Прикарпатський національний університет імені Василя Стефаника
Факультет математики та інформатики
Кафедра комп'ютерних наук та інформаційних систем

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
першого (бакалаврського) рівня освіти
на тему:

“Розробка веб-застосунку туристичних товарів з інтеграцією AI-чату для персоналізованих рекомендацій”

Виконав: студент IV курсу гр. ІСТ-41
спеціальності 126 “Інформаційні системи та технології”
Цибульський О.В.

Науковий керівник: наук. ступінь,
доц. к.т.н. Семаньків М.В.

Рецензент:
доц. к.т.н. Іляш Ю.Ю.

м. Івано-Франківськ – 2025

ЗМІСТ

АНОТАЦІЯ

У бакалаврській роботі представлено розробку веб-платформи для автоматизованого підбору туристичних товарів із використанням штучного інтелекту. Робота спрямована на вирішення проблем пошуку спорядження, забезпечення персоналізованих рекомендацій і підвищення зручності онлайн-покупок. Розроблено адаптивний фронтенд (React, TypeScript, SCSS) з архітектурою Feature-Sliced Design, бекенд на Django і FastAPI із REST API, інтеграцію з OpenAI API для ШІ-рекомендацій, аутентифікацію через GitHub OAuth і кешування в Redis. Система розгорнута в Docker із CI/CD через Jenkins.
Обсяг роботи — 88 сторінок.
Дипломна робота містить 3 розділи, 3 додатки, 3 рисунків.
Ключові слова: туристичні товари, штучний інтелект, веб-платформа, персоналізація, автоматизація.

ABSTRACT

The bachelor’s thesis presents the development of a web platform for automated selection of tourism products using artificial intelligence methods. The work addresses challenges in equipment search, providing personalized recommendations, and enhancing the convenience of online shopping. An adaptive frontend (React, TypeScript, SCSS) with Feature-Sliced Design architecture, a backend on Django and FastAPI with REST API, integration with OpenAI API for AI recommendations, authentication via GitHub OAuth, and caching in Redis were developed. The system is deployed in Docker with CI/CD via Jenkins.
The volume of the work is 88 pages.
The thesis contains 3 chapters, 3 appendices, 36 figures.
Keywords: tourism products, artificial intelligence, web platform, personalization, automation.

ВСТУП

Сучасний розвиток інформаційних технологій відкриває нові можливості для автоматизації та персоналізації в різних сферах, зокрема в туризмі. Традиційні підходи до підбору туристичних товарів часто є трудомісткими, не враховують індивідуальних потреб користувачів і залежать від ручного аналізу пропозицій. Впровадження веб-платформ із інтеграцією штучного інтелекту дозволяє автоматизувати процес підбору товарів, адаптувати рекомендації до запитів користувачів і значно підвищити зручність взаємодії з платформою. Такі рішення стають дедалі актуальнішими в умовах зростання попиту на персоналізовані туристичні продукти.

Актуальність теми. Розробка веб-платформ для підбору туристичних товарів із застосуванням штучного інтелекту є затребуваною в сучасному світі, де користувачі очікують швидких, точних і персоналізованих рішень. Існуючі платформи, такі як Booking або TripAdvisor, часто пропонують загальні рекомендації, які не враховують специфічні потреби користувачів, наприклад, погодні умови, рівень підготовки чи тип подорожі. Інтеграція штучного інтелекту для аналізу запитів користувачів і автоматичної генерації рекомендацій дозволяє створити унікальний користувацький досвід, зменшити час на пошук товарів і підвищити задоволеність клієнтів. Це дослідження спрямоване на розробку версустосунку, який автоматизує підбір туристичних товарів за допомогою штучного інтелекту, що робить процес більш ефективним і доступним.

Об’єктом дослідження є процес автоматизованого підбору туристичних товарів у веб-застосунках.

Предметом дослідження є розробка та впровадження веб-платформи з інтеграцією штучного інтелекту для автоматичної генерації персоналізованих рекомендацій туристичних товарів на основі запитів користувачів.

Метою дослідження є розробка та впровадження веб-платформи для підбору туристичних товарів із застосуванням штучного інтелекту, що забезпечує автоматичну генерацію персоналізованих рекомендацій, підвищуючи ефективність і зручність процесу вибору товарів.

Для досягнення цієї мети необхідно вирішити такі **завдання**:

- проаналізувати існуючі платформи для підбору туристичних товарів, їхні переваги та недоліки;
- розробити архітектуру системи, визначити стек технологій і структуру мікросервісів;
- створити адаптивний веб-інтерфейс платформи для зручного керування функціями;
- реалізувати механізм генерації персоналізованих рекомендацій за допомогою штучного інтелекту на основі запитів користувачів і зовнішніх даних (погода, профіль користувача);
- протестувати ефективність автоматично згенерованих рекомендацій і порівняти їх із традиційними методами підбору;
- оцінити можливості інтеграції платформи з іншими сервісами та перспективи її подальшого розвитку.

Методи дослідження. Для виконання роботи використано такі методи:

- теоретичний аналіз наукових джерел і документацій щодо веб-розробки та штучного інтелекту;
- порівняльний аналіз функціоналу існуючих туристичних платформ;
- проєктування та моделювання мікросервісної архітектури платформи;
- методи машинного навчання для обробки запитів і генерації рекомендацій;
- експериментальне тестування платформи та аналіз отриманих результатів.

Структура роботи. Бакалаврська робота складається з вступу, трьох розділів основної частини, висновків, списку використаних джерел і додатків. У першому розділі аналізуються сучасні підходи до розробки веб-платформ із інтеграцією штучного інтелекту та обґрунтовується необхідність створення власного версустосунку. У другому розділі описується процес розробки платформи, включно з архітектурою, вибором технологій (React, Django, FastAPI), структурою мікросервісів і алгоритмом генерації рекомендацій. У третьому розділі розглядаються результати тестування платформи, аналізуються переваги та недоліки автоматично згенерованих рекомендацій, а також визначаються можливі напрямки подальшого розвитку.

Розділ 1 ДОСЛІДЖЕННЯ ОБЛАСТІ ТА АНАЛІЗ СУЧАСНИХ ПІДХОДІВ

1.1 Опис проблеми

Сучасний ринок туристичних товарів та послуг визначається гострою конкуренцією та збільшенням запиту на персоналізовані пропозиції. Туристи, незалежно від рівня досвіду чи виду подорожі (гірський похід, пляжний відпочинок, екстремальний туризм), прагнуть отримувати швидкі, точні та пристосовані до їхніх потреб рекомендації щодо спорядження, одягу чи інших товарів. [13] Звичайні підходи до підбору туристичних товарів, на кшталт ручного пошуку у фізичних магазинах або перегляду каталогів на веб-сайтах, мають кілька недоліків:

- **Трудомісткість процесу.** Користувачам доводиться самостійно аналізувати величезну кількість інформації, порівнювати товари за характеристиками, враховувати погодні умови, складність маршруту або власні вподобання. Це потребує значного часу та може бути важким для новачків у туризмі.
- **Відсутність персоналізації.** Більшість туристичних платформ пропонують стандартні набори товарів, які не враховують індивідуальних факторів, наприклад, погодні умови, фізична підготовка або особливості маршруту. Наприклад, туристу, який планує похід у Карпати взимку, можуть запропонувати товари, що не відповідають низьким температурам або високій вологості. [1]
- **Обмежена інтерактивність.** Багато платформ не дають можливості уточнити запит у звичній формі (наприклад, "рекомендації для походу в Альпи для початківців"), що ускладнює взаємодію та погіршує зручність використання.
- **Недостатня адаптивність до мобільних пристроїв.** Значна частина користувачів шукає товари зі смартфонів, але не всі платформи забезпечують зручний і швидкий адаптивний інтерфейс, що знижує комфорт користування. [2]

- **Обмежена інтеграція зовнішніх даних.** Традиційні платформи рідко беруть до уваги динамічні фактори, такі як популярність маршрутів чи актуальні відгуки, що знижує релевантність рекомендацій. [13]

Ці проблеми створюють потребу в розробці інноваційної веб-платформи, яка б автоматизувала процес підбору туристичних товарів, використовуючи штучний інтелект для аналізу запитів користувачів та зовнішніх даних. Така платформа має забезпечувати персоналізовані рекомендації, враховуючи індивідуальні потреби (наприклад, тип подорожі, рівень підготовки, погодні умови), бути зручною для користування на різних пристроях та підтримувати інтерактивну взаємодію через текстові запити. У контексті вашого проекту, верстудстосунок має вирішити ці проблеми шляхом інтеграції штучного інтелекту для генерування персоналізованих рекомендацій, реалізації адаптивного інтерфейсу на основі React та мікросервісної архітектури з використанням Django та FastAPI для обробки даних і запитів до ШІ. [12]

Актуальність проблеми підкреслюється ростом популярності туризму, зокрема активного відпочинку, а також поширенням цифрових технологій, які дозволяють користувачам очікувати швидких і точних рішень. Розробка верстудстосунку з інтеграцією ШІ відповідає сучасним вимогам ринку, підвищуючи ефективність вибору товарів та покращуючи користувацький досвід. [13]

1.2 Огляд існуючих рішень і практик застосування ШІ на ринку туристичних платформ

Розвиток ринку туристичного обладнання, одягу та екіпірування в Україні набирає обертів через зростаючий інтерес до активного дозвілля, зокрема до піших мандрівок, гірських сходжень та інших видів туризму. Українські онлайн-магазини пропонують великий вибір продукції, включаючи намети, рюкзаки, спальні мішки, трекінгове взуття та супутні товари, однак їх можливості часто обмежені

через відсутність передових технологій, таких як штучний інтелект (ШІ). ШІ міг

би суттєво покращити індивідуальний підхід, автоматизацію та легкість у виборі товарів, але на даний момент подібні рішення майже не використовуються в українських магазинах. Далі подано огляд чотирьох відомих українських інтернет-магазинів туристичних товарів, їхні позитивні та негативні сторони, а також аналіз практики використання ШІ, що акцентує на прогалинах ринку, які покликаний вирішити ваш застосунок. [12]

1.2.1 Українські туристичні платформи

Gorgany

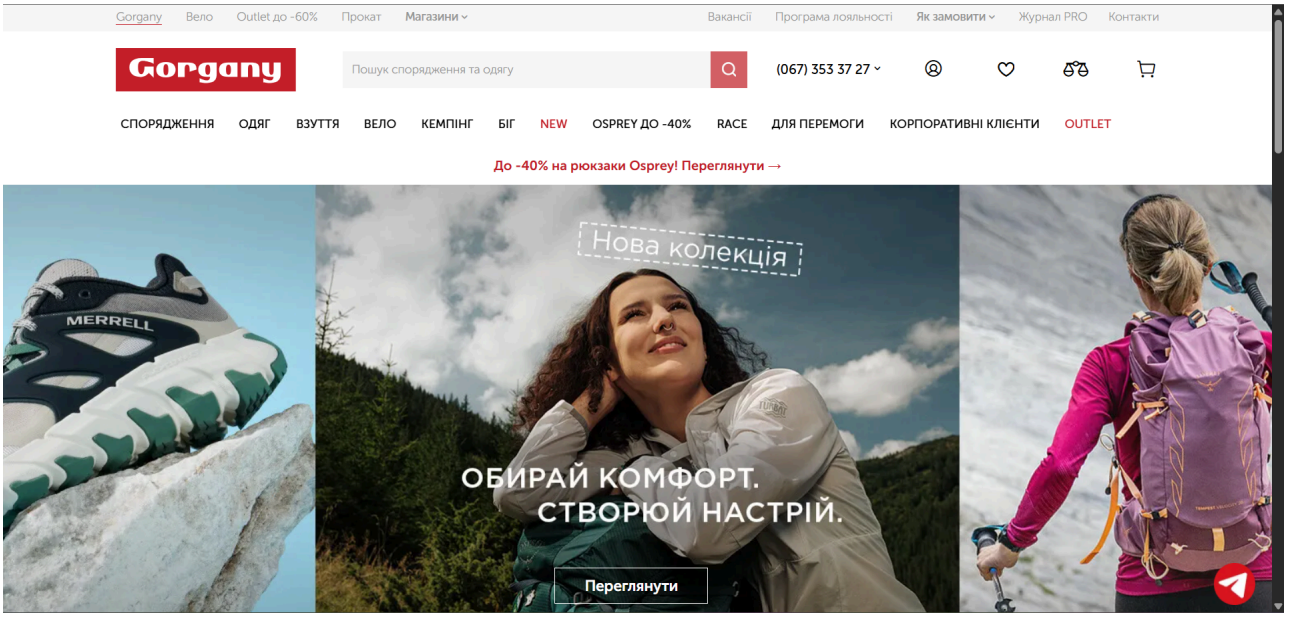

Рисунок 1.1 – Gorgany

Опис: Gorgany – один з найбільших вітчизняних онлайн-магазинів, спеціалізований на туристичному обладнанні, одязі та взутті для походів, скелелазіння, кемпінгу й активного дозвілля. Платформа реалізує продукцію провідних марок, як-от The North Face, Salewa та Deuter, та послуговується базовими алгоритмами для фільтрування та впорядкування товарів за популярністю чи вартістю.

Переваги:

- Широкий вибір товарів, включаючи професійне спорядження для альпінізму й легке екіпірування для походів.
- Адаптивний інтерфейс, який частково підтримує мобільні пристрої, гарантуючи зручність покупок.
- Детальні описи товарів з відгуками користувачів, що сприяють оцінюванню якості. [2]

Недоліки:

- Відсутність ШІ: платформа не залучає штучний інтелект для індивідуальних рекомендацій, що обмежує здатність адаптації пропозицій до особистих потреб, наприклад, погодних умов чи рівня підготовки.
- Немає підтримки текстових запитів природньою мовою (приміром, «спорядження для зимового походу в Карпати»).
- Відсутність інтеграції зовнішніх даних, як-от прогноз погоди чи геолокація, що зменшує релевантність пропозицій.
- Обмежена інтерактивність: користувачі змушені покладатися на ручні фільтри, що ускладнює швидкий підбір товарів.
- Недостатня оптимізація мобільного інтерфейсу для застарілих пристроїв, що може спричиняти незручності. [12]

Terra Incognita

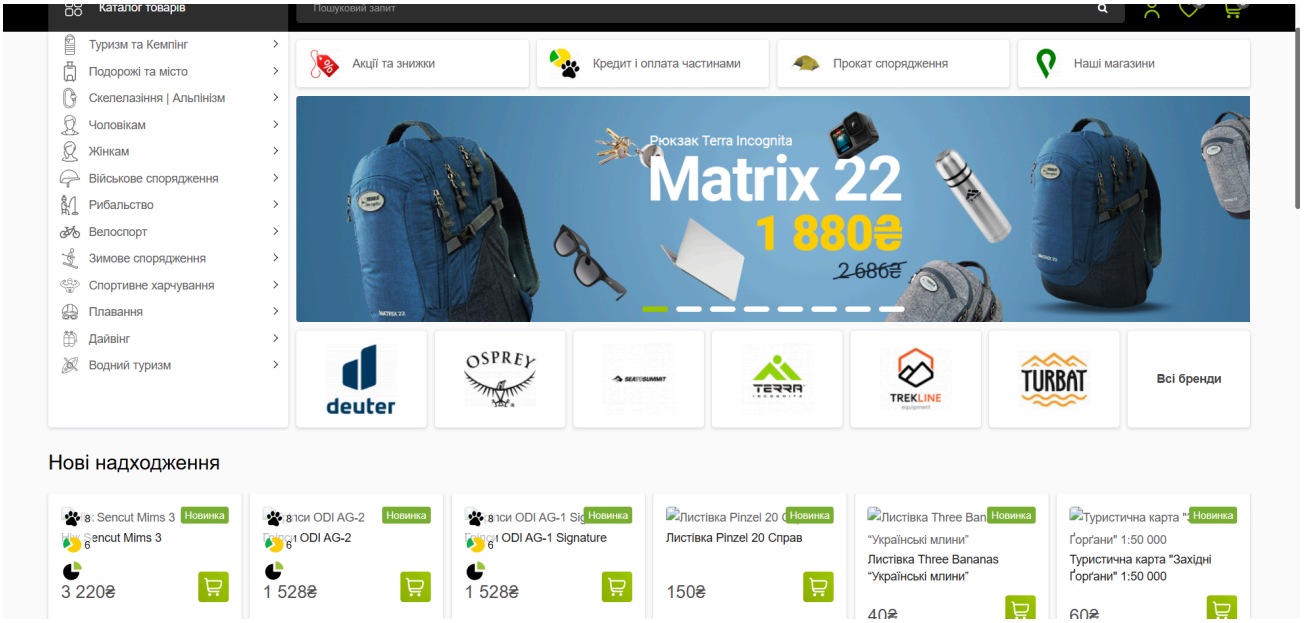

Рисунок 1.2 – Terra Incognita

Опис: Terra Incognita – відомий український інтернет-магазин, що пропонує туристичне спорядження, включаючи намети, спальники, рюкзаки, одяг та аксесуари для альпінізму і кемпінгу. Платформа позиціонує себе як постачальник якісного екіпірування за привабливими цінами, проте її **функціонал базується на ручних фільтрах та категоріях.** [10]

Переваги:

- Великий вибір товарів від світових брендів за конкурентними цінами.
- Наявність професійних консультацій через чат або телефон, що допомагає клієнтам із вибором. [6]

Недоліки:

- Відсутність ШІ: платформа не використовує штучний інтелект для аналізу потреб користувачів чи генерації персоналізованих рекомендацій.
- Немає можливості вводити текстові запити для уточнення вимог (наприклад, «рекомендації для екстремального туризму»).
- Відсутність інтеграції із зовнішніми даними, як-от погодні сервіси, що обмежує контекстну релевантність пропозицій.

- Обмежена адаптивність інтерфейсу для мобільних пристроїв, що зменшує зручність для користувачів смартфонів.
- Відсутність автоматизованих рекомендаційних систем, що змушує клієнтів самостійно аналізувати асортимент. [12]

Palatka

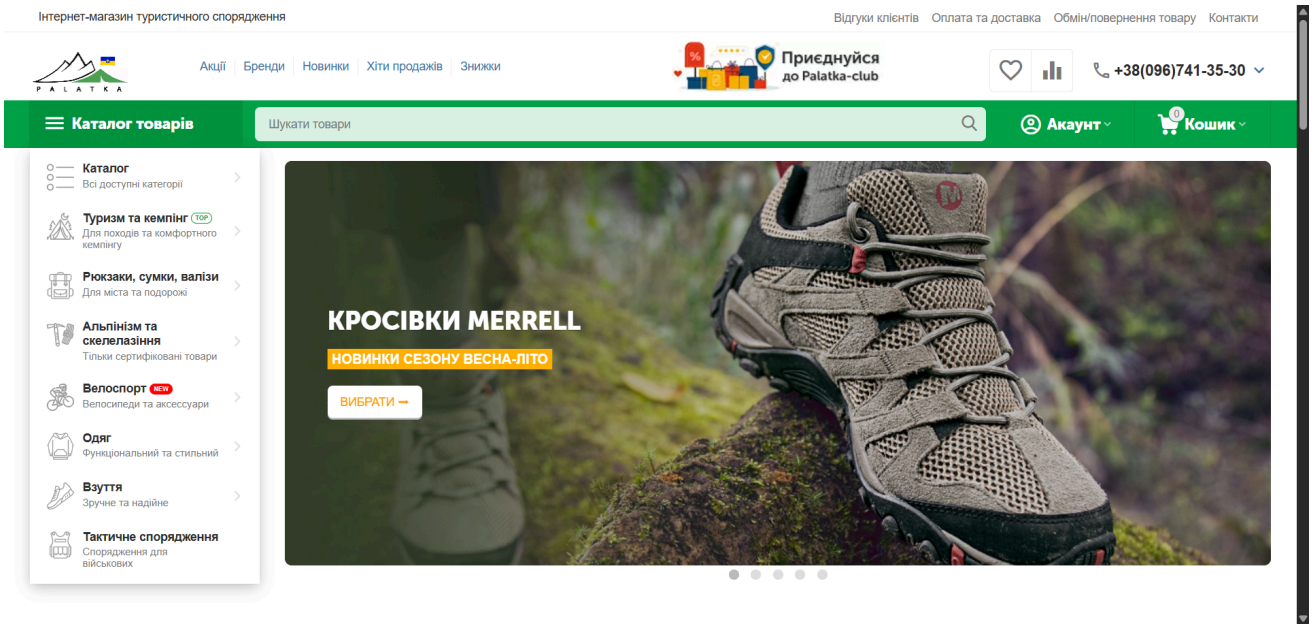

Рисунок 1.3 — Palatka

Опис: Palatka.com.ua – онлайн-магазин, який спеціалізується на туристичному спорядженні, зокрема наметах, спальниках, туристичному посуді та обладнанні для альпінізму. Платформа реалізує сертифіковані товари від відомих виробників, але її функціональність обмежена базовими фільтрами за вартістю та категоріями.

Переваги:

- Спеціалізація на туристичному спорядженні, включно з професійним обладнанням для альпінізму.
- Актуальна наявність товарів і швидка доставка по Україні.

Недоліки:

- Відсутність ШІ: платформа не залучає штучний інтелект для персоналізації чи автоматизації підбору товарів.

- Немає підтримки інтерактивних текстових запитів, що ускладнює швидкий пошук товарів за специфічними потребами.
- Відсутність інтеграції з зовнішніми даними, як-от погода чи відгуки з інших джерел, що знижує точність пропозицій.
- Слабка адаптивність інтерфейсу для мобільних пристроїв, що створює незручності для користувачів.
- Відсутність рекомендаційних алгоритмів, що вимагає від клієнтів ручного перегляду каталогу. [10]

Fram Equipment

Рисунок 1.4 – Fram Equipment

Опис: Fram Equipment – український виробник та інтернет-магазин туристичного спорядження, що пропонує легкі намети, спальники, одяг та аксесуари, протестовані спортсменами. Платформа зосереджується на товарах для тривалих походів та екстремального туризму, але її функціонал обмежений базовими фільтрами й сортуванням.

Переваги:

- Власне українське виробництво, що забезпечує доступні ціни та високу якість.
- Орієнтація на легке спорядження, яке випробуване в реальних умовах.

Недоліки:

- Відсутність ШІ: платформа не використовує штучний інтелект для аналізу запитів чи персоналізації рекомендацій.
- Обмежений асортимент у порівнянні з більшими магазинами, як-от Gorgany.
- Немає підтримки текстових запитів для уточнення потреб користувачів.
- Відсутність інтеграції зовнішніх даних, таких як погодні умови чи геолокація.
- Слабка адаптивність мобільного інтерфейсу, що зменшує зручність для користувачів смартфонів.

1.2.2 Практики застосування ШІ в туристичних платформах

Штучний інтелект (ШІ) в сфері українських туристичних платформ застосовується наразі з певними обмеженнями, але існують приклади ключових практик, які підкреслюють його потенціал:

1. Аналіз даних і рекомендації. Платформи, такі як TripMyDream, вдаються до елементарних алгоритмів машинного навчання з метою зіставлення цін та турпропозицій. Варто зазначити, згідно з Zhang Y. Big Data Analytics in Tourism. Journal of Travel Research, 2023, ці алгоритми, здебільшого, не враховують індивідуальні побажання користувачів, що впливає на їхню продуктивність.
2. Обробка відгуків. TravelHub та схожі платформи використовують ШІ для аналізу зворотного зв'язку, що допомагає в оцінці якості турів та обраних

місць. Проте, в даних рішеннях не застосовуються моделі обробки природної мови (NLP) для глибокого аналізу чи інтерактивних запитів.

