодатку маркетплейсу.

		Улькін В.В.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Годлевський Ю.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		36

3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З МАРКЕТПЛЕЙСУ «Rate»

3.1. Порядок встановлення та налаштування параметрів системи

Для кращого розуміння архітектури вебмаркетплейсу доцільно використовувати діаграму розгортання.

Діаграма розгортання (Deployment diagram) — це схема, яка ілюструє фізичну архітектуру програмної системи, зокрема, як її програмні компоненти (frontend, backend, база даних) розгортаються на різних апаратних пристроях. У контексті маркетплейсу це дає змогу чітко побачити, як саме клієнтська частина (реалізована за допомогою Next.js) взаємодіє з серверною частиною (Express), а також як остання підключається до бази даних (MongoDB).

Дана діаграма демонструє розміщення компонентів на реальних вузлах — наприклад, фронтенд-додаток може бути розміщений на хостингу типу Vercel або іншому сервері, бекенд-сервер — на окремому VPS або хмарному середовищі, а база даних MongoDB — на спеціалізованому сервері баз даних або в хмарі (наприклад, MongoDB Atlas).

Ця діаграма є важливим інструментом при проєктуванні та аналізі інфраструктури вебзастосунку перед його фактичним розгортанням. Вона дозволяє оцінити можливі вузькі місця, спланувати масштабування системи та забезпечити її надійність.

Крім того, діаграма розгортання дає змогу комунікувати технічні деталі архітектури між учасниками команди — розробниками, DevOps-інженерами, аналітиками та зацікавленими сторонами. Вона допомагає швидко зрозуміти, які компоненти взаємодіють між собою, які протоколи або API використовуються для комунікації, а також які сервіси є зовнішніми або внутрішніми.

У випадку масштабного маркетплейсу діаграма також може включати додаткові елементи, такі як CDN для пришвидшення доставки контенту, служби аутентифікації, сервіси обробки платежів, а також засоби моніторингу та логування.

		Улькін В.В.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Годлевський Ю.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		37

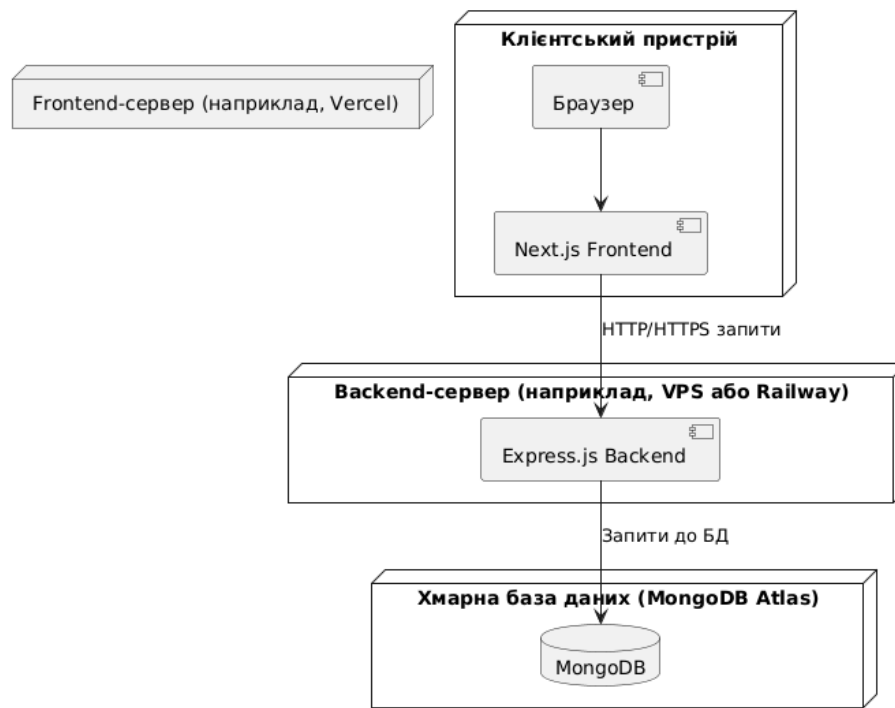


Рисунок 3.1 – Діаграма розгортання

Проаналізувавши дану діаграму, можемо зробити наступні висновки:

Клієнтська частина, реалізована на базі Next.js, звертається до бекенд-сервера для отримання необхідних даних через HTTP-запити.

Під час першого запуску застосунку користувач отримує згенеровану HTML-розмітку (SSR), після чого подальша взаємодія відбувається через асинхронні запити до API для оновлення контенту без перезавантаження сторінки [20].

Бекенд-частина, побудована на Express.js, розгортається на порту 3001, тоді як клієнтська частина працює на порту 3000 під час розробки.

Бекенд взаємодіє з MongoDB, яка розміщена у хмарі (наприклад, MongoDB Atlas), для зберігання та обробки інформації про товари, користувачів та замовлення.

Ця архітектура дозволяє чітко розділити клієнтську та серверну частини вебзастосунку, забезпечуючи їх незалежне розгортання і масштабування. Бекенд виконує бізнес-логіку, обробляє запити та відповідає за безпеку, тоді як клієнтська частина відповідає за візуальне представлення інтерфейсу та інтеракцію з користувачем.

Асинхронна взаємодія з API дозволяє реалізувати динамічне оновлення контенту, що покращує загальний користувацький досвід і забезпечує швидкодію та гнучкість системи.

Ця архітектура дозволяє розділити бекенд і клієнтську частину вебзастосунку, забезпечуючи їх незалежне розгортання та масштабування. Бекенд-частина, реалізована за допомогою Express.js, забезпечує доступ до ресурсів, обробку запитів і реалізацію бізнес-логіки, включаючи роботу з базою даних MongoDB. Клієнтська частина, створена на базі Next.js, відповідає за візуальне представлення інтерфейсу та взаємодію з користувачем.

Звернення до REST API дозволяє клієнтській частині оновлювати дані асинхронно, без необхідності повного перезавантаження сторінки, що забезпечує динамічне оновлення контенту та зручний користувацький досвід [24].

3.2. Структура інтерфейсу та порядок взаємодії з онлайн-сервісом

Коли користувач переходить на сторінку нашого застосунку, його зустрічає головна сторінка з товарами, які доступні для перегляду та покупки. На рис. 3.2 зображено головну сторінку маркетплейсу, де користувач може побачити основну інформацію.

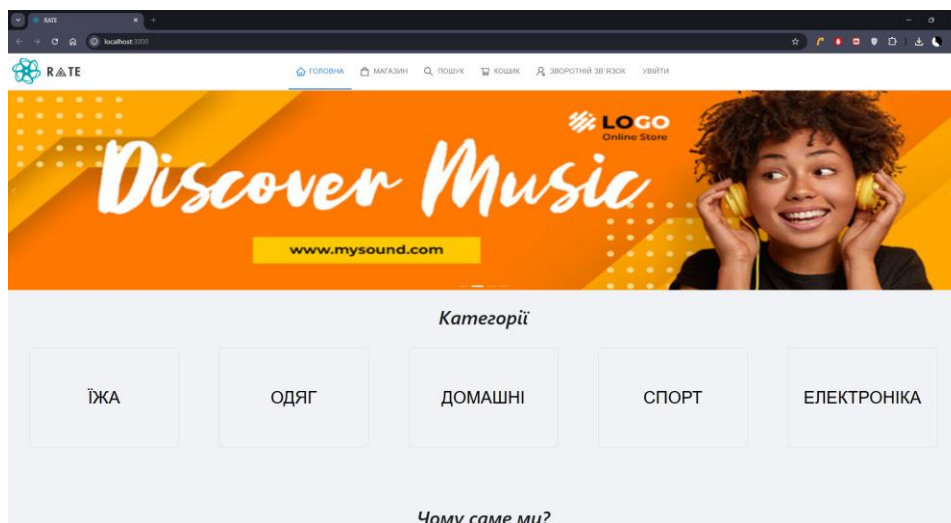


Рис. 3.2 – Головна сторінка застосунку з товарами

Користувач має можливість зробити авторизацію, зареєструватись або ознайомитись з усіма доступними фільмами в прокаті. На рис. 3.3 зображено вікно реєстрації.

