Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук

Бакалаврська кваліфікаційна робота на тему:

РОЗРОБКА ІНТЕРНЕТ-МАГАЗИНУ КОМП'ЮТЕРНОЇ ТЕХНІКИ НА ОСНОВІ ХМАРНИХ ТЕХНОЛОГІЙ

Виконав: студент 4 курсу, групи 4КН-21б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Дидь К. В.
(прізвище та ініціали)

Керівник: к.т.н., доцент каф. КН
Крилик Л. В.
(прізвище та ініціали)
«04» червня 2025 р.

Рецензент: к.т.н., доцент каф. АІІТ
Богач І. В.
(прізвище та ініціали)
«05» червня 2025 р.

Допущено до захисту
Зав. кафедри Яровий А. А.
«06» червня 2025 р.

Вінниця ВНТУ - 2025 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Галузь знань – 12 Інформаційні технології
Спеціальність – 122 Комп'ютерні науки
Освітньо-професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
проф., д.т.н. Яровий А. А.
«05» липня 2025 р.

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
Дидю Костянтину Вікторовичу

1. Тема роботи: Розробка інтернет-магазину комп'ютерної техніки на основі хмарних технологій.
керівник роботи: Крилик Людмила Вікторівна, к.т.н., доц.
затверджені наказом вищого навчального закладу «20» березня 2025 року №97
2. Термін подання студентом роботи 30 червня 2025 р.

3. Вихідні дані до роботи: мова програмування – об'єктно-орієнтована; кількість підтримуваних платформ не менше 3; система має забезпечувати кошик та оформлення замовлення; інтерфейс користувача має бути інтуїтивно зрозумілий; система має використовувати хмарні сервіси.

4. Зміст текстової частини (перелік питань, які потрібно розробити): вступ; обґрунтування доцільності розробки інтернет-магазину комп'ютерної техніки на основі хмарних технологій; проектування інтернет-магазину комп'ютерної техніки на основі хмарних технологій; програмна реалізація інтернет-магазину комп'ютерної техніки на основі хмарних технологій; висновки; список використаних джерел; додатки.

5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень): загальна структурна схема функціональних модулів, UML-діаграма класів фронтенд-частини, UML-діаграма класів мікросервісу API, схема алгоритму роботи інтернет-магазину комп'ютерної техніки, архітектурна діаграма хмарного середовища інтернет-магазину комп'ютерної техніки, ER-модель предметної області «Комп'ютерна техніка», приклад роботи програми.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1–3	Крилик Л. В., к.т.н., доц. каф. КН		

7. Дата видачі завдання 05 травня 2025 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Вступ. Обгрунтування доцільності розробки інтернет-магазину комп'ютерної техніки на основі хмарних технологій	05.05.25	07.05.25	
2	Проектування інтернет-магазину комп'ютерної техніки на основі хмарних технологій	08.05.25	11.05.25	
3	Програмна реалізація інтернет-магазину комп'ютерної техніки на основі хмарних технологій	12.05.25	25.05.25	
4	Висновки та аналіз отриманих результатів	16.05.25	26.05.25	
5	Оформлення додатків	27.05.25	29.05.25	
6	Розробка інструкції користувача	30.05.25	01.06.25	
7	Оформлення матеріалів до захисту БКР	02.06.25	04.06.25	

Студент _____ **Дидь К. В.**
(підпис) (прізвище та ініціали)

Керівник роботи _____ **Крилик Л. В.**
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська кваліфікаційна робота складається з 101 сторінки формату А4, які включають 23 рисунки і 3 таблиці. У списку використаних джерел зазначено 35 найменування.

Метою роботи є розширення функціональних можливостей інтернет-магазину комп'ютерної техніки за допомогою впровадження хмарних технологій.

У загальній частині роботи на основі аналізу ринку інтернет-торгівлі та огляду популярних хмарних платформ обґрунтовано доцільність створення інтернет-магазину саме на базі архітектури мікросервісів із використанням платформи Amazon Web Services. Розроблено функціональну структуру WEB-застосунку, алгоритм роботи, ER-модель бази даних та UML-діаграми класів.

Програмний продукт реалізовано з використанням сучасних WEB-технологій: фронтенд – на основі React, бекенд – з використанням Python, базу даних реалізовано на основі NoSQL. Результати тестування підтвердили працездатність, стабільність та відповідність функціональності поставленим вимогам.

Ключові слова: інтернет-магазин, хмарні технології, AWS, мікросервісна архітектура, React.

ABSTRACT

The bachelor's thesis consists of 101 A4 pages, including 23 figures and 3 tables. The list of references includes 35 titles.

The purpose of the work is to expand the functionality of an online computer store by introducing cloud technologies.

In the general part of the work, based on the analysis of the e-commerce market and a review of popular cloud platforms, the author substantiates the expediency of creating an online store based on the microservices architecture using the Amazon WEB Services platform. The functional structure of the web application, the algorithm of work, the ER-model of the database and UML-diagrams of classes are developed.

The software product is implemented using modern WEB technologies: the frontend is based on React, the backend is based on Python, and the database is implemented on the basis of NoSQL. The test results confirmed the efficiency, stability, and compliance of the functionality with the requirements.

Keywords: online store, cloud technologies, AWS, microservice architecture, React.

ЗМІСТ

ВСТУП

Актуальність досліджень.

Актуальність інтернет-магазинів у сучасному світі важко переоцінити. Все більше людей обирають онлайн-покупки через зручність, швидкість і широкий вибір товарів, що доступні в будь-який час. Широку популярність інтернет-шопінг здобув під час пандемії COVID-19. Онлайн-шопінг продемонстрував себе як основний спосіб отримання доступу до товарів під час обмежень та соціального дистанціювання.

З основних переваг, що стосуються цієї теми, можна виділити:

- швидкий доступ до товарів і послуг. Споживачі прагнуть зручного, персоналізованого та швидкого доступу до товарів, особливо у сфері комп'ютерної техніки. Використання хмарних технологій дозволяє забезпечити високу доступність платформи для клієнтів;

- ефективне управління ресурсами. Хмарні технології забезпечують гнучкість, масштабованість і зниження витрат на інфраструктуру, що є важливим для сучасних онлайн-магазинів;

- зростання конкуренції в індустрії електронної комерції. Розробка онлайн-магазинів на основі хмарних технологій дозволяє значно підвищити конкурентоспроможність за рахунок швидкого впровадження нових функцій, зручності для користувачів та високої надійності.

Отже, тема «Розробка інтернет-магазину комп'ютерної техніки на основі хмарних технологій» є важливою та актуальною в контексті сучасних тенденцій розвитку електронної комерції та впровадження інноваційних технологій.

Мета дослідження є розширення функціональних можливостей інтернет-магазину комп'ютерної техніки шляхом використання хмарних технологій.

Об’єкт дослідження є процес організації та функціонування онлайн-магазину.

Предмет дослідження є програмні засоби для реалізації онлайн-магазину на основі хмарних технологій.

Завдання дослідження:

1. Обґрунтувати доцільність розробки інтернет-магазину комп'ютерної техніки на основі хмарних технологій.
2. Спроектувати архітектуру інтернет-магазину комп'ютерної техніки на основі хмарних технологій.
3. Програмно реалізувати інтернет-магазин комп'ютерної техніки на основі хмарних технологій.
4. Виконати тестування системи та провести аналіз отриманих результатів.
5. Розробити інструкцію користувача.

Апробація роботи.

Результати досліджень було апробовано на LIV Всеукраїнській науково-технічній конференції підрозділів Вінницького національного технічного університету (НТКП ВНТУ) м. Вінниці у 2025 р. [1].

Публікації: за основними результатами досліджень опубліковано тези доповіді на науково-технічній конференції [1] та подано заявку на реєстрацію авторського права на твір у формі комп’ютерної програми – «Інтернет-магазин комп’ютерної техніки на основі хмарних технологій», номер заявки С202502844 від 01.04.2025 р.

1 ОБҐРУНТУВАННЯ ДОЦІЛЬНОСТІ РОЗРОБКИ ІНТЕРНЕТ-МАГАЗИНУ КОМП'ЮТЕРНОЇ ТЕХНІКИ НА ОСНОВІ ХМАРНИХ ТЕХНОЛОГІЙ

1.1 Аналіз сучасного стану ринку інтернет-торгівлі

Останніми роками популярність онлайн-покупок неухильно зростає. У 2021 році глобальні роздрібні продажі в Інтернеті склали майже п'ять трильйонів доларів США, і очікується, що до 2025 року ця цифра перевищить сім трильйонів доларів США [2]. Ці дані свідчать про стабільне зростання популярності онлайн-торгівлі та її поступове перетворення на основний канал роздрібних продажів у багатьох країнах світу. Така тенденція підкреслює важливість впровадження сучасних технологій і розробки інноваційних рішень.

