

БАКАЛАВРСЬКА РОБОТА
БР.ІП-92.00.00.000 ПЗ

Група ІП-22-4

Вірстюк Назар

2026

Івано-Франківський національний технічний університет нафти і газу

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Вірстюк Назар Романович

(прізвище, ім'я, по батькові)

УДК 004.42
(індекс)

БАКАЛАВРСЬКА РОБОТА

**«Розроблення масштабованого веб-застосунку електронної комерції та прокату
гірського спорядження з електронним формуванням договорів»**
(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121– Інженерія програмного забезпечення

(шифр і назва спеціальності)

Робота містить результати власних досліджень, використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело:

Здобувач освітнього ступеня Вірстюк Н.Р.
(підпис, ініціали та прізвище здобувача)

Науковий керівник Дмитрик Тарас Богданович, асистент
(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту
Завідувач кафедри

доц. Бандура В.В.
(посада) (підпис) (дата) (ініціали та прізвище)

м. Івано-Франківськ - 2026 р.

Івано-Франківський національний технічний університет нафти і газу
Інститут, факультет інформаційних технологій
Кафедра інженерії програмного забезпечення
Ступінь вищої освіти бакалавр
Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

Зав. кафедрою ІІЗ

доц. В.В. Бандура

“ ” 2026 р.

ЗАВДАННЯ

НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТОВІ

Вірстюку Назару Романовичу

(прізвище, ім'я, по-батькові)

1. Тема проекту (роботи) «Розроблення масштабованого веб-застосунку електронної комерції та прокату гірського спорядження з електронним формуванням договорів»

керівник проекту (роботи) асист. Дмитрик Тарас Богданович

затверджені наказом закладу вищої освіти від “21” квітня 2026 р. № 179/7

2. Строк подання студентом проекту (роботи) 2026 р.

3. Вихідні дані до проекту (роботи) програмний код Full-Stack застосунку Marmaros, локальне Docker-середовище, база даних каталогу, результати функціонального тестування, конфігурація розгортання для домену marmaros.store.

4. Зміст розрахунково - пояснювальної записки (перелік питань, які потрібно розробити)

1. Аналіз предметної області продажу та оренди туристичного спорядження

2. Формування функціональних і нефункціональних вимог до веб-застосунку Marmaros

3. Проєктування архітектури, бази даних, API та сценаріїв взаємодії

4. Реалізація клієнтської, серверної та адміністративної підсистем

5. Тестування, оцінка результатів і опис розгортання через Coolify та домен marmaros.store

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

1. UML-діаграма випадків використання Marmaros (рис. 2.1, ст. 34)

2. Архітектура Full-Stack системи Marmaros (рис. 3.1, ст. 43)

3. Activity-схема бізнес-процесу замовлення (рис. 3.2, ст. 45)

4. Узагальнена ER-модель бази даних Marmaros (рис. 3.5, ст. 48)

5. Схема впровадження Marmaros на marmaros.store (рис. 5.2, ст. 68)

1. **Консультанти розділів проекту (роботи)**

<i>Розділ</i>	<i>Консультант</i>	<i>Підпис, дата</i>	
		<i>Завдання видав</i>	<i>Завдання прийняв</i>

2. **Дата видачі завдання** _____

Керівник

(підпис)

Дмитрик Т. Б.

(розшифровка підпису)

Завдання прийняв до виконання

(підпис)

Вірстюк Н. Р.

(розшифровка підпису)

КАЛЕНДАРНИЙ ПЛАН

<i>№ з/п</i>	<i>Назва етапів дипломного проекту (роботи)</i>	<i>Строк виконання етапів проекту</i>	<i>Примітка</i>
1	Опрацювання теоретичних матеріалів з теми роботи	27.01.2026	виконано
2	Вибір інструментів, технологій та методів розробки	05.03.2026	виконано
3	Налаштування середовища та конфігурація бази даних	23.03.2026	виконано
4	Розробка компонентів та функціональних модулів системи	13.04.2026	виконано
5	Тестування системи та усунення виявлених помилок	23.04.2026	виконано
6	Підготовка та оформлення бакалаврської роботи	09.06.2026	виконано

Студент – дипломник

(підпис)

Вірстюк Н. Р.

(розшифровка підпису)

Керівник роботи

(підпис)

Дмитрик Т. Б.

(розшифровка підпису)

АНОТАЦІЯ

Бакалаврська робота містить 77 сторінки, 19 рисунків, список використаних джерел із 35 найменувань, 2 додатки.

Метою роботи є розробка Full-Stack веб-застосунку Marmaros для продажу та оренди туристичного спорядження, що забезпечує каталог, купівлю й оренду товарів, доставку, оплату, генерацію PDF-договорів.

Об'єкт дослідження — процес організації продажу та оренди туристичного спорядження через веб-технології.

Предмет дослідження — методи і інструменти проєктування клієнт-серверного веб-застосунку з REST API, реляційною базою, автентифікацією, рольовим доступом та платіжною інтеграцією.

Результати дослідження: створено прототип Marmaros з React, Vite, FastAPI, SQLAlchemy, PostgreSQL, Docker Compose, Coolify та Cloudflare Tunnel, який реалізує каталог, купівлю та оренду, JWT-автентифікацію, email-коди, інтеграцію LiqPay, доставку, PDF-договори, новини, маршрути.

У першому розділі проведено аналіз предметної області.

У другому розділі визначено функціональні та нефункціональні вимоги до Marmaros, описано зацікавлені сторони та постановку задачі.

У третьому розділі спроектовано архітектуру системи, UML-моделі, ER-модель бази даних та REST API і обґрунтовано вибір технологій.

У четвертому розділі описано реалізацію frontend.

У п'ятому розділі проведено системне тестування, оцінку backend-тестів і frontend-build, а також впровадження системи.

Висновки підтверджують, що веб-застосунок Marmaros забезпечує основний функціонал спеціалізованого outdoor-сервісу та готовий до подальшого розвитку і масштабування.

КЛЮЧОВІ СЛОВА: ВЕБ-ЗАСТОСУНОК, ЕЛЕКТРОННА КОМЕРЦІЯ, ОРЕНДА, ТУРИСТИЧНЕ СПОРЯДЖЕННЯ, REACT, VITE, FASTAPI, POSTGRESQL, REST API, DOCKER, JWT, LIQPAY.

ABSTRACT

The bachelor's thesis contains 77 pages, 35 figures, a list of 22 references, and 2 appendices.

The aim of the work is to develop the Marmaros Full-Stack web application for the sale and rental of tourist equipment, providing catalog browsing, purchase and rental, delivery, payment, and PDF rental agreement.

The object of the research is the process of organizing the sale and rental of tourist equipment using web technologies.

The subject of the research is the methods and tools for designing a client-server web application with REST API, relational database, authentication, role-based access control, and payment integration.

Research results: a Marmaros prototype was created using React, Vite, FastAPI, SQLAlchemy, PostgreSQL, Docker Compose, Coolify, and Cloudflare Tunnel, implementing a catalog, purchase and rental processes, JWT authentication, email codes, LiqPay integration, delivery, PDF agreements.

Chapter 1 presents the analysis of the subject area.

Chapter 2 defines functional and non-functional requirements for Marmaros, describes stakeholders, and sets the design task.

Chapter 3 presents the system architecture, UML diagrams, ER database model, REST API, and justifies the choice of technologies.

Chapter 4 describes the frontend implementation.

Chapter 5 presents system testing, backend and frontend-build evaluation, and system deployment.

The conclusions confirm that the Marmaros web application provides the main functionality of a specialized outdoor service and is ready for further development and scaling.

KEYWORDS: WEB APPLICATION, E-COMMERCE, RENTAL, TOURIST EQUIPMENT, REACT, VITE, FASTAPI, POSTGRESQL, REST API, DOCKER, JWT, LIQPAY.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ	8
ВСТУП.....	9
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ, ІСНУЮЧИХ РІШЕНЬ ТА ТЕХНОЛОГІЧНИХ ТЕНДЕНЦІЙ.....	12
1.1 Характеристика предметної області outdoor e-commerce	12
1.2 Основні поняття та терміни розробки веб-застосунків електронної комерції	15
1.3 Аналіз існуючих рішень для продажу та оренди туристичного спорядження	18
1.4 Сучасні технологічні тенденції у розробці веб-систем.....	20
1.5 Висновки по розділу	23
РОЗДІЛ 2. АНАЛІЗ ВИМОГ ТА ПОСТАНОВКА ЗАДАЧІ ПРОЄКТУВАННЯ	24
2.1 Збір та аналіз вимог зацікавлених сторін	24
2.2 Функціональні вимоги до веб-застосунку Marmaros	26
2.3 Нефункціональні вимоги та критерії приймання	29
2.4 Постановка задачі проєктування системи	31
2.5 Висновки по розділу	36
РОЗДІЛ 3. ПРОЄКТУВАННЯ АРХІТЕКТУРИ, МОДЕЛІ ДАНИХ ТА API	39
3.1 Обґрунтування архітектурної моделі системи.....	39
3.2 Моделювання бізнес-процесів і користувацьких сценаріїв.....	41
3.3 Проєктування бази даних та REST API	43
3.4 Вибір і обґрунтування технологічного стеку	45
3.5 Висновки по розділу	48

					БР.ІП – 92.00.00.000 ПЗ			
<i>Змн.</i>	<i>Арк.</i>	<i>№ докум.</i>	<i>Підпис</i>	<i>Дата</i>	Розроблення масштабованого веб-застосунку електронної комерції та прокату гірського спорядження з електронним формуванням договорів Пояснювальна записка	<i>Літ.</i>	<i>Арк.</i>	<i>Акрушів</i>
<i>Розроб.</i>		<i>Вірстюк Н.Р.</i>					7	
<i>Перевір.</i>		<i>Дмитрик Т.Б.</i>				ІФНТУНГ ІП-22-4		
<i>Реценз.</i>		<i>Лютак І.З.</i>						
<i>Н. Контр.</i>		<i>Піх М.М.</i>						
<i>Затверд.</i>		<i>Бандура В. В.</i>						

РОЗДІЛ 4. РЕАЛІЗАЦІЯ ВЕБ-ЗАСТОСУНКУ MARMAROS	51
4.1 Реалізація клієнтської частини	51
4.2 Реалізація серверної та адміністративної підсистем	54
4.3 Організація програмного коду, алгоритми та структури даних.....	60
4.4 Висновки по розділу	63
РОЗДІЛ 5. СИСТЕМНЕ ТЕСТУВАННЯ, ВПРОВАДЖЕННЯ ТА	
ОЦІНКА РЕЗУЛЬТАТІВ.....	64
5.1 Стратегія та методика тестування системи	64
5.2 Результати функціонального, інтеграційного та backend-тестування...	65
5.3 Production-впровадження через Docker, Coolify і Cloudflare Tunnel	67
5.4 Аналіз безпеки, ризиків і надійності системи	69
5.5 Висновки по розділу	70
ВИСНОВКИ	72
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	74
ДОДАТКИ	
БІБЛІОГРАФІЧНА ДОВІДКА	

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

API — Application Programming Interface, інтерфейс прикладного програмування для взаємодії між компонентами системи.

CI/CD — Continuous Integration / Continuous Delivery, автоматизована інтеграція та доставка змін.

CRUD — Create, Read, Update, Delete, базові операції з даними.

DTO — Data Transfer Object, структура передавання даних між шарами системи.

ER — Entity Relationship, модель сутностей і зв'язків бази даних.

JWT — JSON Web Token, формат токена для автентифікації та авторизації.

ORM — Object-Relational Mapping, відображення об'єктів програми на таблиці бази даних.

REST — архітектурний стиль побудови веб-API.

SPA — Single Page Application, односторінковий веб-застосунок.

TTN — товарно-транспортна накладна для відстеження доставки.

UI — User Interface, користувацький інтерфейс.

Frontend — клієнтська частина веб-застосунку, що відповідає за інтерфейс і взаємодію з користувачем.

Backend — серверна частина веб-застосунку, що відповідає за бізнес-логіку, обробку запитів, дані та інтеграції.

JSON — текстовий формат обміну структурованими даними між клієнтом і сервером.

React — JavaScript-бібліотека для побудови користувацьких інтерфейсів.

Node.js — середовище виконання JavaScript на сервері.

					БР.ІП – 92.00.00.000 ПЗ	Арк.
						8
Змн.	Арк.	№ докум.	Підпис	Дата		

ВСТУП

Сучасна електронна комерція у сфері туристичного спорядження потребує не лише каталогу товарів, а й сервісної логіки навколо підготовки до маршруту. Користувачу важливо швидко підібрати спорядження за категорією, ціною, рівнем підготовки й форматом використання, а також мати можливість обрати між купівлею та прокатом. Для українського outdoor-сегмента така модель є особливо актуальною, оскільки спорядження для Карпат часто використовується сезонно, а частина дорогих позицій є раціональнішою для оренди, ніж для одноразової купівлі.

Актуальність теми зумовлена потребою у спеціалізованій платформі, яка поєднує електронну комерцію, прокат спорядження, рекомендаційний контент і адміністративний облік. Універсальні інтернет-магазини зазвичай добре розв'язують задачу продажу, однак слабше підтримують rental-сценарії, договори оренди, повернення спорядження, доставку і контент для планування походів.

Методичні рекомендації до бакалаврських робіт передбачають наявність анотації, вступу, основної частини, висновків, списку джерел і додатків, а також вимагають дотримання стандартів оформлення науково-технічних звітів [1]. Оформлення пояснювальної записки виконано з урахуванням вимог ДСТУ 3008:2015 [2], а бібліографічні записи сформовано відповідно до правил ДСТУ 8302:2015 [3].

Об'єктом дослідження є процес організації продажу та оренди туристичного спорядження засобами веб-технологій.

Предметом дослідження є методи, моделі та інструменти проєктування і реалізації клієнт-серверного веб-застосунку електронної комерції з підтримкою REST API, реляційної бази даних, автентифікації, рольового доступу, платіжної інтеграції та контейнеризованого розгортання.

Метою роботи є розроблення працездатного веб-застосунку Marmaros,

					БР.ІП – 92.00.00.000 ПЗ	Арк.
						9
Змн.	Арк.	№ докум.	Підпис	Дата		

який забезпечує каталог туристичного спорядження, купівлю, оренду, автентифікацію користувачів, оформлення замовлень, інтеграцію платежів, доставку, генерацію договору оренди, особистий кабінет і адміністративне керування.

Для досягнення мети необхідно виконати такі завдання: проаналізувати стан предметної області та існуючі рішення; визначити базові поняття й технологічні тенденції; зібрати та класифікувати вимоги; сформулювати постановку задачі; спроектувати архітектуру, модель даних, API і UML-моделі; реалізувати клієнтську, серверну й адміністративну підсистеми; виконати модульне та системне тестування; описати впровадження на домені marmaros.store.

Методи дослідження включають системний аналіз предметної області, порівняльний аналіз аналогів, аналіз документів і сценаріїв використання, об'єктно-орієнтоване проектування, UML-моделювання, ER-моделювання, прототипування інтерфейсу, модульне та інтеграційне тестування API.

Елементи новизни одержаних результатів полягають у поєднанні в межах одного веб-застосунку класичного e-commerce функціоналу та rental-сценаріїв, що передбачають вибір терміну оренди, формування окремого типу замовлення, генерацію PDF-договору, збереження історії користувача й адміністративне керування замовленнями.

Практичне значення розробки полягає у створенні програмної основи для запуску спеціалізованого сервісу продажу та оренди туристичного спорядження. Розроблена архітектура дозволяє розширювати каталог, додавати нові категорії товарів, підключати додаткові платіжні й логістичні сервіси, удосконалювати адміністративну аналітику та масштабувати систему для подальшого production-використання.

Обмеженням поточної реалізації є те, що частина зовнішніх інтеграцій потребує перевірки з реальними production-ключами платіжного провайдера та служб доставки. Також подальшого розвитку потребують e2e-тестування,

					БР.ІП – 92.00.00.000 ПЗ	Арк.
						10
Змн.	Арк.	№ докум.	Підпис	Дата		

розширені фільтри каталогу, автоматизована аналітика продажів і моніторинг production-середовища.

Робота складається зі вступу, п'яти розділів, висновків, списку використаних джерел і додатків. Перший розділ присвячено аналізу предметної області, понять, існуючих рішень і технологічних тенденцій. Другий розділ містить аналіз вимог і постановку задачі. У третьому розділі наведено архітектуру, UML-моделі, ER-модель, REST API і вибір технологій. Четвертий розділ описує практичну реалізацію, структуру коду, алгоритми та модульне тестування. П'ятий розділ присвячено системному тестуванню, впровадженню, безпеці та оцінці результатів.

					БР.ІП – 92.00.00.000 ПЗ	Арк.
						11
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ, ІСНУЮЧИХ РІШЕНЬ ТА ТЕХНОЛОГІЧНИХ ТЕНДЕНЦІЙ

1.1. Характеристика предметної області outdoor e-commerce

Предметна область роботи охоплює електронну комерцію у сфері туристичного, кемпінгового та гірського спорядження. На відміну від універсального інтернет-магазину, такий сервіс працює з товарами, вибір яких залежить від сезону, маршруту, погодних умов, рівня підготовки користувача і тривалості подорожі. Для покупця туристичного спорядження важливими є не лише ціна і зовнішній вигляд товару, а й вага, об'єм, температурний режим, водостійкість, матеріали, сумісність з іншим спорядженням і доступність сервісної підтримки.

Outdoor e-commerce має поєднувати інформаційну, торговельну та сервісну функції. Інформаційна функція допомагає користувачу зрозуміти характеристики спорядження; торговельна забезпечує купівлю; сервісна підтримує оренду, доставку, повернення, консультації та адміністративний облік. Для Marmaros важливо, що оренда не є другорядною опцією, а окремим бізнес-процесом зі своїми правилами, строками, документами та ризиками.

Туристичне спорядження має виражену сезонність попиту. Перед літніми походами зростає інтерес до наметів, спальників, пальників, рюкзаків і трекінгового взуття; взимку зростає роль теплої одягу, термосів, ліхтарів і засобів безпеки. Це впливає на структуру каталогу, вимоги до фільтрів, контенту, рекомендацій і швидкого оновлення залишків. Система повинна бути готовою до зміни асортименту без втручання розробника в код.

Користувачів outdoor-платформи можна поділити на кілька груп. Перша група - новачки, які не завжди розуміють різницю між типами наметів, спальників або рюкзаків. Друга група - досвідчені туристи, які порівнюють характеристики й очікують точних даних. Третя група - клієнти, які купують базове спорядження для регулярного використання. Четверта група -

					БР.ІП – 92.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		12

користувачі, яким потрібна оренда дорогого або рідко використовуваного спорядження на конкретний маршрут.

Для новачків система повинна зменшувати когнітивне навантаження: показувати зрозумілі категорії, короткі пояснення, фотографії, базові характеристики, рекомендаційний контент і простий checkout. Для досвідчених користувачів важлива точність: вони очікують фільтрів, специфікацій, ваги, ціни оренди за день і швидкого переходу до оформлення. Таким чином, один і той самий інтерфейс має підтримувати різний рівень підготовки аудиторії.

Rental-сценарій суттєво ускладнює предметну область. У звичайному продажі після оплати й доставки товар переходить у власність покупця. В оренді потрібно враховувати строк користування, дату повернення, стан спорядження, відповідальність користувача, договір, заставу або інші умови. Тому в Marmaros оренда моделюється як окремий тип замовлення, а не як звичайний товар зі зниженою ціною.