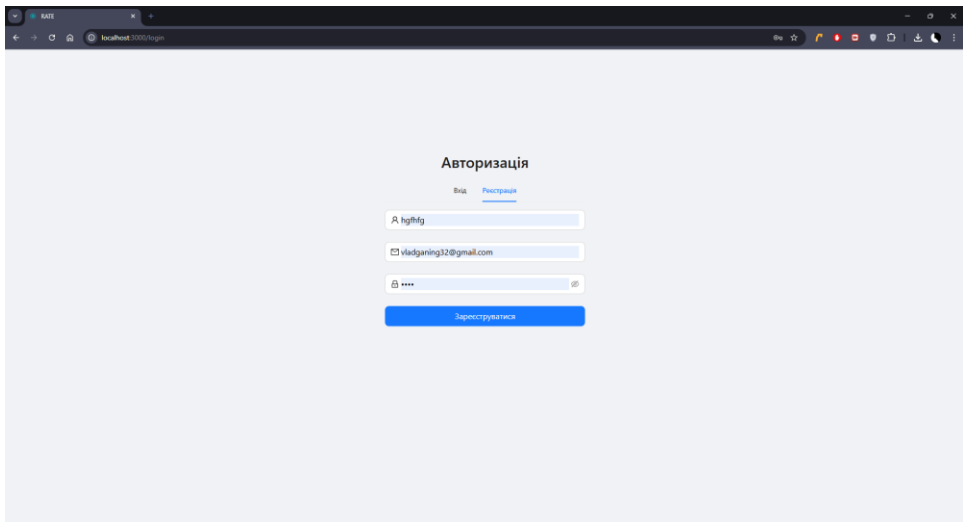


Рис. 3.3 – Сторінка реєстрації

Користувач може зробити авторизацію на сайті ввівши інформацію у відповідні поля. На рис. 3.4 зображено вікно авторизації.

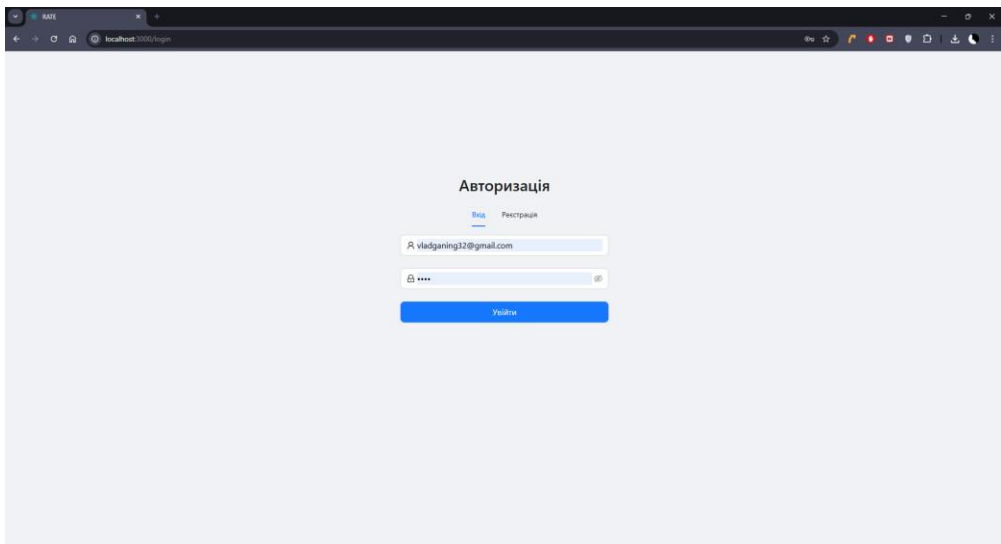


Рис. 3.4 – Сторінка входу

Після реєстрації користувач може замовити товар або переглянути його

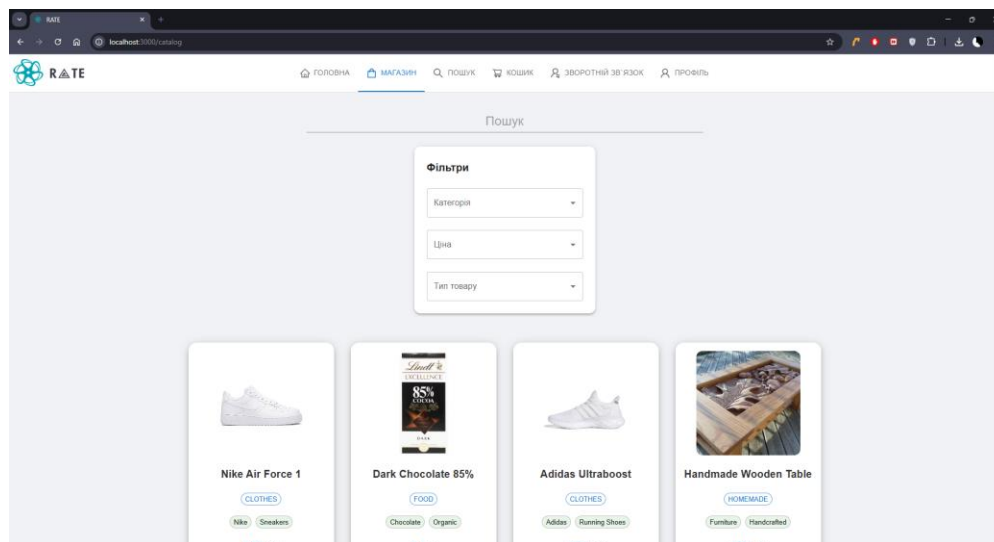


Рис. 3.5 – Сторінка каталогу товарів

Також для більшого ознайомлення з товаром користувач може перейти на сторінку товару та переглянути деталі товару

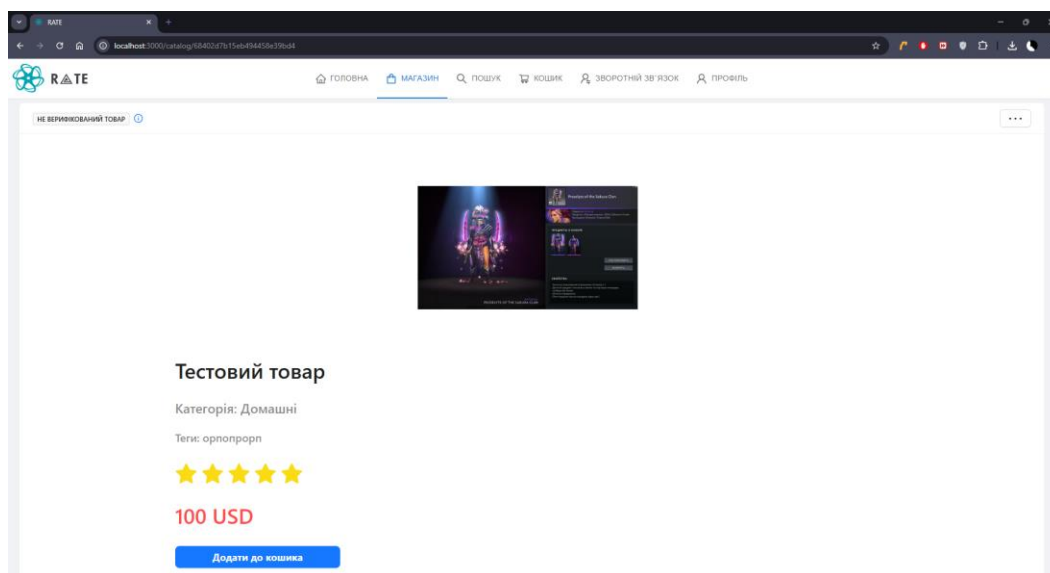


Рис. 3.6 – Сторінка товару

На сторінці товару користувач може не тільки переглядати деталі товару а ще й залишит свій відгук про товар, та оцінити його