3. Чат-боти. Ряд українських туристичних організацій, наприклад, туроператори, інтегрують чат-боти на основі ШІ для відповідей на типові питання користувачів. Водночас, функціональність цих чат-ботів обмежена, і вони не можуть обробляти складні запити, як вказується в Gao J. Conversational AI with Dialogflow. Apress, 2022
4. Прогнозування попиту. Значні туристичні агенції України починають використовувати ШІ для аналізу тенденцій ринку та передбачення попиту на конкретні напрямки. Проте, такі рішення переважно застосовуються на корпоративному рівні, а не у веб-додатках для широкого загалу. [9]

1.3 Аналіз літератури з розробки веб-застосунків

Розробка веб-застосунків є однією з ключових ланок ІТ-сфери, що безперервно еволюціонує через появу нових інструментів, фреймворків та архітектурних підходів. Сучасні веб-застосунки визначаються високою інтерактивністю, адаптивністю до різних пристроїв і здатністю опрацьовувати великі масиви даних у режимі реального часу. Огляд літератури дає змогу виявити ключові тенденції, технології та виклики, які впливають на процес створення таких систем, особливо у контексті інтеграції штучного інтелекту для персоналізації, наприклад, у випадку веб-застосунку для туристичних товарів. [1]

Одним із основних джерел для аналізу є праці, присвячені фронтенд-розробці. Згідно з виданням React in Action (Smith J., Manning Publications, 2020), бібліотека React, розроблена Facebook, стала стандартом для створення динамічних і компонентно-орієнтованих інтерфейсів завдяки своїй гнучкості та підтримці

віртуального DOM, що суттєво збільшує продуктивність. [3]

У публікації Feature-Sliced Design: A Modern Approach to Frontend Architecture (Ivanov A., Medium, 2023) зазначається, що застосування модульних архітектур, на кшталт

Feature-Sliced Design (FSD), дозволяє організувати код таким чином, що спрощує масштабування та підтримку складних проєктів. FSD поділяє код на функціональні модулі (шари, слайси, сегменти), що особливо корисне для веб-застосунків з великою кількістю сторінок та функцій, як у веб-застосунку для туристичних товарів, де є сторінки авторизації, фільтрації, кошика тощо. [4]

На стороні бекенду література підкреслює зростаючу популярність мікросервісної архітектури. У книзі Building Microservices (Newman S., O'Reilly Media, 2021) описано, що мікросервіси дозволяють розділити функціональність застосунку на незалежні компоненти, що збільшує гнучкість і стійкість системи. Для реалізації бекенду часто використовують фреймворки, як-от Django та FastAPI. Згідно з The Definitive Guide to Django (Holovaty A., Kaplan-Moss J., Apress, 2022), Django забезпечує надійну основу для управління користувачами, обробки даних та реалізації бізнес-логіки завдяки вбудованим інструментам, таким як ORM (Object-Relational Mapping) та система аутентифікації. FastAPI, як зазначено в FastAPI: Modern Python Web Development (Ramírez G., Packt Publishing, 2023), є оптимальним рішенням для створення високопродуктивних API, особливо для інтеграції з компонентами штучного інтелекту, оскільки підтримує асинхронне програмування та автоматичну генерацію документації. [6]

Щодо адаптивності, у книзі Responsive Web Design (Coyier C., A Book Apart, 2021) наголошується на важливості створення веб-застосунків, які коректно відображаються на різних пристроях, включно з мобільними телефонами та планшетами. Бібліотеки стилізації, як-от Tailwind CSS, дають змогу швидко створювати адаптивні інтерфейси з мінімальними зусиллями, що є критично важливим для туристичних платформ, де користувачі часто взаємодіють із сайтом з мобільних пристроїв. У книзі Optimizing React Applications (Johnson R., SitePoint, 2022) також розглядаються інструменти для оптимізації продуктивності, такі як ледаче завантаження (lazy loading) у React, що зменшує час завантаження сторінок. [27]

Інтеграція штучного інтелекту у веб-застосунки також широко висвітлюється у літературі. У праці Deep Learning (Goodfellow I., Bengio Y., Courville A., MIT Press, 2016) описано, як бібліотеки машинного навчання, як-от TensorFlow та Hugging Face, дозволяють обробляти текстові запити та генерувати персоналізовані рекомендації. Зокрема, у книзі Natural Language Processing with Python (Brownlee J., Machine Learning Mastery, 2023) наведено приклади використання моделей обробки природної мови (NLP) для аналізу запитів користувачів, що є основою для реалізації рекомендаційних систем у туристичних платформах. Наприклад, аналіз текстових запитів про пункт призначення чи тип подорожі може бути використаний для створення персоналізованих пропозицій. [26]

Окремо варто виділити літературу, присвячену безпеці веб-застосунків. У The Web Application Hacker’s Handbook (Stuttard D., Pinto M., Wiley, 2022) розглядаються методи захисту від поширених атак, таких як XSS (Cross-Site Scripting) та CSRF (Cross-Site Request Forgery), а також важливість використання HTTPS та токенів аутентифікації, таких як JWT. Для туристичних платформ, де обробляються дані користувачів та фінансові транзакції, ці аспекти є критично важливими. [28]

Огляд літератури показує, що сучасна веб-розробка ґрунтується на поєднанні компонентно-орієнтованих фреймворків (React), мікросервісної архітектури (Django, FastAPI) та бібліотек штучного інтелекту (openai). Ці технології дають змогу створювати масштабовані, безпечні та клієнтоорієнтовані платформи, що відповідає потребам розробки веб-застосунку для туристичних товарів. Разом з тим, література вказує на виклики, такі як забезпечення високої продуктивності, оптимізація обробки даних ШІ та захист конфіденційної інформації, які потрібно враховувати під час розробки. [2]

1.4 Постановка задачі та обґрунтування вибору методів

Мета розробки верстування - створення веб-платформи, яка автоматизує процес підбору туристичних товарів, охоплюючи спорядження, одяг та аксесуари. Це досягається за допомогою інтеграції штучного інтелекту для формування персоналізованих рекомендацій. Платформа має забезпечити оперативний, зручний та максимально точний підбір товарів, базуючись на запитах користувачів і враховуючи такі індивідуальні аспекти, як тип подорожі, кліматичні умови, фізична підготовка та особисті вподобання. Для реалізації цієї мети слід вирішити наступні завдання: [4]

1. Розробити адаптивний веб-інтерфейс з усіма необхідними функціями, включаючи сторінки авторизації, фільтрації, кошика, улюблених товарів та детального опису продукції. [2]
2. Впровадити мікросервісну архітектуру, що передбачає наявність головного бекенду, відповідального за управління користувачами,

   товарами та покупками, а також ШІ-бекенду, який обробляє запити та генерує рекомендації.
3. Інтегрувати ШІ для аналізу текстових запитів користувачів, а також зовнішніх даних, таких як погодні умови та геолокація, з метою створення індивідуальних рекомендацій. [10]
4. Забезпечити високий рівень безпеки даних користувачів та фінансових транзакцій шляхом застосування шифрування та заходів захисту від потенційних кіберзагроз.
5. Налагодити автоматизацію тестування та розгортання за допомогою CI/CD, що дозволить полегшити підтримку та оновлення платформи.
6. Провести тестування ефективності рекомендацій, згенерованих ШІ, та порівняти їх з результатами, отриманими за допомогою традиційних методів підбору товарів. [10]

7. Основна задача - розробка масштабованого, безпечного та орієнтованого на користувача веб-застосунку, який вирішує проблеми, характерні для традиційних туристичних платформ, такі як нестача персоналізації, складнощі пошуку та обмежена інтерактивність. [2]

Висновоки:

Розділ перший зосереджується на розробці підходу до веб-додатку, який використовує штучний інтелект для автоматичного підбору туристичних продуктів. Проаналізувавши ситуацію, було чітко визначено проблему, оцінено стан ринку туристичних платформ в Україні та обґрунтовано методологічний підхід до реалізації цього проєкту.

Окреслена проблема виявила недоліки традиційних методів підбору туристичних пропозицій, зокрема їх трудомісткість, брак персоналізації, обмежену інтерактивність та погану адаптивність до мобільних пристроїв. Ці проблеми підкреслюють необхідність створення веб-платформи, яка буде використовувати штучний інтелект для створення персоналізованих рекомендацій, враховуючи індивідуальні побажання користувачів, такі як тип подорожі, погодні умови та рівень підготовки.

Аналіз існуючих рішень, зокрема українських платформ, як-от TripMyDream, TravelHub та Uvidpustku, показав, що більшість з них акцентують увагу на бронюванні турів та готелів, а не на підборі туристичних товарів. Хоча ці платформи використовують базові алгоритми машинного навчання для порівняння цін чи аналізу відгуків, вони мають значні недоліки: обмежена персоналізація, відсутність інтерактивних текстових запитів, недостатня інтеграція зовнішніх даних (наприклад, погоди) і слабка підтримка активного туризму. Ваш веб-додаток відрізняється завдяки підтримці текстових запитів природною мовою, аналізу контекстних даних і спеціалізації на туристичному спорядженні, що заповнює прогалину на ринку.

Постановка задачі чітко окреслила цілі проєкту: створення масштабованого, безпечного та зручного для користувачів веб-застосунку зі штучним інтелектом. Визначено конкретні завдання, включаючи розробку адаптивного інтерфейсу на

базі React, реалізацію мікросервісної архітектури з Django і FastAPI, інтеграцію ШІ для обробки запитів та забезпечення безпеки даних. Обґрунтування методологічного підходу підтвердило доцільність використання Agile-методології для гнучкої розробки, мікросервісної архітектури для оптимізації продуктивності, компонентно-орієнтованого підходу з Feature-Sliced Design для фронтенду та моделей обробки природної мови для ШІ-компонентів. Такий підхід гарантує відповідність платформи сучасним вимогам і потребам користувачів.

Таким чином, проведений аналіз та сформульовані задачі закладають міцний фундамент для практичної реалізації веб-додатку. Унікальність цього проєкту полягає в поєднанні інтерактивності, глибокої персоналізації та адаптивності, що робить його конкурентоспроможним на ринку туристичних платформ, зокрема в Україні.

РОЗДІЛ 2 ПІДХІД ДО РОЗВ’ЯЗАННЯ ТА ІНСТРУМЕНТАЛЬНА БАЗА

2.1 Вимоги до системи

Для розробки веб-платформи, яка автоматизує підбір туристичного спорядження (інвентар, одяг, аксесуари), використовуючи штучний інтелект для створення персоналізованих рекомендацій, критично необхідно чітко окреслити функціональні та нефункціональні вимоги. Ці вимоги формуються на основі потреб цільової аудиторії (туристи, мандрівники, альпіністи), сучасних ринкових стандартів та технічних обмежень. Вони гарантують зручність використання, високу продуктивність, безпеку та масштабованість платформи, що є ключовим фактором для конкурентоспроможного рішення на ринку. [5]

Функціональні вимоги

Функціональні вимоги визначають основні можливості системи, які забезпечують її відповідність цілям проєкту та очікуванням користувачів. Вони охоплюють усі аспекти взаємодії з платформою, від авторизації до генерації рекомендацій і оформлення покупок. [5]

1. Авторизація та керування користувачами.

- Система повинна підтримувати реєстрацію нових користувачів за допомогою електронної пошти та пароля.
- Реалізація авторизації через зовнішні сервіси, наприклад, GitHub OAuth, для полегшення доступу та покращення зручності.
- Підтримка ролей користувачів (звичайний користувач, адміністратор) для розмежування доступу до функціоналу, наприклад, управління каталогом товарів. [26]

2. Каталог і фільтрація товарів.

- Платформа повинна мати структурований каталог туристичних товарів, зокрема намети, рюкзаки, спальники, трекінгове взуття, одяг та аксесуари.
- Реалізація багаторівневих фільтрів за категоріями (наприклад, "для походів", "для альпінізму"), ціною, брендом, технічними характеристиками (вага, матеріал) та типом активності.
- Можливість пошуку товарів за ключовими словами з автодоповненням для зручності.
- Відображення детальної сторінки товару з описом, характеристиками, відгуками, фотографіями та супутніми продуктами.

3. Персоналізовані рекомендації за допомогою ШІ.

- Система повинна обробляти запити користувачів на природній мові (наприклад, "спорядження для зимового походу в Карпати для початківців") та генерувати персоналізовані рекомендації.
- Аналіз контекстних даних, як-от прогноз погоди, рівень фізичної підготовки користувача, тип подорожі (похід, кемпінг, альпінізм) та геолокація маршруту. [11]

4. Кошик та оформлення замовлення.

- Реалізація функціональності кошика з можливістю додавання, видалення та редагування товарів.
- Поп-ап кошик для швидкого перегляду обраних продуктів без переходу на окрему сторінку.
- Сторінка оформлення замовлення з вибором способу доставки (кур'єр, пошта, самовивіз) та оплати (картка, готівка при отриманні).
- Генерація сторінки подяки після успішного оформлення замовлення з деталями покупки та рекомендаціями супутніх товарів. [1]

5. Список улюблених товарів.

- Користувачі повинні мати можливість додавати товари до списку улюблених для збереження їх на майбутнє.
- Синхронізація списку між різними пристроями за допомогою облікового запису користувача.
- Можливість швидкого додавання улюблених товарів до кошика.

6. Адміністрування та аналітика.

- Адміністративна панель для управління каталогом товарів (додавання, редагування, видалення), користувачами (блокування, зміна ролей) та замовленнями (статус, повернення).

- Генерація звітів про продажі, популярність товарів та поведінку користувачів для аналізу ефективності платформи.
- Інструменти модерації відгуків та коментарів для забезпечення якості контенту.

7. Інтерактивний інтерфейс.

- Реалізація різноманітних сторінок, включаючи головну (з акціями та популярними товарами), детальні сторінки товарів, фільтрації, подяки та профілю користувача.
- Підтримка інтерактивних елементів, як-от слайдери, модальні вікна (наприклад, поп-ап кошик) та анімації для покращення користувацького досвіду. [5]

Нефункціональні вимоги

Нефункціональні вимоги визначають якісні характеристики системи, які забезпечують її надійність, продуктивність і зручність використання.

1. Адаптивність і кросбраузерність.

- Інтерфейс має бути повністю адаптивним для коректного відображення на різних пристроях (десктопи, планшети, смартфони) з роздільною здатністю від 320 пікселів.
- Підтримка основних браузерів (Chrome, Firefox, Safari, Edge) для забезпечення широкої доступності.
- Час завантаження сторінок не повинен перевищувати 2 секунди на стандартному інтернет-з’єднанні (4G). [13]

2. Продуктивність.

- Система має обробляти до 1000 одночасних користувачів без значного

падіння швидкості.

- Затримка обробки запитів ШІ не повинна перевищувати 1,5 секунди, що забезпечується кешуванням та оптимізацією моделей.
- Час відгуку бази даних для типових операцій (пошук, фільтрація) має бути меншим за 500 мс. [19]

3. Масштабованість.

- Архітектура системи повинна дозволяти додавання нових мікросервісів та збільшення обсягу даних (наприклад, розширення каталогу до 100 000 товарів).
- Використання хмарної платформи AWS для автоматичного масштабування під час пікових навантажень.

4. Безпека.

- Використання HTTPS та TLS для шифрування даних між клієнтом та сервером.
- Захист від типових атак, таких як XSS, CSRF та SQL-ін’єкції, за допомогою вбудованих механізмів Django та додаткових бібліотек.
- Реалізація JWT для безпечної аутентифікації та авторизації.
- Захист конфіденційних даних користувачів до стандартів GDPR. [25]

5. Надійність і доступність.

- Система повинна забезпечувати uptime ≥ 99.9% для мінімізації простоїв.
- Реалізація механізмів резервного копіювання даних та відновлення після збоїв.
- Логування помилок та моніторинг продуктивності для швидкого виявлення проблем. [20]

Ці вимоги утворюють основу для розробки функціональної, зручної та надійної платформи, яка відповідає потребам сучасних туристів та ринковим стандартам.

2.2 Вибір та обґрунтування програмних технологій

Вибір технологій для втілення версустосунку, що автоматизує добір туристичних товарів (спорядження, одяг, аксесуари) з використанням штучного інтелекту (ШІ), базується на їх здатності задовольняти функціональні й нефункціональні потреби, відповідності сучасним стандартам веб-розробки та вимогам цільової аудиторії. Технологічний стек розподілено на три ключові складові: клієнтська частина, серверна частина та інструменти для реалізації штучного інтелекту. Кожна обрана технологія ретельно обґрунтована з урахуванням її сильних сторін, продуктивності, здатності до масштабування та відповідності цілям проекту, що охоплюють розробку адаптивного інтерфейсу, мікросервісної архітектури та інтерактивної системи рекомендацій на основі текстових запитів користувачів.[12]

2.2.1 Фронтенд: React, TypeScript та SCSS

Для клієнтської частини застосунку, котрий автоматизує підбір туристичних товарів (спорядження, одяг, аксесуари) із використанням API ChatGPT для персоналізованих рекомендацій, було обрано бібліотеку React у парі з TypeScript для типізації коду та SCSS для стилізації інтерфейсу. Цей набір технологій гарантує створення динамічного, адаптивного та високопродуктивного веб-інтерфейсу, що відповідає вимогам проєкту, включно з підтримкою багатьох сторінок (головна, каталог, детальна сторінка товару, кошик, сторінка подяки після покупки), швидкого завантаження (≤2 секунди) та зручності використання на різних пристроях, від смартфонів до десктопів. Вибір React, TypeScript та SCSS обґрунтовано їхньою здатністю забезпечити гнучкість, надійність та візуальну привабливість платформи, що є критично важливим для конкурентоспроможного рішення на ринку туристичних товарів. [2,4]

React було обрано як основну бібліотеку для фронтенду з огляду на її популярність та ефективність у створенні компонентно-орієнтованих

веб-інтерфейсів. У книзі React in Action (Manning Publications, 2020) відзначено, що React забезпечує високу продуктивність завдяки механізму віртуального DOM, який оптимізує оновлення сторінок, мінімізуючи звернення до реального DOM. Це особливо важливо для застосунку з великою кількістю динамічних елементів, як-от фільтри каталогу, спливний кошик, інтерактивні картки товарів та рекомендації, згенеровані на основі текстових запитів до API ChatGPT. Віртуальний DOM дозволяє швидко оновлювати інтерфейс, наприклад, при зміні вмісту кошика чи відображенні нових товарів після застосування фільтрів, що забезпечує плавний користувацький досвід навіть на пристроях з обмеженими ресурсами. [5]

Переваги React для проєкту:

- **Компонентна структура.** React дозволяє створювати компоненти, що використовуються повторно, як-от картки товарів, кнопки авторизації, фільтри або модальні вікна для кошика, що значно прискорює розробку та полегшує підтримку коду. Наприклад, компонент картки товару може відображатися на головній сторінці, у каталозі або у списку улюблених, з мінімальними змінами конфігурації, що знижує дублювання коду та полегшує рефакторинг. [5,6]
- **Оптимізація продуктивності.** React підтримує відкладене завантаження (lazy loading) через React.lazy та Suspense, що дозволяє завантажувати лише необхідні компоненти, наприклад, детальну сторінку товару, коли користувач переходить до неї. Мемоїзація через React.memo запобігає непотрібному повторному рендерингу компонентів, що критично для сторінок з великою кількістю елементів, як-от каталог з сотнями товарів. Ці механізми забезпечують швидке завантаження сторінок (≤2 секунди), що відповідає вимогам адаптивності, особливо для користувачів з повільним інтернет-з’єднанням, котрі здійснюють покупки зі смартфонів у віддалених локаціях. [2,5]

- **Екосистема бібліотек.** React інтегрується з потужними інструментами, як-от React Router для навігації між сторінками (наприклад, від каталогу до сторінки подяки після покупки), Redux Toolkit для управління станом (зберігання даних кошика, фільтрів або списку улюблених товарів) та Axios для асинхронних запитів до API, наприклад, для отримання рекомендацій від FastAPI або даних товарів з Django. Ці бібліотеки забезпечують гнучкість у реалізації складного функціоналу, як-от синхронізація улюблених товарів між пристроями або відображення динамічних рекомендацій на основі ШІ. [9,10,12]
- **Активна спільнота та ресурси.** Завдяки великій спільноті розробників та великій документації, описаній у Learning React (O'Reilly Media, 2020), React дозволяє швидко розв'язувати технічні проблеми, знаходити готові рішення (наприклад, бібліотеки для анімацій чи форм) та адаптувати нових розробників до проєкту, що є важливим для довгострокового розвитку платформи. [5]

Порівняння з альтернативами, такими як Angular та Vue.js, показало, що React є оптимальним вибором для вашого проєкту. Angular, хоча й потужний, має складнішу криву навчання та більший розмір бандлу, що може сповільнити завантаження сторінок, особливо на мобільних пристроях з низькою пропускною здатністю. Vue.js, хоч і легший та швидший у простих проєктах, поступається React за масштабністю екосистеми та кількістю бібліотек, що є критичним для складного застосунку з інтеграцією ШІ, багатьма сторінками та динамічними елементами. React забезпечує баланс між продуктивністю, гнучкістю та підтримкою, що робить його ідеальним для створення інтерактивного інтерфейсу застосунку. [2]

Для підвищення надійності та безпеки коду React використовується разом з TypeScript, мовою, яка додає статичну типізацію до JavaScript. У книзі Programming TypeScript (O'Reilly Media, 2019) підкреслюється, що TypeScript

зменшує кількість помилок на етапі розробки, забезпечуючи перевірку типів та автодоповнення, що особливо важливо для великих проєктів зі складною логікою. У контексті застосунку TypeScript застосовується для типізації пропсів компонентів, стану, API-відповідей та моделей даних, що підвищує стабільність та полегшує масштабування. [4]

Переваги TypeScript у проєкті:

- **Запобігання помилкам.** Статична типізація дозволяє виявляти помилки на етапі компіляції, наприклад, некоректне передавання пропсів до компонента картки товару або неправильний формат відповіді від API ChatGPT. Це знижує ризик багів, як-от відображення некоректних даних у каталозі. [4,13]
- **Покращення підтримки коду.** TypeScript забезпечує автодоповнення та документацію типів у PyCharm Professional, що прискорює розробку та полегшує рефакторинг, наприклад, при зміні структури даних для рекомендацій. Інтерфейси та типи (наприклад, Product, CartItem, Recommendation) чітко визначають структуру даних, що спрощує роботу з Redux Toolkit або Axios. [4]
- **Масштабованість.** TypeScript полегшує додавання нових функцій, як-от підтримка нових типів товарів або інтеграція з платіжними системами, завдяки чітко визначеним типам та контрактам між компонентами. Наприклад, типізація API-запитів до FastAPI гарантує, що фронтенд коректно обробляє рекомендації від ШІ.
- **Інтеграція з React.** TypeScript має вбудовану підтримку React через типи @types/react та @types/react-dom, що дозволяє визначати пропси та стан компонентів з використанням generics, наприклад, React.FC<ProductCardProps>. Це забезпечує безпечну роботу з компонентами, як-от фільтри або кошик, та зменшує час на дебагінг. [4,2]

Порівняння з чистим JavaScript показало, що TypeScript значно підвищує надійність коду, особливо у проєкті з великою кількістю компонентів та складною взаємодією з бекендом (Django, FastAPI). Хоча TypeScript додає початкові витрати на визначення типів, ці зусилля окупаються зменшенням помилок та спрощенням підтримки, що є критичним для довгострокового розвитку платформи. [4]

Для стилізації інтерфейсу було обрано SCSS (Sassy CSS), який є розширенням CSS з підтримкою змінних, вкладених правил та модулів. У книзі Sass and Compass in Action (Manning Publications, 2013) відмічено, що SCSS забезпечує гнучкість та структурованість при створенні стилів, що ідеально підходить для складних веб-застосунків з адаптивним дизайном. У застосунку SCSS використовується для створення візуально привабливого та адаптивного інтерфейсу, що коректно відображається на пристроях з роздільною здатністю від 320 пікселів (смартфони) до 4K (десктопи). [16]

Переваги SCSS у проєкті:

- **Модульність стилів.** SCSS підтримує модулі через @use та @import, що дозволяє створювати окремі файли стилів для різних компонентів, наприклад, _card.scss для карток товарів, _filters.scss для фільтрів та _cart.scss для кошика. Це зменшує дублювання коду та полегшує підтримку, наприклад, при зміні кольорової палітри або розмірів елементів.
- **Змінні та міксини.** Змінні SCSS дозволяють визначати глобальні стилі, як-от кольори бренду або розміри шрифтів, що спрощує їх зміну в усьому проєкті. Міксини, наприклад, для адаптивних стилів (@mixin respond-to($breakpoint)), забезпечують швидке створення медіа-запитів, що відповідає вимогам адаптивності для мобільних та десктопних пристроїв.
- **Вкладені правила.** Вкладеність SCSS дозволяє писати стилі в ієрархічній структурі, що відображає структуру JSX-компонентів, наприклад, стилі для картки товару та її внутрішніх елементів (зображення, ціна, кнопка). Це

підвищує читабельність та зменшує ризик конфліктів стилів.