В Україні ринок онлайн-торгівлі також демонструє стабільне зростання. Крім того, 16% усіх покупок українці наразі здійснюють онлайн. За інформацією пресслужби групи компаній EVO, яка є власником провідних інтернет-магазинів України, обсяг угод у сегменті e-commerce у 2024 році склав майже $4 млрд, що на третину перевищує показники 2023 року [3].

В Україні представлено значну кількість онлайн-магазинів, що спеціалізуються на продажі електроніки. Кожен із них має певний набір переваг і недоліків, які варто розглянути для визначення перспектив розробки нового додатку.

Отож, розглянемо найбільш поширені інтернет-магазини комп'ютерної техніки в Україні.

Telemart − інтернет-магазин, який спеціалізується на продажу комп'ютерної техніки, комплектуючих, периферійних пристроїв та аксесуарів (рис. 1.1). Магазин орієнтується на геймерів, IT-ентузіастів та професійних користувачів [4].

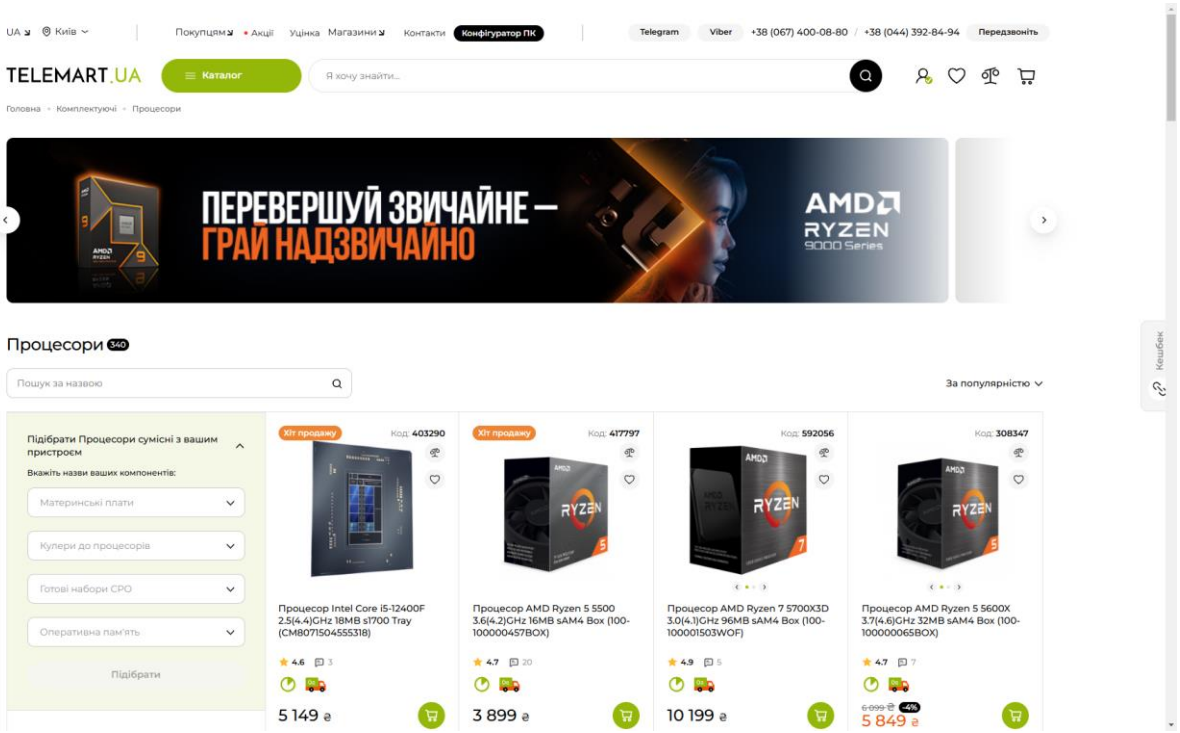

Рисунок 1.1 – Загальний вигляд інтерфейсного вікна інтернет-магазину Telemart

Переваги:

- Широкий асортимент ігрових комплектуючих (відеокарти, процесори, материнські плати).
- Велика кількість акцій та спеціальних пропозицій для геймерів та IT-фахівців.
- Можливість індивідуального складання ПК.
- Зручний фільтр пошуку товарів та докладні характеристики продукції.

Недоліки:

- Обмежений вибір бюджетних товарів.

Artline – це український виробник і онлайн-магазин, що спеціалізується на готових рішеннях у вигляді персональних комп'ютерів (рис. 1.2). Магазин також пропонує послуги складання ПК під замовлення [5].

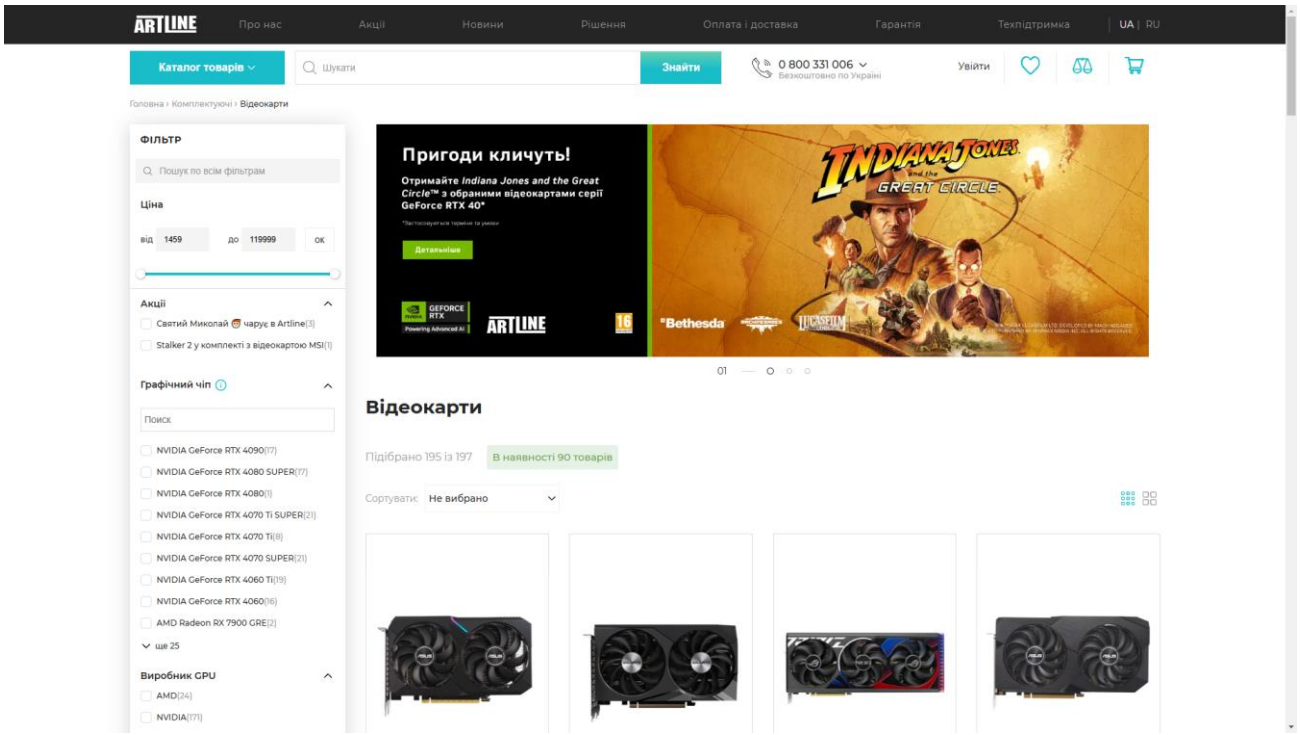

Рисунок 1.2 – Загальний вигляд інтерфейсного вікна інтернет-магазину Artline

Переваги:

- Прямий виробник готових ПК з орієнтацією на геймерів, офісні та професійні системи.
- Гарантія на всі товари та можливість апгрейду техніки.
- Продумана оптимізація системи – ПК приходять повністю готовими до роботи.
- Доступна вартість у сегменті готових рішень.

Недоліки:

- Обмежений вибір комплектуючих для окремого придбання.
- Відсутність фокусу на аксесуарах або периферії.
- Складність у персоналізації замовлення поза рамками запропонованих конфігурацій.