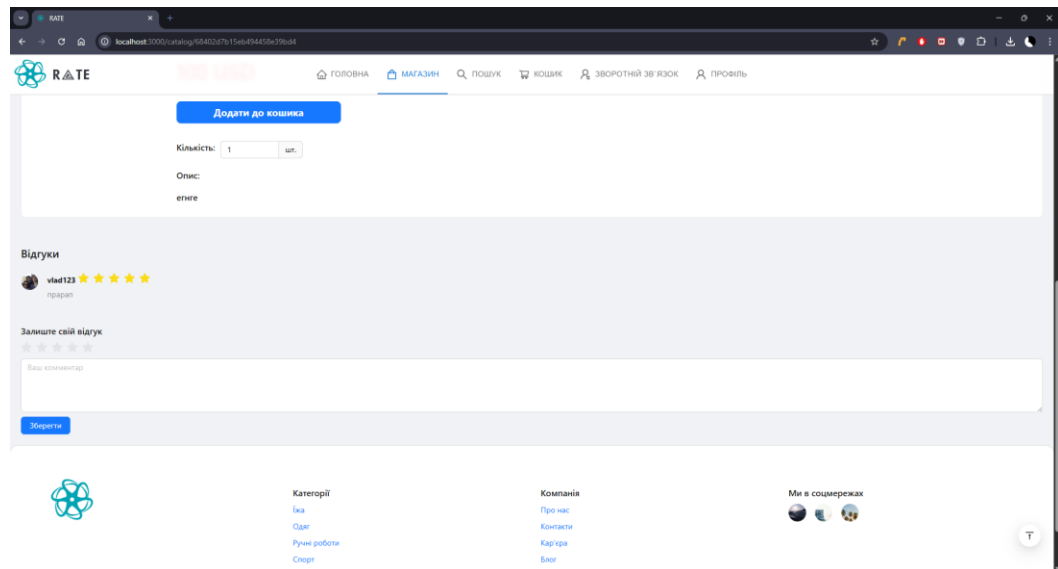


Рис. 3.7 – Інтерфейс коментарів та рейтингу товару

Коли користувач знаходить товар, який хоче придбати, він може натиснути кнопку «Додати до кошика», яка розташована безпосередньо на картці товару. Після цього товар зберігається у локальному сторі браузера і в правому верхньому куті, в хедері, відображається іконка кошика з кількістю доданих товарів.

Користувач має два способи перейти до перегляду кошика:

Через хедер – натиснувши на іконку кошика, відкривається модальне вікно або сторінка з переліком усіх товарів, які були додані. Тут користувач може змінювати кількість одиниць, видаляти товари або перейти до оформлення замовлення.

Через сторінку профілю – користувач також може перейти до розділу профілю, де зберігається історія доданих товарів або оформлених замовлень, якщо він авторизований.

Цей процес забезпечує зручний і швидкий доступ до кошика, дозволяючи користувачу контролювати покупки без необхідності залишати поточну сторінку. Крім того, використання асинхронних запитів до API (при оформленні замовлення) дозволяє оновлювати дані без перезавантаження сторінки, що покращує користувацький досвід.

Також важливо, що дані кошика зберігаються у localStorage браузера, що дозволяє зберігати додані товари навіть після оновлення сторінки або повторного відкриття сайту. Завдяки цьому користувач може повернутись до оформлення замовлення пізніше без втрати обраних позицій.

		Улькін В.В.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Годлевський Ю.О.				42
Змн.	Арк.	№ докум.	Підпис	Дата		

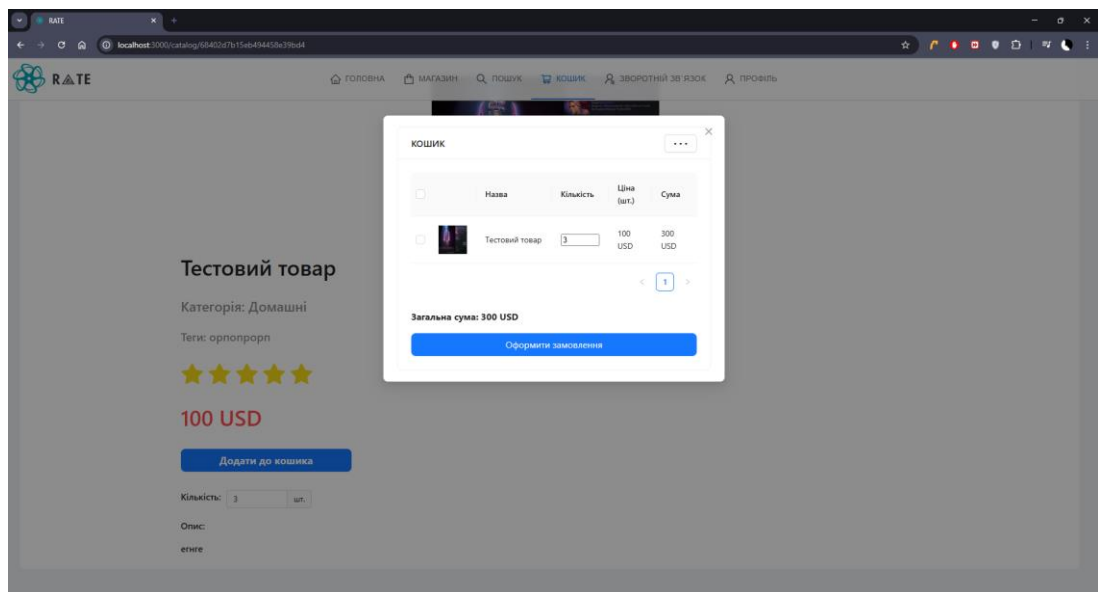


Рис. 3.8 – Інтерфейс кошика (Хедер)

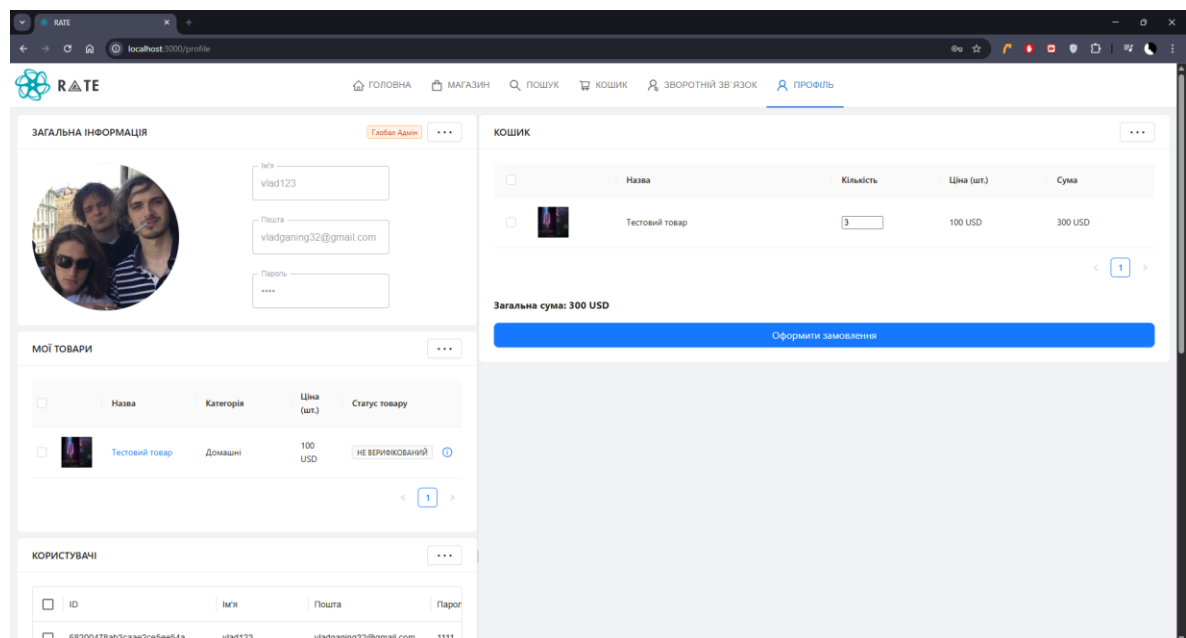


Рис. 3.9 – Інтерфейс кошика (Сторінка профілю)

Після того як користувач додав товари в корзину, він може переглянути список вибраних позицій. Це робиться через модальне вікно корзини або окрему сторінку, яка відкривається після натискання на іконку кошика в хедері.