- **Продуктивність.** SCSS компілюється в оптимізований CSS, що зменшує розмір стилів завдяки мінімізації та видаленню невикористаного коду (за допомогою інструментів, як-от PurgeCSS). Це сприяє швидкому завантаженню сторінок, що є важливим для користувачів з повільним інтернет-з'єднанням. [3]
- **Інтеграція з React.** SCSS легко інтегрується з React через бібліотеки, як-от sass, що дозволяють імпортувати SCSS-файли в компоненти. Це забезпечує локальну область видимості стилів для кожного компонента, знижуючи ризик конфліктів між стилями різних сторінок, як-от каталог та кошик. [16]

Порівняння з альтернативами, як-от CSS-модулі, styled-components або Tailwind CSS, показало, що SCSS є кращим вибором для вашого проєкту. CSS-модулі обмежують гнучкість через локальну область видимості, що ускладнює створення глобальних стилів, як-от теми бренду. Styled-components, хоча й інтегруються з React, збільшують розмір JavaScript-бандлу, що може сповільнити завантаження. Tailwind CSS, попри швидкість стилізації, менш гнучкий для створення складних дизайн-систем та потребує додаткового часу на кастомізацію, тоді як SCSS дозволяє точно налаштувати стилі через змінні, міксини та вкладеність, що ідеально підходить для адаптивного інтерфейсу застосунку. [31]

Поєднання React, TypeScript та SCSS забезпечує створення надійного, продуктивного та візуально привабливого фронтенду. React дозволяє будувати інтерактивний інтерфейс з компонентами, що використовуються повторно, TypeScript підвищує стабільність коду через статичну типізацію, а SCSS забезпечує гнучку та адаптивну стилізацію. Цей стек технологій відповідає вимогам проєкту, включаючи швидке завантаження, підтримку складного функціоналу (фільтри, кошик, рекомендації) і зручність для користувачів на всіх пристроях, що робить версустосунок конкурентоспроможним на ринку туристичних товарів. [2]

2.2.2 Бекенд: Django та FastAPI

Серверна частина версією-додатку базується на мікросервісній архітектурі, що складається з двох ключових блоків: основного бекенду на Django, який займається управлінням користувачами, товарами, замовленнями та аутентифікацією, а також ШІ-бекенду на FastAPI, відповідальному за обробку запитів до API ChatGPT. Вибір мікросервісного підходу був зроблений з метою забезпечення масштабованості, відокремлення функціональності та високої продуктивності, що дозволяє оптимізувати кожен компонент окремо, враховуючи його специфічні потреби. [19]

Django було обрано для основного бекенду, враховуючи його надійність, вбудовані інструменти та широке застосування в комерційних проєктах, про що йдеться у виданні The Definitive Guide to Django (Apress, 2022). Django є повнофункціональним Python-фреймворком, який створює стабільну основу для реалізації більшості серверних функцій платформи. Переваги Django: [18]

- **ORM (Object-Relational Mapping).** Вбудована ORM забезпечує зручну роботу з реляційними базами даних, наприклад, PostgreSQL, для збереження даних про користувачів, товари, замовлення, відгуки та улюблені продукти. Наприклад, модель товару може включати поля для назви, ціни, характеристик і категорії, що робить фільтрацію та пошук простішими. [15]
- **Система аутентифікації.** Django підтримує автентифікацію за допомогою JWT і OAuth (зокрема GitHub OAuth), що відповідає вимогам безпеки та спрощує авторизацію. Вбудовані інструменти дають змогу реалізувати ролі користувачів (звичайний користувач, адміністратор) та обмежити доступ до адміністративної панелі. [17]
- **Адміністративна панель.** Django Admin пропонує готовий інтерфейс для управління каталогом товарів, користувачами та замовленнями, що зменшує час розробки власної панелі. Адміністратори можуть додавати нові товари, змінювати статуси замовлень або модерати відгуки. [5]

- **Безпека.** Вбудовані механізми захисту від XSS, CSRF та SQL-ін’єкцій, а також підтримка HTTPS, забезпечують відповідність вимогам безпеки, зокрема стандартам GDPR для захисту даних користувачів.
- **Екосистема.** Django REST Framework дає можливість створювати RESTful API для взаємодії з фронтендом, забезпечуючи серіалізацію даних і пагінацію для великих каталогів. [10]

Порівняння з альтернативами, як-от Flask або Ruby on Rails, показало, що Django є найкращим вибором через наявність вбудованих інструментів, які скорочують час розробки, та підтримку великих проєктів із високими вимогами до безпеки та стабільності. Flask, хоча й полегшений, вимагає додаткових бібліотек для реалізації автентифікації чи ORM, а Ruby on Rails менш розповсюджений у Python-екосистемі, що ускладнює інтеграцію з ШІ-компонентами. [7, 17]

FastAPI було обрано для ШІ-бекенду, який відповідає за взаємодію з API ChatGPT для обробки текстових запитів та генерування рекомендацій. У книзі FastAPI: Modern Python Web Development (Packt Publishing, 2023) зазначено, що FastAPI є одним з найшвидших Python-фреймворків для створення API завдяки підтримці асинхронного програмування через asyncio. Це робить його ідеальним для обробки ресурсомістких ШІ-запитів. Переваги FastAPI:

- **Висока продуктивність.** Асинхронна обробка дозволяє швидко обробляти велику кількість запитів до API ChatGPT, що критично важливо для аналізу текстових запитів у реальному часі (затримка ≤1,5 секунди). Наприклад, запит «спорядження для зимового походу» обробляється асинхронно, забезпечуючи швидку відповідь.
- **Автоматична документація.** FastAPI генерує Swagger-документацію, що спрощує тестування API та інтеграцію з фронтендом (React) та зовнішніми сервісами, наприклад, погодними API.
- **Інтеграція з ШІ.** FastAPI легко з’єднується з API ChatGPT через бібліотеки,

як-от openai, що дає змогу надсилати текстові запити та отримувати структуровані відповіді для рекомендацій.

- **Масштабованість.** FastAPI підтримує розгортання в контейнерах (Docker) та хмарних платформах (AWS, Google Cloud), що відповідає вимогам масштабованості для обробки зростаючої кількості користувачів.
- **Валідація даних.** Вбудована підтримка Pydantic дозволяє перевіряти формат запитів і відповідей, що знижує ризик помилок при взаємодії з API ChatGPT. [31]

Порівняння з альтернативами, на кшталт Flask чи Node.js (Express), показало, що FastAPI випереджає їх за продуктивністю та зручністю інтеграції з API ШІ. Flask менш ефективний для асинхронних операцій, а Express, хоча й швидкий, вимагає додаткових бібліотек для валідації даних і документації, що ускладнює розробку.

Мікросервісна архітектура забезпечує відокремлення функціональності: Django відповідає за стабільне управління даними (користувачі, товари, замовлення), а FastAPI оптимізує обробку ШІ-запитів, що вимагають значних обчислювальних ресурсів. Це дає змогу масштабувати кожен мікросервіс незалежно, наприклад, розгортаючи FastAPI на більш потужних серверах для обробки запитів до ChatGPT, що відповідає вимогам продуктивності та масштабованості. [19]

2.2.3 Засоби реалізації штучного інтелекту

Для впровадження компонентів штучного інтелекту, що опрацьовують текстові звернення клієнтів та генерують індивідуальні поради, використано ChatGPT API від OpenAI, визнане рішення в галузі обробки природної мови (NLP). Згідно з книгою Natural Language Processing with Python (Machine Learning Mastery, 2023), моделі, що лежать в основі ChatGPT, характеризуються високою точністю аналізу текстових даних та створення релевантних відповідей, що ідеально підходить для роботи з запитами, на зразок "поради для походу в Карпати взимку" або

"спорядження для відпочинку на пляжі". [30]
Ключові аспекти застосування ChatGPT API:

- **Аналіз текстових запитів.** ChatGPT API ідентифікує ключові параметри запиту, наприклад, тип подорожі (пішохідний, альпінізм, кемпінг), метеоумови, рівень фізичної підготовки користувача чи геолокацію, та формує структуровані рекомендації. Наприклад, на запит "спорядження для зимового походу" API може запропонувати список товарів (водонепроникний намет, теплий спальник, термобілизна) з поясненнями щодо їх актуальності. [11]
- **Оптимізація продуктивності.** Для скорочення затримок (≤1,5 секунди) використано кешування запитів через Redis, де зберігаються поширені запити та відповіді на них. Крім того, FastAPI покращує асинхронну взаємодію з API ChatGPT, зменшуючи час очікування.
- **Гнучкість.** API ChatGPT дозволяє налаштовувати параметри, наприклад, стиль відповіді або рівень деталізації, що дає змогу адаптувати поради до стилю платформи (наприклад, лаконічні чи розгорнуті, як зазначено в Deep Learning (MIT Press, 2016). Це надає гнучкість у формуванні рекомендацій, відповідних потребам користувачів. [24]
- **Розгортання**. API ChatGPT – це хмарний сервіс, що звільняє від необхідності локального розгортання моделей, знижуючи витрати на обчислювальні ресурси. Для масштабування обробки запитів використовуються хмарні платформи, такі як AWS API Gateway, що забезпечують стабільну роботу з великою кількістю запитів. [20]

Порівняння з альтернативами, як-от Hugging Face Transformers або локальні моделі на основі TensorFlow, продемонструвало переваги API ChatGPT завдяки простоті інтеграції, високій точності та відсутності потреби в управлінні інфраструктурою. Hugging Face потребує локального розгортання моделей і значних обчислювальних ресурсів, що ускладнює масштабування, тоді як

TensorFlow вимагає більше часу на навчання та оптимізацію моделей. [30, 20, 19]

Для обробки структурованих даних (наприклад, характеристик товарів або профілів користувачів) застосовано бібліотеки Pandas і NumPy, які спрощують підготовку даних для передачі в API ChatGPT. Наприклад, Pandas дозволяє фільтрувати товари за категоріями чи характеристиками перед формуванням рекомендацій. Ці бібліотеки гарантують швидку обробку даних, що відповідає вимогам продуктивності. [1, 10]

Вибір React із Feature-Sliced Design, SCSS, Django, FastAPI та API ChatGPT обґрунтовано їх здатністю забезпечувати модульність, продуктивність, безпеку та інноваційність платформи. Ці технології дозволяють реалізувати адаптивний інтерфейс, мікросервісну архітектуру та інтерактивну систему рекомендацій, що сприяє конкурентоспроможності застосунку на ринку туристичних товарів, пропонуючи унікальний досвід взаємодії для користувачів порівняно з наявними українськими магазинами. [9, 11, 28, 30]

2.3 Методика та інструменти розробки

Розробка туристичної програми, що автоматизує підбір туристичних товарів (спорядження, одяг, аксесуари) із застосуванням API ChatGPT для створення індивідуальних рекомендацій, потребує структурованого підходу та сучасного набору інструментів. Методологія визначає організацію процесу розробки, включаючи планування, ітеративне створення, тестування та розгортання, в той час, як інструменти забезпечують технічне втілення, автоматизацію та перевірку якості. Вибраний підхід та інструментарій враховують складність проекту, який поєднує мікросервісну архітектуру, адаптивний інтерфейс на базі React із Feature-Sliced Design (FSD), серверну частину на Django та FastAPI, а також інтеграцію штучного інтелекту (ШІ) через API ChatGPT. Використання PyCharm Professional, плагінів (nvim, Big Data Tools, Makefile Language тощо), Docker і

WSL оптимізує розробку, забезпечуючи гнучкість, продуктивність і стабільність системи. [14]

Методика розробки

Для реалізації туристичної програми обрано Agile-методологію з фреймворком Scrum, що забезпечує гнучкість та швидке створення прототипів. У книзі Scrum: The Art of Doing Twice the Work in Half the Time (Crown Business, 2014) наголошується, що Scrum ідеально підходить для проєктів із високим рівнем невизначеності, таких як розробка додатків із ШІ, де результати обробки текстових запитів через API ChatGPT можуть вимагати ітеративного покращення. Scrum організовує процес за допомогою коротких ітерацій, регулярних перевірок та зворотного зв'язку, що відповідає потребам проєкту, включаючи створення адаптивного інтерфейсу, мікросервісів та системи рекомендацій. [33]

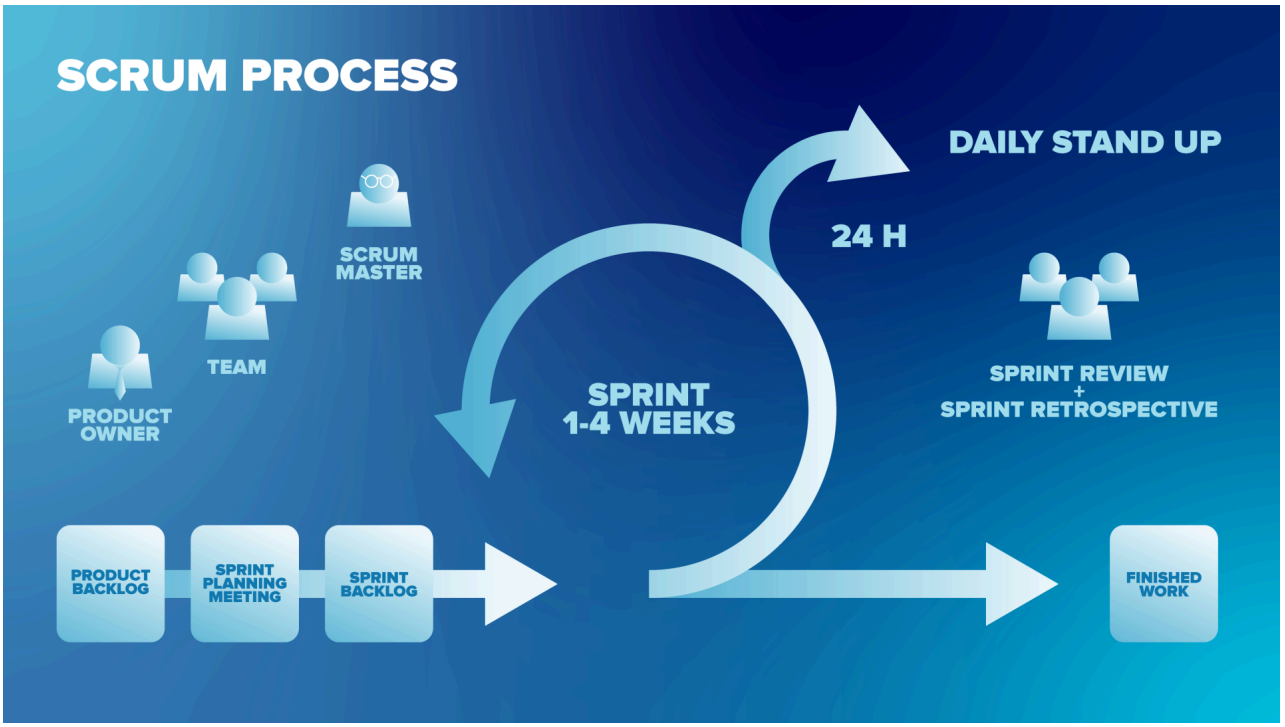

Рисунок 2.1 – Scrum process

Основні принципи застосування Scrum у проекті:

- **Спринти.** Розробка поділена на двотижневі спринти, кожен з яких закінчується створенням робочого функціоналу, наприклад, модуля авторизації через GitHub OAuth, сторінки каталогу з фільтрами, поп-ап кошика чи інтеграції API ChatGPT для рекомендацій. Кожен спринт включає планування, розробку, тестування та демонстрацію результатів, що дозволяє швидко перевіряти гіпотези, наприклад, ефективність ШІ-рекомендацій для зимових походів. [11]

- **Щоденні стендапи.** Щоденні 10–15-хвилинні зустрічі синхронізують команду, дозволяючи обговорювати прогрес (наприклад, завершення компонента кошика) та вирішувати блокери, такі як проблеми з налаштуванням Docker-контейнерів у WSL чи оптимізацією запитів до FastAPI.
- **Ретроспективи.** Після кожного спринту проводяться ретроспективи для аналізу успіхів та проблем, що допомагає покращувати процеси, наприклад, скоротити час на дебагінг ШІ-запитів чи оптимізувати CI/CD-пайплайн.
- **Зворотний зв’язок від користувачів.** Прототипи, такі як сторінка фільтрації чи інтерфейс рекомендацій, тестуються з потенційними користувачами (туристами, альпіністами), що дає змогу коригувати функціонал, наприклад, додавати нові фільтри за вагою спорядження чи уточнювати текстові запити для ШІ. [30]

Agile-методологія забезпечує гнучкість, що критично для інтеграції API ChatGPT, де точність рекомендацій може залежати від налаштування параметрів API чи якості вхідних даних (наприклад, погодних API чи профілю користувача). Порівняння з альтернативами, такими як Waterfall чи Kanban, показало, що Scrum є оптимальним через структуровану ітеративність. Waterfall не підходить через його жорсткий підхід, який ускладнює адаптацію до змін, наприклад, оновлень

API ChatGPT, в той час як Kanban менш структурований і більше підходить для підтримки, а не розробки складних систем. [20]

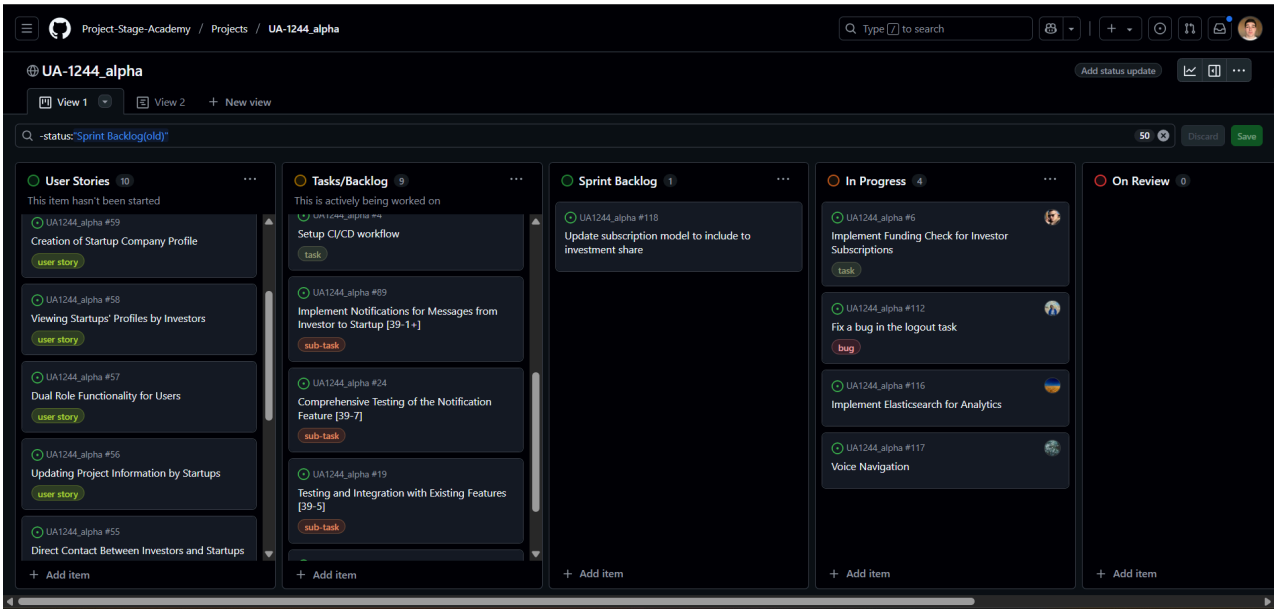

Рисунок 2.2 – GitHub Kanban

Для управління завданнями використано Trello з Kanban-дошкою, яка організовує задачі за статусами: «To Do», «In Progress», «Testing», «Done». Це забезпечує прозорість, дозволяючи відстежувати прогрес, наприклад, розробку сторінки подяки чи тестування мікросервісу FastAPI. Техніка Story Points застосовується для оцінки складності завдань на основі Fibonacci-послідовності (1, 2, 3, 5, 8), що допомагає планувати спринти та розподіляти ресурси. Наприклад, інтеграція API ChatGPT оцінюється у 8 балів через складність налаштування, в той час як створення статичної сторінки — у 2 бали. Для документування вимог і планування використано Confluence, що забезпечує централізоване зберігання специфікацій, діаграм і планів спринтів. [19]

Інструменти розробки

Набір інструментів для розробки туристичної програми підібрано з урахуванням особливостей роботи в PyCharm Professional, використання Docker із WSL, а також потреб у продуктивності, автоматизації та якості коду. Вони охоплюють усі етапи: від написання коду до тестування, розгортання та моніторингу. [22]

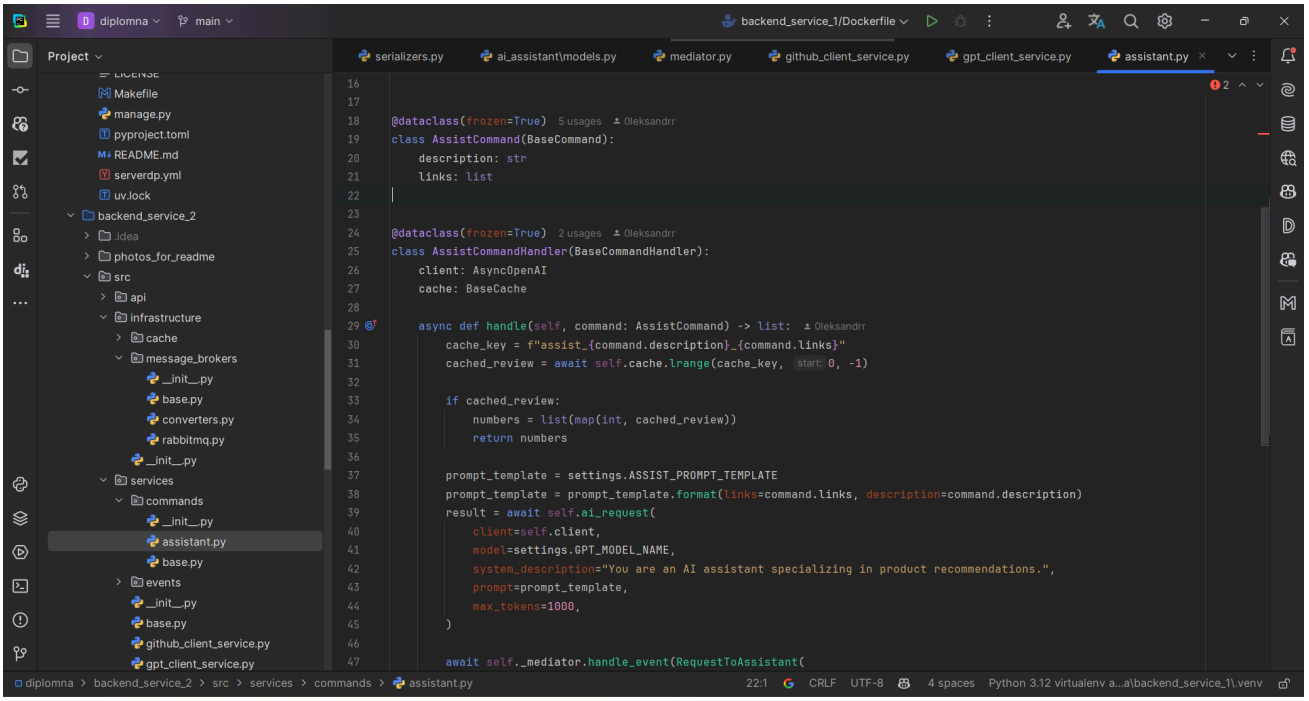

Рисунок 2.3 – IDEA PyCharm Professional

Середовище розробки.