Hotline − це платформа-агрегатор цін (рис. 1.3), що дозволяє користувачам знаходити товари за найкращими цінами серед десятків інтернет-магазинів. Основна спеціалізація − пошук електроніки, зокрема комп'ютерної техніки та комплектуючих [6].

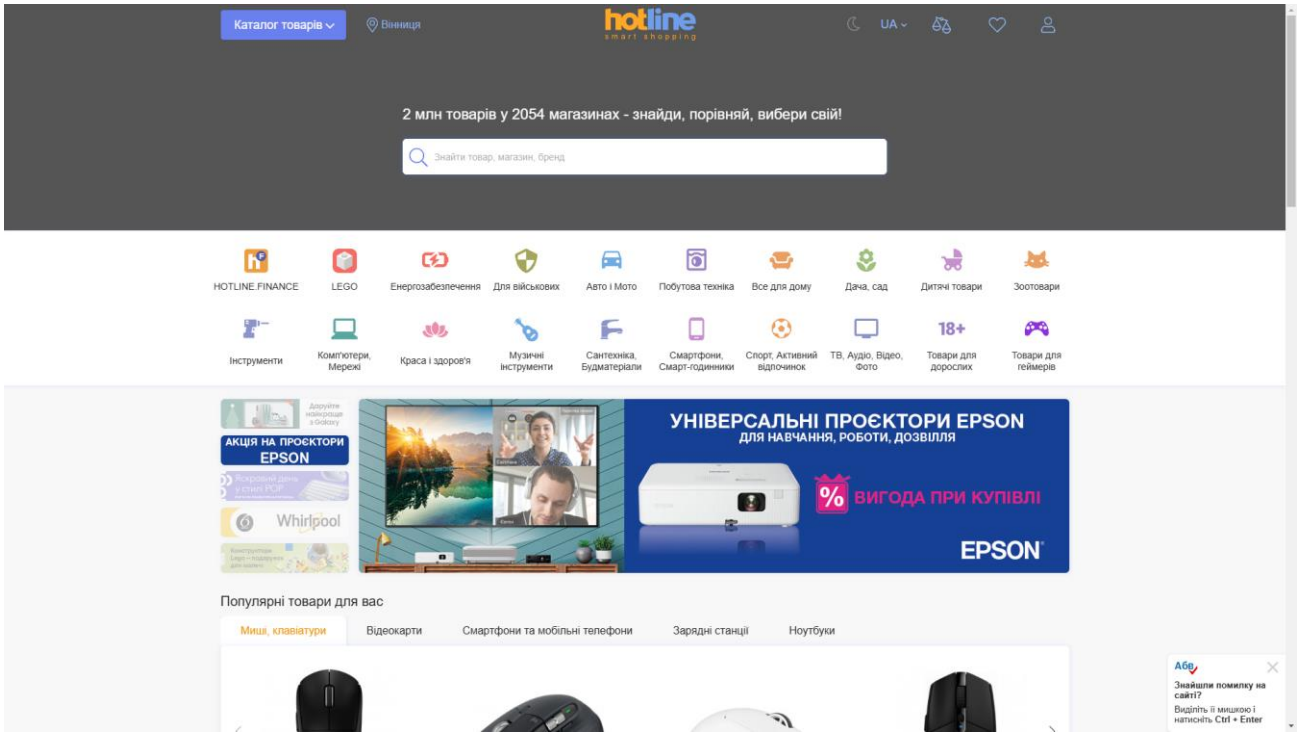

Рисунок 1.3 − Загальний вигляд інтерфейсного вікна інтернет-магазину Hotline

Переваги:

- Порівняння цін із різних магазинів в одному місці.
- Можливість ознайомлення з відгуками користувачів про магазини та товари.
- Зручна система фільтрації для пошуку необхідних характеристик товару.
- Економія часу та коштів завдяки прозорості цін.

Недоліки:

- Відсутність прямого продажу товарів.

- Іноді товари з найнижчою ціною можуть бути недоступними через відсутність актуального оновлення інформації.
- Залежність від надійності магазинів-партнерів, що може впливати на якість обслуговування.

1.2 Огляд доступних хмарних платформ та обґрунтування їх використання

Вибір хмарної платформи є одним із ключових рішень при розробці програмного забезпечення, адже від цього залежить продуктивність, масштабованість і надійність системи. Для реалізації каталогу комп'ютерної техніки важливо обрати платформу, яка забезпечить стабільну роботу, обробку великої кількості запитів та можливість інтеграції з аналітичними інструментами.

Серед сучасних хмарних платформ найчастіше використовуються Amazon Web Services (AWS), Microsoft Azure та Google Cloud Platform (GCP). Кожна з них має свої переваги, недоліки та особливості, які роблять її відповідною для певних типів проєктів. Отже, розглянемо детальніше ці хмарні платформи, щоб визначити оптимальне рішення для реалізації проєкту.

Amazon Web Services (AWS) − це комплексна платформа хмарних обчислень від Amazon (рис. 1.4). Вона пропонує широкий спектр послуг, включаючи обчислювальні потужності, рішення для зберігання даних і бази даних, що дозволяє бізнесу масштабуватися та впроваджувати інновації без необхідності значних початкових інвестицій. AWS, якому довіряють мільйони, відомий своєю надійністю, гнучкістю та розгалуженою глобальною інфраструктурою. AWS може похвалитися широким портфоліо з понад 200 хмарних сервісів, пропонуючи широкий спектр від обчислень і периферійних обчислень до безсерверних рішень [7, 8].

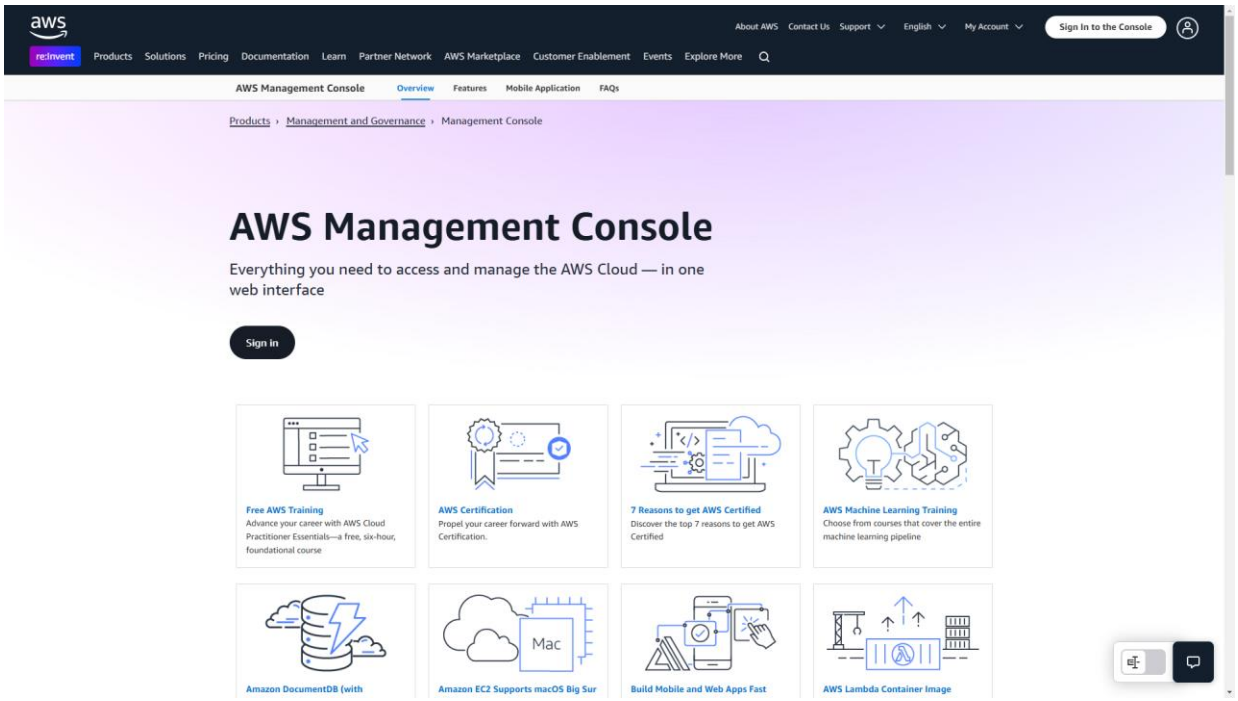

Рисунок 1.4 – Загальний вигляд інтерфейсного вікна платформи AWS

Переваги:

- AWS пропонує понад 200 хмарних сервісів, таких як обчислення, зберігання даних, бази даних;
- AWS має найбільшу кількість регіонів і зон доступності, що забезпечує низькі затримки та високу доступність;
- Опції on-demand, reserved та spot-інстанси дозволяють оптимізувати витрати;
- Найбільша кількість інтеграцій із сторонніми сервісами;
- Велика кількість офіційних посібників, ресурсів і навчальних матеріалів.