У цій корзині користувач бачить назву товару, зображення, обрану кількість і підсумкову ціну.

Якщо деякі товари більше не потрібні, користувач може видалити їх із корзини, натиснувши на відповідну кнопку "Видалити".

Коли список замовлення сформований, користувач натискає кнопку «Оформити замовлення», після чого відкривається форма для заповнення даних

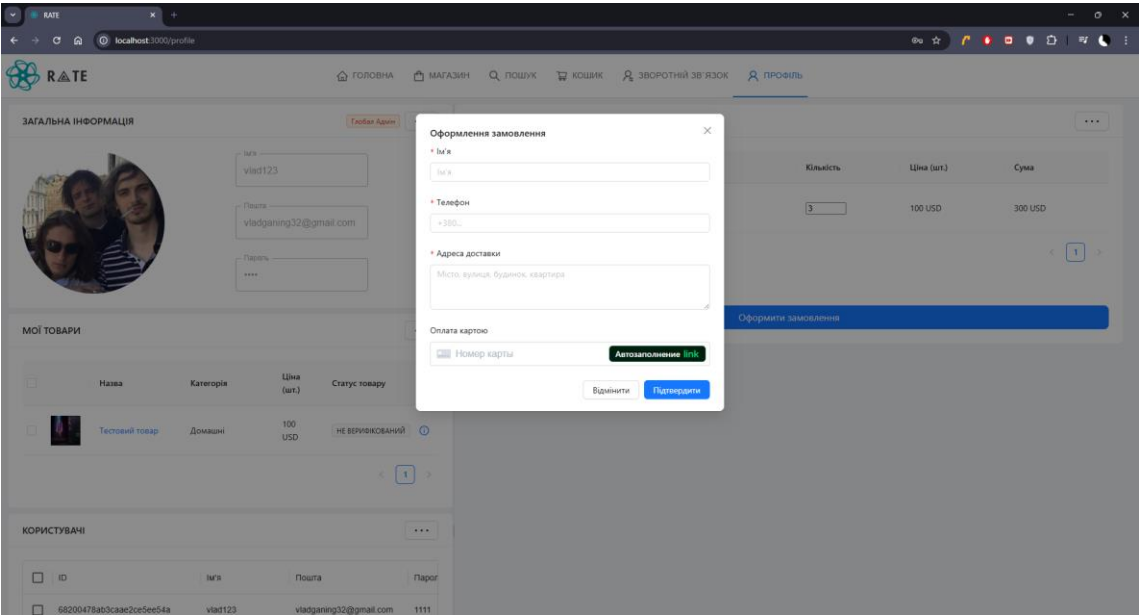


Рис. 3.10 – Інтерфейс оформлення замовлення

Після успішної оплати замовлення, кошти переводяться з банківської карти покупця на корпоративний рахунок компанії, інтегрований через платіжну систему Stripe [5].

Stripe виступає в ролі платіжного шлюзу, який забезпечує: захищене зняття коштів із картки користувача, переведення суми на баланс компанії, зареєстрований у Stripe, генерацію підтвердження оплати та унікального ідентифікатора транзакції.

Усі дані картки обробляються на стороні Stripe, що повністю відповідає вимогам PCI DSS — міжнародного стандарту безпеки платіжних даних. Це забезпечує високий рівень захисту персональної та фінансової інформації користувача.

Таким чином, завершення оплати не тільки підтверджує замовлення, але й зараховує кошти на офіційний акаунт компанії в Stripe, звідки вони можуть бути виведені на банківський рахунок або використані для подальших розрахунків.

У вебзастосунку реалізовано рольову модель доступу, яка ефективно розмежовує права користувачів залежно від їхнього статусу.

Існує три типи ролей:

Global Admin — це користувач з максимальними правами доступу, що виконує функції загальносистемного адміністрування. Він має повний доступ до всієї інформації на сайті, включно з персональними даними інших користувачів, історією замовлень, аналітикою та іншими критичними даними. Global Admin може створювати, редагувати та видаляти будь-які товари на платформі, незалежно від того, кому вони належать. Його роль полягає у підтримці цілісності системи, модеруванні контенту, вирішенні конфліктних ситуацій, а також у забезпеченні безперервної роботи сервісу.

Domain Admin, або продавець. Це користувач, який має обмежені права порівняно з Global Admin, проте володіє повним контролем над власними товарами. Він може створювати нові позиції на маркетплейсі, редагувати наявні, завантажувати зображення, вказувати ціни, опис, а також видаляти товари, якщо вони більше не актуальні.

Звичайний користувач (User) має можливість переглядати каталог товарів, додавати їх у кошик, оформлювати та оплачувати замовлення, але не має прав на редагування чи адміністрування контенту.

Ця ієрархія ролей дозволяє забезпечити гнучку, безпечну та масштабовану модель керування доступом, яка відповідає вимогам типового маркетплейсу або багатокористувацького вебзастосунку.

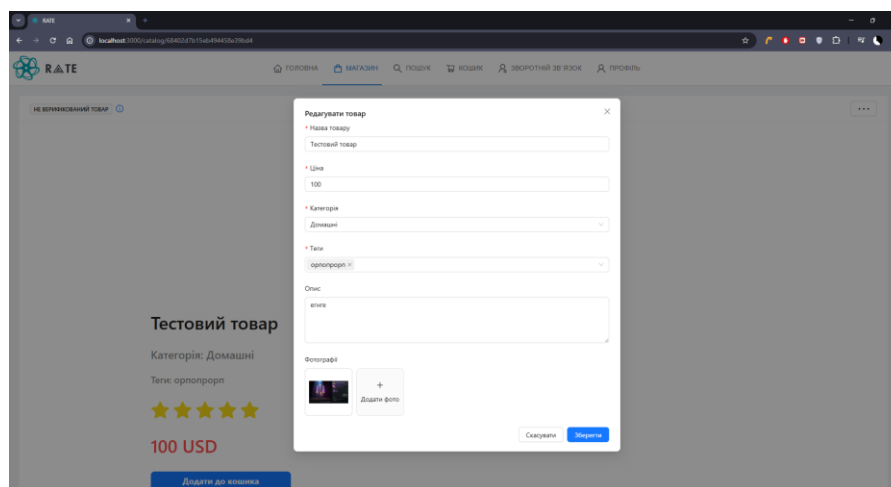


Рис. 3.11 – Інтерфейс редагування товару

		Улькін В.В.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Годлевський Ю.О.				45
Змн.	Арк.	№ докум.	Підпис	Дата		

Кожен користувач вебзастосунку має можливість написати листа на електронну пошту сайту з метою зворотного зв'язку, скарги або пропозицій щодо покращення сервісу.