Ще одна особливість полягає в тому, що туристичне спорядження часто купують або орендують не ізольовано, а як комплект під маршрут. Наприклад, для походу в Карпати користувачу можуть бути потрібні рюкзак, спальник, килимок, пальник, дощовик, ліхтар і аптечка. Це створює простір для майбутніх рекомендацій, але вже на поточному етапі вимагає від каталогу чіткої категоризації та контентного супроводу.

Науково-технічна література з інженерії програмного забезпечення розглядає такі системи як приклад прикладного веб-ПЗ, у якому якість залежить від коректного аналізу вимог, архітектури, модульності та перевірки [16], [17]. Вітчизняні методичні рекомендації наголошують на необхідності поєднання теоретичного аналізу з практичним результатом і належним оформленням доказової бази [1].

У межах роботи Marmaros розглядається не як абстрактна навчальна система, а як прототип прикладного сервісу, який розгорнутий на реальному

					БР.ІП – 92.00.00.000 ПЗ	Арк.
						13
Змн.	Арк.	№ докум.	Підпис	Дата		

домені marmaros.store. Це означає, що під час аналізу предметної області враховуються не лише екрани користувача, а й deployment, адміністрування, секрети середовища, платежі, доставка, тестування і подальший супровід.

У локальній базі Marmaros підготовлено каталог з 255 позицій у 8 категоріях. Категорії охоплюють намети, рюкзаки, спальні системи, одяг, взуття, кемпінг і кухню, аксесуари та електроніку. Для товарів зберігаються назва, slug, опис, ціна купівлі, ціна оренди, зображення, характеристики у JSON-форматі та зв'язок із категорією. Дані особливості описані в таблиці 1.1.

Таблиця 1.1

Особливості предметної області Marmaros

Аспект	Зміст	Вплив на систему
Сезонність	попит залежить від погоди, маршрутів і періоду року	гнучке оновлення каталогу, контент, категорії
Технічність товарів	товари мають вагу, об'єм, температурні режими, матеріали	потрібні характеристики, фільтри, опис
Оренда	частина спорядження потрібна тимчасово	rent_price, rent_days, rental_order, договір
Комплектність	користувач часто збирає набір під маршрут	категорії, рекомендації, маршрути, контент
Доставка	товар має прибути до початку подорожі	перевізники, pickup, місто, відділення, TTN
Адміністрування	каталог і замовлення змінюються щоденно	admin-панель, статуси, ролі
Production	сервіс має бути доступним через домен	Docker, Coolify, Cloudflare Tunnel

Важливим для предметної області є баланс між гнучкістю й контрольованістю даних. Якщо створювати окрему таблицю характеристик для кожної категорії, система стане складною для супроводу. Якщо ж зберігати всі характеристики лише текстом, буде неможливо робити фільтри й порівняння. У Marmaros використано компроміс: базові реляційні поля

зберігають критичні дані, а JSON-характеристики дають змогу описувати різні типи спорядження.

Отже, предметна область Marmaros поєднує електронну комерцію, rental-бізнес, контентну підтримку користувача та операційне адміністрування. Саме ця комбінація формує складність задачі й пояснює, чому розробка не може обмежитися типовою сторінкою каталогу або простою формою замовлення.

1.2. Основні поняття та терміни розробки веб-застосунків електронної комерції

Для коректного розуміння проєкту необхідно визначити базові поняття, які використовуються в роботі. Веб-застосунок - це програмна система, з якою користувач взаємодіє через браузер, а основна бізнес-логіка виконується на сервері або розподіляється між клієнтською і серверною частинами. У Marmaros клієнтська частина реалізована як SPA, а backend надає REST API для каталогу, кошика, замовлень, платежів, доставки, контенту та адміністрування.

Електронна комерція розглядається як сукупність процесів представлення товарів, взаємодії з клієнтом, оформлення замовлення, приймання платежу, організації доставки й післяпродажної підтримки. У випадку Marmaros до цієї моделі додається rental-компонент, тому e-commerce процес розширюється до тимчасового користування товаром, формування договору й контролю повернення.

Архітектурний патерн - це типовий спосіб організації компонентів системи та їхніх взаємодій. Для веб-систем поширеними є клієнт-серверна архітектура, layered architecture, MVC/MVVM, microservices і modular monolith. Практика проєктування корпоративних застосунків показує, що патерни мають обиратися не формально, а відповідно до вимог підтримки,

					БР.ІП – 92.00.00.000 ПЗ	Арк.
						15
Змн.	Арк.	№ докум.	Підпис	Дата		

масштабування і складності бізнес-логіки [20].

Клієнт-серверна архітектура передбачає, що клієнтська частина відповідає за представлення даних і взаємодію з користувачем, а серверна частина - за бізнес-правила, доступ до бази даних, авторизацію, інтеграції та обробку помилок. Такий підхід добре відповідає Marmaros, оскільки checkout, рольовий доступ, LiqPay callback і створення rental-договору не можна довіряти лише frontend-у.

Компонент системи - це логічно відокремлена частина, яка виконує конкретну відповідальність. У Marmaros такими компонентами є frontend, backend API, база даних PostgreSQL, модулі автентифікації, каталогу, кошика, замовлень, платежів, доставки, контенту та адміністративної панелі. Компонентність допомагає локалізувати зміни: наприклад, оновлення delivery API не повинне вимагати переписування каталогу.

Життєвий цикл програмного забезпечення охоплює аналіз, вимоги, проєктування, реалізацію, тестування, впровадження та супровід. За підходами інженерії ПЗ якісна система не може бути отримана лише написанням коду: необхідно документувати вимоги, формувати архітектурні рішення, перевіряти реалізацію й підтримувати зміни [16], [18].

REST API - це спосіб організації взаємодії між клієнтом і сервером через ресурси та HTTP-операції. Для Marmaros ресурсами є products, categories, cart, orders, payments, delivery, news, routes, reviews і users. REST-підхід зручний тим, що відокремлює frontend від backend і дозволяє в майбутньому використати той самий API для мобільного застосунку або зовнішньої інтеграції.

DTO або schema у FastAPI/Pydantic використовується для опису структури даних, які передаються між клієнтом і сервером. Це важливо для перевірки checkout payload, профілю користувача, order item, delivery data та payment callback. Типізовані схеми зменшують кількість помилок, пов'язаних із неправильним форматом запиту [6].

ORM - це технологія відображення об'єктів програми на таблиці реляційної бази даних. SQLAlchemy дає змогу працювати з моделями Python, зв'язками, транзакціями і запитам до PostgreSQL [7]. Для Marmaros ORM важлива, бо система має багато пов'язаних сутностей: users, orders, order_items, products, categories, reviews і cart_items.

У Marmaros JWT використовується як основний механізм автентифікації користувачів після входу в систему. Після успішного логіну або підтвердження email-коду backend формує токен, у якому зберігаються службові дані користувача, зокрема його email та роль. Надалі frontend передає цей токен у заголовок Authorization під час звернення до захищених API-ендпоінтів. На стороні backend токен перевіряється перед виконанням запиту: система контролює його коректність, термін дії та роль користувача. Завдяки цьому звичайний клієнт має доступ лише до особистого кабінету й власних замовлень, а адміністративні функції доступні тільки користувачам з роллю admin.

Нефункціональні характеристики системи описують не окремі функції, а якість їх виконання: продуктивність, безпеку, супроводжуваність, зручність використання, надійність і переносимість. Ці характеристики узгоджуються з моделлю якості програмних продуктів ISO/IEC 25010 [19].

Контейнеризація - це спосіб пакування застосунку та його залежностей у відтворюване середовище. Docker Compose дозволяє описати кілька сервісів як єдину систему [9]. Для Marmaros це frontend, backend, PostgreSQL, mailpit і cloudflared. Такий підхід зменшує різницю між локальною розробкою та production-розгортанням.

CI/CD - це практика автоматизації перевірки, збірки та доставки змін. У дипломній реалізації повноцінний CI/CD pipeline ще не є завершеним, але архітектура підготовлена до нього: є backend tests, frontend build, Docker Compose і deployment через Coolify. Це створює основу для подальшої автоматизації релізів.

					БР.ІП – 92.00.00.000 ПЗ	Арк.
						17
Змн.	Арк.	№ докум.	Підпис	Дата		

Таким чином визначено основні поняття, які потрібні для розуміння роботи Marmaros як повноцінного Full-Stack веб-застосунку. Система складається не лише з візуальної частини сайту, а й із серверної логіки, бази даних, API, механізмів автентифікації, оплати, доставки, тестування та розгортання.

У Marmaros frontend відповідає за інтерфейс користувача: каталог, картку товару, кошик, кабінет і адміністративні сторінки. Backend обробляє бізнес-логіку: створення замовлень, перевірку користувачів, роботу з платежами, доставкою, email-повідомленнями та PDF-договорами. REST API забезпечує обмін даними між клієнтською та серверною частиною, JWT використовується для захищеного доступу до кабінету й адміністративних функцій, а ORM забезпечує роботу Python-моделей із таблицями PostgreSQL.

1.3. Аналіз існуючих рішень для продажу та оренди туристичного спорядження

Існуючі рішення у вибраному класі задач можна умовно поділити на три групи: універсальні маркетплейси, спеціалізовані outdoor-магазини та сервіси локального прокату. Універсальні маркетплейси мають широкий каталог і сильну платіжно-логістичну інфраструктуру, однак не орієнтовані на пояснення туристичних характеристик і договори оренди.

Маркетплейси добре розв'язують задачу пошуку товару за ціною, брендом і наявністю. Вони мають звичний checkout, доставку, рейтинги та оплату. Проте їхня модель не враховує логіку прокату: строк оренди, дату повернення, стан спорядження, PDF-договір, відповідальність користувача, повторну видачу товару іншому клієнту. Через це маркетплейс не є повноцінним рішенням для сервісу туристичного прокату.

Спеціалізовані outdoor-магазини краще працюють із предметною областю: мають категорії спорядження, фільтри, статті та рекомендації. Вони

					БР.ІП – 92.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		18

можуть пояснювати, який спальник підходить для певної температури, який рюкзак зручний для кількаденного походу, або яке взуття доцільне для гірської місцевості. Проте часто такі магазини реалізують лише продаж, а не rental-процеси.

Сервіси локального прокату, навпаки, мають логіку тимчасового користування, але нерідко працюють через форми заявок, месенджери або телефонне підтвердження. Така модель може бути прийнятною для маленької точки прокату, однак вона погано масштабується: складно вести історію, автоматично формувати документи, збирати статистику, підтримувати онлайн-оплату і швидко оновлювати каталог.

Невирішеним або частково вирішеним аспектом є поєднання купівлі й оренди в одному checkout без втрати прозорості для користувача. Якщо в кошику одночасно є позиції купівлі та прокату, система повинна коректно розділяти бізнес-операції, формувати різні типи замовлень і не змішувати договори оренди з купленими товарами.

Іншим частково вирішеним аспектом є документування rental-відносин. У багатьох прокатних сервісах договір формується вручну або поза системою. Це збільшує ризик помилки в даних клієнта, строку оренди або переліку спорядження. У Marmaros PDF-договір розглядається як частина системного сценарію, а не як зовнішній офісний документ.

Ще одним слабким місцем аналогів є операційне адміністрування. Для реального магазину важливо не лише прийняти заявку, а й бачити статус оплати, доставки, TTN, клієнта, суму, rental-документ і контент. Marmaros враховує цю потребу через admin dashboard, AdminOrders, AdminProducts, AdminNews і модерацію відгуків.

З точки зору користувацького досвіду аналогії часто розділяють інформаційний контент і комерційний сценарій. Наприклад, статті про маршрути або спорядження можуть бути окремим блогом, а каталог - окремим магазином. У Marmaros ці частини об'єднуються: користувач може читати

					БР.ІП – 92.00.00.000 ПЗ	Арк.
						19
Змн.	Арк.	№ докум.	Підпис	Дата		

новини й маршрути в межах того самого сервісу, де він обирає спорядження.

З точки зору технологічної архітектури універсальні платформи часто закриті для глибокої адаптації rental-процесу. Якщо використовувати готовий конструктор магазину, можна швидко отримати каталог і оплату, але складніше реалізувати специфічну логіку розділення замовлень, PDF-договору, ролей і майбутніх інтеграцій. Власна Full-Stack реалізація дає більше контролю над бізнес-правилами.

Порівняння аналогів показує, що Marmaros займає проміжну позицію між інтернет-магазином, прокатним сервісом і інформаційною outdoor-платформою. Саме ця комбінація пояснює доцільність розробки окремого застосунку, а не використання лише готового шаблону e-commerce.

Критичний огляд існуючих рішень також показує, що для дипломного проєкту важливо продемонструвати не тільки UI, а й backend-логіку. Якщо система має rental-документи, платежі, доставку, ролі й admin-панель, то основна складність переноситься на проєктування даних, API, тестування і deployment.

Отже, простір для подальшої розробки полягає у створенні системи, яка має достатньо простий користувацький шлях для клієнта, але водночас підтримує складні внутрішні бізнес-процеси магазину. Marmaros реалізує саме такий підхід.

1.4. Сучасні технологічні тенденції у розробці веб-систем

Серед актуальних тенденцій веб-розробки для задач Marmaros найважливішими є SPA-інтерфейси, REST API, контейнеризація, хмарне розгортання, керування секретами, автоматизоване тестування та observability. SPA підхід дає змогу побудувати швидкий клієнтський інтерфейс із плавними переходами між каталогом, кошиком і кабінетом.

React залишається поширеним підходом до побудови компонентних інтерфейсів. Його перевага для Marmaros полягає в тому, що складний UI

					БР.ІП – 92.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		20

можна розділити на незалежні компоненти: header, product card, cart controls, checkout form, admin widgets і modal-вікна [4]. Це зменшує дублювання коду і полегшує подальше розширення інтерфейсу.

Vite відображає тренд на швидші dev/build інструменти для frontend-розробки. Для навчального і практичного проєкту це важливо, оскільки швидкий цикл перевірки змін скорочує час між модифікацією компонента і переглядом результату [5]. У production-режимі Vite формує оптимізовані статичні артефакти, які можуть роздаватися frontend-контейнером.

FastAPI відповідає тенденції типізованих backend API. Він поєднує Python, Pydantic-схеми, dependency injection, автоматичну OpenAPI-документацію та зручне тестування endpoint-ів [6]. Для Marmaros це корисно, бо система має багато доменних модулів і потребує чіткої перевірки структури запитів.

Контейнеризація стала практичним стандартом для відтворюваного запуску застосунків. Docker Compose дозволяє описати frontend, backend, PostgreSQL та допоміжні сервіси як єдину конфігурацію [9]. Для Marmaros це дає змогу однаково запускати систему локально і готувати її до production-розгортання.

Хмарні сервіси та платформи deployment-автоматизації зменшують кількість ручних операцій. Coolify використовується як self-hosted платформа для build/deploy, доменів і змінних середовища [14], а Cloudflare Tunnel забезпечує публікацію marmaros.store та api.marmaros.store без прямого відкриття портів [13].

Тренд мікросервісів не завжди означає, що кожен MVP-проєкт має бути мікросервісним. Для Marmaros доцільнішою є модульна клієнт-серверна архітектура: вона простіша в супроводі, але вже має чіткі межі між компонентами, що дає змогу в майбутньому винести платежі, доставку або аналітику в окремі сервіси.

REST API залишається практичним стандартом для взаємодії SPA з

					БР.ІП – 92.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		21

backend. Для Marmaros REST підходить краще, ніж надмірно складний API-шар, оскільки основні операції мають природну ресурсну структуру: products, cart, orders, payments, users. Це полегшує тестування і документацію endpoint-ів.

Реляційні бази даних залишаються доцільними для систем, де важлива цілісність транзакцій. PostgreSQL підтримує зв'язки, індекси, транзакції, JSON-поля й стабільну роботу з бізнес-даними [8]. Для Marmaros це краще, ніж повністю документна модель, бо замовлення, користувачі, товари й оплата мають чіткі зв'язки.

Автоматизоване тестування є частиною життєвого циклу розробки Marmaros. Для backend використано pytest, а для frontend — production build як базову перевірку збірки. Додатково виконувалася ручна перевірка основних сценаріїв: перегляд каталогу, оформлення замовлення, вибір доставки, оплата, отримання інвойсу, генерація PDF-договору та робота адміністративної панелі.

Безпекові тенденції зосереджені на server-side authorization, захисті API, безпечному зберіганні secrets і перевірці callback-ів зовнішніх сервісів. OWASP API Security наголошує, що некоректний контроль доступу є одним з основних ризиків API-систем [15]. Для Marmaros це означає, що admin-функції повинні перевірятися саме backend-ом.

Observability і моніторинг стають важливими навіть для невеликих веб-систем. Після deployment недостатньо знати, що сайт відкрився один раз; потрібно бачити логи, health checks, помилки, uptime і проблеми з інтеграціями. Поточна версія Marmaros має базовий health endpoint, а майбутній розвиток має включати повноцінний моніторинг.

Отже, технологічний аналіз підтверджує доцільність обраної моделі: React/Vite для інтерфейсу, FastAPI для REST API, PostgreSQL для транзакційних даних, Docker/Coolify для відтворюваного запуску та Cloudflare Tunnel для публічного доступу. Ці рішення достатні для MVP і водночас

					БР.ІП – 92.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		22

залишають простір для поступового розвитку payment, delivery, email та аналітичних модулів.

Невирішеними для подальшого розвитку залишаються автоматизовані рекомендації комплектів спорядження, повний облік стану орендованих товарів після повернення, розширене e2e-тестування, резервне копіювання та моніторинг production-середовища.

1.5 Висновки по розділу

У першому розділі було проведено комплексний аналіз предметної області outdoor e-commerce, зокрема для сервісу Marmaros, який поєднує продаж і оренду туристичного спорядження. Встановлено, що предметна область має високу сезонність, різні категорії користувачів (від новачків до досвідчених туристів) і специфічні вимоги до інтерфейсу, контенту, адміністративного обліку та управління замовленнями. Зокрема, для Marmaros важливо підтримувати одночасно продаж і оренду товарів з формуванням rental-договорів, контролем строків користування, стану спорядження та відповідальності користувача.

Проаналізовано ключові поняття веб-розробки та архітектурні патерни, які використовуються для реалізації повноцінного Full-Stack застосунку. Визначено, що застосунок Marmaros побудований на SPA-клієнті з React/Vite та backend на FastAPI, з REST API для обміну даними, PostgreSQL для зберігання транзакційних даних і використанням JWT для автентифікації користувачів. Архітектура системи передбачає модульність компонентів, що дозволяє локалізувати зміни, ізольовано оновлювати частини системи та забезпечує масштабованість і підтримку складних бізнес-процесів, таких як оренда і генерація PDF-договорів.

					БР.ІП – 92.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		23

РОЗДІЛ 2. АНАЛІЗ ВИМОГ ТА ПОСТАНОВКА ЗАДАЧІ ПРОЄКТУВАННЯ

2.1. Збір та аналіз вимог зацікавлених сторін

Збір вимог для Marmaros виконувався через аналіз предметної області, сценаріїв користувача, існуючого програмного коду, структури бази даних, тестів, інтерфейсів і deployment-конфігурацій. Такий підхід відповідає інженерії вимог, де джерелами вимог можуть бути stakeholders, документи, аналоги, обмеження середовища і сценарії використання [18].