Недоліки:

- Через велику кількість сервісів і функцій, навчання та налаштування може бути складним.
- Хоча є гнучкі моделі, вартість сервісів може швидко зростати без оптимізації.

- AWS більш орієнтований на досвідчених розробників, що може ускладнити використання для бізнес-користувачів.

Microsoft Azure − це платформа хмарних обчислень (рис. 1.5), яка надає широкий спектр послуг, що допомагають організаціям створювати, керувати та розгортати додатки по всьому світу. З акцентом на безперешкодну інтеграцію з екосистемою Microsoft, Azure пропонує надійні рішення для обчислень, аналітики, роботи з мережею та багато іншого. Це надійний вибір для підприємств, що пропонує гібридний підхід і потужну підтримку різноманітних робочих навантажень [7, 9].

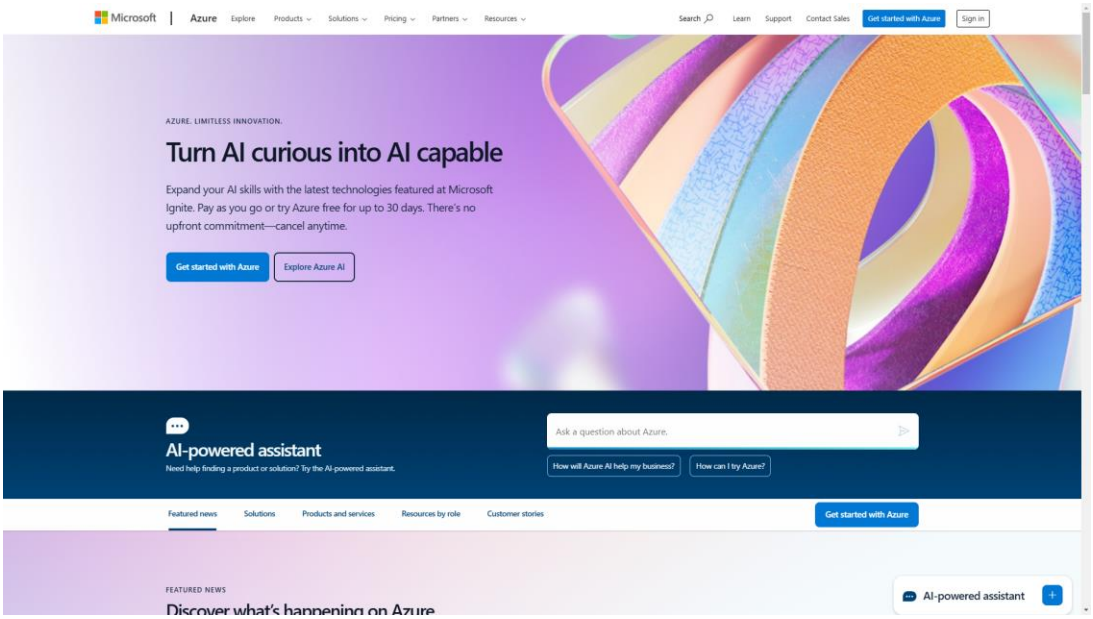

Рисунок 1.5 − Загальний вигляд інтерфейсного вікна платформи Azure

Переваги:

- глибока інтеграція з Windows Server, Active Directory, Office 365 та іншими корпоративними продуктами;

- Azure Arc дозволяє інтегрувати локальну інфраструктуру з хмарними сервісами;

- розвинені сервіси аналітики, підтримка DevOps, машинне навчання;
- Azure має великий список регіонів, що особливо корисно для локальних рішень у Європі, Азії та інших частинах світу.

Недоліки:

- Azure переважно оптимізований для Microsoft-орієнтованих компаній;
- у порівнянні з AWS, документація Azure менш структурована;
- налаштування гібридних рішень може бути надто складним і дорогим для малих компаній.

Google Cloud − це набір хмарних обчислювальних сервісів від Google (рис. 1.6), покликаних допомогти бізнесу використовувати можливості даних та інновацій. Зосереджуючись на аналітиці даних, машинному навчанні та технологіях з відкритим вихідним кодом, Google Cloud Platform (GCP) пропонує гнучку та масштабовану інфраструктуру. Це найкращий вибір для організацій, які шукають передові рішення та міцну основу для цифрової трансформації [7, 10].

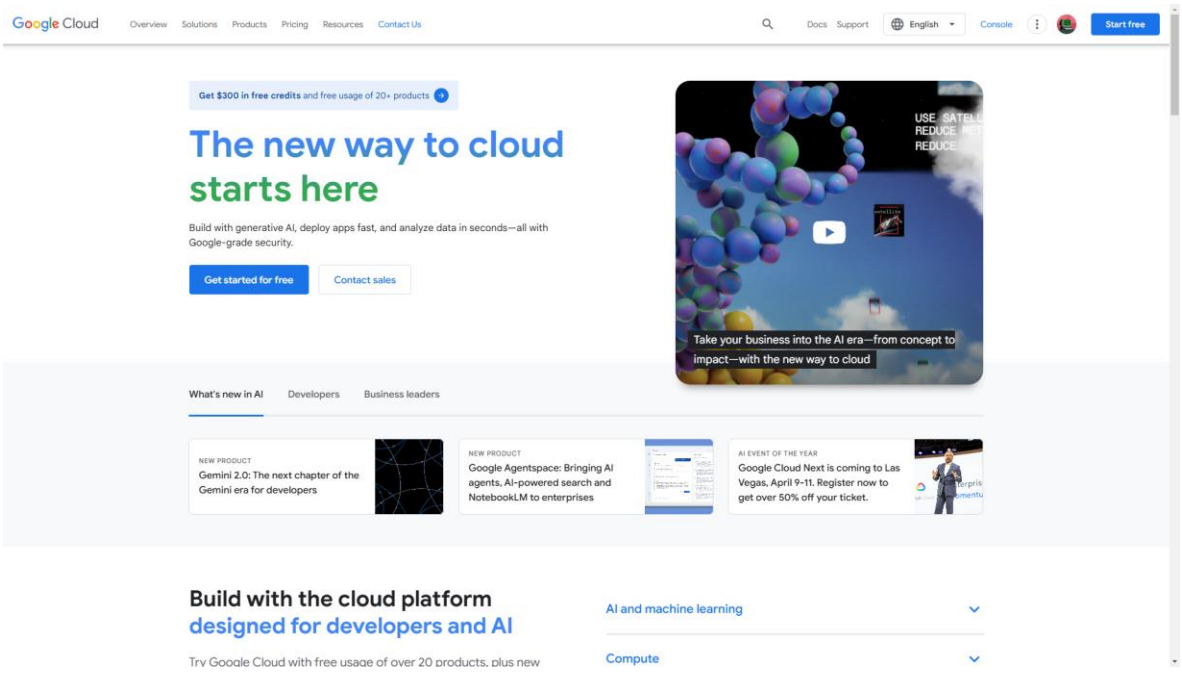

Рисунок 1.6 − Загальний вигляд інтерфейсного вікна платформи GCP

Google Cloud відомий своїми передовими засобами безпеки, що встановлюють галузеві стандарти надійного захисту даних і систем, а також гібридними та мультихмарними рішеннями, що забезпечують безпрецедентну гнучкість для організацій. GCP також пропонує першокласні рішення для сховищ даних, що дозволяють організаціям ефективно зберігати, керувати та аналізувати величезні обсяги даних [7, 10].

Переваги:

- провідні рішення для обробки великих даних та машинного навчання;
- інфраструктура Google забезпечує високу надійність і швидкодію;
- Kubernetes і багато інших рішень для оркестрації контейнерів;
- прозора модель ціноутворення з можливістю заощадження на довгострокових контрактах;
- акцент на використанні відновлювальних джерел енергії для центрів обробки даних.

Недоліки:

- GCP має меншу базу користувачів і спільноту у порівнянні з AWS та Azure;
- менша кількість регіонів та зон доступності у порівнянні з AWS та Azure;
- GCP найкраще підходить для Data Science, AI та аналітики, але може бути недостатньо універсальним для деяких корпоративних сценаріїв.