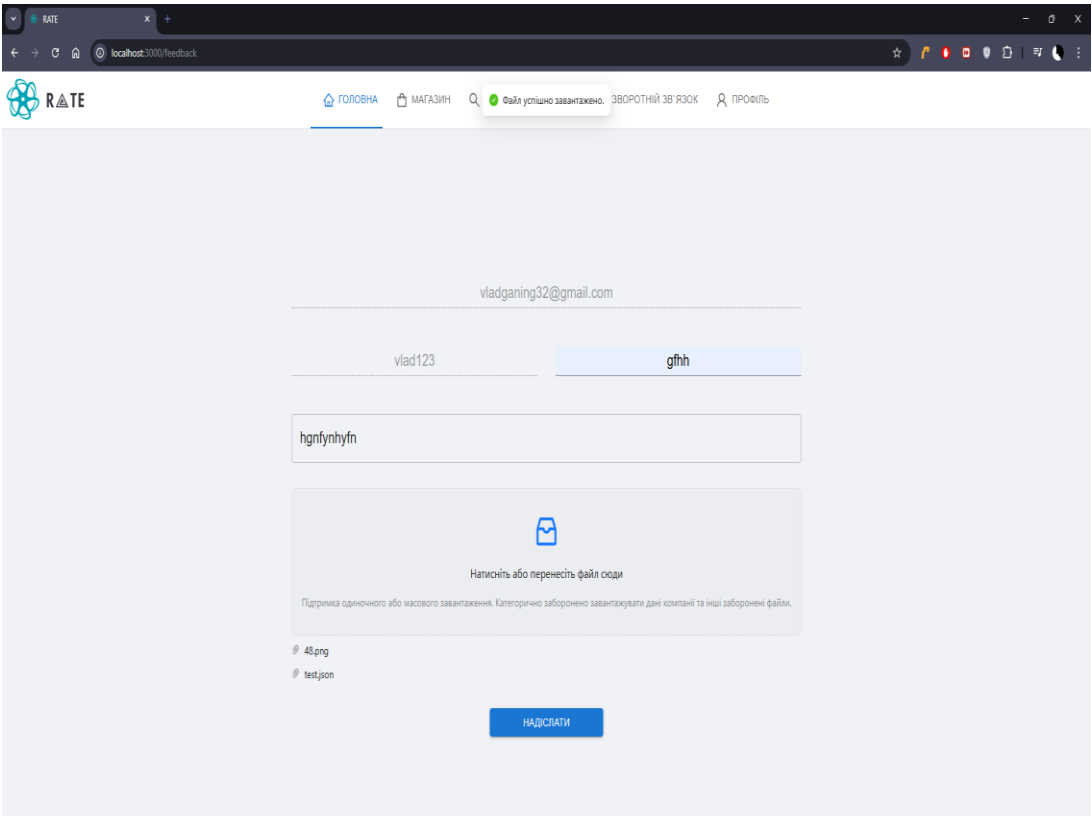


Рис. 3.12 – Інтерфейс сторінки фідбеку

3.3. Тестування роботи програмного продукту

Тестування продукту є важливою частиною розробки веб-застосунку маркетплейсу. Воно дозволяє перевірити правильність реалізації функціональності, такої як додавання товарів до кошика, оплата через Stripe, робота з ролями користувачів тощо. Також тестування допомагає виявити помилки та проблеми ще до релізу, що значно підвищує стабільність та зручність використання системи [5].

Тестування є неперервним процесом, що виконується на різних етапах розробки — починаючи з модульного тестування окремих компонентів (frontend та backend), закінчуючи інтеграційним і системним тестуванням перед запуском проєкту. Воно дозволяє:

Забезпечити високу якість та надійність роботи основних бізнес-функцій (купівля товарів, реєстрація, авторизація, оплата).

Знизити ризики виникнення критичних помилок у роботі застосунку.

Регулярне тестування нових функцій, а також повторне тестування після виправлення багів (регресійне тестування), дозволяє підтримувати веб-застосунок у стабільному та безпечному стані.

Таблиця тест-кейсів

Таблиця 3.1

№	Назва тест-кейсу	Кроки	Очікуваний результат	Роль
1	Додавання товару до кошика	1. Зайти як користувач 2. Натиснути "Додати до кошика" на товарі	Товар з'являється в кошику	User
2	Перегляд кошика	1. Клікнути на іконку кошика в хедері	Відкривається модальне вікно з обраними товарами	User
3	Видалення товару з кошика	1. Відкрити кошик 2. Натиснути "Видалити" біля товару	Товар видаляється з кошика	User
4	Оформлення замовлення	1. Відкрити кошик 2. Натиснути "Оформити замовлення" 3. Заповнити поля доставки 4. Вказати карту	Виводиться повідомлення про успішну оплату	User
5	Перевірка Stripe-оплати	1. Завершити оплату через Stripe	Кошти списані з картки, зараховані на акаунт компанії	User
6	Надсилання зворотного зв'язку	1. Відкрити сторінку "Контакти" 2. Заповнити форму з текстом та прикріпити файл 3. Натиснути "Надіслати"	Повідомлення відправлено на email сайту	User
7	Перегляд товарів глобальним адміністратором	1. Увійти як GlobalAdmin 2. Перейти до панелі адміністрування	Доступ до всіх товарів, користувачів і їх дій	Global Admin
8	Видалення товару адміном	1. Увійти як GlobalAdmin 2. Обрати товар 3. Натиснути "Видалити"	Товар видаляється з каталогу	Global Admin

9	Редагування товару домен-адміном	1. Увійти як DomainAdmin 2. Перейти в профіль 3. Обрати товар 4. Натиснути "Редагувати"	Дані товару оновлюються	Domain Admin
10	Перегляд особистих товарів домен-адміна	1. Увійти як DomainAdmin 2. Перейти в "Мої товари"	Виводиться список власних товарів	Domain Admin
11	Перегляд каталогу товарів користувачем	1. Увійти як User 2. Відкрити головну сторінку	Відображається список товарів	User

Висновки до третього розділу

У даному розділі було наведено послідовність кроків щодо встановлення та налаштування веб-додатку маркетплейсу. Розглянуто як роботу з панеллю адміністратора (включаючи можливості глобального та доменного адміністраторів), так і взаємодію звичайного користувача з основною частиною сайту. Детально проаналізовано ключові елементи інтерфейсу — такі як каталог товарів, кошик, форма оформлення замовлення, профіль користувача, а також можливість зворотного зв'язку через пошту.

Застосунок був протестований з метою виявлення можливих дефектів і підвищення рівня впевненості у якості реалізованого функціоналу. Для цього було створено низку тест-кейсів, що охоплюють основні сценарії користувача: додавання товарів до кошика, оплату через Stripe, обробку ролей, роботу з товарами, реєстрацію, авторизацію тощо. Усі виявлені помилки були своєчасно усунені, після чого проведено повторне тестування для впевненості в їх остаточному виправленні.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи бакалавра було реалізовано веб-додаток маркетплейсу товарів. Проведено аналіз предметної області, а також порівняльний аналіз існуючих аналогічних веб-рішень з метою виявлення їх сильних і слабких сторін. У ході дослідження не було виявлено системи, яка б повністю відповідала поставленим вимогам.

Для реалізації системи було обрано монолітну архітектуру для серверної частини та модульну архітектуру для клієнтської частини, що виявилось оптимальним для даного типу проєкту. Серверна частина реалізована за допомогою Node.js з Express, клієнтська — з використанням Next.js та бібліотек React, Redux Toolkit і Ant Design. Як систему для аутентифікації застосовано NextAuth, а для оплати — Stripe.

Були сформульовані функціональні та нефункціональні вимоги до системи, визначено основні сценарії її використання. Спроектовано ключові компоненти системи, описано взаємозв'язки між ними. Розроблено базу даних на основі MongoDB, у якій зберігаються всі сутності про товари, користувачів, замовлення та ролі.

Було створено чіткий алгоритм взаємодії клієнта з системою: додавання товарів до кошика, оформлення замовлення, заповнення форми доставки, здійснення онлайн-оплати, а також механізм керування товарами для адміністраторів. Передбачено ролі користувачів: звичайний користувач, домен-адміністратор і глобальний адміністратор, кожен з яких має відповідний рівень доступу до функціональності системи.

Інтерфейс додатку є адаптивним і зручним для використання на різних пристроях. Також було описано порядок встановлення та налаштування системи. Система пройшла повноцінне тестування: було створено набір тест-кейсів для перевірки основних сценаріїв взаємодії, виявлені помилки були усунуті.