Основним середовищем обрано PyCharm Professional зі студентською ліцензією від університету, що забезпечує потужний функціонал для роботи з JavaScript (React) та Python (Django, FastAPI). У статті Mastering PyCharm for Python Development (JetBrains Blog, 2023) відмічається, що PyCharm Professional підвищує продуктивність завдяки інтеграції з Docker, WSL та плагінами для аналізу коду. [22]

Використані плагіни:

- **nvim.** Інтеграція Neovim через плагін дозволяє використовувати Vim-подібні команди для швидкого редагування коду, що прискорює роботу з JSX-файлами React чи Python-скриптами FastAPI.
- **Big Data Tools.** Плагін підтримує обробку великих обсягів даних, що корисно для аналізу логів чи підготовки наборів даних для тестування ШІ-рекомендацій, наприклад, списків товарів з характеристиками.
- **Makefile Language.** Забезпечує підтримку Makefile для автоматизації задач, таких як запуск Docker-контейнерів чи виконання CI/CD-скриптів. [22]

- **Python.** Вбудований плагін для автодоповнення, рефакторингу та дебагінгу Python-коду, що спрощує розробку моделей Django чи ендпойнтів FastAPI.
- **JavaScript and TypeScript.** Плагін для аналізу JSX і TypeScript-коду, що забезпечує автодоповнення та перевірку синтаксису в React-компонентах.

PyCharm інтегровано з WSL 2 для запуску Linux-оточення на Windows, що дозволяє використовувати Linux-команди та Docker у звичному середовищі. WSL забезпечує швидке виконання Python-скриптів та контейнерів, що критично для тестування мікросервісів у локальному середовищі.
Контейнеризація та віртуалізація.
Для створення ізольованих середовищ використано Docker з Docker Compose, що працює через WSL. У книзі Docker in Action (Manning Publications, 2020) підкреслюється, що Docker забезпечує однакове середовище для розробки, тестування та продакшену, усуваючи проблеми сумісності. [20]

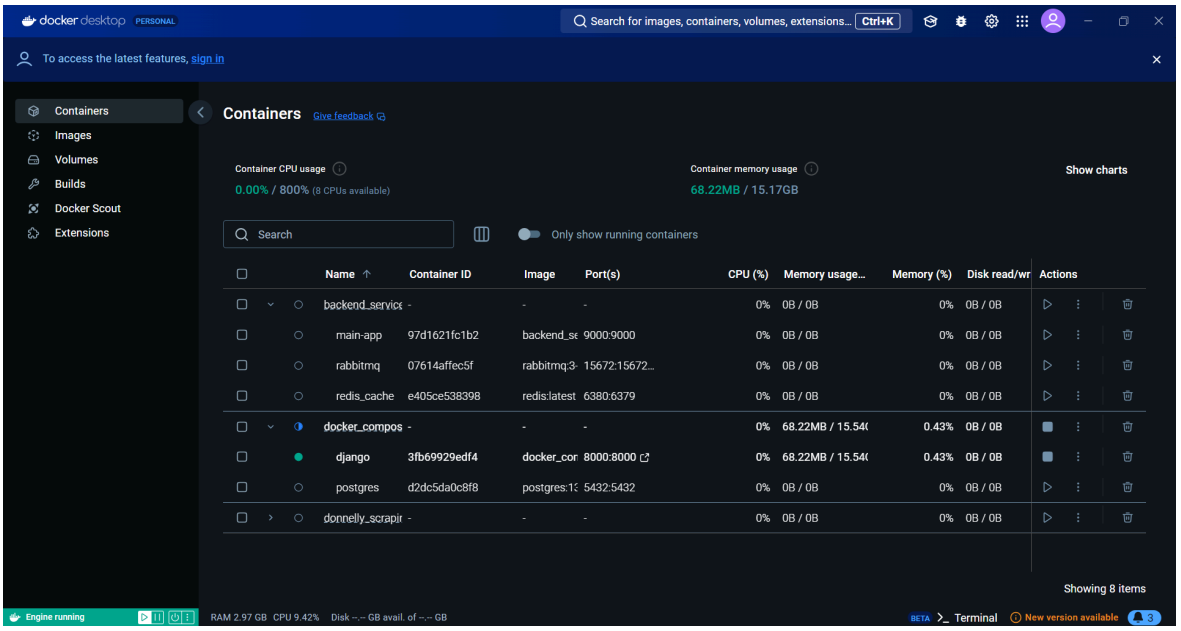

Рисунок 2.4 – Docker Desktop

Використання:

- Окремі контейнери для фронтенду (React з Nginx), основного бекенду (Django з Gunicorn), ШІ-бекенду (FastAPI) та бази даних (PostgreSQL).

- Docker Compose для локального запуску всіх сервісів через єдиний docker-compose.yml, що дозволяє тестувати взаємодію між мікросервісами, наприклад, запит від фронтенду до FastAPI для отримання рекомендацій через API ChatGPT. [20]
- Оптимізація образів через multistage builds, що зменшує розмір контейнерів (наприклад, видаляючи node_modules після побудови React) та прискорює розгортання.
- Інтеграція Docker з PyCharm дозволяє запускати та дебагувати контейнери безпосередньо з IDE, що спрощує тестування, наприклад, ендпойнтів FastAPI.

WSL 2 забезпечує безшовну роботу Docker на Windows, дозволяючи використовувати Linux-команди (bash, make) для управління контейнерами та виконання скриптів, що підвищує продуктивність порівняно з нативним Windows-оточенням. [20]

Контроль версій.

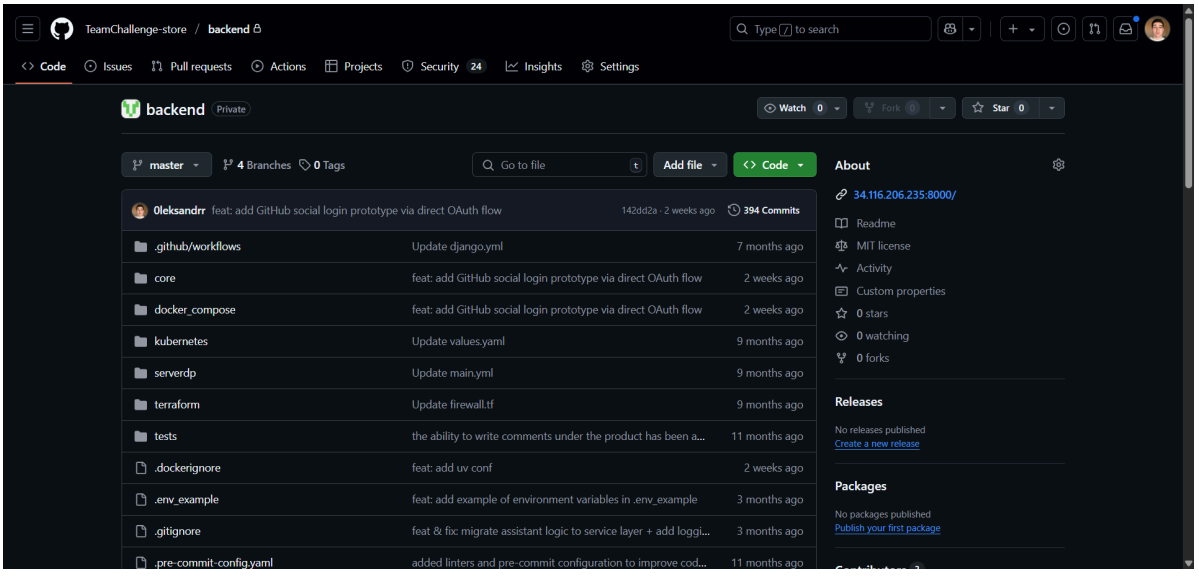

Рисунок 2.5 – GitHub репозиторій проєкту

Для керування кодом використано Git з хостингом на GitHub. У книзі Version

Control with Git (O'Reilly Media, 2019) зазначається, що Git забезпечує надійне зберігання версій та полегшує спільну розробку. GitHub використовується для:

- Збереження окремих репозиторіїв для фронтенду (React), основного бекенду (Django) та ШІ-бекенду (FastAPI), що спрощує управління мікросервісами.
- Керування pull requests з код-рев'ю, що дозволяє перевіряти зміни, наприклад, нову логіку фільтрації чи інтеграцію API ChatGPT, перед злиттям.
- Ведення документації в README та GitHub Wiki для опису структури проекту, інструкцій з запуску Docker-контейнерів та налаштування WSL.
- Інтеграція з PyCharm для виконання Git-команд (commit, push, pull) безпосередньо з IDE, що прискорює робочий процес. [20]

Тестування.

Для забезпечення якості системи використано комплексний підхід до тестування, що включає unit-тести, інтеграційні тести та навантажувальне тестування. Використані інструменти:

- Jest для unit-тестів фронтенду, що перевіряють React-компоненти, наприклад, рендеринг картки товару чи логіку поп-ап кошика. У книзі Testing JavaScript with Jest (Packt Publishing, 2021) підкреслюється, що Jest підтримує знімки (snapshots) для перевірки UI. Jest інтегровано з PyCharm для запуску тестів через інтерфейс IDE.
- Pytest для тестування бекенду (Django, FastAPI), що охоплює API-ендпоінти, ORM-запити та обробку ШІ-запитів. Pytest підтримує параметризацію для тестування різних сценаріїв, наприклад, відповідей API ChatGPT для різних текстових запитів.
- React Testing Library для інтеграційних тестів фронтенду, що перевіряють взаємодію компонентів, наприклад, додавання товару до кошика через поп-ап. [26]

- Locust для навантажувального тестування, що симулює 1000 одночасних користувачів, перевіряючи продуктивність, зокрема швидкість обробки ШІ-запитів через FastAPI чи запитів до PostgreSQL. Locust запускається в Docker-контейнері через WSL для точного відтворення продакшен-оточення. [26]

CI/CD.

Для автоматизації тестування, збирання та розгортання обрано Jenkins – потужний інструмент для побудови CI/CD-каналів. У праці "Jenkins: The Definitive Guide" (O'Reilly Media, 2011) наголошується, що Jenkins пропонує гнучкі налаштування для автоматизації робочих процесів, що ідеально підходить для мікросервісної архітектури проєкту. Jenkins розгорнуто на локальному сервері у контейнері Docker через WSL, що дозволяє інтегрувати його з PyCharm та Docker для керування каналами. [21]

Налаштовані канали включають:

- **Автоматичне виконання тестів.** При кожному push чи pull request у GitHub, Jenkins запускає тести (Jest для фронтенду, Pytest для Django і FastAPI), забезпечуючи якість коду до злиття. Наприклад, тести перевіряють рендеринг React-компонентів або коректність відповідей API ChatGPT.
- **Побудова Docker-образів.** Jenkins збирає Docker-образи для всіх мікросервісів (React, Django, FastAPI) та зберігає їх у Docker Hub. Використання Makefile (підтримується плагіном Makefile Language в PyCharm) автоматизує команди збірки образів, що пришвидшує процес. [21]
- **Розгортання на хмарну платформу.** Після успішного проходження тестів, Jenkins розгортає оновлені Docker-образи на AWS Elastic Beanstalk, забезпечуючи швидке оновлення продакшен-версії. Канал включає перевірку безпеки образів за допомогою інструментів, на кшталт Trivy, для виявлення вразливостей.
- **Інтеграція з PyCharm.** Jenkins інтегровано з PyCharm через плагін Jenkins

Control, що дозволяє переглядати статуси каналів (успіх, помилка) та логи виконання безпосередньо в IDE, спрощуючи моніторинг та дебагінг. Наприклад, розробник може швидко визначити, чому тест API ChatGPT не пройшов, переглянувши логи Jenkins у PyCharm. [21]

Порівняння з альтернативами, такими як GitHub Actions чи GitLab CI, продемонструвало перевагу Jenkins через його гнучкість, підтримку складних каналів та можливість локального розгортання у Docker-контейнері, що зменшує залежність від хмарних сервісів та відповідає вашим потребам у налаштуванні CI/CD. [21]

База даних і кешування.
PostgreSQL використано для збереження даних, як зазначено в "PostgreSQL: Up and Running" (O'Reilly Media, 2021), завдяки підтримці складних запитів і сумісності з Django ORM. Redis пришвидшує обробку частих запитів, зокрема, до API ChatGPT. Обидва сервіси розгорнуто в Docker-контейнерах через WSL. [19]

Моніторинг і логування.
Prometheus та Grafana використовуються для моніторингу продуктивності, як описано в "Monitoring with Prometheus and Grafana" (SysAdmin Magazine, 2022). Sentry логує помилки, інтегруючись з PyCharm для швидкого доступу до логів.

Рисунок 2.6 – Grafana

Інструменти для ШІ.
Бібліотека openai забезпечує інтеграцію з API ChatGPT, а Pandas та NumPy обробляють структуровані дані, як описано в "Natural Language Processing with Python" (Machine Learning Mastery, 2023). Postman використовується для тестування ШІ-запитів. [31]

Висновки:

Другий розділ створив всебічну основу для розробки веб-застусунку, яка автоматизує підбір туристичних товарів з використанням API ChatGPT для персоналізованих рекомендацій. Визначено функціональні вимоги (авторизація, каталог, кошик, ШІ-рекомендації) та нефункціональні вимоги (адаптивність, продуктивність, безпека, масштабованість), які забезпечують відповідність платформи потребам користувачів.

Технологічний стек, включаючи React із Feature-Sliced Design, Tailwind CSS, Django, FastAPI та API ChatGPT, обґрунтовано їхньою здатністю забезпечити модульність, продуктивність та інноваційність. Мікросервісна архітектура ізолює функціонал, дозволяючи масштабувати кожен компонент окремо.
Agile-методологія з фреймворком Scrum забезпечує гнучкість через двотижневі спринти, стендапи та ретроспективи, що дозволяє адаптуватися до змін, наприклад, оновлень API ChatGPT.

Інструменти, включаючи PyCharm Professional з плагінами (nvim, Big Data Tools, Makefile Language), Docker з WSL, GitHub, PostgreSQL, Redis, Prometheus, Grafana, гарантують високу якість коду і стабільність. Використання Jenkins для CI/CD оптимізує автоматизацію тестування (Jest, Pytest), побудови Docker-образів і розгортання на AWS Elastic Beanstalk, забезпечуючи швидке оновлення продакшен-версії. Інтеграція Jenkins з PyCharm спрощує моніторинг каналів, що підвищує продуктивність розробки. Тестування через Jest, Pytest та Locust, а

також інструменти для ШІ (openai, Postman), забезпечують надійність та точність рекомендацій.

Створений підхід та інструментальна база утворюють міцну основу для реалізації верстурботи, що вирізняється інтеграцією ШІ, адаптивним інтерфейсом та високою продуктивністю, перевершуючи українські інтернет-магазини туристичного спорядження завдяки інтерактивним текстовим запитам та контекстній персоналізації.

РОЗДІЛ 3 ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ПЛАТФОРМИ CAMPFIRE

3.1 Архітектура системи

Архітектура веб-додатку, що автоматизує підбір туристичних товарів (спорядження, одяг, аксесуари) з використанням API ChatGPT для персоналізованих рекомендацій, розроблена з врахуванням потреб у продуктивності, масштабованості, адаптивності та безпеці. Система інтегрує клієнтську частину, збудовану на React із застосуванням архітектури Feature-Sliced Design (FSD), та серверну частину, реалізовану через мікросервісну архітектуру з двома головними мікросервісами: основним (на базі Django, з використанням патерну MVC) та AI-мікросервісом (на базі FastAPI, із використанням патерну Domain-Driven Design, DDD). Такий підхід гарантує модульність, відокремлення функціоналу та гнучкість для подальшого розвитку платформи, що є критично важливим для конкурентоспроможного рішення на ринку туристичних товарів. [4]

3.1.1 Архітектура фронтенду: Feature-Sliced Design (FSD)

Для організації клієнтської частини застосунку про туристичні товари, який автоматизує підбір (спорядження, одяг, аксесуари) за допомогою API ChatGPT для персоналізованих порад, обрано архітектуру Feature-Sliced Design (FSD). Цей підхід реалізовано на фронтенді, що створений на React з TypeScript для типізації

та SCSS для стилізації, гарантуючи модульність, масштабованість і простоту підтримки коду. FSD дає змогу ефективно структуризувати складний проєкт із численними сторінками (головна, каталог, детальна сторінка, кошик, подяка, профіль) і функціоналом (авторизація, фільтрація, кошик, ШІ-рекомендації), що відповідає вимогам адаптивності, швидкості завантаження (≤2 секунди) та інтерактивності. Стаття Feature-Sliced Design: A Modern Approach to Frontend Architecture (Medium, 2023) підкреслює, що FSD є сучасним рішенням для структуризації фронтенд-застосунків, оптимізуючи розробку великих проєктів через логічне розділення коду на функціональні блоки. [28]

Структура Feature-Sliced Design

FSD ґрунтується на декомпозиції коду за функціональними областями, що сприяє ізоляції логіки окремих функцій та чіткій організації проєкту. Архітектура ділить код на три основні рівні: шари (Layers), слайси (Slices) і сегменти (Segments), кожен з яких має чітко визначену роль. Ця структура реалізується в директорії src фронтенд-проєкту, де кожен рівень відповідає конкретним задачам застосунку.[15]

1. Шари (Layers):

Шари представляють собою найвищий рівень абстракції, що групує код за його призначенням у застосунку. У застосунку визначені такі основні шари:

- **App:** відповідає за ініціалізацію застосунку, загальну конфігурацію та підключення зовнішніх бібліотек. Містить index.tsx (точка входу), глобальні SCSS-стилі (_global.scss), конфігурацію маршрутизації через React Router і налаштування Redux Toolkit для керування станом. Наприклад, у шарі App ініціалізується Redux Store для зберігання даних кошика або обраних фільтрів. [28]
- **Pages:** містить сторінки застосунку, кожна з яких є контейнером для об’єднання компонентів та логіки. У проєкті це HomePage, CatalogPage,

ProductDetailsPage, CartPage, ThankYouPage і ProfilePage. Кожна сторінка імпортує компоненти з інших шарів, наприклад, ProductCard із шару Entities для відображення товарів на CatalogPage. [28]

- **Entities:** представляє бізнес-сутності, які є ключовими об'єктами застосунку. Це Product (товар), User (користувач), Cart (кошик), Order (замовлення) і Recommendation (ШІ-рекомендація). Містить TypeScript-інтерфейси (наприклад, interface Product { id: number; name: string; price: number }), API-запити через Axios (наприклад, запит до Django для отримання товарів) і нормалізовані моделі даних. [28]
- **Features:** охоплює функціональні модулі, які реалізують окремі можливості застосунку. Тут це Auth (авторизація, зокрема GitHub OAuth), Filters (фільтрація товарів за категорією, ціною чи характеристиками), Cart (додавання/видалення товарів у кошику), Favorites (управління списком улюблених товарів) і Recommendations (відображення ШІ-рекомендацій від FastAPI). Кожен модуль ізольований, зменшуючи залежності. [28]
- **Shared:** містить спільні утиліти та ресурси, що використовуються в інших шарах. У проєкті це SCSS-змінні (наприклад, $primary-color: #2a6f97), загальні хуки (наприклад, useApi для роботи з Axios), допоміжні функції (наприклад, форматування цін) і сторонні бібліотеки, такі як lodash чи axios.

2. Слайси (Slices):

Слайси є підмодулями всередині шару, що відповідають конкретним функціям або сутностям. Наприклад:

- У шарі Entities слайси містять Product, User, Cart, кожен з яких включає логіку, пов’язану з цією сутністю (типи, API-запити, нормалізація). Наприклад, слайс Product містить запит до ендпоінту /api/products/ і TypeScript-тип Product. [28]
- У шарі Features слайси включають Auth, Filters, Cart, Recommendations.

  Наприклад, слайс Filters відповідає за логіку фільтрації товарів, зокрема

  компоненти фільтрів, хуки для управління станом фільтрів і запити до

Django для оновлення каталогу. [28]

Слайси ізолюють функціонал, що дає змогу розробляти та тестувати їх незалежно. Зміни в слайсі Cart (додавання функції редагування кількості товарів) не впливають на слайс Recommendations.

3. Сегменти (Segments):

- Сегменти — це конкретні складові слайсу, такі як компоненти, хуки, типи, API-запити або SCSS-стилі. Наприклад, слайс Cart у шарі Features включає:
- Компонент CartPopup.tsx: відображає модальне вікно кошика.
- Хук useCartItems.ts: керує станом кошика через Redux Toolkit.
- Тип CartItem.ts: TypeScript-інтерфейс для елемента кошика (interface CartItem { productId: number; quantity: number }).
- API-запит cartApi.ts: надсилає дані кошика до Django через Axios.
- Стилі _cart.scss: SCSS-модуль для стилізації кошика. [28]

Сегменти є найменшою одиницею коду, що спрощує їхнє тестування через Jest і рефакторинг.