Вимоги до системи не виникають лише з бажання реалізувати певний набір сторінок. Вони формуються на перетині бізнес-цілей, очікувань користувачів, технічних обмежень, безпекових вимог і майбутньої експлуатації. Тому під час аналізу Marmaros було виділено кілька груп зацікавлених сторін і окремо розглянуто їхні потреби.

Основними зацікавленими сторонами є покупець, користувач прокату, адміністратор магазину, контент-адміністратор, розробник, DevOps-відповідальний і власник бізнесу. Кожна група має різні очікування: користувач хоче швидкого оформлення, адміністратор - контролю замовлень, розробник - модульної архітектури, власник - можливості запуску сервісу на власному домені - marmaros.store.

Покупець очікує, що каталог буде зрозумілим, ціна - прозорою, а процес checkout - коротким. Для нього критично, щоб система не вимагала зайвих дій і не приховувала фінальну суму. Це створює вимоги до UI, підсумку кошика, доступності кнопки підтвердження і повідомлень про помилки.

Користувач прокату має ширший набір очікувань. Йому потрібно знати не лише ціну товару, а й вартість оренди за день, строк користування, умови повернення і можливість отримати договір. Це формує вимоги до rent_price, rent_days, rental_order, rental agreement PDF і відображення історії в кабінеті.

					БР.ІП – 92.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		24

Адміністратор магазину працює не з окремою сторінкою, а з потоком замовлень. Йому потрібні статуси, пошук, дані клієнта, сума, спосіб доставки, оплата, TTN і можливість швидко змінити стан замовлення. Це формує вимоги до admin dashboard, AdminOrders і role-based access control.

Контент-адміністратор відповідає за новини, маршрути, описи й інформаційний супровід. Для outdoor-магазину це важливо, бо контент впливає на вибір спорядження. Вимоги цієї ролі включають створення, редагування, публікацію та архівацію матеріалів, а також збереження image credits.

DevOps-відповідальний або розробник, який підтримує deployment, очікує зрозумілої конфігурації, health checks, secrets, логів і можливості відтворити систему. Це формує вимоги до Docker Compose, Coolify, Cloudflare Tunnel, .env.example і production-змінних середовища.

Методи збору вимог включали аналіз аналогів, побудову user stories, перегляд сценаріїв checkout, аналіз документації методичних вимог, ревізію backend tests і перевірку інтерфейсу через локальний запуск. У результаті вимоги було поділено на функціональні, нефункціональні та обмеження впровадження.

Окремим джерелом вимог стала наявна структура тестів. Якщо в backend test suite існують перевірки auth, cart, orders, payments, dashboard і users_admin, це свідчить про функціональні області, які вже мають бути описані в пояснювальній записці. Таким чином, тести використовуються не лише як перевірка, а й як артефакт уточнення вимог.

Ще одним джерелом вимог став фактичний інтерфейс локального застосунку. Скріншоти головної сторінки, каталогу, картки товару, кошика, кабінету й admin-панелі підтверджують, що вимоги мають не абстрактний характер, а прив'язані до конкретних реалізованих сценаріїв.

Для систематизації вимог використано підхід трасування: кожна ключова вимога повинна мати джерело, опис, критерій приймання і спосіб

					БР.ІП – 92.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		25

перевірки. Це дозволяє уникнути ситуації, коли в роботі описано функцію, але немає доказу її реалізації або тестування.

Вимоги формувалися через аналіз аналогів, user stories, перегляд фактичного коду, backend-тестів, deployment-конфігурації та UI-екранів. У результаті було визначено гібридну модель buy/rent, ключові сценарії checkout, admin-обробки, email-надсилання, доставки та формування договору.

2.2. Функціональні вимоги до веб-застосунку Marmaros

Функціональні вимоги описують, що саме повинна виконувати система. Для Marmaros вони згруповані за модулями: каталог, кошик, замовлення, оренда, автентифікація, доставка, оплата, кабінет, контент і адміністративне керування. Кожна вимога повинна мати критерій приймання, щоб її можна було перевірити тестом або ручним сценарієм.

Функціональні вимоги повинні бути достатньо конкретними. Формулювання на рівні “система повинна мати каталог” є занадто загальним. Для приймання потрібно уточнити: каталог має підтримувати категорії, картки товарів, ціну купівлі, ціну оренди, зображення, характеристики, пагінацію і зв'язок з API.

Основні ролі системи показано на UML-діаграмі випадків використання, наведеній на рисунку 2.1. Діаграма фіксує різницю між гостем, автентифікованим користувачем і адміністратором, а також допомагає уникнути змішування прав доступу на рівні API.

Для гостя система повинна забезпечувати перегляд головної сторінки, каталогу, картки товару, новин і маршрутів. Гість може прийняти рішення про покупку або оренду, але для персоналізованих дій і збереження історії потрібна автентифікація.

Для автентифікованого користувача система повинна підтримувати кошик, checkout, особистий кабінет, історію замовлень, профіль доставки й

					БР.ІП – 92.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		26

доступ до договорів оренди. Особливо важливо, щоб кабінет був пов'язаний із фактичними замовленнями, а не був лише статичною сторінкою.

Для адміністратора система повинна підтримувати перегляд статистики, управління товарами, замовленнями, контентом, користувачами й відгуками. Адміністративний функціонал не повинен бути доступним через frontend-маскування; backend має перевіряти роль користувача перед виконанням операції [15].

Вимога до купівлі й оренди є центральною. Система повинна дозволяти додати товар у режимі купівлі або оренди, зберегти цей режим у кошику, передати його в checkout і створити правильний тип замовлення. Якщо в кошику змішані товари, backend має розділити їх на окремі purchase і rental замовлення.



Рисунок 2.1 - UML-діаграма випадків використання Marmaros

Вимога до платежів включає підготовку LiqPay payload і перевірку callback. Якщо ключі провайдера не налаштовані, система не повинна падати з неочікуваною помилкою. Вона має повернути контрольоване повідомлення про конфігурацію, що важливо для dev і staging середовища.

Вимога до доставки включає самовивіз і інтеграційний сценарій Нової пошти. У поточній реалізації важливо зберігати місто, відділення, тип доставки, дані отримувача та службовий delivery payload.

Контентні вимоги включають новини, маршрути й відгуки. Ці функції

не є критичними для checkout, але підсилюють предметну область, бо користувач може отримати не лише товар, а й інформаційний супровід для походу. Повний список функціональних вимог подано в таблиці 2.1.

Таблиця 2.1

Функціональні вимоги

Код	Вимога	Критерій приймання
FR-01	користувач переглядає каталог і категорії	API повертає 255 товарів і 8 категорій
FR-02	користувач відкриває картку товару	відображаються ціна, опис, зображення, режими
FR-03	користувач додає товар до кошика	позиція зберігається з quantity і mode
FR-04	система підтримує купівлю та оренду	checkout розділяє purchase і rental замовлення
FR-05	користувач проходить автентифікацію	JWT видається після коректних даних
FR-06	підтримується email-код входу	код створюється, перевіряється і має строк дії
FR-07	користувач обирає доставку	місто, відділення або pickup зберігаються в замовленні
FR-08	система формує LiqPay checkout	backend повертає data і signature
FR-09	для оренди формується PDF-договір	у кабінеті доступне завантаження договору
FR-10	адміністратор керує замовленнями	admin бачить статуси, клієнта, суму і TTN
FR-11	адміністратор керує контентом	news/routes створюються й публікуються
FR-12	система підтримує відгуки	reviews створюються і модеруються
FR-13	dashboard показує метрики	відображаються товари, замовлення, виручка
FR-14	система має health endpoint	GET /health повертає status і database

Ключові user stories зводяться до кількох ролей: гість переглядає каталог і картку товару; клієнт додає позиції до кошика, обирає купівлю або оренду, проходить checkout і отримує доступ до історії; адміністратор обробляє замовлення, статуси, доставку й контент; DevOps-профіль відповідає за health check, secrets і deployment.

2.3. Нефункціональні вимоги та критерії приймання

Нефункціональні вимоги визначають якість роботи системи. Для Marmaros критичними є зручність використання, безпека, підтримуваність, переносимість, продуктивність і надійність. Такі характеристики узгоджуються з моделлю якості ISO/IEC 25010 [19]. Повний список нефункціональних вимог подано в таблиці 2.2.

Зручність використання означає, що користувач повинен без інструкцій знайти товар, додати його до кошика і зрозуміти підсумкову суму. Для цього сторінка кошика має показувати товар, кількість, спосіб отримання, доставку, суму товарів, вартість доставки й фінальну суму до сплати.

Безпека означає серверну перевірку ролей, хешування паролів, JWT, контроль CORS і перевірку LiqPay callback. Важливо, що frontend не може бути джерелом істини для прав доступу. Навіть якщо кнопка admin-панелі прихована, endpoint має відхилити запит користувача без ролі admin.

Підтримуваність означає модульну структуру frontend і backend, окремі routers, schemas, models і services. Якщо логіка оплати зміниться, бажано змінювати payments module, а не весь backend. Якщо змінюється дизайн картки товару, це не повинно зачіпати checkout API.

Продуктивність для поточного MVP оцінюється не через великі навантажувальні тести, а через базові ознаки: production build збирається, каталог має пагінацію, API повертає дані без критичних затримок у локальному середовищі. Повноцінні JMeter або k6 тести доцільно додати на наступному етапі.

					БР.ІП – 92.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		29

Надійність означає, що система має контрольовану поведінку при помилках. Якщо база недоступна, health endpoint має це показати. Якщо LiqPay keys не налаштовані, backend має повернути зрозуміле повідомлення. Якщо delivery API недоступний, checkout не повинен втрачати вже введені дані користувача.

Переносимість означає, що проєкт можна запустити не лише на машині розробника. Docker Compose, .env.example, Coolify secrets і Cloudflare Tunnel роблять систему придатною до deployment.

Критерії приймання повинні бути вимірюваними. Наприклад, 'система безпечна' - нечітка вимога. Натомість можна перевірити, що admin endpoint недоступний customer-ролі, LiqPay callback перевіряє signature, а JWT_SECRET не залишається dev-значенням у production.

Для UI критерієм приймання є не лише наявність сторінки, а можливість пройти сценарій. Користувач має відкрити каталог, додати товар, перейти в кошик, заповнити дані, вибрати доставку і побачити підсумок. Саме тому в роботі використано скріншоти не лише головної сторінки, а й кошика, кабінету та admin-панелі.

Для backend критерієм приймання є проходження автоматизованих тестів. У поточній перевірці пройдено 53 тести, які покривають основні API-модулі та підтверджують працездатність ключової серверної логіки. Це не замінює всіх видів тестування, зокрема ручної перевірки та навантажувального тестування, але є важливим доказом коректності базових функцій системи.

Для deployment критеріями приймання є успішний frontend build, запуснені контейнери, health endpoint і коректно налаштований ingress для marmaros.store та api.marmaros.store. Виконання цих умов підтверджує, що застосунок може працювати не лише в локальному середовищі, а й у наближеній до реальної інфраструктурі. Без цих умов система залишається локальним прототипом, а не deployment-ready застосунком.

					БР.ІП – 92.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		30

Нефункціональні вимоги та метрики

Характеристика	Вимога	Метрика/критерій
Продуктивність	каталог не блокує UI	production build успішний, API пагінується
Безпека	admin API захищений	customer не має доступу до admin endpoint
Надійність	backend контролює стан БД	GET /health повертає database ok
Підтримуваність	код поділений за модулями	окремі routers, pages, components, services
Переносимість	система запускається контейнерами	Docker Compose описує frontend/backend/db
Зручність	checkout зрозумілий	користувач бачить товар, доставку, суму й підтвердження
Масштабованість	сервіс можна розширювати	модулі payments/delivery/admin відокремлені
Спостережуваність	є базова перевірка стану	health endpoint і logs

2.4. Постановка задачі проєктування системи

Задача проєктування полягає у створенні Full-Stack веб-застосунку Marmaros, який у межах одного програмного рішення підтримує два основні бізнес-сценарії: продаж туристичного спорядження та його короткострокову оренду. Система має поєднувати функції інтернет-магазину, сервісу прокату, особистого кабінету користувача, адміністративної панелі та інтеграційного середовища для оплати, доставки, email-повідомлень і формування PDF-документів.

Межами системи визначено клієнтську частину, серверну частину, базу даних, адміністративну підсистему та зовнішні інтеграційні точки. До клієнтської частини належать сторінки головного інтерфейсу, каталог, картка товару, кошик, checkout, авторизація, особистий кабінет, новини, маршрути та

адміністративні сторінки. Backend API відповідає за бізнес-логіку, обробку запитів, роботу з користувачами, товарами, замовленнями, платежами, доставкою, email-повідомленнями та rental-документами. База даних PostgreSQL використовується для зберігання користувачів, товарів, категорій, кошика, замовлень, позицій замовлення, відгуків і контенту.

Кінцевим результатом проєктування має бути не статичний UI-макет, а працездатний прототип веб-застосунку, який можна запустити локально через Docker і підготувати до production-публікації на домені marmaros.store. Система повинна мати реальні API endpoint-и, реляційну базу даних, рольову модель доступу, автоматизовані backend-тести, production-збірку frontend, скріншоти основних сценаріїв, deployment-схему та документаційні додатки. Це дозволяє розглядати Marmaros як deployment-ready MVP, а не лише як навчальний приклад або локальну лабораторну роботу.

Основна проєктна задача полягає в тому, щоб об'єднати купівлю та оренду в єдиній моделі користувача, товару й замовлення. Для цього товар не повинен дублюватися окремо як товар для продажу і як товар для оренди. Натомість одна товарна позиція повинна містити дані, необхідні для обох сценаріїв: ціну купівлі, ціну оренди, опис, характеристики, зображення, категорію та службові поля для відображення у каталозі. Такий підхід спрощує підтримку каталогу й дозволяє адміністратору керувати асортиментом без створення паралельних сутностей.

Особливу увагу потрібно приділити checkout-сценарію. У звичайному інтернет-магазині checkout завершується створенням замовлення на купівлю. У Marmaros він має підтримувати складнішу логіку: користувач може додати до кошика товари для купівлі, товари для оренди або змішаний набір позицій. Backend повинен коректно обробити такий кошик, зберегти режим кожної позиції, розрахувати суму, врахувати доставку й за потреби розділити purchase-замовлення та rental-замовлення. Це потрібно для того, щоб куплені товари не змішувалися з орендованими, а для rental-сценарію можна було

					БР.ІП – 92.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		32

сформувати окремий договір.

Для rental-сценарію система повинна підтримувати строк оренди, дані користувача, перелік орендованого спорядження, суму, умови повернення та PDF-договір. Генерація договору має бути частиною системної логіки, а не зовнішньою ручною дією. Після оформлення rental-замовлення користувач повинен мати доступ до відповідного документа в особистому кабінеті, а адміністратор — бачити таке замовлення в admin-панелі разом зі статусом, клієнтськими даними та службовою інформацією.

На рівні ролей необхідно передбачити щонайменше три базові стани користувача: гість, автентифікований користувач і адміністратор. Гість може переглядати головну сторінку, каталог, картки товарів, новини та маршрути. Автентифікований користувач може працювати з кошиком, оформлювати замовлення, переглядати особистий кабінет, історію замовлень і rental-документи. Адміністратор отримує доступ до керування товарами, категоріями, замовленнями, користувачами, контентом, відгуками та основними показниками dashboard. Захист адміністративних функцій має виконуватися на backend-рівні, а не лише через приховування елементів інтерфейсу.

На рівні API необхідно забезпечити стабільну й логічну структуру endpoint-ів із версіонованим префіксом ``/api/v1``. Endpoint-и мають бути поділені за предметними областями: auth, products, categories, cart, orders, payments, delivery, dashboard, users, news, routes і reviews. Такий поділ спрощує тестування, документування, супровід і майбутнє розширення системи. Крім того, версіонування API дозволяє в майбутньому змінювати контракт окремих модулів без негайного порушення роботи клієнтської частини.

На рівні frontend необхідно реалізувати інтерфейс, який відповідає основним користувацьким сценаріям. Головна сторінка має презентувати сервіс і вести користувача до каталогу. Каталог має показувати товари та категорії. Картка товару повинна відображати характеристики, ціну купівлі,

					БР.ІП – 92.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		33

ціну оренди й доступні дії. Кошик має містити позиції, кількість, режим використання, контактні дані, доставку та підсумкову суму. Особистий кабінет має показувати профіль користувача, історію замовлень і доступ до договорів. Адміністративні сторінки повинні забезпечувати керування основними даними системи.

На рівні бази даних необхідно спроектувати структуру, яка забезпечує цілісність зв'язків між основними сутностями. Користувачі мають бути пов'язані із замовленнями, замовлення — з позиціями замовлення, позиції — з товарами, товари — з категоріями, а відгуки — з товарами та користувачами. Така модель дозволяє зберігати історію дій користувача, формувати адміністративні звіти, перевіряти замовлення та розширювати функціональність без повної перебудови бази даних.

Система повинна підтримувати зовнішні інтеграційні точки, але не повинна залежати від них неконтрольовано. Для оплати передбачено інтеграцію з LiqPay, для доставки — інтеграційний шар під службу доставки, для email — серверне надсилання кодів входу та повідомлень, для deployment — Docker, Coolify і Cloudflare Tunnel. Реальні ключі LiqPay, email-сервісів і служб доставки мають задаватися лише через production-змінні середовища. У репозиторії не повинні зберігатися приватні ключі, паролі або службові токени.

Припущеннями проєктування є наявність стабільного інтернет-з'єднання, доступу до PostgreSQL, production-домену marmaros.store, піддомену api.marmaros.store, налаштованих secrets у Coolify та коректної конфігурації Cloudflare Tunnel. Також передбачається, що платіжні callback-и LiqPay перевіряються відповідно до sandbox або production-конфігурації провайдера [11]. Якщо зовнішній сервіс тимчасово недоступний, система повинна поводитися контрольовано: не втрачати введені користувачем дані та повертати зрозуміле повідомлення про помилку.

Архітектурно система повинна залишатися достатньо простою для

					БР.ІП – 92.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		34

реалізації й підтримки в межах бакалаврської роботи, але водночас достатньо модульною для подальшого розвитку. Тому обрано клієнт-серверну модель із модульним backend замість повної мікросервісної архітектури. Такий підхід зменшує складність deployment, тестування й супроводу, але дозволяє ізолювати основні частини системи: каталог, замовлення, оплату, доставку, користувачів, контент і адміністрування.

Окремою задачею є забезпечення тестованості системи. Backend повинен мати автоматизовані тести для ключових модулів: автентифікації, кошика, замовлень, платежів, товарів, категорій, користувачів, dashboard і role-based access. Frontend має проходити production-збірку, що підтверджує коректність імпортів, маршрутизації, компонентів і bundle-артефактів. Тестування повинно бути пов'язане з функціональними та нефункціональними вимогами, щоб було зрозуміло, яка саме перевірка підтверджує конкретну можливість системи.

На рівні впровадження необхідно підготувати Docker/Coolify-схему, змінні середовища, Cloudflare Tunnel ingress і доменну модель marmaros.store/api.marmaros.store. Docker забезпечує відтворюваний запуск frontend, backend і бази даних. Coolify використовується для керування deployment і production-змінними. Cloudflare Tunnel забезпечує публічний доступ до frontend і backend без прямого відкриття портів. Така схема дозволяє продемонструвати практичний шлях від локальної розробки до публічного середовища.