Реалізований веб-додаток може бути легко масштабований для використання різними компаніями, підтримує багатфункціональну взаємодію з користувачем, продаж товарів та управління асортиментом, що робить його придатним для впровадження у реальні бізнес-процеси.

		Улькін В.В.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Годлевський Ю.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		49

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Next.JS / The React Framework for the Web [Електронний ресурс] – Режим доступу до ресурсу: <https://nextjs.org/>
2. React – A JavaScript library for building user interfaces [Електронний ресурс] – Режим доступу до ресурсу: <https://reactjs.org/>
3. Redux Toolkit / The official, opinionated, batteries-included toolset for efficient Redux development [Електронний ресурс] – Режим доступу до ресурсу: <https://redux-toolkit.js.org/>
4. Ant Design / A design system for enterprise-level products [Електронний ресурс] – Режим доступу до ресурсу: <https://ant.design/>
5. Stripe / Online payment processing for internet businesses [Електронний ресурс] – Режим доступу до ресурсу: <https://stripe.com/>
6. Node.js / JavaScript runtime built on Chrome's V8 JavaScript engine [Електронний ресурс] – Режим доступу до ресурсу: <https://nodejs.org/>
7. Express / Fast, unopinionated, minimalist web framework for Node.js [Електронний ресурс] – Режим доступу до ресурсу: <https://expressjs.com/>
8. MongoDB / Developer data platform [Електронний ресурс] – Режим доступу до ресурсу: <https://www.mongodb.com/what-is-mongodb>
9. Mongoose / Elegant MongoDB object modeling for Node.js [Електронний ресурс] – Режим доступу до ресурсу: <https://mongoosejs.com/>
10. NextAuth.js / Authentication for Next.js [Електронний ресурс] – Режим доступу до ресурсу: <https://next-auth.js.org/>
11. JavaScript | MDN Web Docs [Електронний ресурс] – Режим доступу до ресурсу: <https://developer.mozilla.org/en-US/docs/Web/API>
12. React Icons / Include popular icons in your React projects [Електронний ресурс] – Режим доступу до ресурсу: <https://react-icons.github.io/react-icons/>
13. React Hook Form / Performant, flexible and extensible forms [Електронний ресурс] – Режим доступу до ресурсу: <https://react-hook-form.com/>

		Улькін В.В.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Годлевський Ю.О.				50
Змн.	Арк.	№ докум.	Підпис	Дата		

14. React Query / Powerful asynchronous state management [Електронний ресурс] – Режим доступу до ресурсу: <https://tanstack.com/query>
15. JSON Web Tokens / Introduction [Електронний ресурс] – Режим доступу до ресурсу: <https://jwt.io/>
16. Postman / API platform for building and using APIs [Електронний ресурс] – Режим доступу до ресурсу: <https://www.postman.com/>
17. GitHub / Where the world builds software [Електронний ресурс] – Режим доступу до ресурсу: <https://github.com/>
18. Git / Free and open source distributed version control system [Електронний ресурс] – Режим доступу до ресурсу: <https://git-scm.com/>
19. Prettier / Code formatter [Електронний ресурс] – Режим доступу до ресурсу: <https://prettier.io/>
20. TypeScript / Typed JavaScript at Any Scale [Електронний ресурс] – Режим доступу до ресурсу: <https://www.typescriptlang.org/>
21. HTML | MDN Web Docs / HyperText Markup Language [Електронний ресурс] – Режим доступу до ресурсу: <https://developer.mozilla.org/en-US/docs/Web/Guide/HTML>
22. CSS | MDN Web Docs / Cascading Style Sheets [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Learn/CSS/First_steps
23. MongoDB Compass / The GUI for MongoDB [Електронний ресурс] – Режим доступу до ресурсу: <https://www.mongodb.com/docs/compass/current/>
24. REST API Tutorial / Representational State Transfer [Електронний ресурс] – Режим доступу до ресурсу: <https://restfulapi.net/>
25. CORS / Cross-Origin Resource Sharing [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy
26. dotenv / Loads environment variables from .env [Електронний ресурс] – Режим доступу до ресурсу: <https://www.npmjs.com/package/dotenv>
27. Cookie | MDN Web Docs / HTTP Cookies [Електронний ресурс] – Режим доступу до ресурсу: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie>

28. Vice R., Horton A. Mastering React. – Packt Publishing, 2016. – 260 с.
29. Bank C. Full-Stack Web Development with MongoDB and Node.js. – Packt Publishing, 2022. – 320 с
30. Flanagan D. JavaScript: The Definitive Guide. – O'Reilly Media, 2020. – 706 с.

		Улькін В.В.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Годлевський Ю.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		52

ДОДАТКИ

		Улькін В.В.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Годлевський Ю.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		53

Лістинг GoodsTable:

```
import React, { useState, useContext } from 'react'
import {
  Card,
  CardContent,
  CardActionArea,
  Typography,
  Chip,
  Pagination,
  Box,
  Stack,
  Tooltip,
} from '@mui/material'
import { Grid } from '@mui/system'
import { Image } from 'antd'
import { useGetAllGoodsQuery, useCreateGoodsMutation } from
'@/app/routes/goodsApi'
import FilterContext from '@/app/common/dashboards/Catalog/FilterContext'
import { useRouter } from 'next/navigation'

const GoodsTable = () => {
  const router = useRouter()
  const [currentPage, setCurrentPage] = useState(1)
  const itemsPerPage = 10
  const [createGoods] = useCreateGoodsMutation()
  const { searchValue, categoryValue, priceValue, productType } =
useContext(FilterContext)
  console.log(searchValue)

  const { data, isLoading, isError } = useGetAllGoodsQuery({
    search: searchValue,
    category: categoryValue,
    price: priceValue,
    productType: productType,
  })

  const indexOfLastItem = currentPage * itemsPerPage
  const indexOfFirstItem = indexOfLastItem - itemsPerPage
  const currentItems = data?.slice(indexOfFirstItem, indexOfLastItem) || []

  const handleChangePage = (event, value) => {
```

```

    setCurrentPage(value)
  }

return (
  <Box
    sx={{
      width: '80%',
      px: 4,
      py: 4,
      borderRadius: 4,
    }}
  >
    <Grid container spacing={4} justifyContent="center">
      {currentItems?.map((item) => (
        <Grid item key={item._id}>
          <CardActionArea onClick={() => router.push(`/catalog/${item._id}`)}>
            <Card
              sx={{
                width: 280,
                height: 450,
                borderRadius: 4,
                boxShadow: 6,
                transition: '0.3s',
                '&:hover': {
                  transform: 'scale(1.03)',
                },
              }}
            >
              <CardContent
                sx={{
                  display: 'flex',
                  flexDirection: 'column',
                  alignItems: 'center',
                  height: '100%',
                  gap: 2.5,
                  p: 2,
                }}
              >
                {item.images && item.images.length > 0 ? (
                  <Image
                    src={`http://localhost:3001${item.images[0]}`}
                    alt={item.title}
                    width={200}
                    height={200}
                    preview={false}

```

```

        style={{ borderRadius: '12px', objectFit: 'cover' }}
      />
    ) : (
      <Box
        sx={{
          width: 200,
          height: 200,
          borderRadius: '12px',
          backgroundColor: '#f0f0f0',
          display: 'flex',
          alignItems: 'center',
          justifyContent: 'center',
          color: '#999',
        }}
      >
        No Image
      </Box>
    )}
  <Typography
    variant="h6"
    align="center"
    sx={{ fontWeight: 600, color: '#333' }}
  >
    {item.title}
  </Typography>
  <Chip
    label={item.category}
    color="primary"
    variant="outlined"
    size="small"
  />
  <Stack
    direction="row"
    spacing={1}
    useFlexGap
    flexWrap="wrap"
    justifyContent="center"
  >
    {item.tags.map((tag, index) => (
      <Chip
        key={index}
        label={tag}
        size="small"
        variant="outlined"
        sx={{ bgcolor: '#e8f5e9' }}