Принципи FSD у застосунку

FSD базується на декількох ключових принципах, адаптованих до потреб застосунку:

- **Ізоляція.** Кожен слайс (наприклад, Filters, Cart) автономний, що зменшує залежності між модулями. Наприклад, логіка фільтрації товарів ізольована від логіки кошика, дозволяючи змінювати фільтри (додавати нові параметри, як-от вага спорядження) без впливу на інші функції.
- **Односпрямований потік даних.** Дані передаються від вищих шарів (App, Pages) до нижчих (Entities, Features), забезпечуючи передбачуваність.

Наприклад, сторінка CatalogPage імпортує компоненти з Filters і Product, але не навпаки.

- **Повторне використання.** Загальні утиліти в шарі Shared (наприклад, SCSS-змінні чи хук useApi) використовуються в різних слайсах, що зменшує дублювання коду.
- **Чітка відповідальність.** Кожен шар і слайс має визначену роль, що полегшує навігацію по проєкту. Уся логіка авторизації зосереджена в слайсі Auth, а відображення рекомендацій — у слайсі Recommendations.

У веб-застосунку "Campfire" архітектура FSD реалізована за допомогою TypeScript для типізації пропсів, стану і відповідей API, що значно підвищує надійність коду. SCSS забезпечує модульне оформлення, де кожен слайс має власний SCSS-файл (наприклад, _filters.scss, _cart.scss), що відповідає компонентам. React Router керує навігацією між сторінками, а Redux Toolkit відповідає за глобальний стан, зберігаючи фільтри або вміст кошика. Axios застосовується для запитів до мікросервісів Django (дані про товари) та FastAPI (ШІ-рекомендації), ізольованих у слайсах Product і Recommendations. [28]

Чому обрано Feature-Sliced Design

Вибір FSD для фронтенду "Campfire" базується на її здатності відповідати складності проекту, вимогам до масштабованості, зручності підтримки та прискорення розробки. Далі представлено докладний аналіз обґрунтування вибору FSD у порівнянні з альтернативними підходами. [28]

1. Модульність та ізоляція.

"Campfire" включає численні функції (авторизація, фільтрація, кошик, ШІ-рекомендації) та сторінки, що потребує чіткої організації коду. FSD ізолює кожну функціональність в окремий слайс, що зменшує залежності та полегшує розробку. Наприклад, слайс Recommendations ізольовано обробляє запити до FastAPI для отримання ШІ-рекомендацій, не впливаючи на слайс Cart. Це дозволяє паралельно розробляти різні функціональності, як-от один розробник працює над фільтрами, а інший — над кошиком, уникаючи конфліктів. У книзі "React Design Patterns and Best Practices" (Packt Publishing, 2021) підкреслено, що модульність є критичною для великих React-проєктів, і FSD надає її на всіх рівнях. [27]

2. Масштабованість.

FSD дає змогу легко розширювати проект, що відповідає довгостроковим цілям "Campfire", як-от додавання нових функцій (порівняння товарів, чат-бот для уточнення рекомендацій) або підтримка нових мов інтерфейсу. Наприклад, для додавання сторінки блогу досить створити новий слайс у шарі Pages (BlogPage) та підключити компоненти з Entities чи Features, не змінюючи вже існуючий код. Структура FSD підтримує зростання проекту без значних рефакторингів, що є ключовим для платформи, яка планує розширення до тисяч одночасних користувачів. [29]

3. Зручність підтримки та адаптація розробників.

Чітка структура FSD (шари, слайси, сегменти) полегшує навігацію в проекті, що зменшує час на адаптацію нових розробників. Наприклад, розробник може швидко знайти логіку кошика в слайсі Cart у шарі Features, а стилі - у файлі _cart.scss. Це особливо важливо для проекту, який може підтримуватися різними командами у майбутньому. Активна спільнота FSD, описана в "Feature-Sliced Design: A Modern Approach to Frontend Architecture" (Medium, 2023), надає документацію та приклади, що спрощують навчання. [4]

4. Інтеграція з TypeScript та SCSS.

FSD гармонійно інтегрується з TypeScript та SCSS, що є ключовими технологіями фронтенду "Campfire". TypeScript використовується для створення типів у слайсах Entities (наприклад, Product, CartItem) та Features (наприклад, пропси для Filters), що підвищує надійність коду. SCSS-модулі, прив’язані до певних слайсів (наприклад, _card.scss для ProductCard), відповідають принципу ізоляції FSD, забезпечуючи локальну видимість стилів та зменшуючи конфлікти. [4]

5. Порівняння з альтернативами.

Для вибору архітектури були розглянуті альтернативи, такі як Atomic Design, Redux-centric architecture та проста модульна структура.

- Atomic Design (описана в "Atomic Design" від Brad Frost, 2016) зосереджується на компонентах різних рівнів (атоми, молекули, організми), але менш придатна для організації бізнес-логіки та API-запитів, що є важливим для "Campfire" з інтеграцією ШІ. FSD, навпаки, чітко розділяє сутності, функції та сторінки, що краще відповідає складності проекту.
- Redux-centric architecture прив'язує код до структури Redux (actions, reducers, selectors), що ускладнює масштабування та ізоляцію функціональності, як-от фільтрація чи рекомендації. FSD більш гнучка, дозволяючи використовувати Redux Toolkit лише для глобального стану, а логіку ізолювати в слайсах.
- Проста модульна структура (розподіл на папки components, pages, utils) надто загальна та не забезпечує чіткої ієрархії для складних проектів. Наприклад, логіка кошика та фільтрів може змішатися, що ускладнює підтримку. FSD пропонує структуровану декомпозицію, що вирішує цю проблему. [4]

Порівняння продемонструвало, що FSD є оптимальним вибором завдяки її модульності, підтримці складних проектів і відповідності принципам React,

TypeScript та SCSS. Вона перевершує альтернативи за здатністю ізолювати функціональність, спрощувати тестування та забезпечувати масштабованість.

Реалізація FSD у "Campfire"

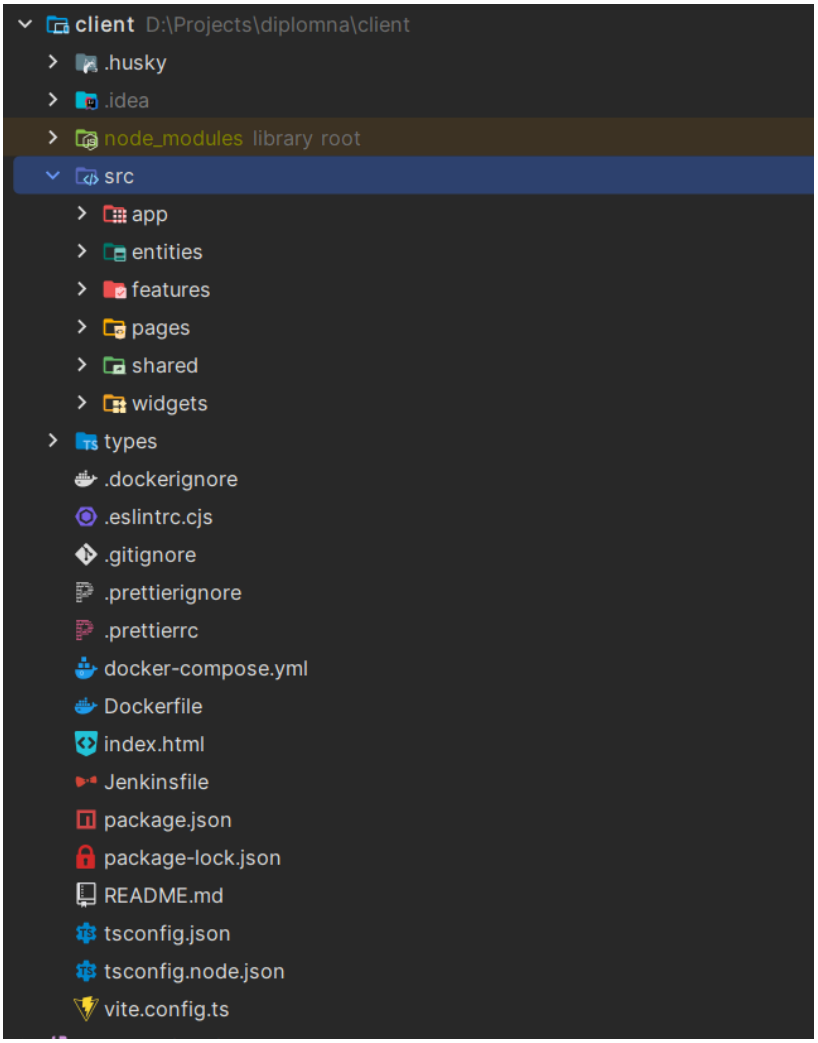

Рисунок 3.1 – Frontend структура

У "Campfire" FSD реалізовано з врахуванням специфіки проекту:

- **Сторінки.** Шар Pages містить сторінки, наприклад, CatalogPage, яка імпортує компоненти ProductCard (з Entities/Product) та Filters (з Features/Filters). React Router забезпечує навігацію, наприклад, перехід від /catalog до /products/:id. [3]

- **Сутності.** Шар Entities визначає типи та API-запити. Наприклад, слайс Product містить Product.ts (TypeScript-інтерфейс), productApi.ts (Axios-запити до Django) та ProductCard.tsx (компонент картки).
- **Функціональність.** Шар Features ізолює логіку. Наприклад, слайс Recommendations містить Recommendations.tsx (компонент для відображення ШІ-рекомендацій), useRecommendations.ts (хук для запитів до FastAPI) та recommendationsApi.ts (Axios-запити). [28]
- **Загальні утиліти.** Шар Shared містить SCSS-змінні (_variables.scss), хуки (useApi.ts) та утиліти (formatPrice.ts).
- **Тестування.** Jest тестує сегменти, наприклад, useCartItems чи ProductCard, а React Testing Library перевіряє інтеграцію, наприклад, додавання товару до кошика.
- **CI/CD.** Jenkins автоматизує запуск Jest-тестів при кожному push чи pull request, що гарантує якість коду. [21]

Архітектура головного мікросервісу (Django): MVC

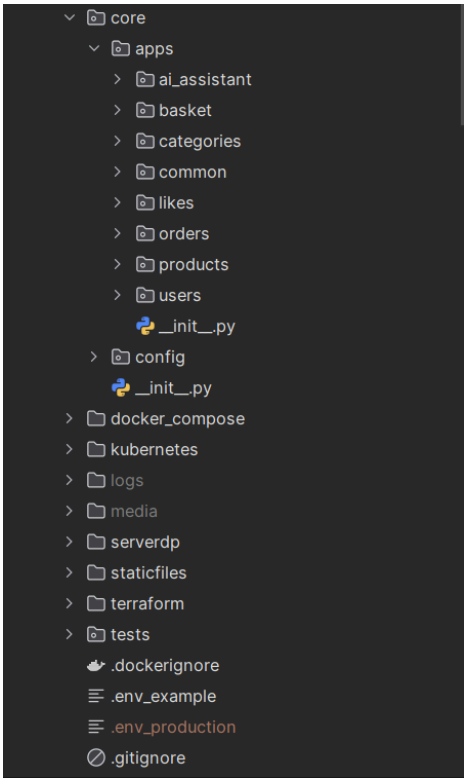

Рисунок 3.2 – Django структура

Головний мікросервіс вебзастосунку, який відповідає за адміністрування користувачів, товарів, замовлень, категорій, корзини, вподобань і адміністративних функцій, реалізовано на фреймворку Django, використовуючи архітектурний шаблон Model-View-Controller (MVC). Згідно з виданням The Definitive Guide to Django (Apress, 2022), MVC є стандартним підходом для Django, що гарантує чітке розділення даних, представлення та управління запитами, ідеально підходячи для структурованих вебзастосунків з реляційними базами даних. У контексті вебзастосунку, MVC дозволяє ефективно організувати код для обробки комплексного функціоналу, такого як аутентифікація через GitHub OAuth, фільтрація товарів за категоріями, управління корзиною та оформлення замовлень, забезпечуючи масштабованість, тестування та зручність підтримки. [1]

3.1.2 Структура MVC у Django-мікросервісі

Структура каталогів Django-мікросервісу (backend_service_1) демонструє модульний підхід, де кожна програма (app) відповідає за конкретну бізнес-логіку. Відповідно до наданої структури (ai_assistant, basket, categories, common, likes, orders, products, users) видно, що код розподілений за функціональними областями, що узгоджується з принципами MVC та полегшує інтеграцію з фронтендом і ШІ-мікросервісом. [10]

1. Model (Модель):

Моделі визначають дані та бізнес-логіку, впроваджені за допомогою Django ORM для взаємодії з базою даних PostgreSQL. Кожна програма містить моделі, що стосуються її домену:

- users: Модель User розширює стандартну модель Django (AbstractUser) та включає поля для імені, email, ролі (користувач, адміністратор) та даних аутентифікації (JWT-токени, GitHub OAuth). [10]

- products: Модель Product описує товари (назва, ціна, опис, характеристики, зображення, категорія), підтримує фільтрацію та пошук.
- categories: Модель Category визначає категорії товарів (наприклад, намети, рюкзаки) з ієрархічною структурою (батьківські та дочірні категорії).
- orders: Модель Order зберігає дані про замовлення (користувач, товари, статус, спосіб доставки, загальна сума).
- basket: Модель Basket (або Cart) містить товари, додані користувачем до корзини, з полями кількості та прив'язки до користувача.
- likes: Модель Like пов’язує користувачів з товарами, які вони додали до списку вподобань.
- ai_assistant: Модель RecommendationHistory (за потреби) зберігає історію запитів до ШІ-мікросервісу для аналізу або повторного використання.
- common: Містить загальні моделі або утиліти, наприклад, TimestampModel для автоматичного додавання полів created_at і updated_at. [1]

Моделі забезпечують валідацію даних (наприклад, перевірка унікальності email чи позитивності ціни) та підтримують складні запити, наприклад, Product.objects.filter(category__name="tents", price__lte=5000) для фільтрації товарів. Django ORM оптимізує доступ до даних, використовуючи індекси та кешування через Redis для частих запитів, наприклад, списку популярних товарів.

2. View (Представлення):

Представлення обробляють HTTP-запити від фронтенду через Django REST Framework (DRF) та повертають JSON-відповіді. Кожна програма містить свої власні представлення, реалізовані у вигляді класів (Class-Based Views) для покращеної структуризації. [1] Основні приклади:

- users: UserViewSet обробляє реєстрацію, аутентифікацію (JWT, GitHub OAuth) та оновлення профілю. Ендпоінт /api/users/me/ повертає дані поточного користувача.
- products: ProductViewSet надає список товарів (/api/products/), детальну інформацію (/api/products/:id/) та підтримку пагінації та фільтрації (наприклад, ?category=tents&max_price=5000).
- categories: CategoryViewSet повертає ієрархію категорій (/api/categories/) для відображення в фільтрах.
- orders: OrderViewSet створює замовлення (/api/orders/) та повертає історію (/api/orders/history/).
- basket: BasketViewSet управляє корзиною (/api/basket/), дозволяючи додавати, видаляти чи змінювати товари.
- likes: LikeViewSet додає або видаляє товари зі списку вподобань (/api/likes/).
- ai_assistant: За потреби містить представлення для зберігання історії ШІ-рекомендацій, інтегруючись з FastAPI-мікросервісом. [10]

Представлення використовують серіалізатори DRF для перетворення моделей у JSON та забезпечують захист від атак (XSS, CSRF) за допомогою вбудованих механізмів Django. Наприклад, ендпоінти захищені JWT-аутентифікацією, а запити перевіряються за допомогою серіалізаторів, які перевіряють формат даних.

3. Controller (Контролер):

У Django роль контролера виконують URL-маршрути (визначені в urls.py) та логіка представлень. Кожна програма має свій власний urls.py, що направляє запити до відповідних представлень. Наприклад:

- Запит GET /api/products/ викликає ProductViewSet.as_view({'get': 'list'}).
- Запит POST /api/basket/ додає товар до корзини через BasketViewSet.
- Запит POST /api/auth/github/ обробляє авторизацію через GitHub OAuth.

URL-маршрути відповідають RESTful-підходу, обробляючи HTTP-методи (GET, POST, PUT, DELETE) та забезпечуючи валідацію вхідних даних через DRF. [10]

Реалізація MVC

- База даних. PostgreSQL зберігає дані моделей, а Django ORM забезпечує абстракцію для запитів. Наприклад, фільтрація товарів реалізована через Product.objects.filter(...), а зв’язки між моделями (наприклад, Order і Product) використовують ForeignKey. [9]
- API. DRF реалізує RESTful API з підтримкою пагінації (наприклад, 20 товарів на сторінку), фільтрації (через django-filter) та сортування. Ендпоінти оптимізовані для швидкості за допомогою кешування в Redis.
- Автентифікація. Використовується django-allauth для GitHub OAuth та djangorestframework-simplejwt для JWT-токенів, що забезпечують безпечний доступ до ендпоінтів, таких як /api/orders/.
- Адміністрування. Django Admin надає інтерфейс для управління товарами, категоріями, замовленнями та користувачами, спрощуючи модерацію.
- Кешування. Redis кешує часті запити, наприклад, список популярних товарів або категорій, зменшуючи затримки до <500 мс. [9]

Переваги MVC

- **Чітка структура.** Розділення моделей, представлень і контролерів полегшує розробку та підтримку. Наприклад, зміна логіки фільтрації в ProductViewSet не впливає на модель Product. [9]
- **Тестування.** Ізольовані компоненти полегшують модульне тестування (Pytest) та інтеграційне тестування (перевірка API-ендпоінтів).
- **Повторне використання.** Моделі та серіалізатори використовуються в різних представленнях, що зменшує дублювання коду.

- **Безпека.** Вбудовані механізми Django (захист від SQL-ін'єкцій, CSRF) забезпечують відповідність вимогам безпеки. [9]

MVC ідеально підходить для Django-мікросервісу, оскільки забезпечує структуроване керування каталогом, замовленнями, корзиною та аутентифікацією, спрощуючи інтеграцію з фронтендом через REST API.

Архітектура ШІ-мікросервісу (FastAPI): DDD

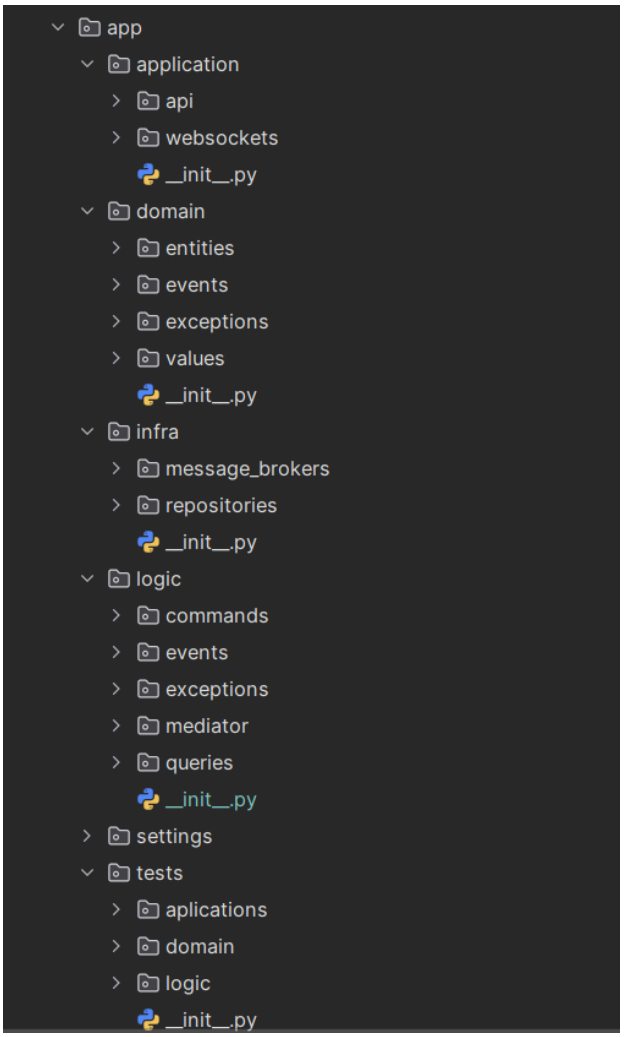

Рисунок 3.3 FastApi структура

ШІ-мікросервіс, створений на FastAPI, відповідає за обробку текстових запитів користувачів та генерування персоналізованих рекомендацій за допомогою API ChatGPT, доповнених зовнішніми даними (наприклад, прогноз погоди). Для

організації коду використано патерн Domain-Driven Design (DDD), який зосереджується на моделюванні бізнес-логіки, як описано в Domain-Driven Design: Tackling Complexity in the Heart of Software (Addison-Wesley, 2003). DDD оптимально підходить для ШІ-мікросервісу, оскільки дає змогу чітко визначити домен «персоналізовані рекомендації» та ізолювати складну ШІ-логіку від інших частин системи. [11]

3.1.3 Структура DDD у FastAPI-мікросервісі

Структура каталогів FastAPI-мікросервісу (fastapi-websockets-kafka-py3.12) демонструє підхід DDD, де код поділяється на доменні, прикладні та інфраструктурні шари. Згідно з наявною структурою (application, domain, infra, logic, settings, tests, message_brokers, repositories, entities, events, exceptions, values, commands, queries, mediator) видно, що архітектура зосереджена на домені рекомендацій з чітким розподілом логіки, інфраструктури та обробки подій. [11]

1. Bounded Context:

ШІ-мікросервіс є окремим контекстом, що відповідає за домен «персоналізовані рекомендації». Він відокремлений від Django-мікросервісу, щоб уникнути змішування логіки керування товарами та ШІ. Контекст включає:

- Обробку текстових запитів користувачів (наприклад, «спорядження для зимового походу»).
- Інтеграцію з API ChatGPT для аналізу запитів та генерування рекомендацій.
- Збагачення запитів зовнішніми даними, наприклад, прогнозом погоди з OpenWeatherMap.
- Кешування рекомендацій у Redis для швидкого доступу. [1]

2. Entities (Сутності):

Сутності, визначені в каталозі domain/entities, є основними об'єктами домену:

- Recommendation: містить список рекомендованих товарів, текстове пояснення (наприклад, «Ці намети підходять для зимових умов») та метадані (тип подорожі, погодні умови).
- UserQuery: описує запит користувача (текст, контекст, параметри, такі як геолокація чи рівень підготовки).
- ProductReference: спрощена модель товару, що містить ID та основні характеристики, отримані від Django-мікросервісу.

  Сутності мають унікальні ідентифікатори (UUID) та бізнес-логіку, наприклад, валідацію формату запиту чи перевірку відповідності рекомендації запиту. [23]

3. Репозиторії (Repositories):

Репозиторії, що розташовані в domain/repositories, забезпечують доступ до даних:

- RecommendationRepository: Зберігає рекомендації в Redis для кешування (з TTL 24 години) і в PostgreSQL для історії запитів.
- ProductRepository: Отримує інформацію про товари з Django-мікросервісу за допомогою API-запиту (/api/products/).

  Репозиторії абстрагують джерела даних, дозволяючи змінювати сховище (наприклад, замінити Redis на Memcached) без впливу на доменну логіку.

4. Application Layer (Рівень застосунку):

Рівень застосунку, реалізований у application, містить FastAPI-ендпоінти, які викликають доменні сервіси. Основні ендпоінти:

- POST /api/recommendations/: Приймає текстовий запит (наприклад, { "query": "спорядження для зимового походу" }), активує

RecommendationGeneratorService і повертає JSON з рекомендаціями.