Критеріями приймання системи є наявність працездатного каталогу, коректне повернення товарів і категорій через API, можливість додавання товарів до кошика, створення purchase- та rental-замовлень, збереження доставки й суми, формування PDF-договору для оренди, захист admin-функцій роллю, проходження backend-тестів, успішна production-збірка frontend і наявність описаної deployment-схеми. Додатковими доказами готовності є скріншоти основних сторінок, результати тестування та приклади

					БР.ІП – 92.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		35

конфігурацій у додатках.

Водночас система має чітко визначені межі. У межах бакалаврського MVP не реалізуються повноцінна бухгалтерія, складський WMS, CRM, автоматичне списання застави, production-моніторинг, навантажувальні тести, бухгалтерський експорт і повний облік фізичного стану спорядження після повернення. Ці функції не подаються як завершені, а визначаються як напрями подальшого розвитку. Такий підхід дозволяє уникнути завищення результатів і коректно відокремити реалізовану функціональність від перспективних можливостей.

Отже, задача проєктування Marmaros зводиться до створення модульної Full-Stack системи, у якій каталог, кошик, checkout, оплата, доставка, email-повідомлення, rental-договір, особистий кабінет і admin-панель працюють через єдину модель користувача, товару та замовлення. Система повинна бути достатньо функціональною для демонстрації основних бізнес-процесів outdoor-сервісу, достатньо структурованою для тестування й документування та достатньо підготовленою для подальшого production-впровадження.

2.5 Висновки по розділу

У другому розділі бакалаврської роботи проведено комплексний та систематизований аналіз вимог до веб-застосунку Marmaros, який є основою для подальшого архітектурного проєктування та реалізації системи. У межах розділу сформовано цілісне уявлення про предметну область, визначено коло зацікавлених сторін, структуровано їхні потреби та перетворено їх у формалізовані функціональні й нефункціональні вимоги.

Особливістю проведеного аналізу є орієнтація не лише на класичну модель електронної комерції, але й на розширений rental-сценарій, який суттєво впливає на логіку системи. Визначено, що система Marmaros повинна

					БР.ІП – 92.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		36

одночасно підтримувати два бізнес-напрями — продаж і оренду туристичного спорядження — що накладає додаткові вимоги до моделі даних, процесів оформлення замовлень, обробки платежів та формування документів. Саме поєднання цих двох сценаріїв формує унікальність системи та визначає її відмінність від типових e-commerce рішень.

У процесі аналізу було виділено ключові групи зацікавлених сторін: покупці, користувачі оренди, адміністратори системи, контент-адміністратори, розробники та DevOps-відповідальні. Для кожної групи визначено окремі функціональні очікування. Зокрема, користувачі очікують простого та інтуїтивного процесу вибору товарів і оформлення замовлення, адміністратори — повного контролю над замовленнями, товарами та користувачами, а технічні спеціалісти — стабільної, модульної та легко розгортальної системи з передбачуваною поведінкою в production-середовищі.

На основі потреб зацікавлених сторін сформовано детальний перелік функціональних вимог, який охоплює основні модулі системи: каталог товарів, кошик, checkout-процес, систему автентифікації, управління замовленнями, доставку, оплату, кабінет користувача, а також адміністративну та контентну частини. Для кожної вимоги визначено критерії приймання, що дозволяє забезпечити їх перевірюваність та подальше трасування під час тестування. Такий підхід є важливим з точки зору інженерії програмного забезпечення, оскільки дозволяє уникнути неоднозначності у трактуванні вимог і забезпечує їх відповідність реалізованому функціоналу.

Окрему увагу приділено нефункціональним вимогам, які визначають якість роботи системи. Серед них виділено продуктивність, безпеку, масштабованість, підтримуваність, зручність використання, переносимість та надійність. Було встановлено, що система повинна забезпечувати стабільну роботу API, швидку обробку запитів до каталогу, безпечну обробку персональних даних користувачів та захищений доступ до адміністративних функцій. Крім того, передбачено вимоги до контейнеризації та можливості

					БР.ІП – 92.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		37

відтворюваного розгортання через Docker та Coolify, що є важливою умовою для сучасних веб-застосунків.

Важливим результатом розділу є формування чітких меж системи. Визначено, що Marmaros реалізується як MVP-рішення, яке охоплює основні бізнес-процеси, але не включає повноцінні корпоративні системи рівня WMS, CRM або бухгалтерського обліку. Такий підхід дозволяє сфокусуватися на ключовій цінності системи — поєднанні продажу та оренди туристичного спорядження з автоматизацією основних операцій. Додаткові складні підсистеми розглядаються як напрям подальшого розвитку.

Отримані результати аналізу вимог дозволили сформувавши основу для подальшого проєктування архітектури системи. Зокрема, визначено необхідність клієнт-серверної моделі з модульним backend, використання REST API з версіонуванням, а також реляційної бази даних для забезпечення цілісності бізнес-даних. Окремо обґрунтовано потребу в розділенні ролей користувачів і впровадженні серверної авторизації як ключового механізму безпеки системи.

Також у межах розділу закладено основу для подальшого тестування системи. Формалізовані вимоги безпосередньо пов'язані з майбутніми тест-кейсами, що дозволяє реалізувати принцип трасування вимог: кожна функція системи може бути перевірена через відповідний набір автоматизованих або ручних тестів. Це підвищує контрольованість розробки та спрощує валідацію результатів на етапі впровадження.

Узагальнюючи, другий розділ виконує ключову роль у структурі бакалаврської роботи, оскільки він перетворює загальне уявлення про систему Marmaros на формалізовану систему вимог. Саме на основі цих вимог у наступних розділах виконується проєктування архітектури, моделі даних, API та користувацьких сценаріїв. Розділ забезпечує логічну цілісність роботи, визначає межі системи та створює основу для її подальшої реалізації, тестування та впровадження.

					БР.ІП – 92.00.00.000 ПЗ	Арк.
						38
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 3. ПРОЄКТУВАННЯ АРХІТЕКТУРИ, МОДЕЛІ ДАНИХ ТА API

3.1. Обґрунтування архітектурної моделі системи

На основі вимог, сформованих у другому розділі, для Marmaros обрано клієнт-серверну архітектуру з окремою SPA-клієнтською частиною, backend API та реляційною базою даних. Така модель відповідає фактичній реалізації проєкту: frontend працює на React + Vite, backend реалізовано на FastAPI, а дані зберігаються у PostgreSQL.

Клієнт-серверна модель дозволяє чітко розділити відповідальність. Frontend відповідає за інтерфейс, навігацію, стан кошика, форми та взаємодію з користувачем. Backend відповідає за бізнес-правила, автентифікацію, ролі, checkout, оплату, доставку, роботу з базою даних і адміністративні операції. Такий поділ важливий для безпеки: критичні рішення не повинні прийматися лише на клієнті.

Повна мікросервісна архітектура для поточного етапу не обрана, оскільки вона ускладнила б deployment, тестування і супровід без реальної потреби. Натомість backend організовано як модульний застосунок: окремі routers і services відповідають за auth, products, categories, cart, orders, payments, delivery, dashboard, users, news, routes і reviews. Це дає переваги модульності без надмірної інфраструктурної складності.

Архітектурна схема системи наведена на рисунку 3.1. Вона відображає браузер користувача, frontend React + Vite, backend API FastAPI, PostgreSQL, модуль автентифікації, зовнішні сервіси, deployment через Docker/Coolify і публікацію через marmaros.store. Схема відповідає реальній структурі проєкту й не включає компонентів, які фактично не реалізовані.

					БР.ІП – 92.00.00.000 ПЗ	Арк.
						39
Змн.	Арк.	№ докум.	Підпис	Дата		

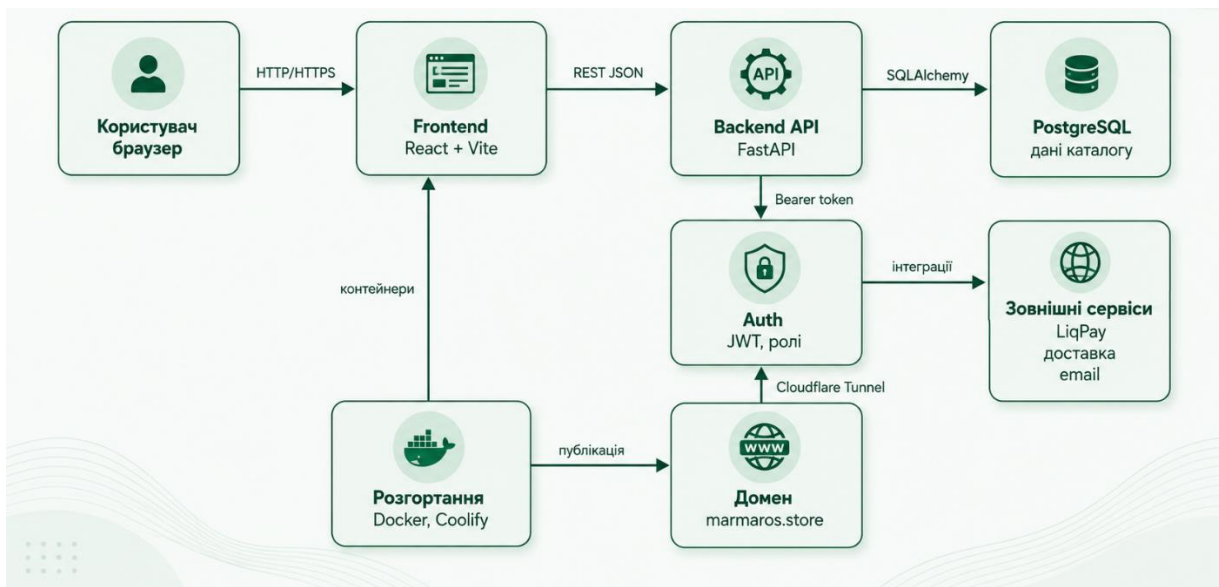


Рисунок 3.1 - Архітектура Full-Stack системи Marmaros

У межах цієї архітектури frontend не звертається напряму до бази даних. Усі операції проходять через backend API. Це дає змогу централізовано перевіряти права доступу, валідувати payload, вести бізнес-логіку замовлень і не розкривати структуру бази даних клієнтській частині.

PostgreSQL використовується як основне сховище транзакційних даних. Для e-commerce і rental-сценаріїв це доцільно, оскільки замовлення, користувачі, товари, категорії, позиції замовлення й відгуки мають чіткі зв'язки. Реляційна модель також полегшує подальшу аналітику для admin dashboard.

Зовнішні сервіси в архітектурі обмежені тим, що реально передбачено проєктом: LiqPay для платежів, delivery-інтеграційний шар для служб доставки, email-сервіс для кодів входу та повідомлень, Cloudflare Tunnel для публічного доступу. У роботі не описуються сторонні сервіси, які не закладені в реалізацію.

Важливим архітектурним рішенням є збереження версіонованого API-префікса /api/v1. Це дає змогу в майбутньому змінювати API без негайного руйнування клієнтської частини. Наприклад, якщо в наступній версії зміниться структура checkout або delivery response, можна буде додати /api/v2,

залишивши /api/v1 для старих клієнтів.

У проєкті чітко розділено development і production-режими. Локально система запускається через Docker Compose і Vite dev server, що зручно для перевірки UI, API та бази даних. Для production-середовища передбачено окремі змінні FRONTEND_URL, API_PUBLIC_URL, CORS_ORIGINS і секрети інтеграцій, які не повинні зберігатися у відкритому коді.

Архітектура не приховує компромісів MVP. Система не розбита на окремі мікросервіси для каталогу, платежів і доставки, оскільки це збільшило б витрати на підтримку. Натомість модульний backend дає достатню ізоляцію відповідальностей: кожен router можна розвивати окремо, а deployment залишається простим і контрольованим.

Для даних обрано гібридний підхід: транзакційні сутності зберігаються у нормалізованій реляційній структурі, а гнучкі характеристики товарів можуть зберігатися у JSON-полях. Це відповідає предметній області outdoor-спорядження, де різні категорії мають різні параметри, але самі замовлення, користувачі та оплати потребують суворої узгодженості.

Таким чином, архітектура Marmaros орієнтована на практичну демонстрацію і подальший розвиток. Вона не є надмірно складною, але вже містить головні production-ready елементи: контейнеризацію, окремий API, реляційну БД, ролі, інтеграційні точки, доменну схему і можливість винесення секретів у середовище deployment.

3.2. Моделювання бізнес-процесів і користувацьких сценаріїв

Моделювання бізнес-процесів виконано навколо основного сценарію: вибір товару, додавання до кошика, вибір режиму купівлі або оренди, введення даних отримувача, вибір доставки, підтвердження замовлення, оплата або післяплата, адміністративна обробка. Activity-схему цього процесу наведено на рисунку 3.2.

					БР.ІП – 92.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		41

У сценарії покупки користувач додає товар до кошика, обирає кількість, заповнює контактні дані, обирає доставку і підтверджує замовлення. У сценарії оренди до цих кроків додається строк користування і подальше формування rental-договору. Саме відмінність цих сценаріїв пояснює потребу в order_type і спеціальній обробці rental_lines.

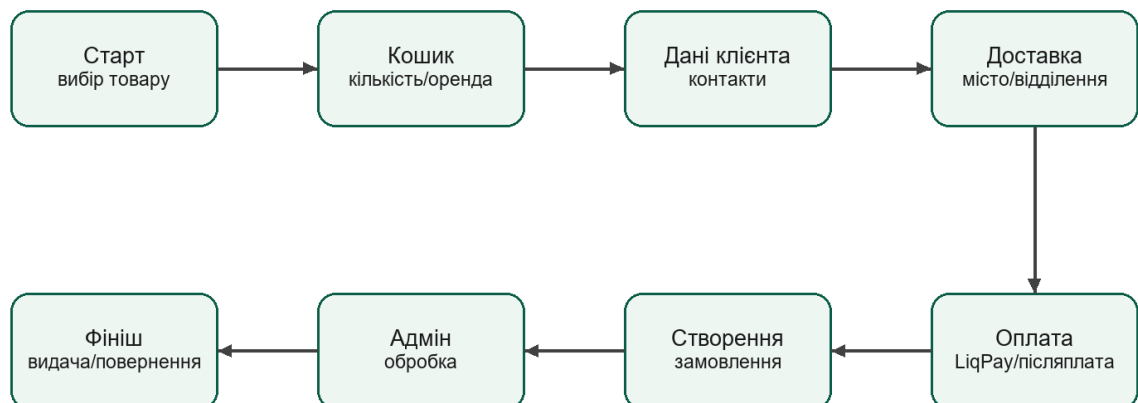


Рисунок 3.2 - Activity-схема бізнес-процесу замовлення

Sequence-схема на рисунку 3.3 деталізує взаємодію під час checkout. Користувач формує cart payload у frontend, після чого frontend надсилає POST /orders/checkout до Orders API. Backend створює записи orders і order_items у PostgreSQL, формує або готує дані для оплати/доставки, повертає статус замовлення і робить його доступним для admin-панелі.

Важливо, що backend залишається центральним вузлом сценарію. Він не лише зберігає замовлення, а й приймає рішення про розділення купівлі й оренди. Якщо кошик містить обидва типи позицій, payload розділяється на purchase-order і rental-order. Це спрощує життєвий цикл замовлень і не змішує документи оренди з купленими товарами.

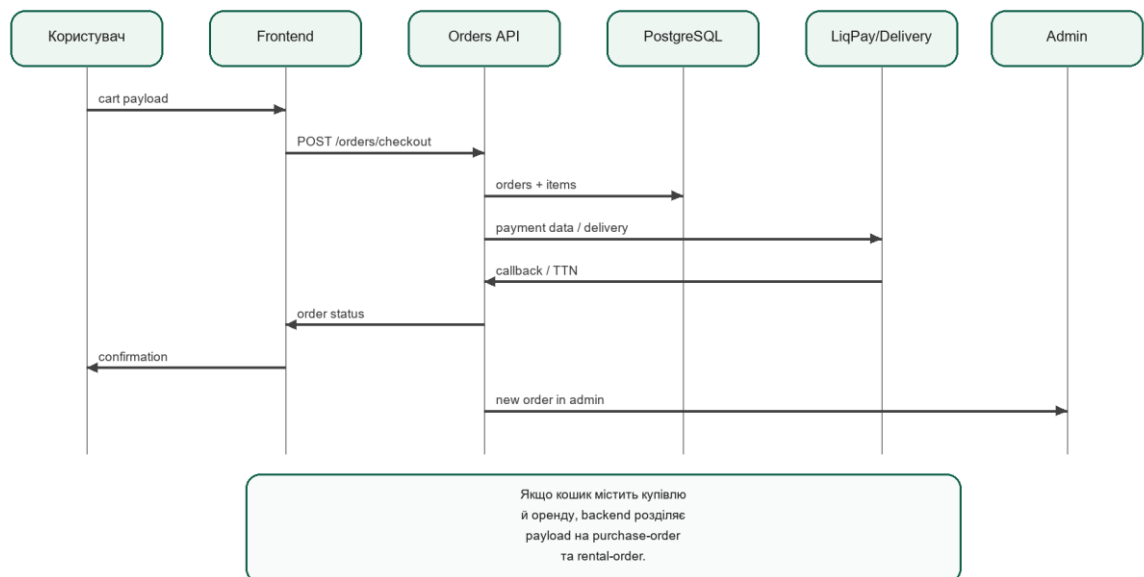


Рисунок 3.3 - Sequence-схема checkout купівлі та оренди

Прикладну схему checkout наведено на рисунку 3.4. Вона показує, що потік користувача не закінчується на кошику: після підтвердження замовлення система переходить до оплати, договору оренди, адміністративної обробки та історії в кабінеті.

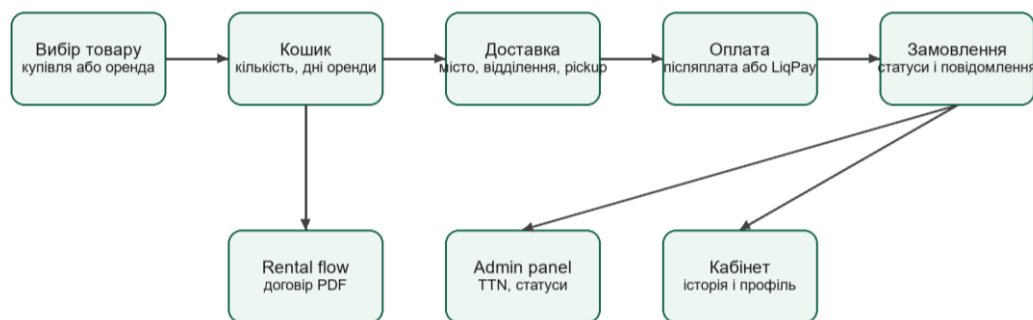


Рисунок 3.4 - Сценарій оформлення замовлення Marmaros

3.3. Проєктування бази даних та REST API

Модель даних Marmaros побудована навколо сутностей users, products, categories, cart_items, orders, order_items, reviews, news, routes, contacts і

newsletter. Реляційна модель обрана через потребу в транзакційності замовлень, зв'язках між товарами й категоріями, історії користувача та звітах для адміністратора [8].