1.3 Формулювання вимог та постановка задачі

З кожним роком асортимент комп'ютерної техніки стає дедалі більшим, що ускладнює процес вибору потрібного обладнання. Особливо це стосується ситуацій, коли покупці намагаються знайти пристрої, що відповідають їхнім специфічним вимогам, таким як технічні характеристики, бренд чи ціновий діапазон. Додатково ускладнює вибір наявність численних платформ, кожна з яких має свої переваги,

недоліки та різні підходи до відображення інформації про товари. Для спрощення процесу вибору комп'ютерної техніки необхідне рішення, яке забезпечить доступ до структурованої інформації та дозволить порівнювати товари за різними критеріями.

На ринку України існують такі популярні платформи, як Artline, Hotline, Telemart та інші, які вже надають функції пошуку й порівняння техніки. Однак вони мають певні обмеження. Наприклад, Artline зосереджується на продажу готових ПК, а не на детальному порівнянні компонентів, що може бути незручним для тих, хто шукає комплектуючі або специфічні модифікації техніки. Hotline, є агрегатором, часто показує неточні або застарілі дані про наявність та ціни. Крім того, багато платформ не дозволяють створювати персоналізовані списки техніки чи налаштовувати оповіщення про зниження цін. Це створює потребу у створенні нового сервісу, який би усунув зазначені недоліки та запропонував користувачам зручний спосіб роботи з каталогом комп’ютерної техніки.

У розроблюваному каталозі комп'ютерної техніки будуть реалізовані такі основні функції:

- створення персональних списків обраних товарів;
- пошук і порівняння товарів за широким спектром характеристик;
- збереження фільтрів пошуку;
- персоналізовані рекомендації на основі уподобань користувачів.

Для реалізації каталогу необхідно забезпечити:

- розробку WEB-інтерфейсу, який дозволить користувачам легко шукати, порівнювати й зберігати інформацію про товари;
- використання хмарних технологій для забезпечення масштабованості, доступності та збереження великих обсягів даних;
- інтеграцію з базами даних, щоб автоматично оновлювати інформацію про наявність та ціни;
- реалізацію модулів для персоналізації;

- тестування системи на зручність використання та стабільність роботи.

Створення такого каталогу забезпечить споживачів зручним інструментом для вибору комп’ютерної техніки, враховуючи їхні потреби та індивідуальні вимоги. Система сприятиме зменшенню часу на пошук і підвищенню задоволеності користувачів.

1.4 Висновок

У цьому розділі проведено аналіз сучасного стану ринку онлайн-торгівлі комп'ютерною технікою, який показав значний попит на зручні інструменти для пошуку та порівняння товарів. Було розглянуто найбільш популярні хмарні платформи: Amazon Web Services, Microsoft Azure та Google Cloud Platform. Визначено їх переваги, недоліки та особливості, це дозволить обрати оптимальне рішення для реалізації проєкту. Крім того, проведено формулювання вимог до системи, визначено основні функції каталогу та окреслено задачі, необхідні для розробки.

Каталог комп’ютерної техніки є актуальним рішенням, оскільки сучасні платформи не завжди задовольняють потреби користувачів, зокрема через незручності пошуку та обмеження в порівнянні товарів. Розроблюваний каталог допоможе спростити процес вибору техніки, забезпечуючи персоналізовані рекомендації, можливість збереження фільтрів та отримання актуальної інформації про товари. Це сприятиме економії часу користувачів та підвищенню їх задоволеності.

У цьому розділі також були поставлені основні задачі для подальшої реалізації проєкту.

2 ПРОЕКТУВАННЯ ІНТЕРНЕТ-МАГАЗИНУ КОМП'ЮТЕРНОЇ ТЕХНІКИ НА ОСНОВІ ХМАРНИХ ТЕХНОЛОГІЙ

2.1 Обґрунтування вибору функціональних і нефункціональних вимог до інтернет-магазину комп'ютерної техніки

Вибір функціональних і нефункціональних вимог є важливим етапом розробки інтернет-магазину комп'ютерної техніки. Функціональні вимоги визначають основні завдання, які система має виконувати для забезпечення зручності користувачів.

Однією з ключових функцій є каталог товарів, який дозволяє переглядати, шукати та фільтрувати комп'ютерну техніку за характеристиками, такими як ціна, бренд, технічні параметри або інші показники. Це забезпечує користувачам швидкий доступ до потрібної інформації.

Інша важлива функція – збереження персональних списків обраних товарів. Користувачі зможуть створювати списки бажаних позицій, редагувати їх та легко повертатися до збережених товарів у зручний для них час. Крім цього, система має підтримувати можливість порівняння товарів, що є корисним для ухвалення обґрунтованих рішень під час покупки. Такий функціонал допоможе користувачам оцінити технічні характеристики кількох товарів одночасно.

Також система має використовувати алгоритми для формування персоналізованих рекомендацій на основі уподобань та історії дій користувача. Це забезпечить індивідуальний підхід до кожного покупця.

Поряд із функціональними вимогами важливу роль відіграють нефункціональні вимоги, які визначають якість та стабільність роботи системи. Масштабованість є одним із ключових аспектів, оскільки інтернет-магазин має підтримувати зростання кількості користувачів та обсягу даних без зниження

продуктивності. Доступність системи також є критично важливою. Магазин має працювати цілодобово, забезпечуючи безперебійний доступ користувачів з будь-якого регіону.

Безпека даних є обов’язковою вимогою для захисту конфіденційної інформації, зокрема персональних та платіжних даних користувачів. Для цього використовуватимуться сучасні протоколи шифрування, такі як SSL/TLS. Швидкодія також має вирішальне значення. Наприклад, час завантаження сторінок чи обробки пошукових запитів не має перевищувати двох секунд. Це забезпечить комфортний досвід користувачів.

Стабільність роботи системи під час пікових навантажень, наприклад у період розпродажів, є ще одним важливим аспектом. Для цього планується використання хмарних сервісів, які забезпечують автоматичне балансування навантаження, резервне копіювання та швидке відновлення у разі збоїв.

Завдяки цим вимогам можна сформулювати типові сценарії використання. Наприклад, користувач може шукати ноутбуки у визначеному ціновому діапазоні, зберігати шаблон пошуку та отримувати повідомлення про зниження цін на обрані моделі. Інший користувач може створювати списки товарів для порівняння їх характеристик перед покупкою. Система також наддасть персоналізовані рекомендації, адаптовані до вподобань кожного покупця.

Реалізація цих функціональних і нефункціональних вимог дозволить створити зручний, стабільний і безпечний інтернет-магазин, який відповідатиме сучасним очікуванням користувачів.

2.2 Розробка архітектури інтернет-магазину комп’ютерної техніки

Вибір архітектури є ключовим етапом розробки інтернет-магазину комп'ютерної техніки, оскільки від цього залежить ефективність роботи системи, її

масштабованість, гнучкість і легкість у підтримці. Для інтернет-магазину комп'ютерної техніки необхідно враховувати високі вимоги до продуктивності та можливість обробки значної кількості запитів у режимі реального часу.

Одним із можливих підходів є мікросервісна архітектура. Це підхід до створення програмних продуктів, що будується на реалізації незалежних одне від одного модулів (мікросервісів).

Кожен такий модуль невеликий і націлений на вирішення лише одного бізнес-завдання. Мікросервіси тісно взаємодіють одне з одним через спеціалізовані інтерфейси, зберігаючи при цьому незалежність та власну монолітність. Кожен мікросервіс можна оновлювати, змінювати, розгортати та масштабувати без ризиків для цілісності та працездатності системи [11].

Однією з переваг мікросервісної архітектури є гнучкість у виборі технологічного стеку. Кожен компонент в мікросервісній архітектурі працює як невеликий окремий додаток, який можна реалізувати на будь-яких фреймворках, мовах програмування та інструментах. Відтак бізнес може обирати оптимальні технології під будь-яке завдання та уникати зайвих компромісів, спричинених застарілим стеком вже наявного продукту чи несумісністю окремих технологій [11].

Ще однією важливою перевагою є масштабованість. Автономні сервіси можна масштабувати відокремлено від інших компонентів системи, залежно від навантаження та вимог до обчислювальної потужності. Додавати нові сервіси та функції до модульної системи простіше та економічно вигідніше, ніж масштабувати монолітний продукт [11].

Також важливою перевагою є автоматизоване розгортання та придатність до підтримки. Продукт на базі мікросервісів набагато простіше підтримувати та обслуговувати, адже модифікації в одному модулі не зачіпають усієї системи. Кожен мікросервіс може оновлюватися і розгортатися незалежно від інших. Сучасні

інструменти DevOps та кращі практики CI/CD для автоматизованого розгортання дозволяють пристосувати цей процес до мінливих потреб будь-якої команди [11].