FastAPI застосовує асинхронні методи (async def) для паралельної обробки запитів, що є критично важливим для ресурсомістких операцій з API ChatGPT. [33]

5. Інфраструктура та допоміжні компоненти:

- **message_brokers:** Застосовується Kafka для асинхронної обробки подій, наприклад, збереження рекомендацій у PostgreSQL після їх генерації. Директорія message_brokers містить конфігурацію Kafka-продюсерів і консьюмерів.
- **events:** Містить доменні події, як-от RecommendationGeneratedEvent, котрі публікуються в Kafka для обробки (наприклад, збереження в БД чи відправка сповіщень).
- **commands** та queries: Реалізують CQRS (Command Query Responsibility Segregation). Приміром, команда GenerateRecommendationCommand ініціює генерацію рекомендації, а запит GetRecommendationHistoryQuery повертає історію.
- **mediator:** Медіатор (logic/mediator) керує викликами команд і запитів, забезпечуючи їх ізольованість.
- infra: Включає інтеграції з зовнішніми сервісами (OpenWeatherMap, API ChatGPT) та налаштування Redis/PostgreSQL.
- **exceptions:** Визначає кастомні винятки, такі як InvalidQueryError для некоректних запитів.
- **settings:** Зберігає конфігурацію (API-ключі, URL-адреси Django, параметри Kafka). [1, 10, 11]

Переваги DDD

- Фокус на домені. DDD чітко моделює бізнес-логіку рекомендацій, відокремлюючи її від інфраструктури, що полегшує додавання нових функцій, наприклад, аналізу геолокаційних даних.
- Ізоляція. Bounded Context ізолює ШІ-логіку від Django-мікросервісу, зменшуючи залежності та спрощуючи масштабування.
- Гнучкість. Доменні сервіси та CQRS дозволяють адаптувати логіку, наприклад, замінити API ChatGPT на іншу модель ШІ без зміни фронтенду чи Django.
- Тестування. Ізольовані сутності, сервіси та команди полегшують тестування, зокрема, перевірку RecommendationGeneratorService на різні сценарії запитів. [33]

DDD ідеально підходить для ШІ-мікросервісу завдяки складності домену, що включає аналіз тексту, інтеграцію зовнішніх даних та контекстну персоналізацію, забезпечуючи гнучкість і масштабованість.

3.2 Frontend: розробка сторінок

Головна сторінка (HomePage) вебзастосунку є центральним елементом взаємодії користувача, що відображає основний функціонал платформи: навігацію, рекламні банери, популярні та нові товари, акційні пропозиції, форму підписки на розсилку та доступ до ШІ-асистента. Сторінка міститься в директорії src/pages/home-page відповідно до шару Pages архітектури FSD та використовує компонент Layout (ui/HomePage/Layout.tsx) для структурування контенту. Реалізація сторінки забезпечує адаптивний дизайн, швидке завантаження та інтерактивність, що є критично важливим для залучення користувачів, зокрема туристів, які переглядають сайт з мобільних пристроїв у віддалених локаціях. Усі компоненти сторінки інтегровані з бекендом через REST API, що оптимізує продуктивність завдяки кешуванню в Redis. [4]

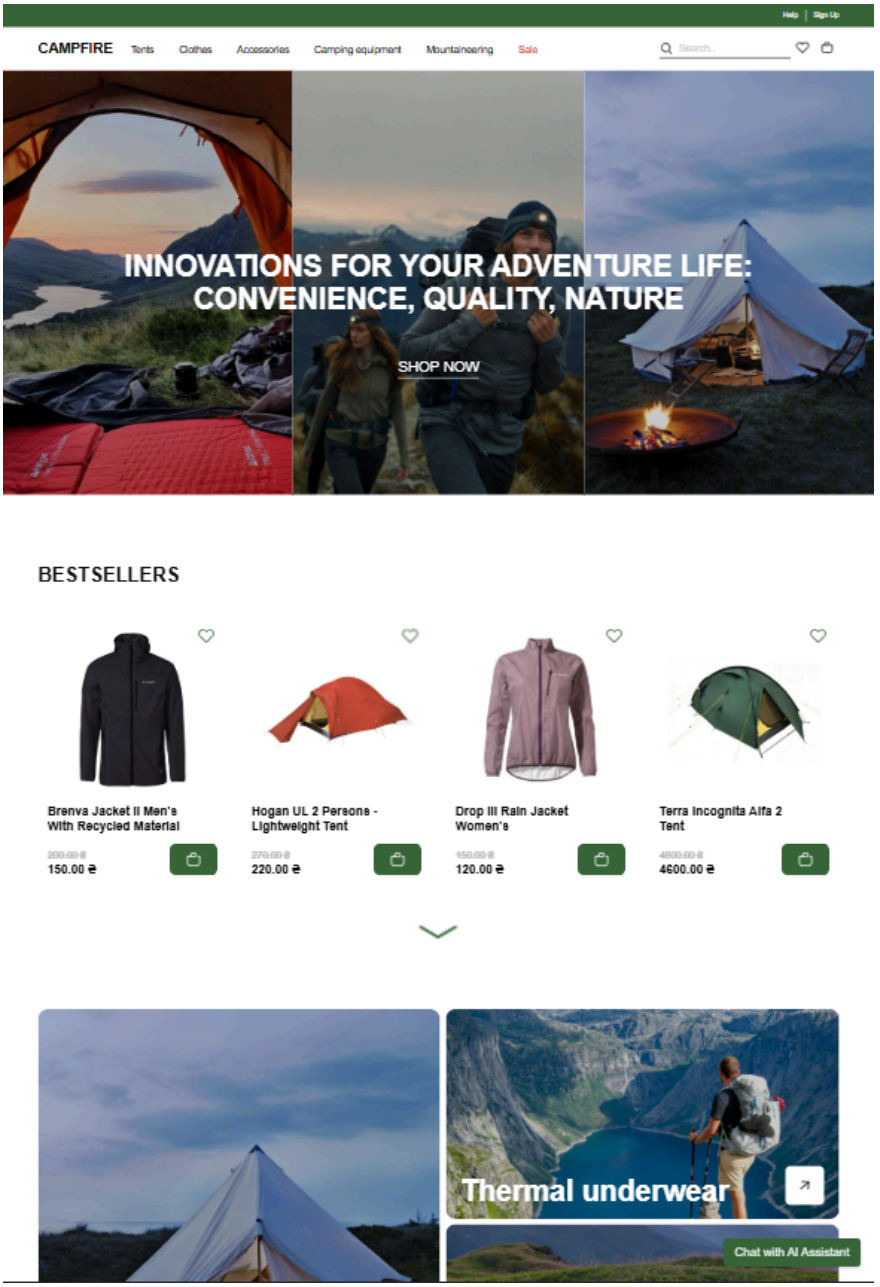

Рисунок 3.4 – Основна сторінка сайту Campfire

Головна сторінка має чітку структуру, що поєднує навігаційні, рекламні, інформаційні та інтерактивні елементи, створюючи зручний та привабливий інтерфейс. Нижче подано опис кожного елемента сторінки, його призначення та реалізація. [4]

Елементи сторінки

1. Хедер

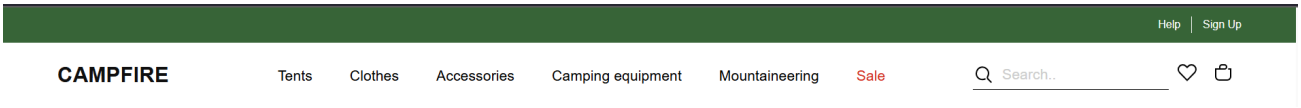

Рисунок 3.5 – Хедер

Хедер (шар Shared) містить:

- Навігацію посередині з такими категоріями: «Tents», «Clothes», «Accessories», «Camping Equipment», «Mountaineering», «Sale» (посилання через React Router). На мобільних пристроях меню згортається у бургер-меню.
- Іконки праворуч: улюблені товари (/favorites) та кошик (/cart) з лічильниками, керованими Redux Toolkit. [4]

2. Основний банер

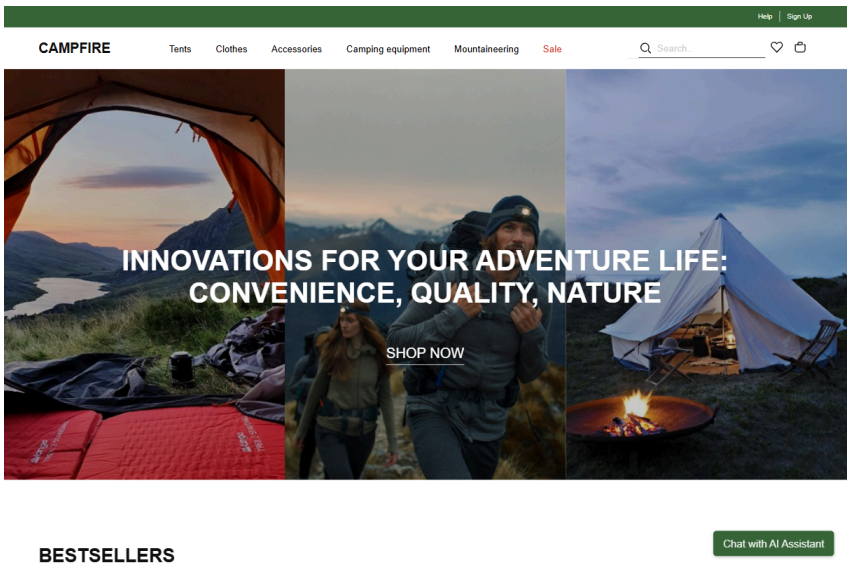

Рисунок 3.6 – Банер сайту Campfire

Компонент Banner (~widgets/banner) — це слайдер (Swiper), який демонструє зображення (гори, туристи, намет) з автопрокруткою (2000 мс).

3. Бестселери

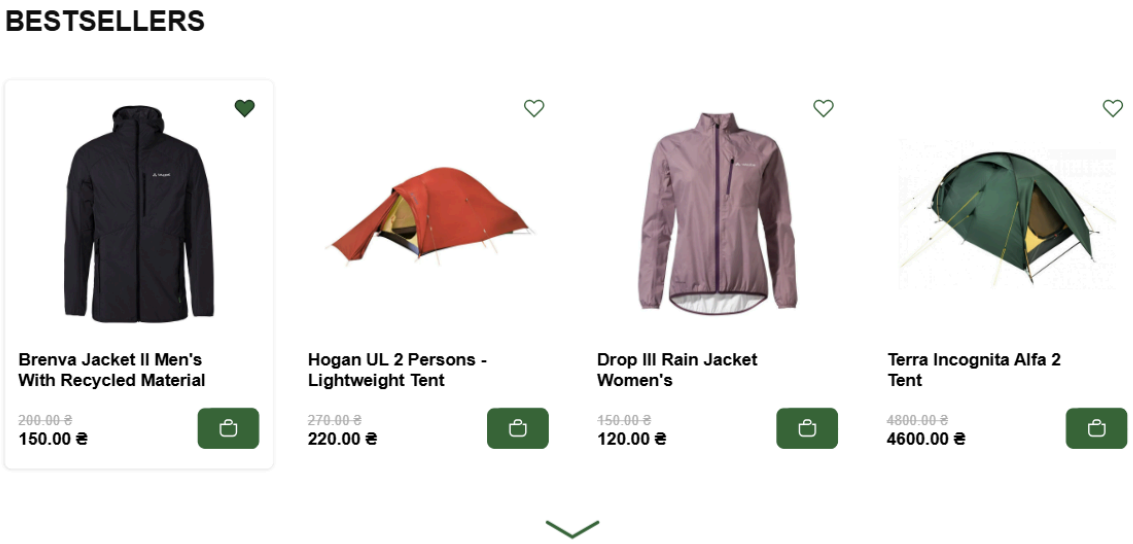

Рисунок 3.7 – Секція товарів які найбільше продаються

Блок Bestsellers (~widgets/bestsellers) показує 8 популярних товарів (ендпоінт /api/products/?sort=rate) у вигляді карток (ProductCard). Кнопка «Показати більше» веде до /products?sort=rate. Адаптивна сітка (SCSS) та кешування (RTK Query) забезпечують швидкість роботи. [32]

4. Тематичні банери

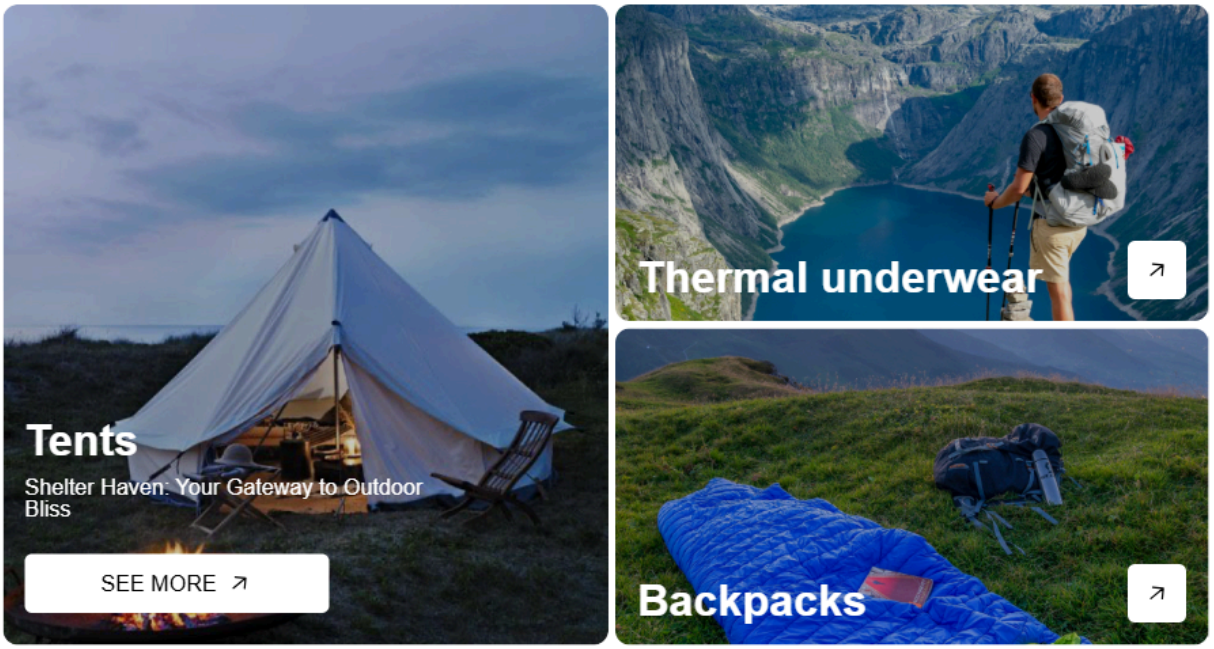

Рисунок 3.8 – Банери з категоріями

Компонент CategoriesBanner відображає три банери (намети, термоодяг, термомішки) з посиланнями на категорії (наприклад, /products/tents). На мобільних пристроях банери гортаються як слайдер (Swiper). [32]

5. Новинки

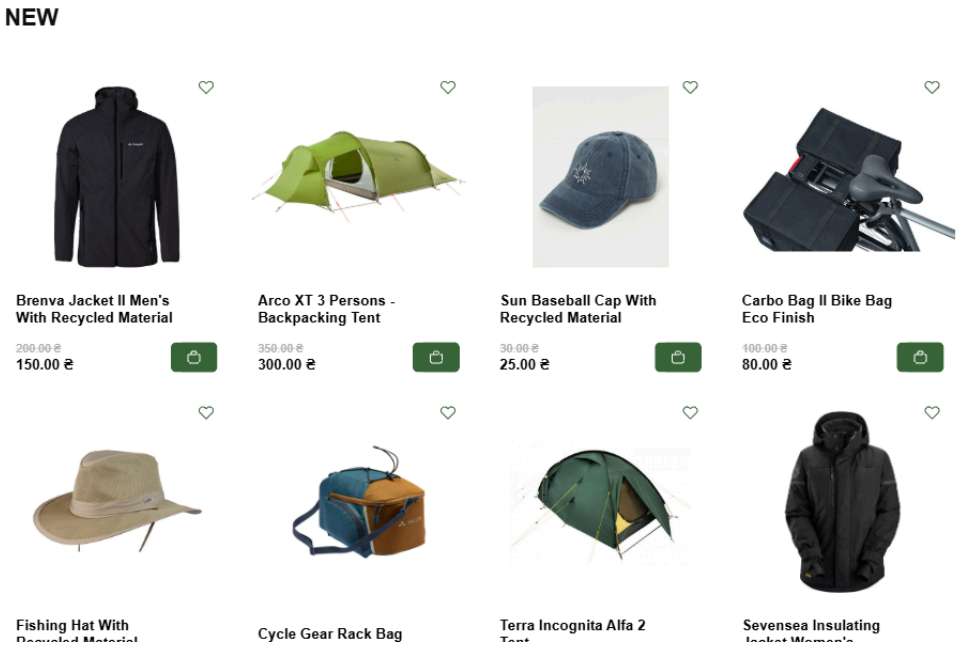

Рисунок 3.9 – Секція з новими товарами

Блок NewBlock (~widgets/newBlock) презентує 8 нових товарів (ендпоінт /api/products/?sort=new) з кнопкою «Показати більше» (/products?sort=new). Адаптивність та кешування аналогічні бестселерам.

6. Акційні пропозиції

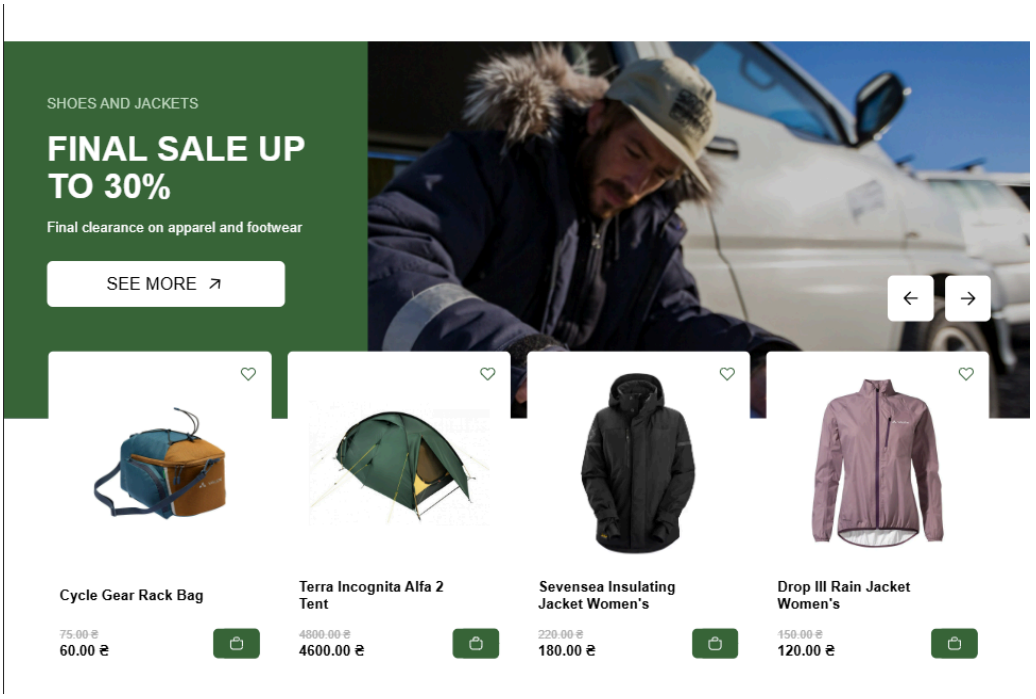

Рисунок 3.10 — Секція з акційними пропозиціями

PromotionalOffers (~widgets/promotional-offers) демонструє товари зі знижками (ендпоінт /api/products/?sale=true) у слайдері (Swiper, 1–4 товари в залежності від екрану). Банер із текстом «Final sale up to 30%» та кнопкою /products/sale мотивує до покупок.

7. Форма підписки

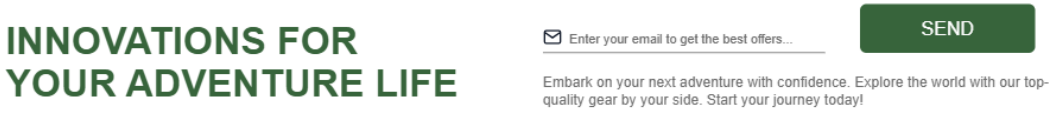

Рисунок 3.11 – Підписка на розсилку акційних пропозицій

SubscribeBlock (~widgets/subscribe-block) містить форму (SubscribeForm, шар Features) для введення email, який надсилається на /api/subscriptions/. Заголовок та опис стилізовані через SubscribeBlock.module.scss. [32]

8. Футер

Рисунок 3.12 – Campfire футер

Футер (шар Shared) включає контакти, інформацію про компанію, посилання на соціальні мережі (Instagram, Facebook, Twitter) та навігацію. Адаптивний дизайн економить місце на мобільних пристроях.

9. ШІ-асистент

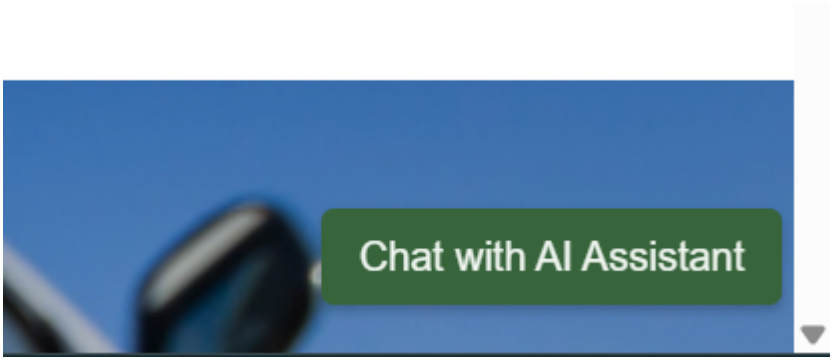

Рисунок 3.13 – Кнопка для відкриття вікна з ШІ

Кнопка в лівому нижньому куті (ChatWidget, ~widgets/ai-assistant) відкриває чат із FastAPI (/api/recommendations/), доступний авторизованим користувачам. Чат підтримує зміну розміру та показує рекомендації з API ChatGPT. [32]

Розробка попап-вікна ШІ-асистента

Спливаюче вікно ШІ-асистента, реалізоване компонентом ChatWidget (~widgets/ai-assistant/ChatWidget.tsx), розміщено у нижньому лівому куті головної

сторінки (шар Widgets в FSD). Воно дозволяє автентифікованим користувачам отримувати поради через FastAPI-мікросервіс, використовуючи RTK Query.

Однакові запити зберігаються в кеші Redis, що значно прискорює відповідь (затримки <500 мс для кешованих запитів, <1,5 секунди для нових). [12]

Візуальні та функціональні складові

1. Кнопка виклику

Кнопка (css.chatButton) з текстом "Chat with AI Assistant" (або "Close AI Assistant" при відкритті) стилізована за допомогою ChatWidget.module.scss. Розташована внизу зліва, вона відкриває/закриває попап за допомогою toggleChat.