Узагальнену ER-модель наведено на рисунку 3.5. Категорія має багато товарів, користувач має кошик і замовлення, замовлення має рядки order_items, а products пов'язані з reviews та order_items. Така модель відповідає реальній структурі проєкту і підтримує як продаж, так і оренду без дублювання товарних позицій.

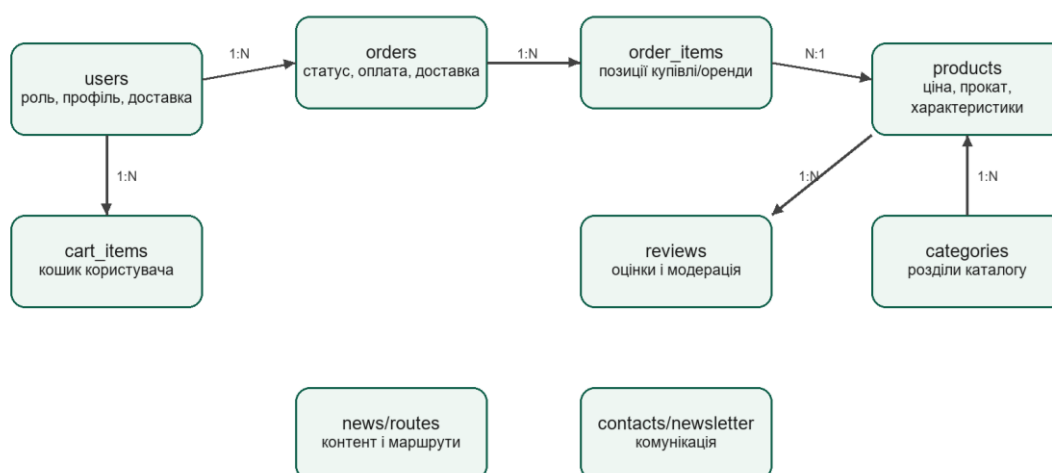


Рисунок 3.5 - Узагальнена ER-модель бази даних Marmaros

Для товарів використано поєднання стандартних полів і JSON-характеристик. Стандартні поля потрібні для ціни, назви, slug, опису, категорії та зображення. JSON-характеристики дають змогу зберігати різні параметри для різних категорій спорядження без створення надмірної кількості таблиць.

REST API спроектовано з префіксом /api/v1. Такий підхід дає змогу версіонувати API та не змішувати службові маршрути з публічними endpoint-ами. Роутери згруповані за відповідальністю, що відповідає реальній організації backend-коду. Основні групи REST API виділено в таблиці 3.1.

Основні групи REST API

Група API	Призначення	Приклади операцій
auth	автентифікація та профіль	login, email-code, me, update profile
products/categories	каталог	list, detail, create, update, delete
cart/orders	кошик і checkout	add item, checkout, order history
payments	LiqPay	checkout payload, callback
delivery	доставка	cities, warehouses, TTN data
dashboard/users	адміністрування	stats, users, role access
news/routes/reviews	контент і соціальні функції	publish, moderate, list

Окремо для уточнення взаємозв'язків між frontend, backend, базою даних і зовнішніми сервісами побудовано діаграму залежностей API. Вона показує, що браузер працює з внутрішнім REST API, а Resend, LiqPay і Нова пошта викликаються тільки серверною частиною. Це важливо для безпеки, оскільки секретні ключі не потрапляють у JavaScript-код клієнта.

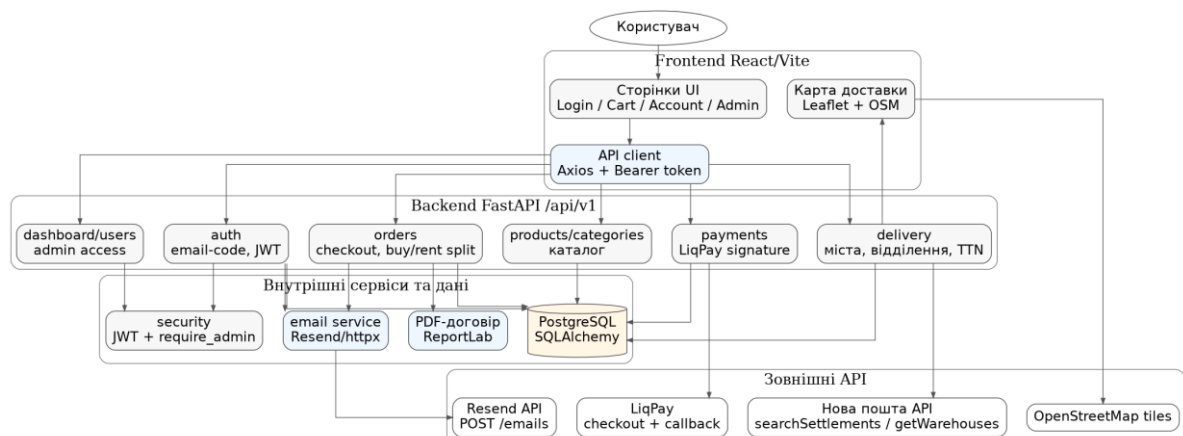


Рисунок 3.6 - Діаграма залежностей API та зовнішніх інтеграцій Marmaros

3.4. Вибір і обґрунтування технологічного стеку

Вибір технологічного стеку для веб-застосунку Marmaros здійснювався

на основі вимог до функціональності, масштабованості, підтримуваності та можливості подальшого розгортання системи в production-середовищі.

Frontend частина системи реалізована з використанням React і Vite. React обрано завдяки компонентній архітектурі, широкій екосистемі та підтримці повторного використання UI-компонентів [4]. Це особливо важливо для Marmaros, оскільки інтерфейс включає каталог товарів, картки продуктів, кошик, checkout, особистий кабінет, новини, маршрути та адміністративні сторінки. Vite використовується як сучасний інструмент збірки, що забезпечує швидкий dev-server і оптимізований production build, скорочуючи час розробки та тестування інтерфейсу [5].

Додатково важливим фактором вибору React є його придатність для побудови масштабованих SPA-додатків із чітким розділенням стану та представлення. Використання компонентного підходу дозволяє ізолювати логіку окремих частин системи (каталог, кошик, checkout, кабінет), що спрощує супровід коду та зменшує зв'язаність між модулями. Це є критично важливим для Marmaros, оскільки інтерфейс містить як комерційні сценарії (купівля/оренда), так і інформаційні сторінки (новини, маршрути), які повинні розвиватися незалежно.

Backend реалізовано на основі FastAPI, який підтримує типізовані схеми даних, dependency injection, автоматичну генерацію OpenAPI-документації та зручну структуру routers [6]. Це дозволяє чітко розділити бізнес-домени системи: автентифікацію, товари, кошик, замовлення, оплату, доставку та адміністративні функції. SQLAlchemy використовується як ORM-рівень для взаємодії з PostgreSQL, що дозволяє працювати з базою даних через об'єктну модель Python і спрощує управління складними зв'язками між сутностями [7]. PostgreSQL застосовується як основна реляційна база даних для зберігання транзакційних даних, замовлень, користувачів і зв'язків між ними, забезпечуючи надійність і цілісність даних [8].

Важливою особливістю обраного стеку є його внутрішня узгодженість:

					БР.ІП – 92.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		46

FastAPI, SQLAlchemy та PostgreSQL утворюють єдину типізовану backend-екосистему, у якій дані проходять перевірку на рівні схем, ORM та бази даних. Це зменшує кількість помилок, пов'язаних із невідповідністю форматів запитів, і підвищує стабільність API. Додатково використання асинхронної моделі FastAPI дозволяє ефективно обробляти паралельні запити користувачів без блокування основного потоку виконання, що є важливим для сценаріїв каталогу, checkout та інтеграцій із зовнішніми сервісами.

Для розгортання системи використано Docker Compose, який дозволяє запускати frontend, backend і базу даних як єдину контейнеризовану систему [9]. Це забезпечує відтворюваність середовища та однакову поведінку системи в development і production. Для production-деплою використовується Coolify як платформа керування контейнерами та змінними середовища [14], а Cloudflare Tunnel забезпечує безпечний доступ до системи через домени marmaros.store і api.marmaros.store без відкриття портів на сервері [13].

Для автентифікації використовується JWT (JSON Web Token), що забезпечує безпечну передачу даних користувача та реалізацію рольової моделі доступу [10]. Платіжна інтеграція реалізована через LiqPay, який використовується для формування checkout-запитів та перевірки callback-підписів [11]. Для доставки передбачено інтеграційний шар із використанням API служб доставки, зокрема Нової пошти, що дозволяє отримувати міста, відділення та формувати дані доставки [12]. Email-повідомлення реалізуються через серверний сервіс розсилок (наприклад, Resend), який використовується для надсилання кодів входу, підтверджень замовлень та службових повідомлень користувачам [22].

Обґрунтування технологічного стеку полягає у тому, що React і Vite забезпечують швидкий та зручний SPA-інтерфейс, FastAPI — продуктивний і типізований REST API, SQLAlchemy і PostgreSQL — надійне зберігання та обробку транзакційних даних, Docker і Docker Compose — відтворюване середовище виконання, Coolify і Cloudflare Tunnel — простий і безпечний

					БР.ІП – 92.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		47

production deployment, а JWT, LiqPay та інтеграції доставки забезпечують реалізацію ключових бізнес-процесів системи.

Таким чином, обраний стек технологій є узгодженим, сучасним і практично орієнтованим, що дозволяє реалізувати повноцінний Full-Stack веб-застосунок із можливістю подальшого масштабування та розвитку без суттєвої зміни архітектури.

3.5 Висновки по розділу

У третьому розділі бакалаврської роботи здійснено комплексне проєктування архітектури веб-застосунку Marmaros, моделі даних та REST API, що є ключовим етапом переходу від формалізованих вимог до практичної реалізації системи. На основі аналізу предметної області та вимог, визначених у попередньому розділі, сформовано цілісну архітектурну модель, яка забезпечує підтримку як функціоналу електронної комерції, так і сценаріїв оренди туристичного спорядження.

Обґрунтовано вибір клієнт-серверної архітектури з модульним backend-застосунком і SPA-клієнтом. Такий підхід дозволяє чітко розмежувати відповідальність між компонентами системи: frontend відповідає за інтерфейс користувача та взаємодію, backend — за бізнес-логіку, обробку даних, авторизацію, інтеграції та безпеку, а база даних забезпечує централізоване зберігання транзакційних даних. Відмова від повної мікросервісної архітектури на користь модульного моноліту є обґрунтованою для масштабу бакалаврського проєкту, оскільки дозволяє зберегти баланс між складністю реалізації та можливістю подальшого розвитку системи.

У межах розділу побудовано логічну та фізичну модель даних, яка відображає основні сутності предметної області: користувачів, товари, категорії, кошик, замовлення, позиції замовлення, відгуки, контентні модулі (новини та маршрути). Визначено зв'язки між сутностями, що забезпечують цілісність даних та підтримують як сценарії купівлі, так і оренди без

					БР.ІП – 92.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		48

дублювання товарних записів. Особливістю моделі є підтримка гібридного підходу до зберігання характеристик товарів, де критичні атрибути реалізовані у реляційній структурі, а гнучкі параметри — через JSON-поля. Це дозволяє адаптувати систему до різних типів туристичного спорядження без ускладнення схеми бази даних.

Окремо спроектовано REST API із версіонованим префіксом `/api/v1`, що забезпечує стабільність контракту між клієнтською та серверною частинами. Endpoint-и структуровано за предметними доменами, такими як `auth`, `products`, `cart`, `orders`, `payments`, `delivery`, `dashboard` та інші. Така декомпозиція підвищує зрозумілість системи, спрощує тестування, підтримку та подальшу масштабованість. Версіонування API також створює можливість еволюційного розвитку системи без порушення сумісності з існуючими клієнтами.

У процесі проектування визначено ключові бізнес-процеси системи, зокрема сценарії покупки та оренди. Особливу увагу приділено логіці `checkout`, яка передбачає обробку змішаних кошиків і розділення замовлень на `purchase` та `rental` типи. Це рішення дозволяє уникнути змішування різних бізнес-моделей у межах одного замовлення та забезпечує коректну обробку орендних договорів, строків користування та супровідної документації.

Важливим результатом розділу є формалізація архітектурної взаємодії між компонентами системи. Визначено, що `frontend` не має прямого доступу до бази даних, а всі операції виконуються через `backend API`. Такий підхід підвищує рівень безпеки, дозволяє централізовано контролювати бізнес-логіку та забезпечує можливість масштабування системи у майбутньому. Окремо визначено інтеграційні точки із зовнішніми сервісами, такими як платіжна система `LiqPay`, служби доставки, email-сервіси та інфраструктура розгортання через `Docker`, `Coolify` і `Cloudflare Tunnel`.

Архітектурні рішення також враховують вимоги до безпеки та керування доступом. Реалізовано концепцію рольової моделі доступу, де

					БР.ІП – 92.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		49

розмежовано права гостей, автентифікованих користувачів та адміністраторів. Це забезпечує контроль над доступом до критичних функцій системи, таких як керування товарами, замовленнями та контентом. Додатково враховано необхідність захисту API через JWT-автентифікацію та перевірку прав доступу на серверному рівні.

У розділі також сформовано базові принципи для подальшого тестування системи. Структура API, модульність backend та чітке визначення бізнес-логіки створюють умови для реалізації автоматизованих тестів, що перевіряють коректність роботи ключових функцій. Це дозволяє забезпечити трасування між вимогами, архітектурою та реалізацією, що є важливим принципом інженерії програмного забезпечення.

Таким чином, третій розділ закладає фундамент практичної реалізації системи Marmaros. Він визначає структурну організацію застосунку, спосіб взаємодії компонентів, модель даних та принципи побудови API. Сформована архітектура є достатньо простою для реалізації в межах бакалаврської роботи, але водночас гнучкою для подальшого масштабування та розширення функціональності.

Отже, результати третього розділу забезпечують цілісне технічне бачення системи та створюють основу для реалізації клієнтської та серверної частин у наступному розділі. Визначені архітектурні принципи, модель даних та API-контракти гарантують узгодженість між компонентами системи та дозволяють реалізувати Marmaros як повноцінний Full-Stack веб-застосунок із можливістю подальшого розвитку у production-середовищі.

					БР.ІП – 92.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		50

РОЗДІЛ 4. РЕАЛІЗАЦІЯ ВЕБ-ЗАСТОСУНКУ MARMAROS

4.1. Реалізація клієнтської частини

Клієнтська частина Marmaros реалізована як SPA на React. Основними сторінками є HomePage, ShopPage, ProductPage, CartPage, LoginPage, AccountPage, NewsPage, RoutesPage і адміністративні сторінки. Компонентна структура дозволяє повторно використовувати Header, Footer, ProductCard, RentCheckoutModal, layout-компоненти та API-клієнт.

Головна сторінка демонструє готовий UI, навігацію, категорії, промо-блоки та картки товарів. Вона допомагає користувачеві швидко орієнтуватися у категоріях і акціях - рисунок 4.1.

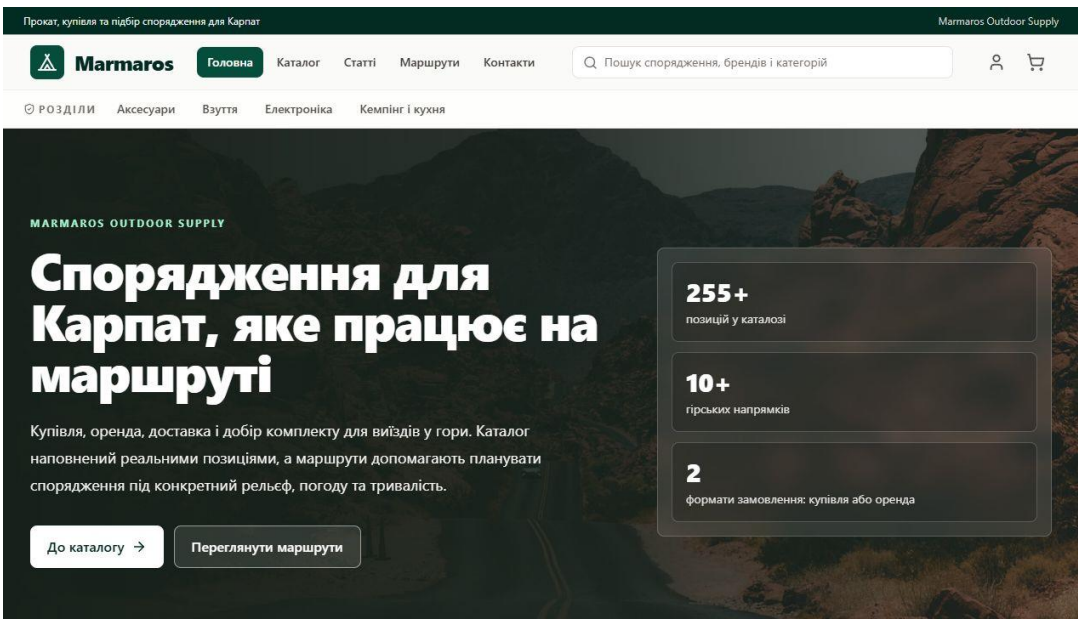


Рисунок 4.1 - Головна сторінка Marmaros

Для покращення сприйняття інформації на сторінці використовуються повторно компоненти, такі як Header, Footer і ProductCard. Це спрощує масштабування та підтримку UI і логічно підводить до демонстрації каталогу, що зображено на рисунку 4.2.

					БР.ІП – 92.00.00.000 ПЗ	Арк.
						51
Змн.	Арк.	№ докум.	Підпис	Дата		

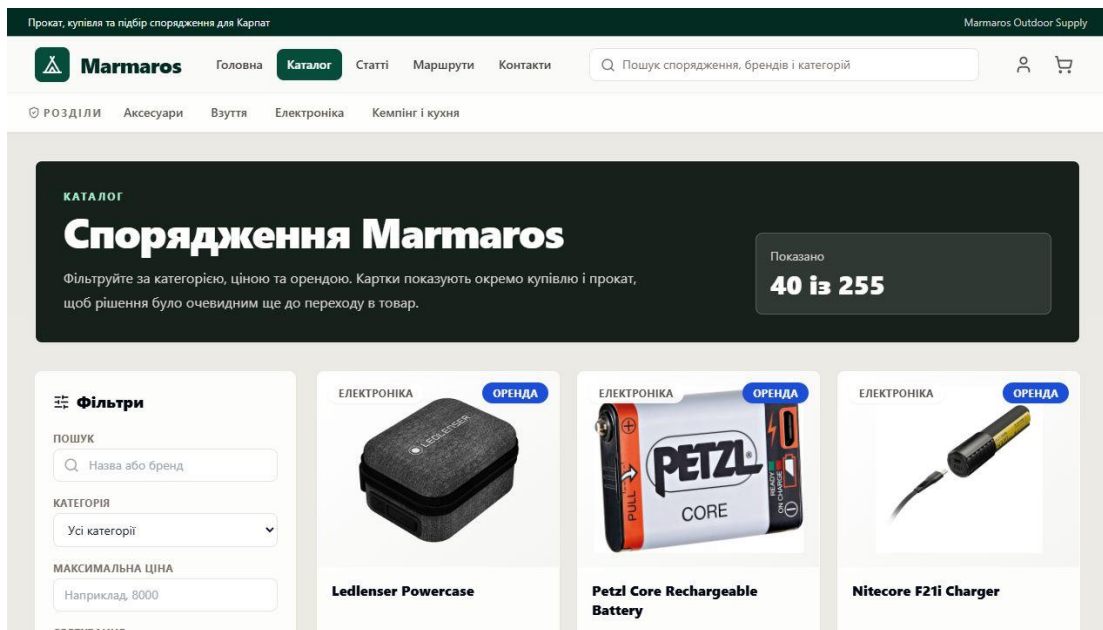


Рисунок 4.2 - Каталог спорядження з фільтрами

Каталог показує категорії товарів, можливість сортування, швидкий доступ до купівлі або оренди та інтеграцію з backend API.