```

```

        />
      )))
    </Stack>
    <Stack direction="row" alignItems="center" gap={0.5}>
      <Typography variant="h6" fontWeight="bold" color="#1976d2">
        {item.price}
      </Typography>
      <Typography variant="body2" color="text.secondary">
        {item.currency}
      </Typography>
    </Stack>
  </CardContent>
</Card>
</CardActionArea>
</Grid>
  )))
</Grid>
<Pagination
  count={Math.ceil(data?.length / itemsPerPage)}
  page={currentPage}
  onChange={handleChangePage}
  color="primary"
  sx={{ mt: 4, display: 'flex', justifyContent: 'center' }}
/>
</Box>
)
}

```

export default GoodsTable

Лістинг goods API:

```
import express, { Request, Response } from 'express'
import { connectToMongo } from '@server/db'
import GoodsSchema from '@server/models/Goods'
import multer from 'multer'
import path from 'path'
import { Types } from 'mongoose'

const router = express.Router()
connectToMongo()

const storage = multer.diskStorage({
  destination: (req, file, cb) => {
    cb(null, path.join(__dirname, '../../server/uploads'))
  },
  filename: (req, file, cb) => {
    const uniqueSuffix = Date.now() + '-' + Math.round(Math.random() * 1E9)
    const ext = path.extname(file.originalname)
    cb(null, file.fieldname + '-' + uniqueSuffix + ext)
  }
})

const upload = multer({ storage })

router.get('/', async (req: Request, res: Response) => {
  try {
    const {
      search,
      category,
      price,
      productType,
    } = req.query

    const filter: any = {}

    if (search !== '') {
      filter.title = { $regex: search, $options: 'i' }
    }

    if (category !== 'null' && category !== '') {
      filter.category = category
    }
  }
})
```

```

    }

    if (price != 'null' && price !== '') {
      const parsedPrice = parseFloat(price as string)
      if (!isNaN(parsedPrice)) {
        filter.price = { $lte: parsedPrice }
      }
    }

    if (productType != 'null' && productType !== '') {
      filter.verified = productType === 'true'
    }

    const items = await GoodsSchema.find(filter)

    res.json(items)
  } catch (error) {
    console.error('Error fetching goods:', error)
    res.status(500).json({ success: false, message: 'Server error' })
  }
})

router.get('/userId', async (req: Request, res: Response) => {
  try {
    const { _id = '' } = req.query

    const domainObjectId = new Types.ObjectId(`${_id}`)

    const items = await GoodsSchema.find({ domain: domainObjectId })

    res.json({ success: true, items })
  } catch (error) {
    console.error('Error fetching product by ID:', error)
    res.status(500).json({ success: false, message: 'Server error' })
  }
})

router.get('/:id', async (req: Request, res: Response) => {
  try {
    const { id } = req.params
    const item = await GoodsSchema.findById(id)

    res.json({ success: true, data: item })
  } catch (error) {
    console.error('Error fetching product by ID:', error)
  }
})

```

```

    res.status(500).json({ success: false, message: 'Server error' })
  }
})

router.post('/', upload.array('images'), async (req: Request, res: Response) => {
  try {
    const files = req.files as Express.Multer.File[]

    const { title, userId, category, price, description, tags } = req.body

    const imagesPaths = files.map(f => `/uploads/${f.filename}`)

    const newGood = new GoodsSchema({
      title,
      domain: userId,
      category,
      price: Number(price),
      description,
      tags: JSON.parse(tags),
      images: imagesPaths,
    })

    await newGood.save()

    res.json({ success: true, data: newGood })
  } catch (error) {
    res.status(500).json({ success: false, message: `Server error: ${error}` })
  }
})

router.patch('/', upload.array('images'), async (req: Request, res: Response) => {
  try {
    const files = req.files as Express.Multer.File[]
    const {
      _id,
      title,
      userId,
      category,
      price,
      description,
      tags: rawTags,
    } = req.body

    if (!_id) {
      res.status(400).json({ success: false, message: 'ID is required' })
    }
  }
})

```

```

    }

    const tags = rawTags ? JSON.parse(rawTags) : []

    const updatedFields: any = {
      title,
      userId,
      category,
      price,
      description,
      tags,
    }

    const updatedGood = await GoodsSchema.findByIdAndUpdate(_id,
    updatedFields, { new: true })

    if (!updatedGood) {
      res.status(404).json({ success: false, message: 'Product not found' })
    }

    res.status(200).json({ success: true, data: updatedGood })
  } catch (error) {
    res.status(500).json({ success: false, message: 'Internal server error' })
  }
})

router.delete('/', async (req: Request, res: Response) => {
  try {
    const { _id } = req.body
    const item = await GoodsSchema.findByIdAndDelete(_id)

    res.json({ success: true, data: item })
  } catch (error) {
    res.status(500).json({ success: false, message: error })
  }
})

export default router

```

Лістинг mail API:

```
dotenv.config()

const router = express.Router()
const upload = multer({ storage: multer.memoryStorage() })

router.post('/', upload.array('files'), async (req: Request, res: Response) => {
  const { name, email, theme, desc } = req.body
  const files = req.files as Express.Multer.File[]

  try {
    const transporter = nodemailer.createTransport({
      host: process.env.SMTP_HOST,
      port: Number(process.env.SMTP_PORT),
      secure: true,
      auth: {
        user: process.env.SMTP_USER,
        pass: process.env.SMTP_PASS,
      },
    })

    await transporter.sendMail({
      from: `"${name}" <${email}>`,
      to: process.env.MAIL_TO,
      subject: theme,
      text: desc,
      attachments: files.map(file => ({
        filename: file.originalname,
        content: file.buffer,
      })))
  }

  res.status(200).json({ success: true, message: 'Mail sent successfully' })
} catch (error) {
  res.status(500).json({ success: false, message: 'Fail :( ' })
}
})

export default router
```

Лістинг LoginPage:

```
'use client'

import {
  Button,
  Input,
  Flex,
  Typography,
  Tooltip,
  message,
  Form,
  Tabs
} from 'antd'
import {
  UserOutlined,
  LockOutlined,
  ExclamationCircleOutlined,
  MailOutlined
} from '@ant-design/icons'
import React, { useState } from 'react'
import { signIn } from 'next-auth/react'
import '@ant-design/v5-patch-for-react-19'
import { useCreateUserMutation } from '@app/routes/userApi'

const { Title } = Typography

const LoginPage = () => {
  const [passwordError, setPasswordError] = useState(false)
  const [formLogin] = Form.useForm()
  const [formRegister] = Form.useForm()
  const [ createUser, { isLoading: isCreating, isError } ] = useCreateUserMutation()

  const handleLogin = async (values: any) => {
    const response = await signIn('credentials', {
      email: values.email,
      password: values.password,
      redirect: true,
      callbackUrl: '/',
    })
  }

  if (response?.error) {
    message.error('Невірний логін або пароль')
  } else if (response?.ok) {
```

```
    message.success('Успішний вхід')
  }
}
```

```
const handleRegister = async (values: any) => {
  try {
    const res: any = await createUser({
      email: values.email,
      name: values.name,
      password: values.password,
      role: 'User',
    })

    if(res?.error) {
      message.error(res?.error?.data?.message)
      throw new Error(res?.error?.data?.message)
    }

    await signIn('credentials', {
      email: values.email,
      password: values.password,
      redirect: true,
      callbackUrl: '/',
    })

  } catch (error: any) {
    return
  }
}
```

```
const loginForm = (
  <Form
    form={formLogin}
    name="login"
    onFinish={handleLogin}
    autoComplete="on"
    layout="vertical"
  >
    <Form.Item
      name="email"
      rules={[{ required: true, message: 'Введіть email' }]}
    >
      <Input
        size="large"
        placeholder="Пошта"
```

```

    prefix={<UserOutlined />}
    autoComplete="email"
  />
</Form.Item>

<Form.Item
  name="password"
  rules={[{ required: true, message: 'Введіть пароль' }]}
>
  <Input.Password
    size="large"
    placeholder="Пароль"
    prefix={<LockOutlined />}
    autoComplete="current-password"
    onChange={(e) => {
      setPasswordError(e.target.value.length > 0 && e.target.value.length <= 3)
    }}
    suffix={passwordError &&
      <Tooltip title="Пароль повинен бути більше 3 символів" color="red">
        <ExclamationCircleOutlined />
      </Tooltip>}
  />
</Form.Item>

<Form.Item>
  <Button type="primary" htmlType="submit" size="large" block>
    Увійти
  </Button>
</Form.Item>
</Form>
)

const registerForm = (
  <Form
    form={formRegister}
    name="register"
    onFinish={handleRegister}
    autoComplete="on"
    layout="vertical"
  >
    <Form.Item
      name="name"
      rules={[{ required: true, message: 'Введіть ім'я' }]}
    >
      <Input size="large" placeholder="Ім'я" prefix={<UserOutlined />} />
    </Form.Item>
  </Form>
)