2. Спливаюче вікно

Вікно чату (css.chatWindow) включає:

- Заголовок: "AI Assistant" (css.chatHeader).
- Область повідомлень: Відображає привітання ("Hi! How can I assist you today?") або історію чату (css.chatContent). Повідомлення користувача (css.userMessage) та ШІ (css.serverMessage) містять текст, посилання на товари (buildProductUrl) та час (formatTimestamp).
- Поле введення: Текстове поле (css.chatInput) та кнопка "Send" (css.sentMessage) відправляють запити до /api/recommendations/.

3. Автентифікація

- Неавтентифіковані користувачі: Показується повідомлення (css.unauthenticatedMessage) із посиланням на /sign_up через React Router.
- Автентифіковані користувачі: Після авторизації через логін/пароль або GitHub OAuth (перевірка через sessionStorage.getItem('accessToken')) користувачі вводять запити, які обробляються FastAPI через API ChatGPT.

  [17, 18]

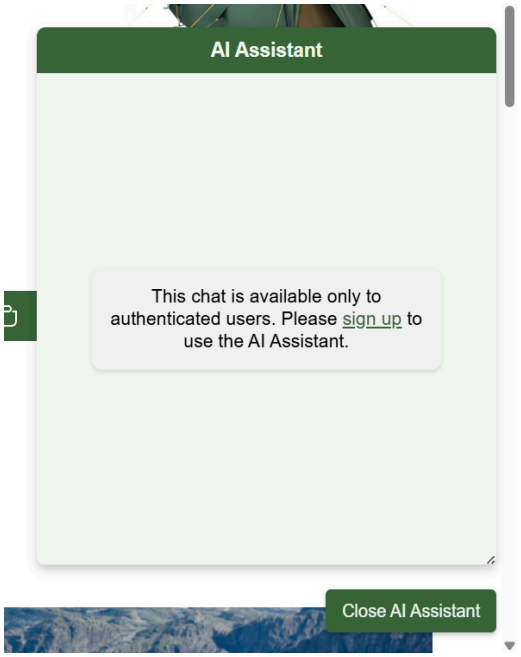

Рисунок 3.14 – Вікно для рекомендацій ШІ

4. Зміна розміру

Вікно підтримує зміну розміру (ширина 300–600 пікселів, висота 350–650 пікселів) через обробник (css.resizeHandle) з подіями mousedown, mousemove, mouseup. [4]

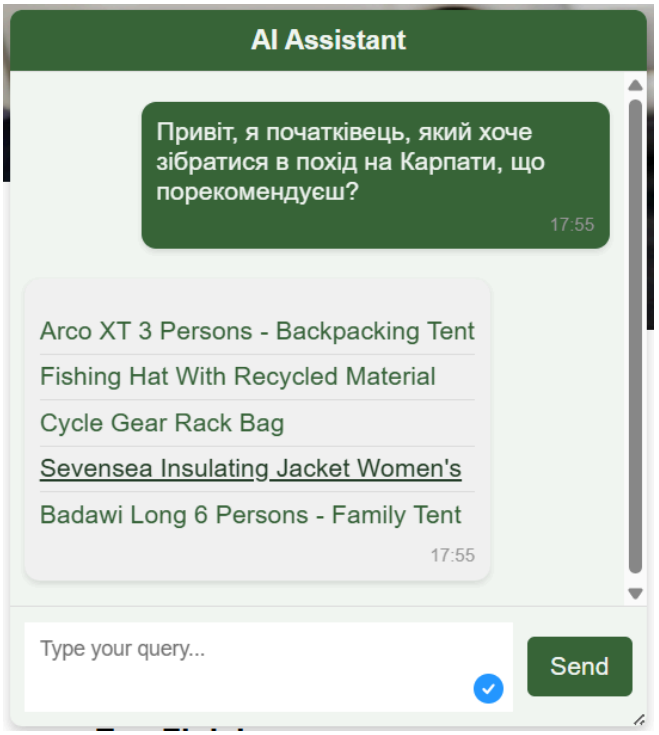

Рисунок 3.15 – Вікно для рекомендацій ШІ

Користувач вводить запит: "Привіт, я новачок, котрий прагне в похід на Карпати, що порадите?". Після автентифікації FastAPI через API ChatGPT аналізує запит (новачок, похід, Карпати) та надає поради, наприклад:

- Намет Naturehike Cloud-Up 2: Легкий (1,8 кг), водонепроникний, простий в установці.
- Трекінгові черевики Salomon X Ultra 3: Зручні, підтримують щиколотку, для карпатських стежок.
- Рюкзак Osprey Talon 33: Компактний, з вентиляцією, для одноденних походів.

Кожна порада містить назву, опис, переваги ("легкий", "водонепроникний") та посилання (наприклад, http://localhost:5173/product/123), відображені як список (css.linkList). Якщо користувач повторює схожий запит (наприклад, "спорядження для походу в Карпати"), FastAPI перевіряє Redis і повертає кешовану відповідь за <500 мс, минаючи API ChatGPT, що значно прискорює обробку. [19]

Спливаючі вікна списку бажань та кошика, реалізовані як компоненти WishlistPopup (~widgets/wishlist/WishlistPopup.tsx) та CartPopup (~widgets/cart/CartPopup.tsx) в шарі Widgets FSD, виводяться при взаємодії з іконками в хедері головної сторінки. Вони дозволяють як авторизованим, так і анонімним користувачам додавати товари до списку бажань чи кошика, зберігаючи інформацію через API.

Візуальні та функціональні особливості

1. Кнопки виклику

Іконки списку бажань та кошика, що знаходяться у верхньому правому куті хедера (шар Shared), стилізовані за допомогою SCSS (наприклад, Header.module.scss). Іконки (IconFavorite, IconCart) містять лічильники (кількість товарів), керуються Redux Toolkit та відкривають відповідні спливаючі вікна при натисканні.

2. Спливаюче вікно списку бажань (Wish List)

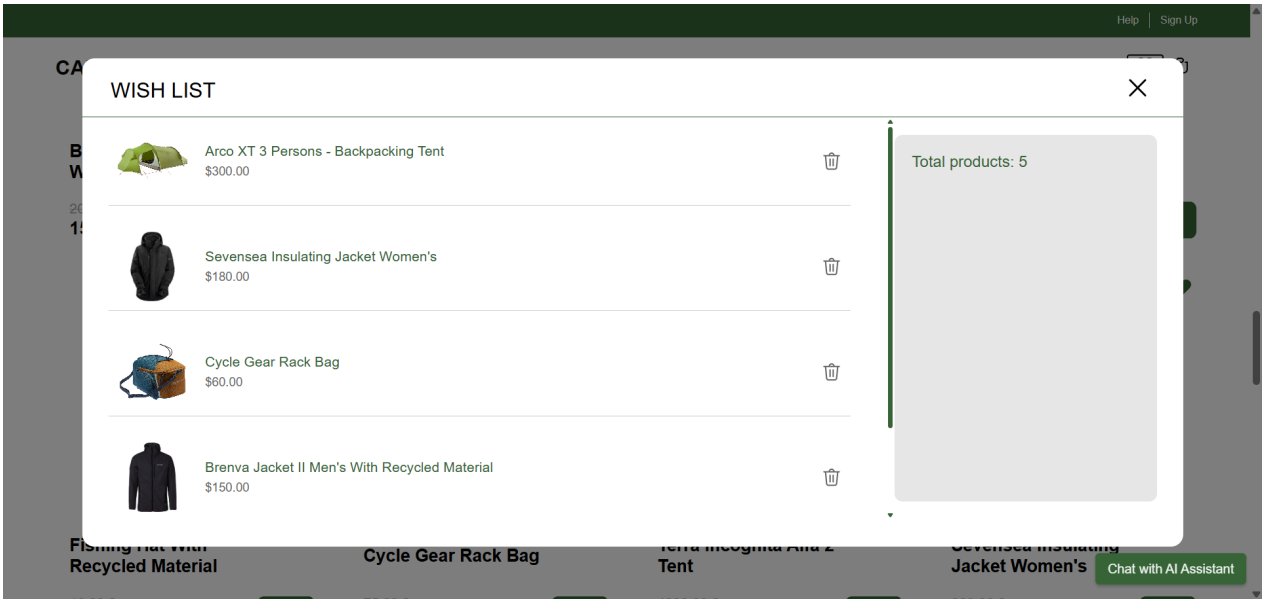

Рисунок 3.16 – Pop-up вікно з товарами які сподобались

WishlistPopup показує товари, додані до списку бажань через ендпоінт /api/likes.

Елементи:

- Заголовок: «Your Wishlist», стилізований через WishlistPopup.module.scss.
- Перелік товарів: Картки товарів (ProductCard з шару Entities/product) з зображенням, назвою, ціною та кнопкою видалення.
- Кнопка вподобання: Біля кожного товару в каталозі розташована іконка лайку, що додає товар до /api/likes (POST-запит).
- Повідомлення: Після додавання товару з'являється сповіщення у верхньому правому куті («Added to Wishlist»), стилізоване як тост-нотифікація. [25]

3. Спливаюче вікно кошика (Cart)

CartPopup відображає товари в кошику через ендпоінт /api/basket.

Елементи:

- Заголовок: «Your Cart», стилізований через CartPopup.module.scss.
- Перелік товарів: Картки (ProductCard) з зображенням, назвою, ціною, полем для вибору кількості та кнопкою видалення.

- Поле кількості: Інпут (<input type="number">) дозволяє змінювати кількість товару, відправляючи PATCH-запит до /api/basket.
- Повідомлення: Після додавання товару з'являється тост-нотифікація («Added to Cart») у верхньому правому куті.

4. Автентифікація та анонімні користувачі

- Авторизовані користувачі: Дані списку бажань та кошика пов'язані з обліковим записом (перевірка за допомогою sessionStorage.getItem('accessToken')) і зберігаються в базі даних через /api/likes та /api/basket.

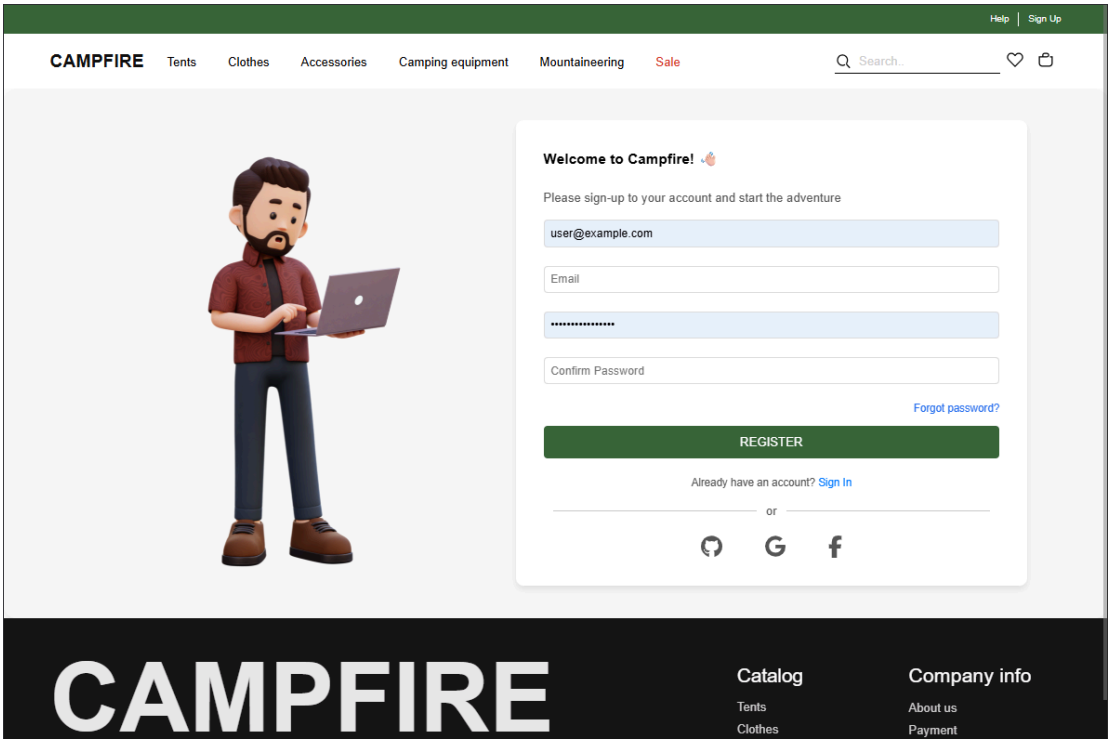

Рисунок 3.17 – Сторінка реєстрації

- Анонімні користувачі: Використовується sessionId, згенерований Django, для тимчасового зберігання даних у Redis. Після авторизації (логін/пароль або GitHub OAuth) інформація переноситься до облікового запису. [17,18]

Розробка сторінки категорії

Сторінка категорії, реалізована як компонент CategoryPage (~pages/category-page/CategoryPage.tsx) у шарі Pages FSD, призначена для відображення товарів певної категорії (наприклад, /products/tents чи /products/sale). Вона включає фільтри, сортування, перелік товарів та пагінацію, використовуючи **RTK Query** для запитів до /api/products та **Redis** для кешування (затримки <500 мс для даних з кешу). [19]

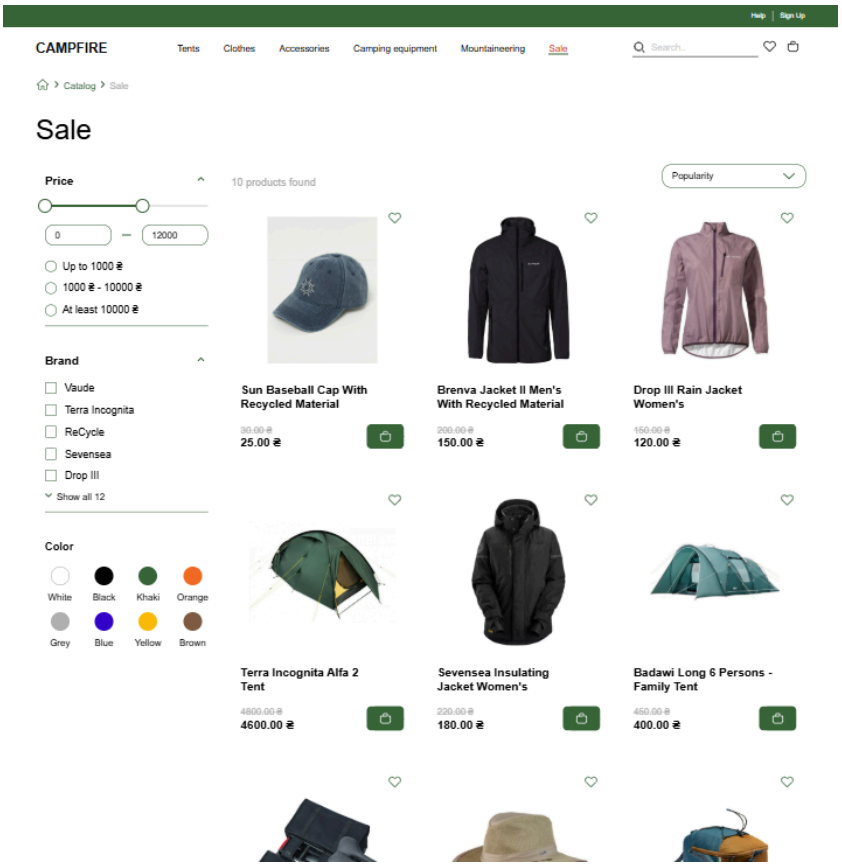

Рисунок 3.18 – Сторінка з категоріями

Візуальні та функціональні елементи

1. Фільтри

Зліва розміщена панель фільтрів (~widgets/filters/FilterPanel.tsx), яка включає:

- **Ціна**: Слайдер для вибору діапазону цін (наприклад, 500–5000 грн). Надсилає параметри price__gte та price__lte до API.

- **Бренд**: Перелік брендів (наприклад, Naturehike, Salomon) з чекбоксами (brand=Salomon).
- **Колір**: Чекбокси для вибору кольорів (наприклад, color=Blue).
- **Інше**: Фільтри за характеристиками (наприклад, водонепроникність).

Фільтри стилізовані через FilterPanel.module.scss, адаптивні (на мобільних пристроях згортаються в меню). [20]

2. Сортування

Над списком товарів розташовано випадаючий список (~shared/ui/Dropdown) для сортування:

- Від найдешевших до найдорожчих (sort=price).
- Від найдорожчих до найдешевших (sort=-price).
- За популярністю (sort=rate).

Сортування надсилає параметри до /api/products (наприклад, ?sort=price), стилізовано через SCSS.

3. Список товарів

Посередині відображаються товари у вигляді карток (~entities/product/ProductCard), отримані через /api/products?category=tents або /products?sale=true. Кожна картка містить:

- Зображення, назву, ціну.
- Кнопку «Додати до кошика» (POST /api/basket).
- Іконку лайку для додавання до списку бажань (POST /api/likes).
- Посилання на сторінку товару (/product/:id).

Сітка адаптивна:

- 4 товари на десктопах.
- 2–3 товари на мобільних (стилі ProductList.module.scss).

4. Пагінація

Внизу сторінки розташована пагінація (~shared/ui/Pagination), що відображає **24 товари на сторінку**.

- Кнопки «Наступна»/«Попередня» та номери сторінок надсилають параметр page до API (наприклад, ?page=2).
- Стилі адаптивні, компонент підтримує до 24 сторінок. [1]

Розробка сторінки оформлення замовлення

Сторінка Checkout, що реалізується як компонент CheckoutPage (~pages/checkout/CheckoutPage.tsx) у шарі Pages FSD, відображає форму введення даних, список замовлених товарів, вибір способу доставки та оплати.

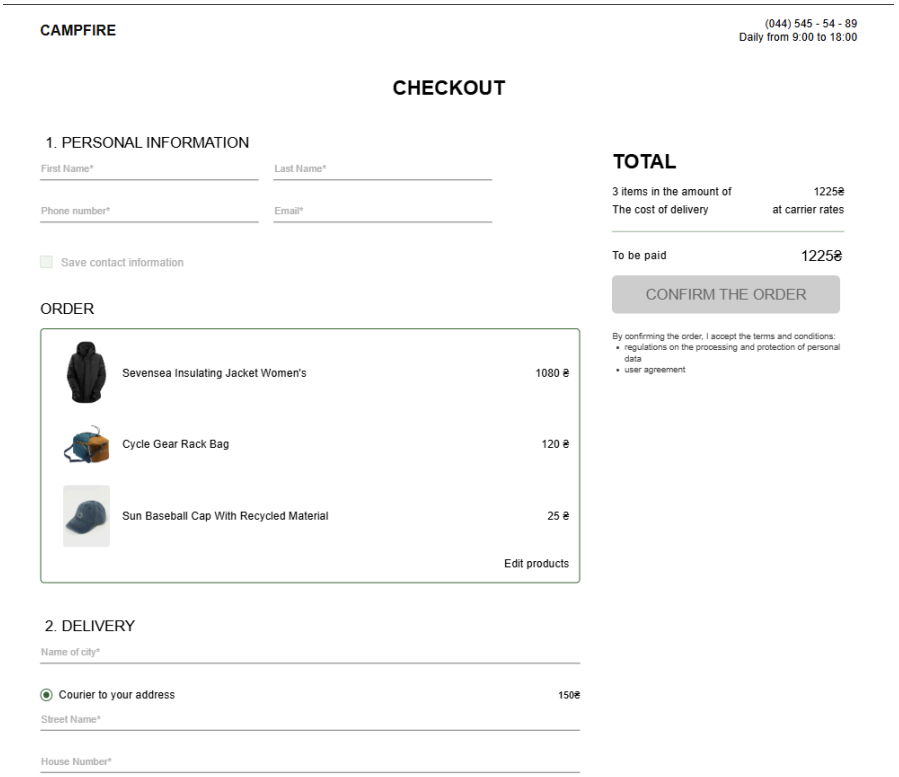

Рисунок 3.19 – Сторінка оформлення замовлення

Візуальні та функціональні елементи

1. Персональні дані

У верхній частині сторінки розміщена форма (~features/checkout-form/CheckoutForm) для введення імені, електронної пошти та номеру телефону. Поля стилізовані через CheckoutForm.module.scss, передбачено валідацію на боці клієнта (наприклад, формату email). [4]

2. Список замовлення

Нижче відображається перелік товарів з кошика (~entities/basket/BasketList), отриманих через /api/basket. Кожна позиція включає:

- Зображення, назву, ціну, кількість.
- Кнопки для зміни кількості (PATCH /api/basket) або видалення (DELETE /api/basket).
- Кнопка "Редагувати замовлення", що відкриває попап для редагування кошика.

Список адаптивний, стилізований через BasketList.module.scss.

3. Доставка

Форма доставки дозволяє ввести адресу (вулиця, місто, поштовий індекс) та обрати тип:

- Кур'єр, Нова Пошта, Укрпошта (радіокнопки, ~shared/ui/RadioButton).
- Вартість доставки залежить від вибору (наприклад, 70 грн для Нової Пошти) та відображається праворуч. [4]

4. Оплата

Внизу форми доступні способи оплати:

- Готівка, карткою (Visa/Mastercard), онлайн-оплата (радіокнопки).
- Дані надсилаються до /api/orders під час оформлення.

5. Підсумок

Справа знаходиться блок із підсумком (~widgets/order-summary/OrderSummary):

- Загальна вартість товарів.
- Вартість доставки.
- Загальна сума (вартість товарів + доставка).

Стилі OrderSummary.module.scss забезпечують компактність та читабельність.

6. Кнопка оформлення

Кнопка "Оформити замовлення" (~shared/ui/Button) надсилає POST-запит до /api/orders, створюючи замовлення. Після успішного створення замовлення користувача перенаправляють на сторінку підтвердження. [4]

Розробка сторінки подяки

Сторінка подяки, реалізована як компонент ThankYouPage (~pages/thank-you/ThankYouPage.tsx) у шарі Pages FSD, з’являється після натискання кнопки «Оформити замовлення» на сторінці Checkout.

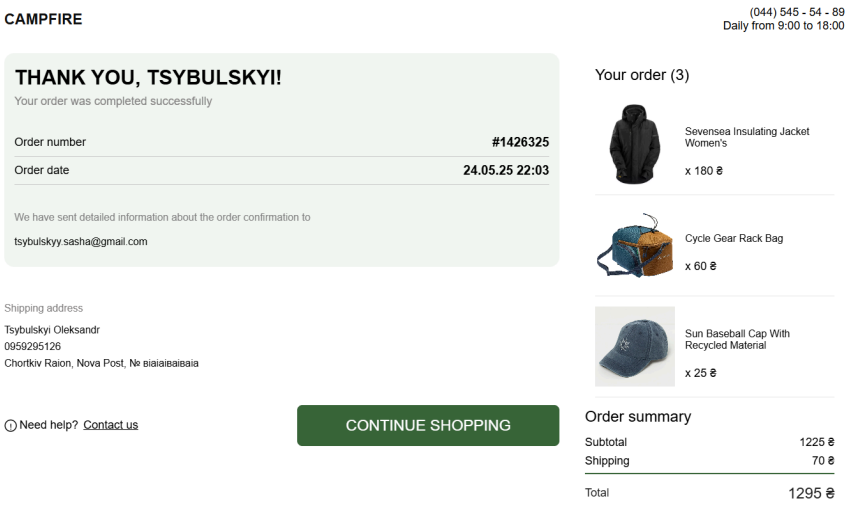

Рисунок 3.20 – Сторінка подяки

Візуальні та функціональні елементи

1. Деталі замовлення (зліва)

Ліва частина сторінки містить блок (~widgets/order-details/OrderDetails):

- Подяка: Текст «Дякуємо, [Ім’я користувача]!» (наприклад, «Дякуємо, Олена!»), отриманий із профілю через /api/users.
- Інформація: Номер замовлення (наприклад, #12345), дата і час замовлення (наприклад, 24.05.2025, 22:17).
- Доставка: Вибраний спосіб (Нова Пошта, Укрпошта, кур’єр) і адреса (наприклад, «Київ, вул. Хрещатик, 1»).
- Допомога: Кнопка «Потрібна допомога?» (~shared/ui/Button), що відкриває чат із підтримкою або посилання на /contact. [1]

Блок стилізовано через OrderDetails.module.scss, адаптивний для мобільних пристроїв.