Останньою але не менш важливою перевагою є надійність та мінімізація ризиків. Мікросервісна архітектура відрізняється стабільністю та відмовостійкістю, оскільки усі її компоненти є автономними. Якщо в одному з сервісів виникне збій, то він щонайменше не вплине на роботу інших модулів та не спричинить суттєвих проблем для системи в цілому [11].

Водночас, мікросервісна архітектура має і певні недоліки. Один із головних недоліків – це підвищена складність розгортання та адміністрування, адже кожен мікросервіс потребує окремої інфраструктури та налаштування. Налагодження додатку з мікросервісною архітектурою може бути доволі складним та потребує потужних інструментів автоматизації.

Альтернативою є монолітна архітектура – це традиційний підхід до розробки програмного забезпечення, при якому весь додаток розробляється як одна єдина технологічна система. Всі компоненти додатку взаємодіють один з одним і розгортання відбувається на одному сервері або групі серверів.

Однією з визначальних характеристик монолітної архітектури є її тісно зв'язаний дизайн. Усі компоненти, від користувацького інтерфейсу до бізнес-логіки та управління базами даних, взаємопов’язані [12].

Однією з ключових переваг монолітної архітектури є простота розгортання. Завдяки тому, що застосунок являє собою єдиний виконуваний файл або директорію, процес розгортання стає швидким і легким. Це особливо важливо для невеликих команд або проєктів, де немає потреби в складних процедурах інтеграції.

Об’єднана кодова база спрощує процес розробки, сприяючи кращій співпраці між членами команди. Всі компоненти застосунку працюють разом, що полегшує взаємодію між різними частинами коду. Розробники можуть швидше вносити зміни та бачити їх вплив на всю систему [12].

Однак, монолітна архітектура має і свої недоліки. Серйозним обмеженням монолітного підходу є неможливість масштабувати окремі компоненти. Це означає, що у разі зростання навантаження на певну функцію потрібно масштабувати всю систему, що є менш ефективним і витратним у порівнянні з мікросервісною архітектурою.

Збій одного модуля може призвести до повної відмови системи. У монолітній архітектурі всі компоненти тісно взаємопов’язані, тому навіть незначна проблема в одному з них може вплинути на стабільність роботи всього застосунку.

Ще одним викликом є оновлення фреймворків або мов програмування. У монолітних застосунках це може вимагати оновлення всієї системи, що створює додаткові витрати часу й ресурсів. Перехід на нові технології може стати складним і затратним процесом [12].

З огляду на вимоги до розробки, оптимальним варіантом є використання мікросервісної архітектури. Цей підхід дозволить інтегрувати систему з хмарними сервісами для забезпечення автоматичного масштабування окремих компонентів.

Вибір мікросервісної архітектури також обґрунтований вимогами до персоналізованих рекомендацій та інтеграції зі сторонніми API. Окремі сервіси, що відповідають за ці функції, можуть використовувати спеціалізовані технології, наприклад, моделі машинного навчання або сторонні сервіси аналітики, без впливу на інші компоненти системи.

Таким чином, мікросервісний підхід забезпечує високу гнучкість, масштабованість і надійність системи, що є необхідним для успішного функціонування інтернет-магазину комп’ютерної техніки в умовах високого навантаження та змінних вимог користувачів.

Модульний підхід передбачає поділ програмної системи на окремі функціональні блоки (модулі), кожен з яких виконує конкретні завдання. Такий підхід дозволяє забезпечити гнучкість, масштабованість та спрощення управління

системою. У рамках розроблюваного інтернет-магазину комп'ютерної техніки було виділено такі основні модулі:

- модуль головної сторінки,
- модуль рекомендованих товарів,
- модуль каталогу товарів,
- модуль вподобаних товарів,
- модуль корзини.

На рисунку 2.1. подано загальну структурну схему функціональних модулів.

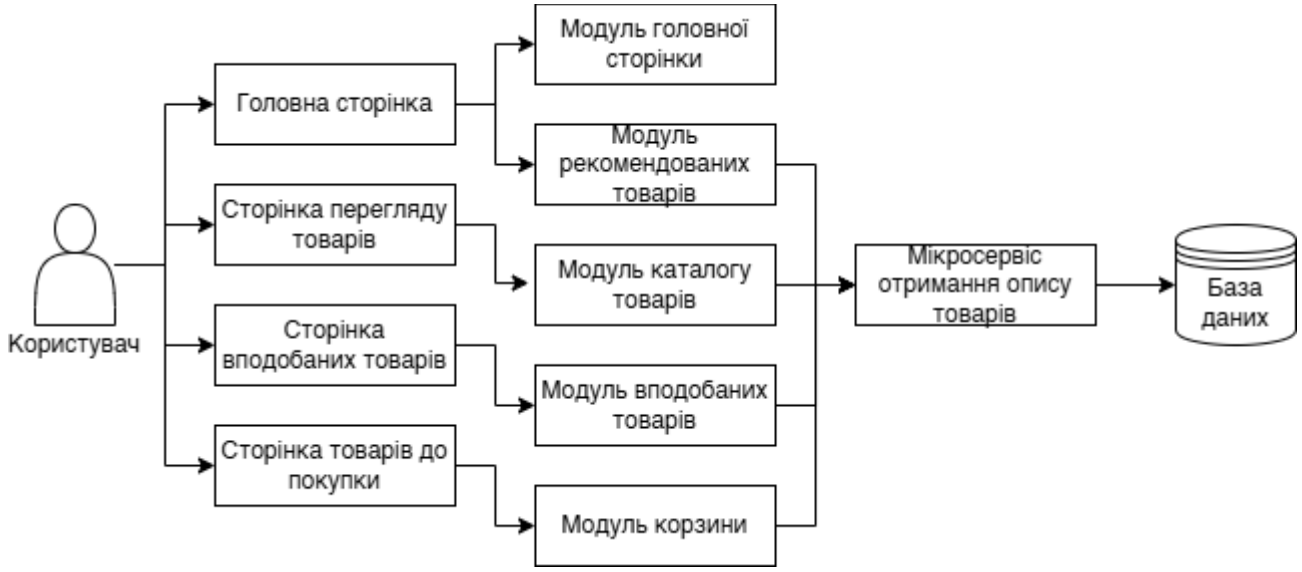

Рисунок 2.1 − Загальна структурна схема функціональних модулів

На рисунку 2.1 зображено основні елементи архітектури WEB-застосунку та їх взаємодію. Взаємодія користувача з системою починається з доступу до різних сторінок, кожна з яких має відповідний функціональний модуль.

Отож, розглянемо кожну сторінку:

- Головна сторінка. Користувач взаємодіє з головною сторінкою, яка містить модуль для відображення рекомендованих товарів. Цей модуль отримує дані з іншого модуля – модуля вподобаних товарів, а потім через мікросервіс, який

здійснює запити до бази даних. Таким чином, користувач бачить персоналізовані рекомендації на основі доступної інформації;

- Сторінка перегляду товарів. Ця сторінка дозволяє переглядати каталог товарів із деталізацією. Відповідний модуль перегляду товарів отримує дані про товари, такі як назва, опис, ціна та характеристики через мікросервіс, що виконує запити до бази даних. Це дозволяє забезпечити актуальність інформації та гнучкість в її оновленні;

- Сторінка вподобаних товарів. Користувач може додавати товари до списку вподобань. Ця функціональність реалізована через модуль вподобаних товарів, який також взаємодіє з мікросервісом. Завдяки цьому користувач має змогу зберігати вибрані товари для подальшого перегляду чи покупки;

- Сторінка товарів до покупки. Для управління кошиком користувача передбачена сторінка товарів до покупки. Модуль цієї сторінки дозволяє переглядати вибрані товари, змінювати їх кількість або видаляти їх із кошика. Як і в інших модулях, усі дані проходять через мікросервіс, що здійснює доступ до бази даних.

Усі модулі системи інтегруються через центральний мікросервіс, який відповідає за отримання, обробку та передачу даних із бази даних. Цей мікросервіс побудований на основі REST API, що забезпечує стандартизовану взаємодію між клієнтською частиною та сервером.

REST API – це архітектура інтерфейсу прикладних програм, яка використовує HTTP-запити для доступу до ресурсів та їхнього використання. Такими HTTP-запитами є GET, PUT, POST і DELETE. Вони дають змогу, відповідно, читати, змінювати, створювати й видаляти ресурси [13].

REST API дозволяє обмінюватися даними у форматі JSON, що робить його простим у використанні та легко інтегрованим у клієнтські додатки.