```

```

</Form.Item>

<Form.Item
  name="email"
  rules={[{ required: true, type: 'email', message: 'Введіть дійсний email' }]}
>
  <Input size="large" placeholder="Пошта" prefix={<MailOutlined />} />
</Form.Item>

<Form.Item
  name="password"
  rules={[{ required: true, message: 'Введіть пароль' }]}
>
  <Input.Password
    size="large"
    placeholder="Пароль"
    prefix={<LockOutlined />}
  />
</Form.Item>

<Form.Item>
  <Button type="primary" htmlType="submit" size="large" block>
    Зареєструватися
  </Button>
</Form.Item>
</Form>
)

return (
  <Flex vertical justify="center" align="center" style={{ height: '80vh' }}>
    <Title level={2}>Авторизація</Title>
    <Tabs
      centered
      style={{ width: 400 }}
      items={[
        {
          key: 'login',
          label: 'Вхід',
          children: loginForm,
        },
        {
          key: 'register',
          label: 'Реєстрація',
          children: registerForm,
        },
      ]}
    </Tabs>
  </Flex>
)

```

```
    }  
  />  
</Flex>  
)  
}
```

```
export default LoginPage
```

Лістинг UserProductCard:

```
import React, { useState } from 'react'
import {
  Card,
  Image,
  Button,
  Dropdown,
  Table,
  Popconfirm,
  notification,
  Tag,
  Tooltip
} from 'antd'
import {
  DeleteOutlined,
  AppstoreAddOutlined,
  InfoCircleOutlined
} from '@ant-design/icons'
import {
  useGetProductByUserQuery,
  useDeleteGoodsMutation,
} from '@app/routes/goodsApi'
import { useSession } from 'next-auth/react'
import Link from 'next/link'
import ProductModal from '@app/common/components/ProductModal'
import { useCreateGoodsMutation } from '@app/routes/goodsApi'

export default function UserProductsCard() {
  const [selectedRowKeys, setSelectedRowKeys] = useState([])
  const [messageApi, contextHolder] = notification.useNotification()
  const [isModalOpen, setIsModalOpen] = useState(false)
  const { data: session } = useSession()
  const [ deleteGoods, { isLoading: isDeleting } ] = useDeleteGoodsMutation()
  const [ createGoods, { isLoading: isCreating, } ] = useCreateGoodsMutation()

  const { data, refetch } = useGetProductByUserQuery(
    { _id: session?.user?.id },
    { skip: !session?.user?.id }
  )

  const products = data?.items || []
```

```

const handleDeleteProductFromCart = async () => {
  try {
    await Promise.all(selectedRowKeys.map((_id) =>
deleteGoods({_id}).unwrap()))
    messageApi.success({
      message: 'Товари успішно видалені',
      duration: 2,
    })
    setSelectedRowKeys([])
    refetch()
  } catch (err) {
    messageApi.error({
      message: 'Помилка при видаленні товарів',
      duration: 2,
    })
  }
}

```

```

const handleConfirmModal = async (newProduct) => {
  try {
    const formData = new FormData()

    formData.append('title', newProduct?.title)
    formData.append('userId', session?.user?.id)
    formData.append('category', newProduct?.category)
    formData.append('price', newProduct?.price.toString())
    formData.append('description', newProduct?.description)

    formData.append('tags', JSON.stringify(newProduct?.tags))

    newProduct?.fileList?.forEach(file => {
      formData.append('images', file.originFileObj || file)
    })
    await createGoods(formData)
    messageApi.success({
      message: 'Товар успішно додано',
      duration: 2,
    })
    refetch()
  } catch (err) {
    messageApi.error({
      message: 'Сталася помилка',
      duration: 2,
    })
  }
}

```

```

    setIsModalOpen(false)
  }

const handleCancelModal = () => {
  setIsModalOpen(false)
}

const columns = [
  {
    title: "",
    dataIndex: 'images',
    width: 70,
    render: (image, record) => (
      <Image preview={false} src={`http://localhost:3001${image}`}
alt={record.title} width={50} height={50} />
    ),
  },
  {
    title: 'Назва',
    dataIndex: 'title',
    width: 200,
    render: (text, record) => (
      <Link href={` /catalog/${record._id}`} style={{ color: '#1677ff' }}>
        {text}
      </Link>
    ),
  },
  {
    title: 'Категорія',
    dataIndex: 'category',
    width: 200,
  },
  {
    title: 'Ціна (шт.)',
    dataIndex: 'price',
    width: 100,
    render: (price) => `${price} USD`,
  },
  {
    title: 'Статус товару',
    dataIndex: 'verified',
    width: 180,
    render: (verified) => (
      <div style={{ display: 'flex', alignItems: 'center', gap: 8 }}>
        <Tag color={verified ? 'green' : ''}>

```

```

      {verified ? 'ВЕРИФІКОВАНИЙ' : 'НЕ ВЕРИФІКОВАНИЙ'}
    </Tag>
    <Tooltip
      title={
        verified
          ? 'Верифікований товар — це товар, який підтверджено
сертифікатом та постачається оригінальним постачальником.'
          : 'Неверифікований товар — це товар, постачається місцевим
постачальником.'
      }
    >
      <InfoCircleOutlined style={{ fontSize: '16px', color: '#1677ff', cursor:
'pointer' }} />
    </Tooltip>
  </div>
),
},
]

```

```

const DropMenu = (
  <Dropdown
    menu={{
      items: [
        {
          key: 'create',
          icon: <AppstoreAddOutlined style={{ fontSize: '16px' }} />,
          disabled: isDeleting,
          label: "Додати товар",
          onClick: () => setIsModalOpen(true))
        },
        {
          key: 'delete',
          danger: true,
          icon: <DeleteOutlined style={{ fontSize: '16px' }} />,
          disabled: isDeleting || selectedRowKeys.length === 0,
          label: (
            <Popconfirm
              title="Видалити обрані товари?"
              onConfirm={handleDeleteProductFromCart}
              okText="Так"
              cancelText="Ні"
            >
              Видалити
            </Popconfirm>
          ),

```

```

        },
      ],
    }}
  >
  <Button>•••</Button>
</Dropdown>
)

return (
  <Card
    title="МОЇ ТОВАРИ"
    hoverable
    extra={DropMenu}
  >
    {contextHolder}
    {session?.user?.role !== 'User' ? (
      <>
        <Table
          rowKey={(record) => record._id}
          columns={columns}
          dataSource={products}
          pagination={{ pageSize: 5 }}
          rowSelection={{
            selectedRowKeys,
            onChange: setSelectedRowKeys,
          }}
          locale={{ emptyText: 'Поки немає власних товарів' }}
        />
        <ProductModal
          create={true}
          visible={isModalOpen}
          onSave={handleConfirmModal}
          onCancel={handleCancelModal}
          product={[]}
        />
      </>
    ) : (
      <div style={{ padding: '1rem', fontSize: '16px' }}>
        Для цієї функції необхідно мати статус продавця
      </div>
    )}
  </Card>
)
}

```