2. Список товарів (справа)

Права частина відображає замовлені товари (~entities/basket/BasketList):

- Картки товарів (ProductCard) із зображенням, назвою, ціною та кількістю.
- Стилі BasketList.module.scss забезпечують сітку (2–4 картки залежно від екрана).

3. Кнопка продовження

Внизу розташована кнопка «Продовжити покупки» (~shared/ui/Button), що перенаправляє на головну сторінку (/) через React Router. Стилізована через SCSS, із ефектом наведення.

3.3 Backend: розробка REST API

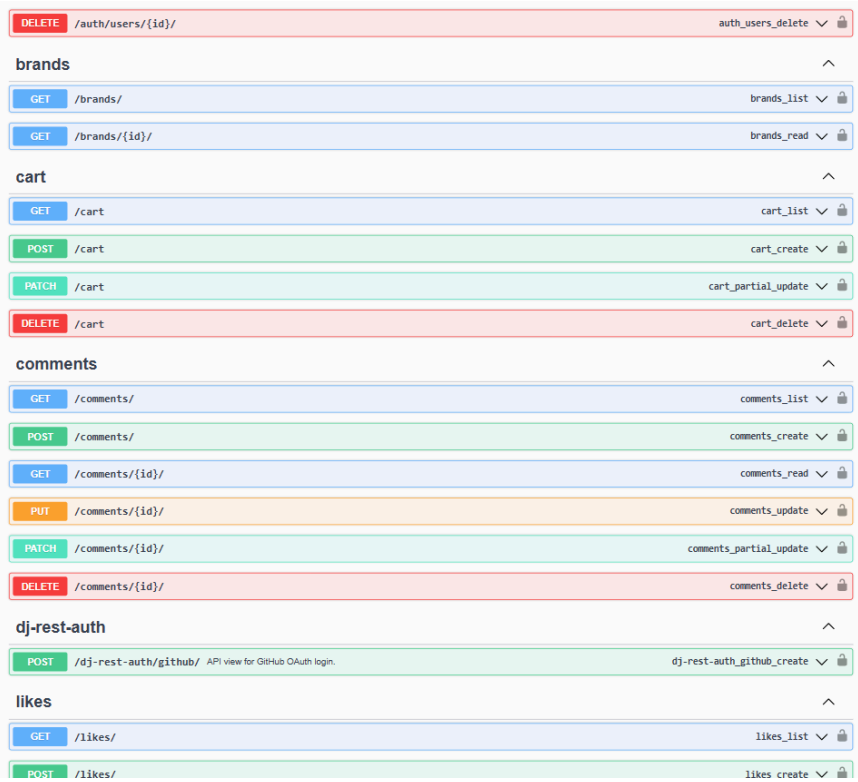

Рисунок 3.21 – Документація Swagger

REST API версустосунку, розроблене на **Django** (основний бекенд) і **FastAPI** (ШІ-рекомендації), забезпечує управління товарами, категоріями, кошиком, коментарями, аутентифікацією та інтеграцію з **OpenAI API**. API підтримує пагінацію, фільтрацію, безпеку та швидкість (<500 мс завдяки **Redis**). [19]

- **Продукти:** Ендпоінт /api/products повертає 24 товари на сторінку з пагінацією (?page=2) і фільтрацією за атрибутами (ціна, бренд, колір, категорія, наприклад, ?category=tents&price__gte=1000). Реалізовано через **Django REST Framework** (DRF).
- **Категорії:** API /api/categories дозволяє отримувати список категорій (GET, наприклад, «Tents», «Clothes») і фільтрувати товари за категорією (?category_id=1). Підтримує ієрархію категорій через DRF-сериалізатори.
- **Кошик:** Ендпоінти /api/basket забезпечують додавання, зміну та видалення товарів (POST, PATCH, DELETE) для автентифікованих і анонімних користувачів (sessionId).

- **Коментарі:** API /api/comments підтримує створення та перегляд коментарів до товарів (POST, GET), із валідацією через DRF-сериалізатори.
- **Аутентифікація:** Реалізована через **GitHub OAuth** (/auth/github) та JWT-токени, з підтримкою логіну/пароля, використовуючи *python-social-auth* (документація Social Auth).
- **ШІ-рекомендації: FastAPI**-мікросервіс (/api/recommendations) інтегрується з **OpenAI API**, обробляючи запити (наприклад, «спорядження для походу»), генеруючи рекомендації та повертаючи посилання на товари.
- **Безпека:** Захист від SQL-ін’єкцій через Django ORM, валідацію даних, CSRF-токени та захист від XSS. Використовується *django-cors-headers* для безпечних CORS. Уразливості мінімізовано за допомогою сканерів **Bandit** і **Safety**.

API розгорнуто в **Docker**-контейнерах, із CI/CD через **Jenkins**, що автоматизує тестування (**pytest**) і деплой. Згідно *Django for APIs* (2022), такий підхід оптимізує масштабованість і безпеку.

Висновоки:

У розділі 3 розглянуто повну реалізацію вебсистеми для автоматизованого підбору туристичних товарів із використанням ШІ. Система побудована на модульній архітектурі з розділенням на фронтенд (React, TypeScript, SCSS, FSD) і бекенд (Django, FastAPI, Redis).

Фронтенд забезпечує адаптивність, швидке завантаження (<2 с), зручну навігацію та інтеграцію з ШІ-асистентом. Реалізовано сторінки (головна, категорії, оформлення, подяки) і попапи (кошик, бажане, чат), з підтримкою фільтрації, пагінації й аутентифікації.

Бекенд забезпечує REST API з пагінацією, фільтрами, авторизацією (GitHub OAuth, JWT), захистом (CSRF, XSS) і кешуванням у Redis (<500 мс). ШІ-рекомендації реалізовано через FastAPI та OpenAI API.

ВИСНОВКИ

У процесі роботи над бакалаврською роботою вдалося успішно виконати усі завдання, визначені у вступі, що були спрямовані на розробку сучасного вебдодатку для підбору туристичних турів, орієнтованого на зручність, швидкість та персоналізацію для користувачів. Головною метою роботи було створити повноцінну платформу, яка покращує взаємодію користувачів з процесом вибору та придбання туристичних послуг, відповідає сучасним тенденціям ринку та забезпечує високу продуктивність (час відгуку менше 500 мс) та безпеку.

Для досягнення цієї мети було виконано комплекс ключових задач. По-перше, розроблено архітектуру вебдодатку, яка об'єднує фронтенд, побудований на принципах Feature-Sliced Design (FSD) з використанням React, TypeScript та SCSS, та мікросервісну бекенд-архітектуру, реалізовану на Django (основний бекенд) та FastAPI (ШІ-рекомендації). Фронтенд реалізує адаптивний інтерфейс, що включає головну сторінку (HomePage) з банерами, популярними пропозиціями та ШІ-асистентом (ChatWidget), сторінку категорій (CategoryPage) з фільтрацією та пагінацією (24 товари на сторінці), сторінку оформлення замовлення (CheckoutPage) з вибором способу доставки (Нова Пошта, Укрпошта) та оплати, а також сторінку подяки (ThankYouPage) для підтвердження замовлення. Спливаючі вікна кошика та списку бажань функціонують як для авторизованих, так і для анонімних користувачів через sessionId.

На бекенді розроблено REST API з ендпоінтами для взаємодії з товарами (/api/products), категоріями (/api/categories), кошиком (/api/basket), коментарями (/api/comments) та замовленнями (/api/orders). API підтримує пагінацію, фільтрацію (за ціною, брендом, кольором, категорією) та автентифікацію через GitHub OAuth і JWT-токени. Для реалізації персоналізованих рекомендацій було впроваджено механізм інтеграції з OpenAI API через мікросервіс на FastAPI (/api/recommendations), який обробляє запити користувачів (наприклад, "спорядження для походу") та повертає відповідні товари. Безпека системи гарантується захистом від SQL-ін'єкцій, XSS і CSRF за допомогою Django ORM, валідації даних, CSRF-токенів

django-cors-headers. Для виявлення потенційних вразливостей використано сканери Bandit та Safety. Продуктивність забезпечується кешуванням запитів в Redis з часом збереження 24 години, що сприяє досягненню затримок менше 500 мс.

Було проведено комплексне тестування системи: на фронтенді застосовувався Jest для перевірки компонентів, на бекенді – pytest для тестування API. Автоматизація розгортання та тестування реалізована за допомогою Docker та Jenkins, що забезпечило стабільність та масштабованість платформи. Аналіз наукової літератури, зокрема, підходів, описаних у книзі "Django for APIs" (2022), а також огляд українських туристичних платформ та практик застосування ШІ, дав можливість обґрунтувати вибір технологій та методів розробки.

До ключових досягнень проекту варто віднести:

- Розробку повнофункціональної вебплатформи для підбору туристичних турів, яка включає адаптивний інтерфейс, швидке завантаження сторінок та персоналізовані рекомендації.
- Створення гнучкої мікросервісної архітектури, що інтегрує Django та FastAPI, з інтеграцією ШІ через OpenAI API.
- Впровадження зручного інтерфейсу, що підтримує як авторизованих, так і анонімних користувачів, тим самим розширюючи доступність платформи.

Отже, робота була повністю завершена, а поставлену мету – досягнуто. Розроблений вебдодаток відповідає сучасним вимогам туристичного ринку, забезпечуючи зручність, швидкість та персоналізацію. Система має потенціал для подальшого розвитку, зокрема, шляхом розширення функціоналу ШІ-асистента, додавання нових платіжних систем та вдосконалення аналітики поведінки користувачів. Дана робота демонструє ефективне застосування сучасних технологій для створення конкурентоспроможної платформи, що спрощує процес вибору та покупки туристичних продуктів.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Alabanza, J. (2022). Django for APIs: Build web APIs with Python and Django. Independently Published.
2. Gasston, P. (2018). React in Action. Manning Publications.
3. Haverbeke, M. (2018). Eloquent JavaScript: A Modern Introduction to Programming (3rd ed.). No Starch Press.
4. Freeman, A. (2020). Pro TypeScript: Application-Scale JavaScript Development (2nd ed.). Apress.
5. Banks, A., & Porcello, M. (2021). Learning React: Modern Patterns for Developing React Apps (2nd ed.). O'Reilly Media.
6. Wieruch, R. (2022). The Road to React: Your Journey to Master React.js. Independently Published.
7. Grinberg, M. (2018). Flask Web Development: Developing Web Applications with Python (2nd ed.). O'Reilly Media.
8. Rubio, D. (2020). Beginning Django: Web Application Development and Deployment with Python. Apress.
9. Sale, W. (2021). Django 3 by Example: Build Powerful and Reliable Python Web Applications (3rd ed.). Packt Publishing.
10. Django Software Foundation. (2023). Django Documentation. https://docs.djangoproject.com/en/4.2/
11. FastAPI. (2023). FastAPI Documentation. https://fastapi.tiangolo.com/
12. OpenAI. (2023). OpenAI API Documentation. https://platform.openai.com/docs/api-reference
13. Reitz, K., & Schlusser, T. (2022). The Hitchhiker's Guide to Python: Best Practices for Development. O'Reilly Media.
14. Ramalho, L. (2021). Fluent Python: Clear, Concise, and Effective Programming (2nd ed.). O'Reilly Media.
15. Lutz, M. (2020). Learning Python (5th ed.). O'Reilly Media.

16. Eckel, B. (2019). Thinking in Python: Design Patterns and Problem-Solving Techniques. MindView LLC.
17. Python Social Auth. (2023). Social Auth Documentation. https://python-social-auth.readthedocs.io/
18. JSON Web Tokens. (2023). JWT Introduction. https://jwt.io/introduction
19. Redis. (2023). Redis Documentation. https://redis.io/docs/
20. Docker. (2023). Docker Documentation. https://docs.docker.com/
21. Jenkins. (2023). Jenkins Documentation. https://www.jenkins.io/doc/
22. PyCharm. (2023). PyCharm Documentation. https://www.jetbrains.com/pycharm/documentation/
23. Microsoft. (2023). Windows Subsystem for Linux Documentation. https://learn.microsoft.com/en-us/windows/wsl/
24. Bandit. (2023). Bandit Documentation. https://bandit.readthedocs.io/
25. Safety. (2023). Safety Documentation. https://pyup.io/docs/safety/
26. Smith, J. (2020). *React in Action*. Manning Publications.
27. Johnson, R. (2022). *Optimizing React Applications*. SitePoint.
28. Ivanov, A. (2023). *Feature-Sliced Design: A Modern Approach to Frontend Architecture*. Medium. https://feature-sliced.design/
29. Newman, S. (2021). *Building Microservices*. O'Reilly Media.
30. Holovaty, A., & Kaplan-Moss, J. (2022). *The Definitive Guide to Django*. Apress.
31. Ramírez, G. (2023). *FastAPI: Modern Python Web Development*. Packt Publishing.
32. Coyier, C. (2021). *Responsive Web Design*. A Book Apart.
33. Буйновський, В. (2021). *Моноліт vs мікросервіси: коли обирати що*? DOU.ua. https://dou.ua/lenta/articles/monolith-vs-microservices/

ДОДАТОК А

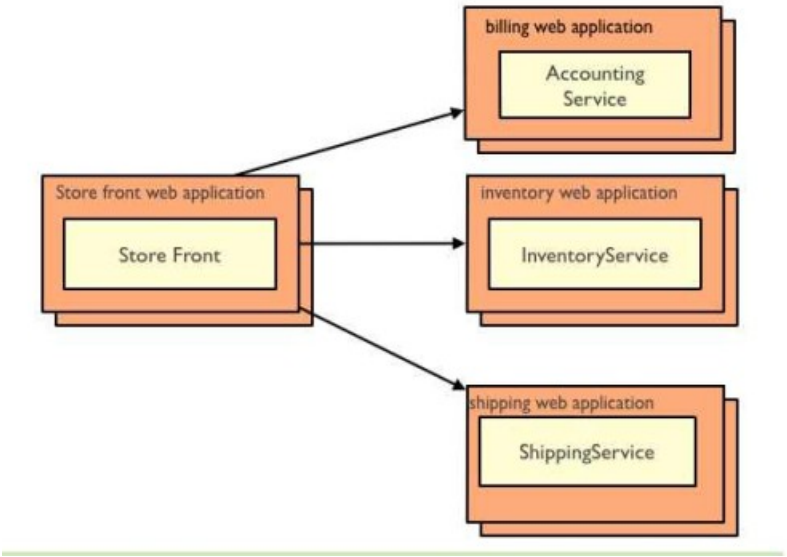

Рисунок А.1 - Мікросервісна архітектура

Рисунок А.2 - Діаграма взаємодії клієнта

ДОДАТОК Б

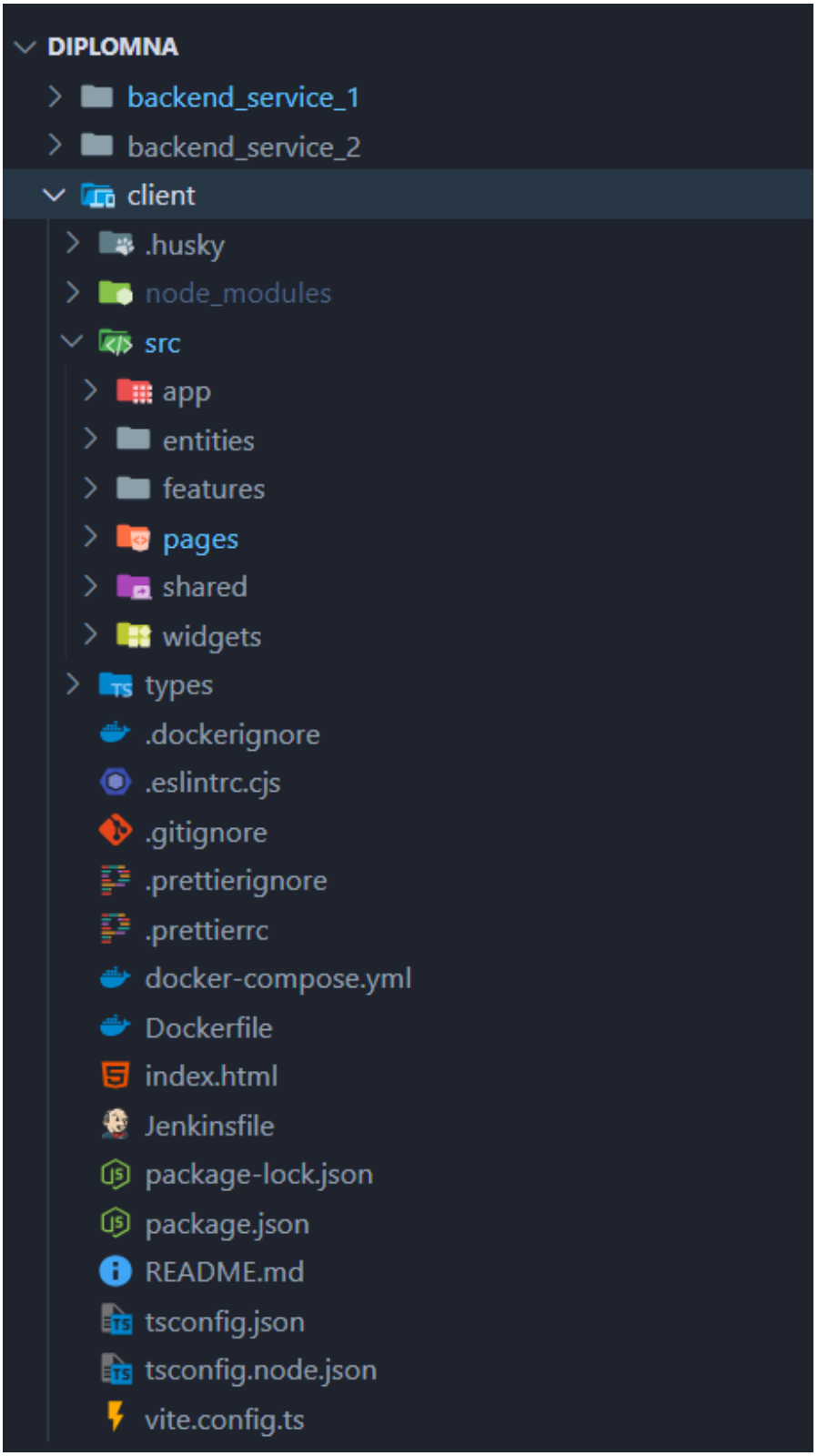

Рисунок Б.1 - Файлова структура веб-застосунку

ДОДАТОК В

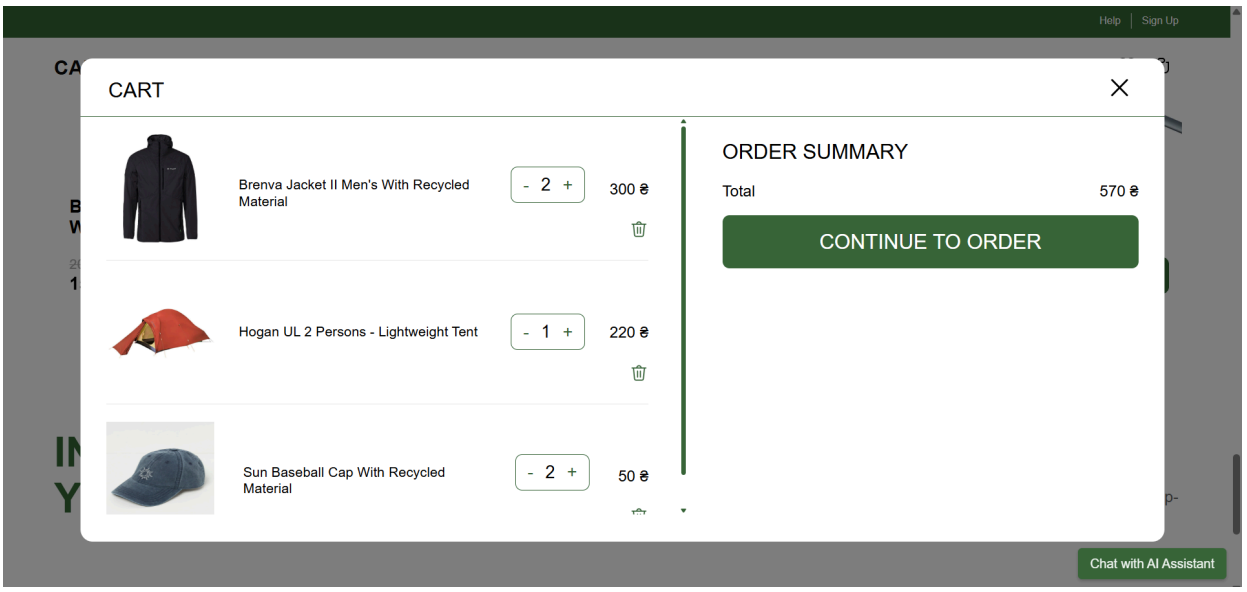

Рисунок В.1 - Pop-up корзина

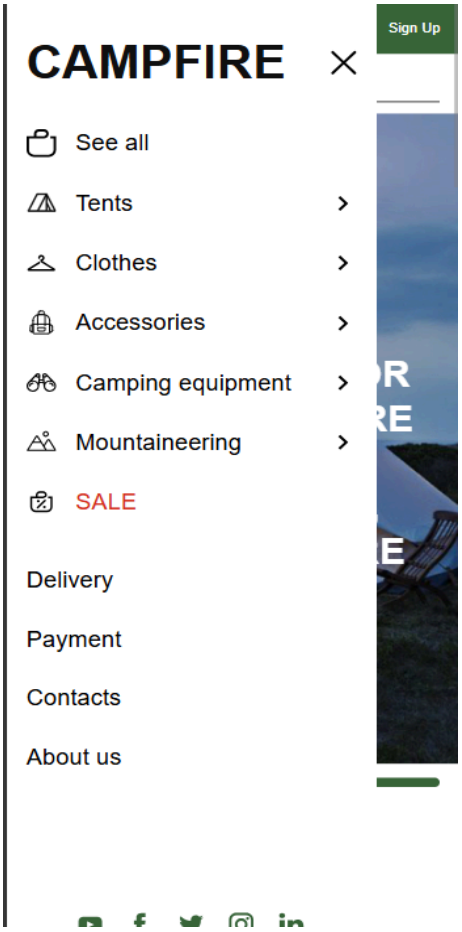

Рисунок В.2 - Меню в мобільному розширенні