Мікросервіс виконує кілька ключових функцій:

- обробка запитів клієнтів: отримання запитів від різних модулів (наприклад, для перегляду товарів, додавання до списку бажань чи управління кошиком);

- доступ до бази даних: виконання операцій читання та запису, включаючи отримання інформації про товари, оновлення стану замовлень та управління даними користувачів;

- форматування даних: перетворення даних із бази даних у формат, придатний для відображення у клієнтському інтерфейсі.

База даних виконує центральну роль у функціонуванні системи, забезпечуючи зберігання та управління всією необхідною інформацією. Вона є надійним сховищем, яке підтримує цілісність і доступність даних для всіх компонентів системи. Основною задачею бази даних є зберігання детальної інформації про товари.

2.3 Розробка UML-діаграми класів

Розробка UML-діаграми класів є ключовим етапом проєктування програмного забезпечення, оскільки вона надає візуальне представлення структури системи та взаємодії її основних компонентів. У рамках проєкту UML-діаграма класів відображає об'єкти, їх атрибути, методи та зв'язки між ними, що дозволяє чітко зрозуміти архітектуру системи.

Для реалізації серверної частини проєкту використовуються класи, побудовані на основі об'єктно-орієнтованого підходу, який є основним принципом розробки програм на мові Python. Основними принципами, що використовуються при проєктуванні, є інкапсуляція, поліморфізм, наслідування та абстракція. Ці принципи забезпечують модульність, гнучкість і можливість повторного використання коду.

Отже, розглянемо класи, які використовуються в розроблюваному мікросервісі API. У його структурі передбачені такі класи:

Клас AppController відповідає за ініціалізацію та запуск додатка. Він налаштовує CORS, реєструє схеми та забезпечує метод для запуску застосунку.

Методи:

- __init__(): ініціалізує додаток і налаштовує його;
- _configure_cors(): встановлює конфігурацію CORS для API, дозволяючи зовнішнім клієнтам доступ до сервера;
- _register_blueprints(): реєструє blueprints, які містять маршрути для обробки запитів;
- run(host, port): запускає додаток на вказаному хості та порту.

Клас ProductAPI відповідає за обробку API-запитів, пов’язаних із продуктами. Він ініціалізується із схемами, визначає маршрути та реалізує методи для отримання списку продуктів, окремого продукту та перевірки стану API.

Методи:

- __init__(self, blueprint): ініціалізує API з переданим схемами;
- api_status(): обробляє API-запит для перевірки стану сервера;
- get_all_products(): отримує всі продукти з бази даних;
- get_product(product_id): отримує один продукт за його унікальним ідентифікатором.

Клас Product визначає структуру об’єкта продукту.

Атрибути:

- id: унікальний ідентифікатор продукту (відповідає _id у MongoDB);
- name: назва продукту;
- description: опис продукту;
- price: ціна продукту;
- imageUrl: посилання на зображення продукту;

- category: категорія товару;
- characteristics: додаткові характеристики продукту (наприклад, тип пам’яті, потужність тощо).

Клас Config містить конфігураційні параметри для додатка, такі як URI MongoDB, назву бази даних і колекції. Він отримує ці параметри із змінних середовища.

Атрибути:

- MONGO_URI: посилання для підключення до MongoDB;
- DB_NAME: назва бази даних;
- COLLECTION_NAME: назва основної колекції.

Клас Database реалізує патерн. Одинак і керує підключенням до MongoDB. Він гарантує, що в додатку існує лише один екземпляр підключення до бази даних.

Методи:

- __init__(): ініціалізує підключення до бази даних MongoDB, використовуючи параметри конфігурації;
- get_collection(collection_name): отримує колекцію бази даних за її назвою;

На рисунку 2.2 зображено UML-діаграму класів мікросервісу API. Розглянемо класи, які використовуються у фронтенд-частині застосунку.

Клас App є кореневим компонентом, який обгортає всю програму. Він включає глобальні компоненти, такі як Header і Footer.

Атрибути:

- layout: ReactNode визначає основний макет додатка.

Клас CartProvider забезпечує глобальний контекст для управління кошиком покупок, надаючи методи для додавання, видалення товарів і розрахунку загальної вартості.

Атрибути:

- cart: CartItem[] – масив товарів у кошику.

Методи:

- addToCart(product: Product) – додає продукт у кошик;
- removeFromCart(productId: string) – видаляє продукт із кошика;
- clearCart() – очищує кошик;
- getCartTotal(): number – повертає загальну вартість кошика.

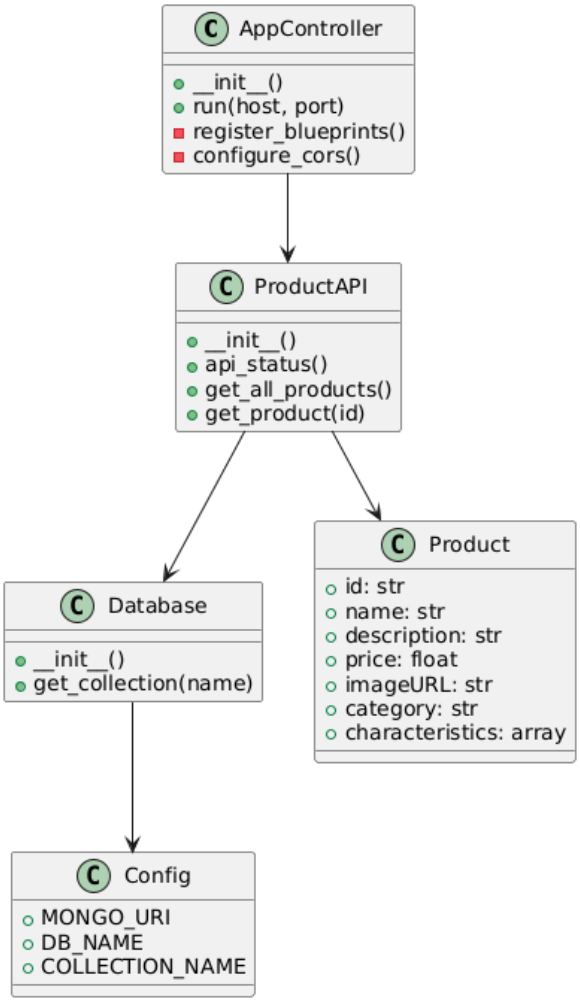

Рисунок 2.2 – UML-діаграма класів мікросервісу API

Клас WishlistProvider дозволяє керувати списком бажаного на рівні додатка, використовуючи глобальний контекст.

Атрибути:

- wishlist: Product[] – масив продуктів у списку бажаного.

Методи:

- addToWishlist(product: Product) – додає продукт до списку бажаного;
- removeFromWishlist(productId: string) – видаляє продукт зі списку бажаного;
- isInWishlist(productId: string): boolean – перевіряє, чи є продукт у списку бажаного.

Клас ProductsPage відповідає за сторінку товарів, керує пошуком, фільтрацією та сортуванням.

Атрибути:

- products: Product[] – список продуктів;
- selectedCategory: ProductCategory | null – обрана категорія;
- searchTerm: string – пошуковий запит;
- minPrice: string – мінімальна ціна;
- maxPrice: string – максимальна ціна;
- sortBy: string – критерій сортування;
- viewMode: "grid" | "list" – режим перегляду.

Методи:

- fetchProducts() – отримує список продуктів;
- filterProducts(): Product[] – фільтрує продукти;
- sortProducts(products: Product[]): Product[] – сортує продукти;
- handleAddToCart(product: Product) – додає продукт у кошик;
- handleWishlist(product: Product) – додає продукт у список бажаного;
- renderProductCard(product: Product) – відображає картку продукту;
- renderProductListItem(product: Product) – відображає продукт у списковому вигляді.

Клас ProductDetailPage відповідає за сторінку продукту.

Атрибути:

- product: Product | null – обраний продукт.

Методи:

- fetchProduct() – отримує дані про конкретний продукт;
- handleAddToCart() – додає продукт у кошик;
- handleWishlist() – додає продукт у список бажаного;
- renderProductCharacteristics() – відображає характеристики товару.

Клас CartPage відповідає за сторінку кошика, дозволяючи переглядати, редагувати та видаляти товари.

Атрибути:

- cart: CartItem[] – список товарів у кошику.

Методи:

- removeFromCart(productId: string) – видаляє продукт із кошика;
- clearCart() – очищує кошик;
- getCartTotal(): number – повертає загальну вартість кошика.

Клас WishlistPage відповідає за управління списком бажаного.