Картка товару зображена на рисунку 4.3 та демонструє підтримку одночасно купівлі та оренди товарів — ключову відмінність Marmaros від звичайного інтернет-магазину.

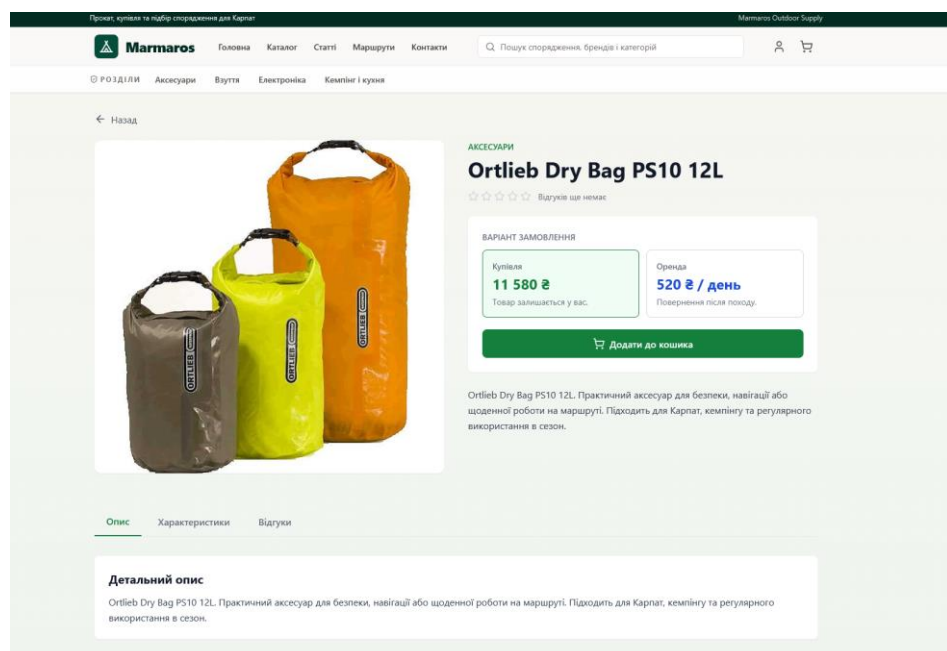


Рисунок 4.3 - Картка товару з режимами купівлі та оренди

					БР.ІП – 92.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		52

Сторінка кошика є ключовим frontend-модулем, оскільки вона об'єднує товари, кількість, контактні дані, доставку, підсумок і підтвердження. Її вигляд наведено на рисунку 4.4. У межах реального проєкту саме CartPage є точкою, де користувацький інтерфейс найближче взаємодіє з бізнес-логікою backend.

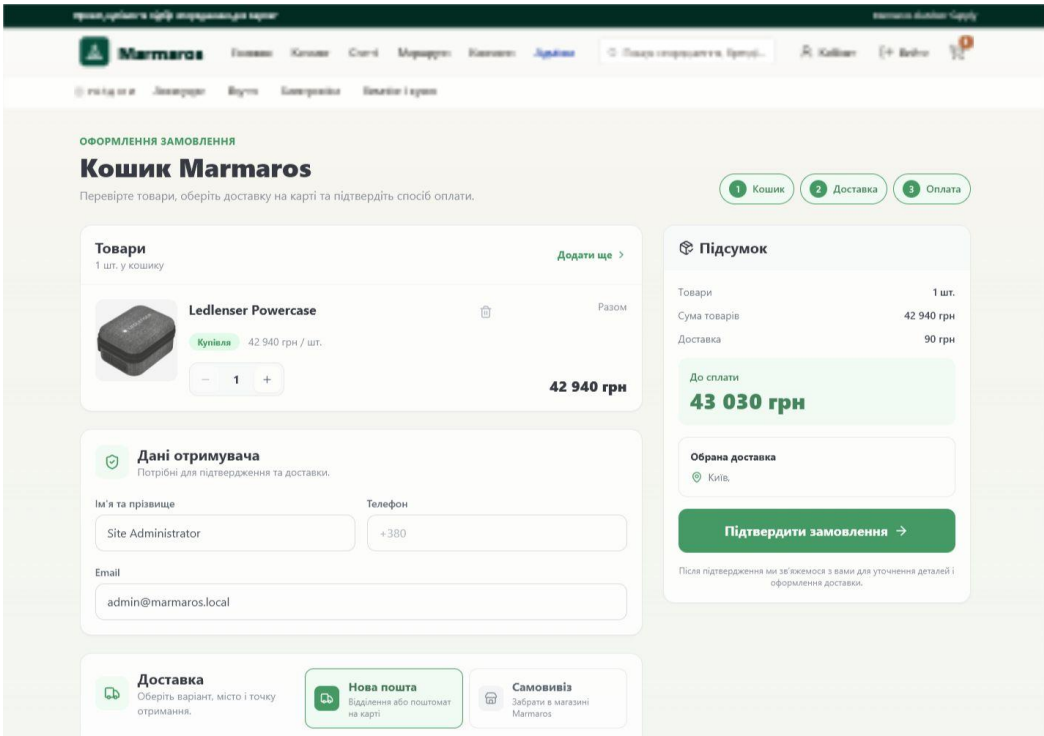


Рисунок 4.4 - Кошик і форма оформлення замовлення Marmaros

Особистий кабінет показаний на рисунку 4.5. Він потрібен не для декоративного профілю, а для відображення історії замовлень, активних операцій, профілю доставки і договорів оренди. Приклад місця для скріншота згенерованого PDF-договору наведено далі в розділі 4, щоб доказ цієї функції був не лише у додатку.

Практична реалізація frontend побудована навколо повторного використання API-клієнта. Це важливо, оскільки сторінки каталогу, кабінету, кошика й admin-панелі не повинні дублювати логіку HTTP-запитів. Єдиний API-шар спрощує обробку базового URL, токена, помилок і форматування відповіді.

Стан користувача зберігається на рівні auth-контексту та

використовується для відображення профілю, що на рисунку 4.5, а також історії замовлень і адміністративних елементів інтерфейсу. Водночас UI-логіка не підміняє backend-захист: навіть якщо користувач приховає або відкриє кнопку в браузері, серверні admin-endpoint-и мають окремо перевіряти роль.

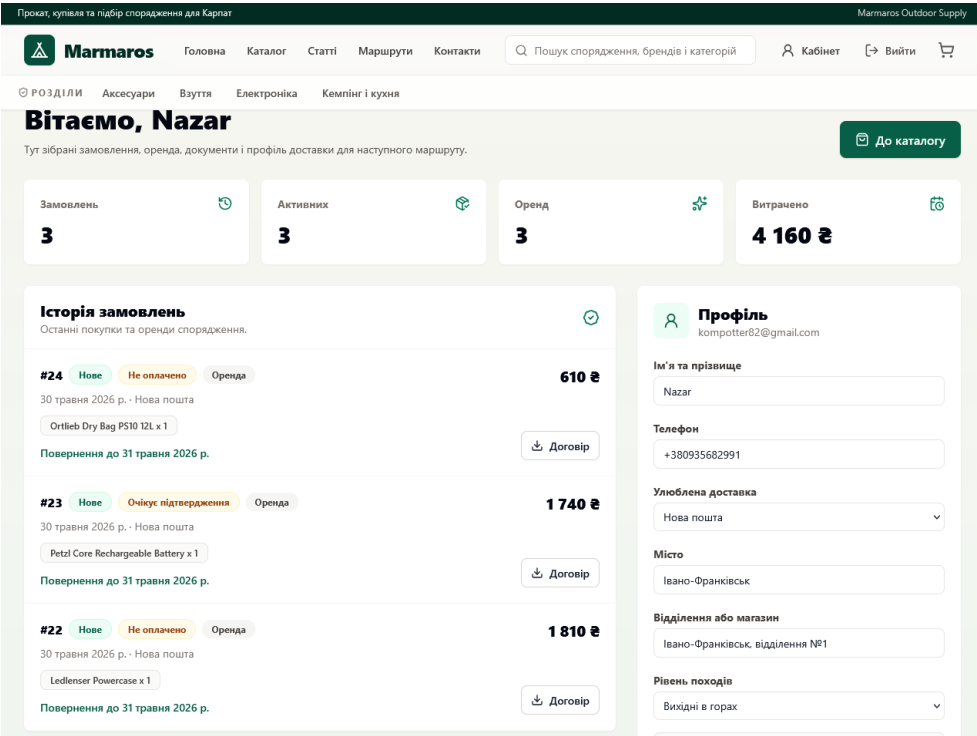


Рисунок 4.5 - Особистий кабінет користувача

4.2. Реалізація серверної та адміністративної підсистем

Backend реалізовано як FastAPI-застосунок з application factory create_app. Під час створення застосунку підключаються middleware для CORS і correlation-id, реєструються обробники помилок, а всі API-роутери підключаються з префіксом /api/v1. Це дає змогу зберігати стабільну структуру API та версіонувати контракт.

Серверна частина містить модулі auth, cart, products, categories, orders, payments, delivery, dashboard, reviews, news, routes, contacts, newsletter і users. Адміністративні маршрути захищені перевіркою ролі admin, тому зміна

каталогу, новин, маршрутів, статусів замовлень і користувачів недоступна звичайному клієнту.

Модуль orders реалізує основну бізнес-логіку. Під час checkout він працює з даними кошика, створює замовлення, розділяє купівлю та оренду, зберігає delivery data і готує замовлення до подальшої оплати або адміністративної обробки.

Модуль payments відповідає за LiqPay payload і callback. У роботі описується саме реалізована логіка: формування data/signature і перевірка підпису. Не стверджується, що система проходила повну production-сертифікацію платіжного провайдера, оскільки це потребує реальних ключів і окремого процесу.

Окреме місце займає генерація rental-договору. Для оренди недостатньо лише створити order record, тому backend формує PDF-документ на основі даних rental-order, контактів клієнта, строку користування та переліку позицій. Договір пов'язаний із замовленням і доступний користувачу в кабінеті через endpoint rentalAgreementUrl, що робить rental-сценарій документально завершеним. Приклад договору зображено на рисунку 4.6.

Адміністративна підсистема реалізована як операційний інтерфейс, а не як окремий маркетинговий сайт. Її основна задача - швидко показати замовлення, статуси, користувачів, статистику і контент, який потребує підтримки. Робочі екрани admin dashboard і керування замовленнями подано на рисунку 4.6.

На рівні backend важливим є також централізоване налаштування CORS, correlation-id і обробників помилок. Це допомагає під час налагодження: помилка API повертається у передбачуваному форматі, а запит можна відстежити за ідентифікатором у логах. Для дипломного MVP це практичніше, ніж розгортати складну систему observability.

					БР.ІП – 92.00.00.000 ПЗ	Арк.
						55
Змн.	Арк.	№ докум.	Підпис	Дата		

ДОГОВІР ПРОКАТУ (НАЙМУ) ТУРИСТИЧНОГО СПОРЯДЖЕННЯ № R-000024

м. Івано-Франківськ, 31.05.2026

Правова підстава: глава 58 Цивільного кодексу України, зокрема статті 759, 762, 763, 773, 779, 785, 787-791; Закон України «Про електронні документи та електронний документообіг»; Закон України «Про електронну ідентифікацію та електронні довірчі послуги».

1. Сторони та реквізити замовлення

Наймодавець / Орендодавець	Marmaros Outdoor Supply, м. Івано-Франківськ, вул. Незалежності, 41
Наймач / Орендар	Nazar, телефон +380935682991, email: kompotter82@gmail.com
Замовлення	№ 24, створене 31.05.2026, 01:34
Доставка / передача	Нова пошта, Івано-Франківськ, Івано-Франківськ, відділення №1
Місце повернення	Marmaros Івано-Франківськ, вул. Незалежності, 41. Якщо спорядження отримували доставкою, повернення можна узгодити з менеджером через Нову пошту.

2. Предмет договору та акт приймання-передачі

Наймодавець передає, а Наймач приймає у строкове платне володіння та користування рухоме майно (туристичне спорядження), визначене нижче. Ця таблиця є актом приймання-передачі спорядження, підтверджує перелік, кількість, тариф, строк користування та стан майна на момент передачі.

№	Спорядження та ідентифікація	К-сть	Днів	Тариф/день	Сума	Стан / комплектність
1	Ortlieb Dry Bag PS10 12L	1	1	520.00 грн	520.00 грн	ID товару: 216. Комплектація за описом позиції замовлення. Стан при передачі: справний, без видимих пошкоджень, якщо інше не зазначено у коментарі до замовлення.
Строк користування		з 31.05.2026, 01:34 до 01.06.2026, 01:34				
Вартість прокату		520.00 грн				
Доставка		90.00 грн				
Разом до сплати		610.00 грн				
Дата формування документа		31.05.2026, 01:34				

3. Порядок оплати, передачі та повернення

3.1. Плата за користування спорядженням визначена тарифами Наймодавця та зафіксована у цьому договорі. Оплата здійснюється способом, обраним у замовленні.

3.2. Повернення спорядження здійснюється не пізніше 01.06.2026, 01:34 за адресою/точкою: Marmaros Івано-Франківськ, вул. Незалежності, 41. Якщо спорядження отримували доставкою, повернення можна узгодити з менеджером через Нову пошту..

3.3. Під час повернення сторони звіряють кількість, комплектність, чистоту та технічний стан спорядження. Зауваження щодо стану фіксуються менеджером у замовленні або окремому акті.

Рисунок 4.6 - PDF-договір оренди

Під час реалізації збережено межу між фактичним кодом і майбутніми можливостями. Наприклад, у системі описано delivery integration layer і поля доставки, але не заявлено повну автоматичну генерацію накладних для всіх служб. Такий підхід робить пояснювальну записку коректною щодо реального стану репозиторію.

					БР.ІП – 92.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		56

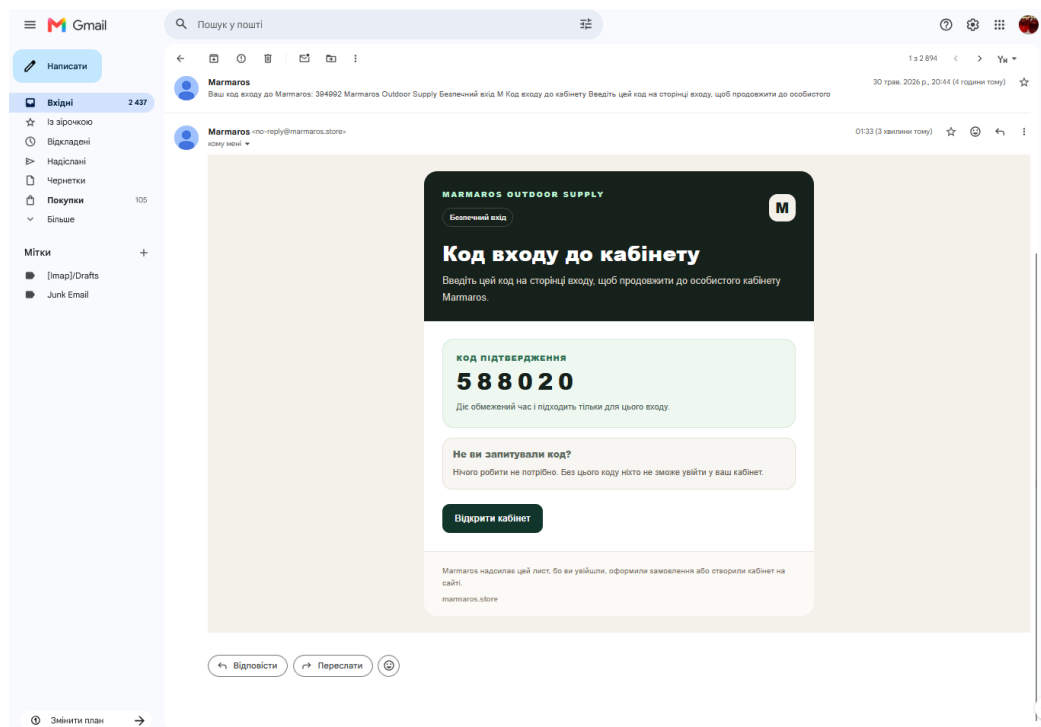


Рисунок 4.7 - Лист з email-кодом входу

Email-сервіс реалізовано на backend-рівні, а не у браузері. Через `backend/app/services/email.py` виконуються надсилення одноразового коду входу, welcome-листів, листів для guest account і підтвердження замовлення. Після `checkout orders router` ставить відправлення чека або інвойсу у `BackgroundTasks`, а у відповіді повертає `receipt_email_queued`, `account_email_queued` та `email_notice`. `RESEND_API_KEY` і `RESEND_FROM_EMAIL` залишаються у змінних середовища, тому секрети не потрапляють у frontend [22]. Приклад листа для авторизації входу зображено на рисунку 4.7. А сам інвойс після платежу на рисунку 4.8.

Інтеграція з Новою поштою винесена у backend-модуль `delivery`. Frontend викликає внутрішні endpoint-и `/api/v1/delivery/cities` і `/api/v1/delivery/warehouses`, а backend формує запити до офіційного JSON API Нової пошти: `searchSettlements/getCities` для міст і `getWarehouses` для відділень.

					БР.ІП – 92.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		57

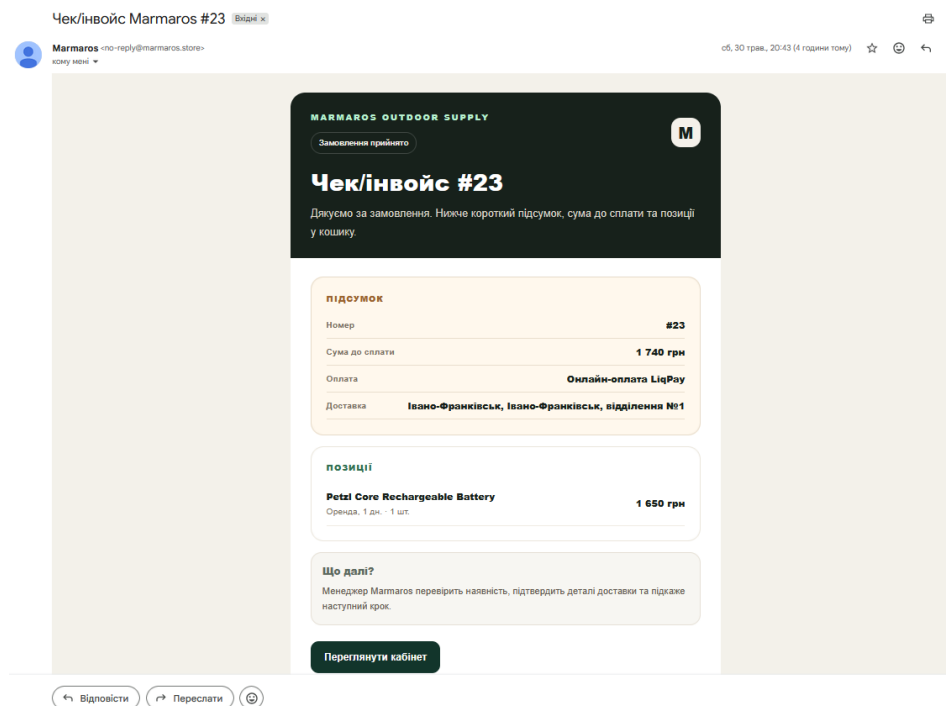


Рисунок 4.8 - Лист з інвойсом або підтвердженням оплати

Відповідь нормалізується до ref, name і meta, а координати відділень використовуються для карти. NOVA_POSHTA_API_KEY залишається на сервері; якщо зовнішній сервіс недоступний, система використовує local_fallback [12]. Візуальний приклад продемонстровано на рисунках 4.9 та 4.10.

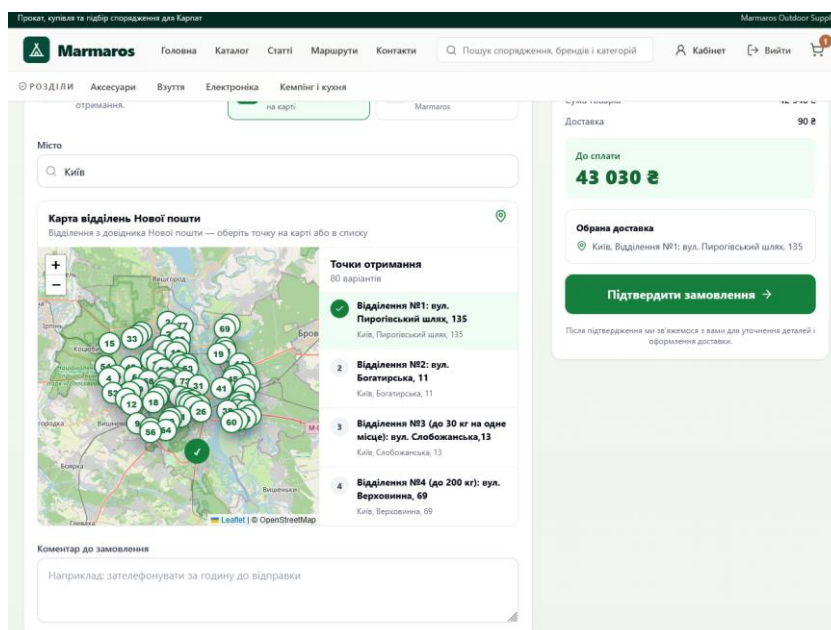


Рисунок 4.9 - Вибір відділення Нової пошти

					БР.ІП – 92.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		58

Демонстраційні матеріали, що підтверджують роботу основних функцій Marmaros у публічному середовищі, наведено в додатку А. До них належать екрани головної сторінки, каталогу, checkout, адміністративної панелі, email-надсилання, інтеграції Нової пошти та PDF-договору оренди.

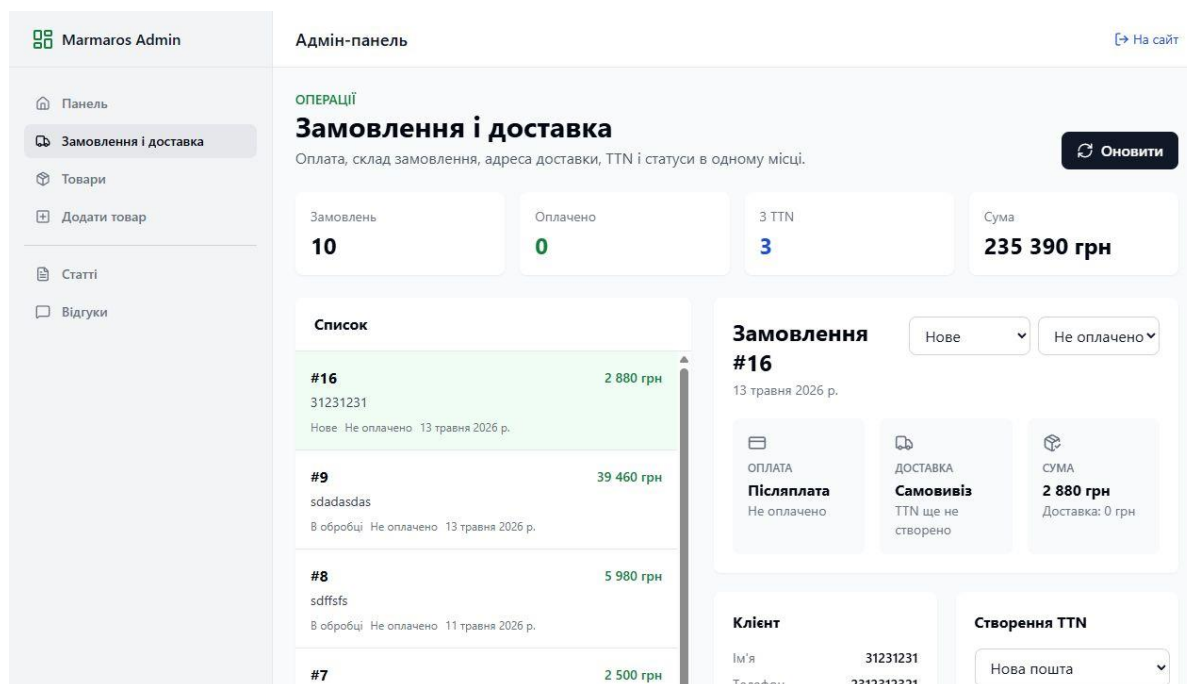


Рисунок 4.10 - Керування замовленнями та доставкою

4.3. Організація програмного коду, алгоритми та структури даних

Організація коду відповідає модульному принципу. У frontend логіка поділена на pages, components, context, api і utils. У backend виділено app/models, app/schemas, app/routers, app/services, app/config.py і міграції. Такий поділ підтримує принцип single responsibility і спрощує модульне тестування.

Ключовим алгоритмом є checkout split: backend аналізує рядки кошика й розділяє їх на purchase_lines і rental_lines. Якщо в одному кошику є купівля та оренда, створюються два замовлення різного типу. Це зменшує складність

подальшої обробки, тому що rental-замовлення має договір і дату повернення, а purchase-замовлення - ні.

Другий важливий алгоритм - формування підпису LiqPay. Backend формує payload, кодує його у base64 і підписує приватним ключем за правилами провайдера [11]. Callback приймається лише після перевірки signature, що запобігає несанкціонованій зміні статусу оплати.

Структури даних продукту включають JSON-характеристики. Це дає змогу зберігати різні технічні параметри для наметів, рюкзаків, електроніки чи одягу без створення окремої таблиці для кожного типу товару. Для критичних транзакційних даних, зокрема замовлень і користувачів, використовується нормалізована реляційна модель.

Лістинг 4.1 - Підключення API-роутерів у FastAPI

```
api_prefix = "/api/v1"
for router in (
    auth.router, cart.router, products.router, categories.router,
    news.router, routes.router, reviews.router, orders.router,
    payments.router, delivery.router, contacts.router,
    newsletter.router, dashboard.router, users.router,
):
    app.include_router(router, prefix=api_prefix)
```

Лістинг 4.2 - Формування підпису LiqPay

```
def _sign(data: str) -> str:
    raw =
    f'{settings.LIQPAY_PRIVATE_KEY}{data}{settings.LIQPAY_PRIVATE_KEY}'
    return base64.b64encode(hashlib.sha1(raw).digest()).decode("ascii")
```

Модульне тестування та налагодження

Модульне тестування backend виконано за допомогою pytest. Тести згруповано за функціональними областями: auth, cart, categories, contacts, dashboard, email_service, news, newsletter, orders, payments, products, reviews, routes і users_admin. Така структура тестів відповідає модульній організації

					БР.ІП – 92.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		60

backend.

Налагодження виконувалося через локальні Docker-контейнери, health endpoint, логи backend, перевірку production build фронтенду і ручний перегляд ключових сторінок. Для frontend критичною перевіркою є npm run build, оскільки вона виявляє помилки імпортів, JSX, маршрутизації та збірки bundle [5].

У роботі не заявляється про виконання повного e2e-набору в Selenium або Playwright як про завершений результат. Такі тести визначені як напрям розвитку. Фактично виконаними перевірками є backend pytest suite, frontend production build, health endpoint і ручна перевірка основних екранів через локально запусканий сайт.

Модульні тести мають найбільшу цінність для backend, тому що саме там зосереджені правила замовлень, ролей, платежів і даних. Для frontend у межах роботи основним автоматизованим gate є успішна production-збірка. Вона не замінює e2e, але підтверджує, що клієнтська частина компілюється і може бути розгорнута. Приклади модульних тестів та їх результатів описано в таблиці 4.1.

Таблиця 4.1

Приклади модульних тестових кейсів

Модуль	Що перевіряється	Очікуваний результат
auth	login, email code, current user	коректний JWT і профіль
cart	додавання, кількість, видалення	стан кошика відповідає діям
orders	checkout purchase/rental	створюються правильні типи замовлень
payments	LiqPay payload/signature	data і signature валідні
dashboard	admin statistics	агреговані показники доступні admin
users_admin	рольовий доступ	customer не має admin-доступу

Налагодження Docker-середовища також є частиною практичної реалізації. Під час локального запуску перевіряється, що frontend бачить API, backend підключається до PostgreSQL, а health endpoint повертає коректний стан. Це важливо, бо помилки такого рівня не завжди проявляються у чистих unit-тестах.

4.4 Висновки по розділу

У четвертому розділі детально описано практичну реалізацію Marmaros, включаючи клієнтську частину, серверний backend та адміністративну підсистему. Клієнтська частина реалізована як SPA на React із компонентною структурою, що дозволяє повторно використовувати Header, Footer, ProductCard, RentCheckoutModal і layout-компоненти. Основні сторінки включають HomePage, ShopPage, ProductPage, CartPage, LoginPage, AccountPage, NewsPage, RoutesPage та адміністративні сторінки. Така організація дозволяє забезпечити зручну навігацію, повторне використання компонентів і підтримку масштабованого UI.

Демонстраційні рисунки показують готовий інтерфейс, включно з головною сторінкою, каталогом із фільтрами, карткою товару з режимами купівлі та оренди, кошиком і особистим кабінетом користувача. Кожен із цих модулів відображає реальну взаємодію frontend із бізнес-логікою backend і підтримує ключові функції системи, включаючи відображення замовлень, формування PDF-договорів для оренди та інтеграцію з платіжними і логістичними сервісами.

Серверна частина реалізована на FastAPI із модульною структурою, включаючи роутери auth, cart, products, orders, payments, delivery, dashboard та інші. Основні алгоритми, такі як розділення purchase та rental замовлень та формування підпису LiqPay, забезпечують коректну обробку бізнес-логіки та безпечну взаємодію з зовнішніми сервісами. Backend контролює ролі

					БР.ІП – 92.00.00.000 ПЗ	Арк.
						62
Змн.	Арк.	№ докум.	Підпис	Дата		

користувачів, перевірку доступу до адміністративних функцій та генерацію rental-договорів у PDF.

Модульне тестування і налагодження підтверджують стабільність та коректність роботи системи. Backend-тести покривають основні модулі та бізнес-логіку, а frontend build забезпечує перевірку збірки інтерфейсу. Налаштування через Docker-контейнери, health endpoint та логування дозволяє контролювати стан системи та швидко виявляти помилки.

					БР.ІП – 92.00.00.000 ПЗ	Арк.
						63
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 5. СИСТЕМНЕ ТЕСТУВАННЯ, ВПРОВАДЖЕННЯ ТА ОЦІНКА РЕЗУЛЬТАТІВ

5.1. Стратегія та методика тестування системи

Системне тестування Marmaros спрямоване на перевірку готового рішення як цілісної системи. Воно охоплює функціональне тестування, інтеграційне тестування, backend API тести, smoke-перевірки, перевірку безпеки доступу, production build і перевірку deployment-конфігурації.

Методика тестування побудована за принципом трасування вимог: кожна ключова вимога з розділу 2 має відповідний спосіб перевірки. Модель перевірки якості наведено на рисунку 5.1. Вона показує перехід від вимог до unit/API tests, frontend build, smoke checks, manual UX, admin checks і release gate.

У межах перевірки Marmaros було виконано не лише модульне тестування backend і перевірку production-збірки frontend, а й додаткову ручну та інструментальну перевірку ключових сценаріїв. Для backend використано pytest, для перевірки працездатності інтерфейсу — ручне проходження основних користувацьких сценаріїв, а для оцінки поведінки системи під навантаженням і в браузерному середовищі можуть бути застосовані JMeter та Selenium. Таке поєднання дозволяє перевірити як внутрішню логіку API, так і практичну роботу системи з боку користувача.

Методика тестування охоплює перевірку backend-модулів, frontend build, health check, Docker-середовища, основних екранів системи, сценаріїв checkout, оплати, доставки, email-повідомлень і генерації PDF-договору. Це підтверджує, що Marmaros працює як цілісний MVP: серверна частина обробляє запити, клієнтська частина коректно збирається, база даних доступна, а ключові функції можуть бути продемонстровані під час захисту.

Критерії приймання сформовано так, щоб їх можна було перевірити фактами: кількість пройдених тестів, результат build, відповідь health endpoint,

					БР.ІП – 92.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		64

наявність товарів і категорій в API, screenshots реальних сторінок, а також коректна deployment-схема для marmaros.store і api.marmaros.store.

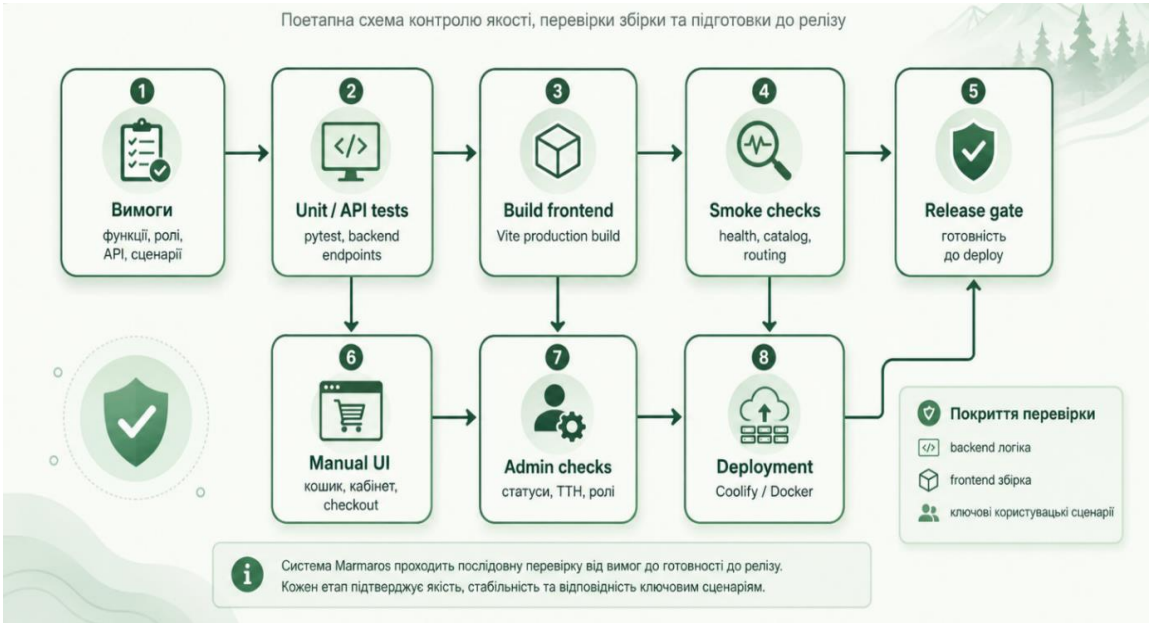


Рисунок 5.1 - Модель перевірки якості Marmaros

У поточній роботі перевірка охоплює backend API tests, frontend production build, ручне функціональне тестування, інтеграційну перевірку основних потоків та smoke-перевірку deployment. Повноцінне e2e, навантажувальне й security-тестування залишено як напрям подальшого розвитку.

5.2. Результати функціонального, інтеграційного та backend-тестування

Практична перевірка виконувалася в локальному Docker-середовищі. Контейнери frontend, backend, PostgreSQL і cloudflared були запущені; backend health endpoint повернув status ok і database ok. Це підтверджує базову доступність системи та зв'язок backend із базою даних.

Frontend production build було виконано у контейнері marmaros-frontend. Збірка завершилася успішно: оброблено 1748 модулів, сформовано

index.html, CSS bundle 42.52 kB і JS bundle 458.22 kB. Це підтверджує, що клієнтський код збирається у production-артефакти без помилок компіляції.

Backend test suite було виконано у контейнері marmaros-backend після тимчасового копіювання каталогу tests, оскільки production Docker image не містить tests через .dockerignore. Запущено 53 тести у 14 файлах. Результат виконання: 53 passed за 50.19 с. Частка успішних тестів становить 100% для наявного набору автоматизованих backend-перевірок.

Попередження під час тестування стосувалися переважно datetime.utcnow і cache path у тимчасовій директорії контейнера. Вони не блокують проходження тестів, однак у production-версії варто перейти на timezone-aware UTC, щоб уникнути ризиків у rental-днях, даті повернення і часових фільтрах.

Функціональні сценарії перевірялися також через інтерфейс: головна сторінка, каталог, картка товару, кошик, кабінет користувача, admin dashboard і admin orders. Додатково для фінальної версії роботи передбачено фіксацію скріншотами таких інтеграційних доказів: надсилання email-коду входу, листа/інвойсу після оформлення замовлення, листа після оплати, вибору міста й відділення Нової пошти та відкриття згенерованого PDF-договору оренди. Результати цих тестувань можемо бачити в таблиці 5.1.

Таблиця 5.1

Результати тестування

Перевірка	Кількість/показник	Результат
Backend tests	53 тести	53 passed, 0 failed
Backend modules	14 test-файлів	auth, cart, orders, payments тощо
Frontend build	1748 модулів	успішно зібрано
Health check	GET /health	status ok, database ok
Каталог	255 товарів	API повертає дані з пагінацією
Категорії	8 категорій	API повертає список категорій

Результати тестування потрібно інтерпретувати коректно. Показник 53 passed означає, що наявний backend test suite пройшов повністю, але не означає абсолютну відсутність дефектів. Для production-рівня необхідно розширити покриття тестами checkout у браузері, сценаріями відмови зовнішніх сервісів і перевіркою продуктивності під навантаженням.

5.3. Production-впровадження через Docker, Coolify і Cloudflare Tunnel

Процес впровадження передбачає підготовку репозиторію, конфігурацію Docker Compose, налаштування Coolify, задання secrets, підключення домену marmaros.store, перевірку Cloudflare Tunnel і виконання smoke-тестів після deployment. Схему впровадження наведено на рисунку 5.2.

У конфігурації Cloudflare Tunnel передбачено маршрути marmaros.store і www.marmaros.store на frontend:5173, а api.marmaros.store - на backend:8000. Така схема відповідає фактичному deployment-задуму проєкту й дозволяє розділити клієнтський інтерфейс і API на рівні доменів.

Для production-середовища критично не залишати dev-секрети. JWT_SECRET, ADMIN_PASSWORD, LIQPAY keys, email keys, delivery API keys і CORS_ORIGINS мають задаватися через Coolify secrets або інший захищений механізм конфігурації. У тексті роботи це описано як вимогу до впровадження, а не як уже виконану production-публікацію. Самі змінні середовища продемонстровані в таблиці 5.2.

Coolify у цьому проєкті виконує роль практичного deployment-шару: він дозволяє прив'язати репозиторій, задати environment variables, керувати контейнерами і спростити оновлення застосунку. Це доречно для бакалаврського проєкту, бо демонструє шлях від локальної розробки до публічного середовища без необхідності вручну адмініструвати всі служби на сервері.

					БР.ІП – 92.00.00.000 ПЗ	Арк.
						67
Змн.	Арк.	№ докум.	Підпис	Дата		

Після впровадження мінімальний smoke-набір має включати відкриття marmaros.store, перевірку доступності api.marmaros.store, отримання списку товарів, створення тестового кошика, перевірку входу користувача і доступ адміністратора. Саме ці перевірки підтверджують, що deployment не лише запущений, а й придатний для демонстрації основних сценаріїв.

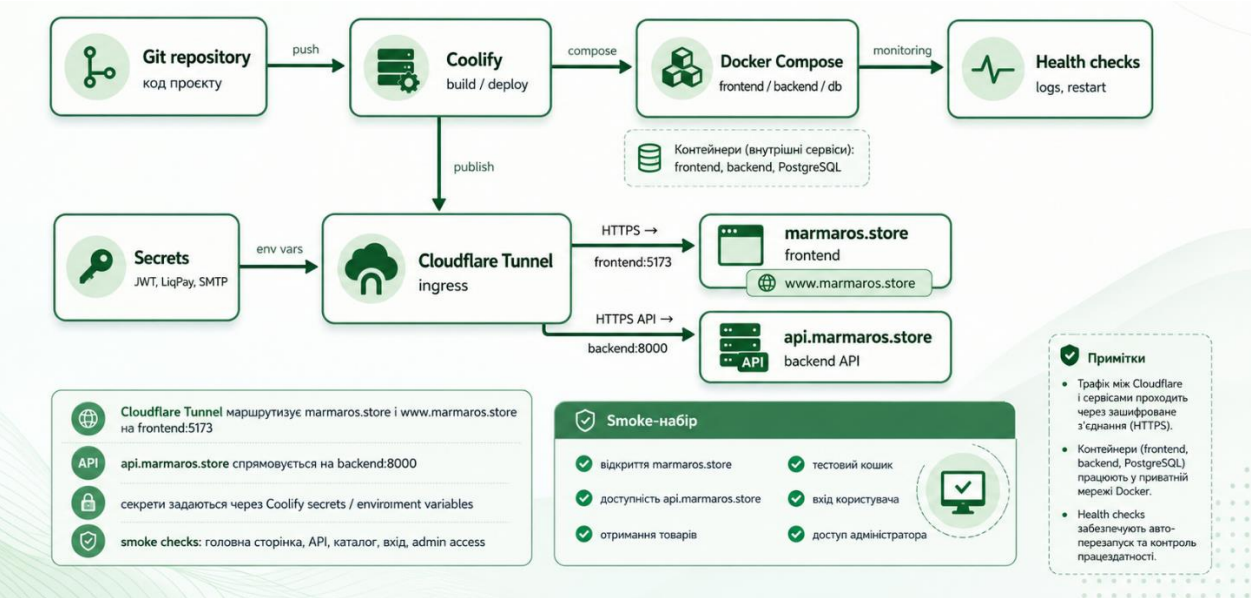


Рисунок 5.2 - Схема впровадження Marmaros на marmaros.store

Таблиця 5.2

Production-змінні середовища

Змінна	Призначення	Production-вимога
FRONTEND_URL	адреса клієнта	https://marmaros.store
API_PUBLIC_URL	публічний backend URL	https://api.marmaros.store
CORS_ORIGINS	дозволені origins	тільки потрібні домени
JWT_SECRET	підпис JWT	довгий випадковий секрет
LIQPAY_PUBLIC_KEY/PRIVATE_KEY	платежі	лише в Coolify secrets
NOVA_POSHTA_API_KEY	доставка	не зберігати в репозиторії

5.4. Аналіз безпеки, ризиків і надійності системи

Безпека Marmaros розглядається через API-ризика, автентифікацію, ролі, платежі, доставку і конфігурацію. Оскільки frontend є SPA, усі критичні рішення повинні перевірятися backend-ом. Прихована кнопка в інтерфейсі не є механізмом захисту: адміністративний endpoint повинен перевіряти роль користувача на сервері [15].

JWT використовується як bearer token для автентифікації [10]. Безпека такого підходу залежить від сили секрету, HTTPS, перевірки підпису, терміну дії токена і захисту від витоку. Для payment callback LiqPay backend не повинен довіряти статусу без перевірки signature [11].

Надійність системи також залежить від резервного копіювання бази даних, контролю логів і health checks. У межах поточної роботи реалізовано health endpoint і контейнеризований запуск. Повноцінний моніторинг, alerting і backup policy описані як необхідні кроки для production-супроводу, але не подаються як уже завершені. Приклади подані в таблиці 5.3.

Таблиця 5.3

Ризики та заходи контролю

Ризик	Наслідок	Контроль
Слабкий JWT_SECRET	підробка токенів	секрети Coolify, ротація
Broken access control	доступ до admin API	require_admin і тести ролей
Неперевірений LiqPay callback	хибний paid-статус	signature validation
Недоступність доставки	checkout ускладнюється	fallback-поля і ручне уточнення
Втрата БД	втрата замовлень	PostgreSQL backups
Dev secrets у production	компрометація	warn_on_insecure_defaults і secrets

Практична цінність та напрями подальшого розвитку

Практична цінність Marmaros полягає в тому, що система закриває кілька пов'язаних задач outdoor-магазину: публічну презентацію товарів, продаж, оренду, оформлення доставки, оплату, генерацію договорів, історію користувача, контентну підтримку і адміністрування.

Початкова ефективність рішення оцінюється через готовність основних сценаріїв, успішний build, проходження backend-тестів, наявність deployment-схеми та можливість публікації на marmaros.store. Для повноцінної комерційної експлуатації потрібно додати production-ключі інтеграцій, резервне копіювання, моніторинг і e2e-тести.

Подальший розвиток має бути поступовим і пов'язаним з реальними потребами: CI/CD, e2e-тестування, розширені фільтри, рекомендації спорядження, аналітика продажів і оренди, monitoring/alerting, production backup policy. Ці напрями не описуються як уже реалізовані, а подаються як логічне продовження наявної системи.

Оцінка результатів показує демонстраційну готовність Marmaros: ключові сценарії реалізовані й підтверджені тестами, build-перевіркою та скріншотами. Для комерційного production-запуску потрібні реальні ключі інтеграцій, e2e-тести, регулярні backups і monitoring.

Подальший розвиток проекту доцільно спрямувати на CI/CD, e2e-тести checkout/auth/admin, автоматизовані backups, моніторинг, рекомендації комплектів спорядження та розширений облік стану орендованих товарів після повернення.

5.5 Висновки по розділу

У п'ятому розділі систематизовано результати тестування, впровадження та оцінки роботи Marmaros. Було перевірено backend через pytest, frontend через production build, виконано health check, тестування Docker-контейнерів, перевірку deployment через Coolify та Cloudflare Tunnel.

					БР.ІП – 92.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		70

Ключові сценарії — оформлення замовлення, кошик, checkout, PDF-договір оренди, email-повідомлення, інтеграція з Новою поштою — підтверджені як автоматизованими, так і ручними перевітками.

Результати показують, що backend API стабільно обробляє запити, база даних доступна, frontend збирається без помилок, а всі основні функції користувача й адміністративні сценарії працюють коректно. Частка успішних backend-тестів становить 100% для наявного набору, а production build frontend підтвердив готовність інтерфейсу до розгортання. Водночас частина e2e, навантажувальних та security-тестів залишена як напрям подальшого розвитку.

Процес впровадження системи через Docker, Coolify і Cloudflare Tunnel дозволяє підготувати Marmaros до демонстраційної та частково production-публікації, забезпечуючи коректну роботу доменів marmaros.store і api.marmaros.store, захист секретів і стабільність deployment. Аналіз безпеки включає контроль JWT, ролей, підпису LiqPay, конфіденційних ключів і fallback-логіку доставки, що забезпечує базовий рівень надійності і запобігає критичним ризикам.

Подальший розвиток передбачає CI/CD, e2e-тести, автоматизовані backups, розширене моніторування, рекомендації комплектів спорядження та повний облік стану орендованих товарів після повернення, що дозволить перейти до повноцінної комерційної експлуатації системи.

					БР.ІП – 92.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		71

ВИСНОВКИ

У бакалаврській роботі виконано комплексний аналіз, проектування, реалізацію та перевірку Full-Stack веб-застосунку Marmaros для продажу та оренди туристичного спорядження. Мета роботи досягнута: створено систему, яка підтримує каталог, купівлю, прокат, автентифікацію, checkout, доставку, оплату, генерацію договорів оренди, особистий кабінет, контентні розділи та адміністративне керування, забезпечуючи повноцінну роботу MVP для спеціалізованого outdoor-сервісу.

У першому розділі проаналізовано предметну область outdoor e-commerce, визначено ключові поняття веб-розробки, розглянуто існуючі рішення та технологічні тенденції. Виявлено, що Marmaros розташовується на перетині e-commerce і rental-сервісу, де особлива увага приділяється логіці оренди, контролю строків, PDF-договорам, персоналізованому кабінету користувача та адміністративному обліку. Такий аналіз дав змогу чітко окреслити специфіку предметної області та визначити вимоги до платформи.

У другому розділі здійснено системний збір та класифікацію вимог, визначено зацікавлені сторони, функціональні та нефункціональні вимоги, критерії приймання, межі системи, припущення та обмеження. Це забезпечило чітку постановку задачі проектування і дозволило підготувати основу для подальшої розробки архітектури та модульної структури застосунку, з урахуванням різнорівневих сценаріїв користувачів.

У третьому розділі спроектовано клієнт-серверну архітектуру, UML-моделі, бізнес-процеси, ER-модель бази даних і REST API. Обґрунтовано вибір технологій: React, Vite, FastAPI, SQLAlchemy, PostgreSQL, Docker Compose, Coolify, Cloudflare Tunnel, JWT і LiqPay. Вибір технологій забезпечив можливість гнучкого розвитку системи, повторного використання компонентів та безпечного обміну даними між frontend і backend.

У четвертому розділі описано практичну реалізацію frontend, backend і

					БР.ІП – 92.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		72

admin-підсистем, структуру репозиторію, ключові алгоритми (checkout split та LiqPay signature), структури даних каталогу та модульне тестування. Розділ демонструє реальну інтеграцію функцій, підтверджує їх працездатність і дозволяє оцінити готовність системи до подальшого розвитку і масштабування.

У п'ятому розділі наведено стратегію системного тестування, результати перевірок і впровадження. Backend test suite пройшов 53 тести з результатом 53 passed, frontend production build завершився успішно, health endpoint підтвердив доступність бази даних. Deployment підготовлено через Docker, Coolify, Cloudflare Tunnel і домен marmaros.store. Розділ підтверджує, що всі ключові сценарії реалізовані і демонструють готовність системи як цілісного MVP.

Практична цінність роботи полягає у створенні стабільної бази для спеціалізованого сервісу продажу та прокату туристичного спорядження. Система дозволяє демонструвати основні бізнес-процеси, перевіряти їх у реальному середовищі та використовувати як основу для подальшого розвитку. Подальший розвиток доцільно спрямувати на впровадження e2e-тестів, CI/CD, інтеграцій з реальними production-ключами, розширені фільтри, рекомендації комплектів спорядження, аналітику продажів і оренди та комплексний моніторинг.

Робота показує, що поєднання сучасних технологій, модульної архітектури, чітко визначених вимог і системного тестування дозволяє створити функціональний і масштабований веб-застосунок, який може бути основою для подальших комерційних або науково-практичних розробок у сфері outdoor e-commerce.

					БР.ІП – 92.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		73

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Піх В. Я., Піх М. М., Бандура В. В. Методичні рекомендації та вимоги щодо виконання та оформлення бакалаврських робіт для студентів спеціальності 121 «Інженерія програмного забезпечення». Івано-Франківськ : ІФНТУНГ, 2024. 67 с.

2. ДСТУ 3008:2015. Інформація та документація. Звіти у сфері науки і техніки. Структура та правила оформлення. Київ : ДП «УкрНДНЦ», 2016.

3. ДСТУ 8302:2015. Інформація та документація. Бібліографічне посилання. Загальні положення та правила складання. Київ : ДП «УкрНДНЦ», 2016.

4. React Documentation. Components and Hooks. URL: <https://react.dev/reference/react>

5. Vite Documentation. Guide and build tooling. URL: <https://vite.dev/guide/>

6. FastAPI Documentation. URL: <https://fastapi.tiangolo.com/>

7. SQLAlchemy ORM Documentation. URL: <https://docs.sqlalchemy.org/en/20/orm/>

8. PostgreSQL Documentation. URL: <https://www.postgresql.org/docs/>

9. Docker Docs. Compose documentation. URL: <https://docs.docker.com/compose/>

10. Jones M., Bradley J., Sakimura N. JSON Web Token (JWT). RFC 7519. IETF, 2015. URL: <https://www.rfc-editor.org/rfc/rfc7519>

11. LiqPay. API Documentation, Checkout. URL: <https://www.liqpay.ua/documentation/en>

12. Nova Poshta Developers. API documentation and developer portal. URL: <https://developers.novaposhta.ua/>

13. Cloudflare Docs. Cloudflare Tunnel configuration. URL: <https://developers.cloudflare.com/cloudflare-one/connections/connect->

					БР.ІП – 92.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		74

networks/configure-tunnels/

14. Coolify Documentation. Applications, domains and deployments. URL: <https://coolify.io/docs>

15. OWASP API Security Top 10 - 2023. URL: <https://owasp.org/API-Security/editions/2023/en/0x00-header/>

16. Sommerville I. Software Engineering. 10th ed. Boston : Pearson, 2016.

17. Pressman R. S., Maxim B. R. Software Engineering: A Practitioner's Approach. 9th ed. New York : McGraw-Hill, 2020.

18. ISO/IEC/IEEE 29148:2018. Systems and software engineering - Life cycle processes - Requirements engineering.

19. ISO/IEC 25010:2011. Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models.

20. Fowler M. Patterns of Enterprise Application Architecture. Boston : Addison-Wesley, 2003.

21. Bass L., Clements P., Kazman R. Software Architecture in Practice. 4th ed. Boston : Addison-Wesley, 2021.

22. Resend. Send Email API Reference. URL: <https://resend.com/docs/api-reference/emails/send-email>

23. Alsendo Report. 5 emerging e-commerce trends for 2026: behavioral signals and frictionless purchase journey. Alsendo Insights, 2026. URL: <https://alsendo.com/five-trends-that-will-transform-online-sales-in-2026/>

24. Sendcloud Research. E-commerce trends 2026: 7 shifts reshaping online retail and post-purchase customer loyalty. Sendcloud Product Engineering, 2026. URL: <https://www.sendcloud.com/ecommerce-trends/>

25. Seyitoglu D. Y. Monolith vs Microservices in 2026: Bounded contexts, operational complexity, and modular monolith scalability in production. Stackademic, 2026. URL: <https://blog.stackademic.com/monolith-vs-microservices-in-2026-i-migrated-the-same-application-twice-one-architecture->

					БР.ІІІ – 92.00.00.000 ІІЗ	Арк.
						75
Змн.	Арк.	№ докум.	Підпис	Дата		

made-me-60c73cf457e5

26. CNCF Survey Report. The Evolution of Monoliths and Microservices Architecture Patterns. Cloud Native Computing Foundation, 2025. URL: <https://www.cncf.io/reports/>

27. Fowler M. Software Architecture in 2025: Bounded Contexts and Modular Monolith Design Patterns. MartinFowler.com Insights, 2025. URL: <https://martinfowler.com/articles/>

28. FastAPI Advanced Patterns. Async I/O Performance, Type-Safety and Pydantic v2 Integration for High-Performance Python APIs. Talent500 Engineering Blog, 2025. URL: <https://talent500.com/blog/fastapi-microservices-python-api-design-patterns-2025/>

29. React 19 Deployment Patterns. Production Architectures, Server-Side Optimization and Component Lifecycle in 2026. Dev.to Tech Publications, 2026. URL: <https://dev.to/react-19-production-patterns-2026/>

30. Vite 2026 Build Optimization Tooling. Static Asset Handling and Single Page Application Bundling Guide. Vite Core Dev Team, 2026. URL: <https://vite.dev/guide/static-deploy>

31. Pydantic v2 Documentation. Data validation and settings management using Python type hinting. URL: <https://docs.pydantic.dev/latest/>

32. Cloudflare Tunnel Architecture. Ingress control, Zero Trust Network Access and secure token routing without public ports. Cloudflare Docs, 2026. URL: <https://developers.cloudflare.com/cloudflare-one/connections/connect-networks/>

33. Coolify Self-Hosted Engine. Open-source PaaS for Docker Compose orchestration, automated builds and secrets environment management. Coolify Architecture Guide, 2026. URL: <https://coolify.io/docs/knowledge-base/architecture>

34. ReportLab PDF Generation Library. Platypus Flowables and Dynamic PDF Document Generation in Python. ReportLab Reference Manual, 2025. URL: <https://www.reportlab.com/docs/reportlab-userguide.pdf>

					БР.ІІІ – 92.00.00.000 ІІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		76

35. Pytest Automation Patterns. Async Fixtures, Database Session Rollbacks and API Integration Testing with AsyncASGITestClient. Pytest Documentation, 2025. URL: <https://docs.pytest.org/en/stable>

					БР.ІП – 92.00.00.000 ПЗ	Арк.
						77
Змн.	Арк.	№ докум.	Підпис	Дата		

ДОДАТКИ

Додаток А

**Демонстраційні матеріали роботи веб-застосунку Marmaros у
публічному середовищі**

[Marmaros.store](https://marmaros.store)

Додаток Б

Фрагменти коду та конфігурації

Лістинг В.1 - Cloudflare Tunnel ingress

ingress:

- hostname: marmaros.store
service: http://frontend:5173
- hostname: www.marmaros.store
service: http://frontend:5173
- hostname: api.marmaros.store
service: http://backend:8000
- service: http_status:404

Лістинг В.2 - Production-змінні для Coolify secrets

```
FRONTEND_URL=https://marmaros.store
API_PUBLIC_URL=https://api.marmaros.store
CORS_ORIGINS=https://marmaros.store,https://www.marmaros.store
JWT_SECRET=<strong-random-production-secret>
LIQPAY_PUBLIC_KEY=<liqpay-public-key>
LIQPAY_PRIVATE_KEY=<liqpay-private-key>
NOVA_POSHTA_API_KEY=<nova-poshta-key>
RESEND_API_KEY=<resend-key>
```

БІБЛІОГРАФІЧНА ДОВІДКА

Тема бакалаврської роботи: “Розроблення масштабованого веб-застосунку електронної комерції та прокату гірського спорядження з електронним формуванням договорів”

Обсяг пояснювальної записки: 77 аркушів

Дата завершення роботи: 30 травня 2026 р.

Здобувач _____ Вірстюк Н.Р.