Атрибути:

- wishlist: Product[] – список бажаного.

Методи:

- removeFromWishlist(productId: string) – видаляє продукт зі списку бажаного;
- addToCart(product: Product) – додає продукт у кошик.

Інтерфейс Product визначає структуру об'єкта товару.

Атрибути:

- id: string – унікальний ідентифікатор;
- name: string – назва;
- category: ProductCategory – категорія;
- price: number – ціна;
- description: string – опис;

- imageUrl: string – зображення;
- slug: string – унікальний ідентифікатор URL;
- characteristics: Record<string, string> – характеристики.

Інтерфейс CartItem визначає структуру товару в кошику.

Атрибути:

- product: Product – товар;
- quantity: number – кількість.

На рисунку 2.3 зорбражено UML-діаграму класів фронтенд-частини.

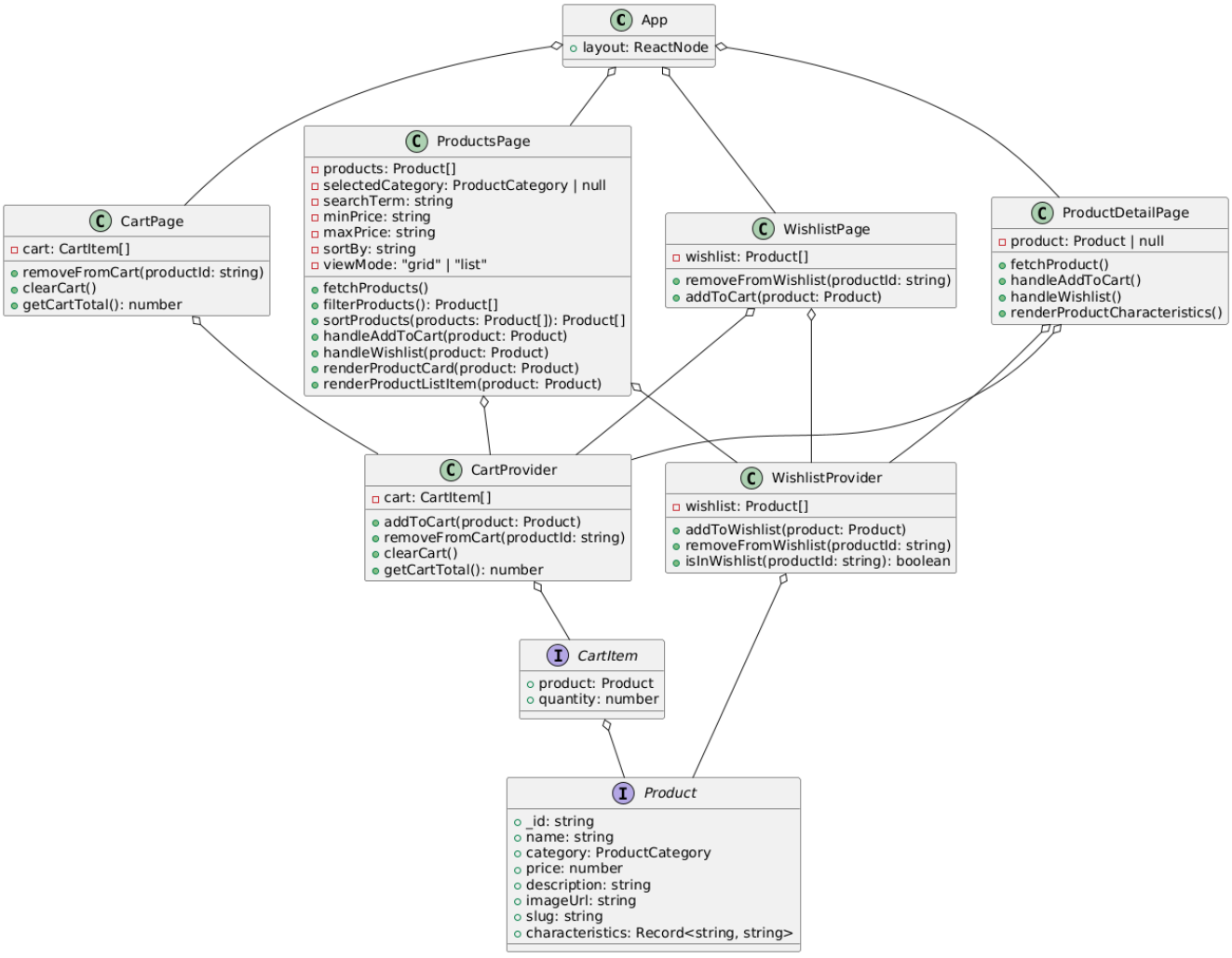

Рисунок 2.3 – UML-діаграма класів фронтенд-частини

2.4 Розробка ER-моделі бази даних

ER-модель є концептуальною основою для створення структури бази даних. Вона відображає об'єкти, їхні властивості та взаємозв’язки у вигляді графічної діаграми. Цей підхід дозволяє глибше осмислити предметну область ще до реалізації технічної частини, що значно спрощує перехід до фізичного проектування бази даних [14].

В універсальне відношення потрібно включити атрибути, що описують такі сутності: продукт, категорія, бренд, характеристики процесора, характеристики відеокарти, характеристики оперативної пам’яті, характеристики накопичувача, характеристики материнської плати, характеристики блоку живлення, характеристики кейсу.

Універсальне відношення для реалізації бази даних буде мати такий вигляд: R (назва продукту, опис, ціна, зображення, категорія, назва категорії, опис, назва бренду, опис, бренд, ядра, потоки, базова частота, турбочастота, тип роз’єму, тепло провідність, інтегрована графіка, модель, бренд, пам’ять, частота ядра, турбочастота ядра, споживана потужність, інтерфейс, тип пам’яті, бренд, пам’ять, швидкість, затримка, напруга, тип накопичувача, пам’ять, швидкість читання, швидкість запису, інтерфейс, модель, бренд, типу сокету, форм-фактор, чіпсет, кількість слотів пам’яті, максимум оперативної пам’яті, бренд, вольтаж, сертифікат ефективності, модульність, бренд, форм-фактор, матеріал, підтримка вентиляторів, підтримка радіаторів).

В представленій базі даних наявний один типу зв’язку – ОДИН-ДО-БАГАТЬОХ (1:Б). Цей тип зв’язку наявний між сутностями: Категорія-Продукт, Характеристики Процесора-Продукт, Характеристики Відеокарти-Продукт, Характеристики Оперативної Пам’яті-Продукт, Характеристики Накопичувача-Продукт, Характеристики Материнської Плати-Продукт, Характеристики Блоку

Живлення-Продукт, Характеристики Кейсу-Продукт, Бренд-Характеристики Процесора, Бренд-Характеристики Відеокарти, Бренд-Характеристики Оперативної Пам’яті, Бренд-Характеристики Накопичувача, Бренд-Характеристики Материнської Плати, Бренд-Характеристики Блоку Живлення, Бренд-Характеристики Кейсу. Характеристику зв’язків подано в таблиці 2.1.

Таблиця 2.1 – Характеристики відношень між сутностями

Сутність 1	Сутність 2	Тип зв’язку	Ім’я зв’язку	Клас належності
Продукт	Категорія	Б:1	Відноситься	Обов’язковий
Характеристики Процесора	Продукт	1:Б	Має	Не обов’язковий
Характеристики Відеокарти	Продукт	1:Б	Має	Не обов’язковий
Характеристики Оперативної Пам’яті	Продукт	1:Б	Має	Не обов’язковий
Характеристики Накопичувача	Продукт	1:Б	Має	Не обов’язковий
Характеристики Материнської Плати	Продукт	1:Б	Має	Не обов’язковий
Характеристики Блоку Живлення	Продукт	1:Б	Має	Не обов’язковий
Характеристики Кейсу	Продукт	1:Б	Має	Не обов’язковий
Характеристики Процесора	Бренд	Б:1	Належить	Обов’язковий
Характеристики Відеокарти	Бренд	Б:1	Належить	Обов’язковий
Характеристики Оперативної Пам’яті	Бренд	Б:1	Належить	Обов’язковий
Характеристики Накопичувача	Бренд	Б:1	Належить	Обов’язковий
Характеристики Материнської Плати	Бренд	Б:1	Належить	Обов’язковий
Характеристики Блоку Живлення	Бренд	Б:1	Належить	Обов’язковий
Характеристики Кейсу	Бренд	Б:1	Належить	Обов’язковий

На рисунку 2.4 зображено ER-модель предметної області «Комп’ютерна техніка».

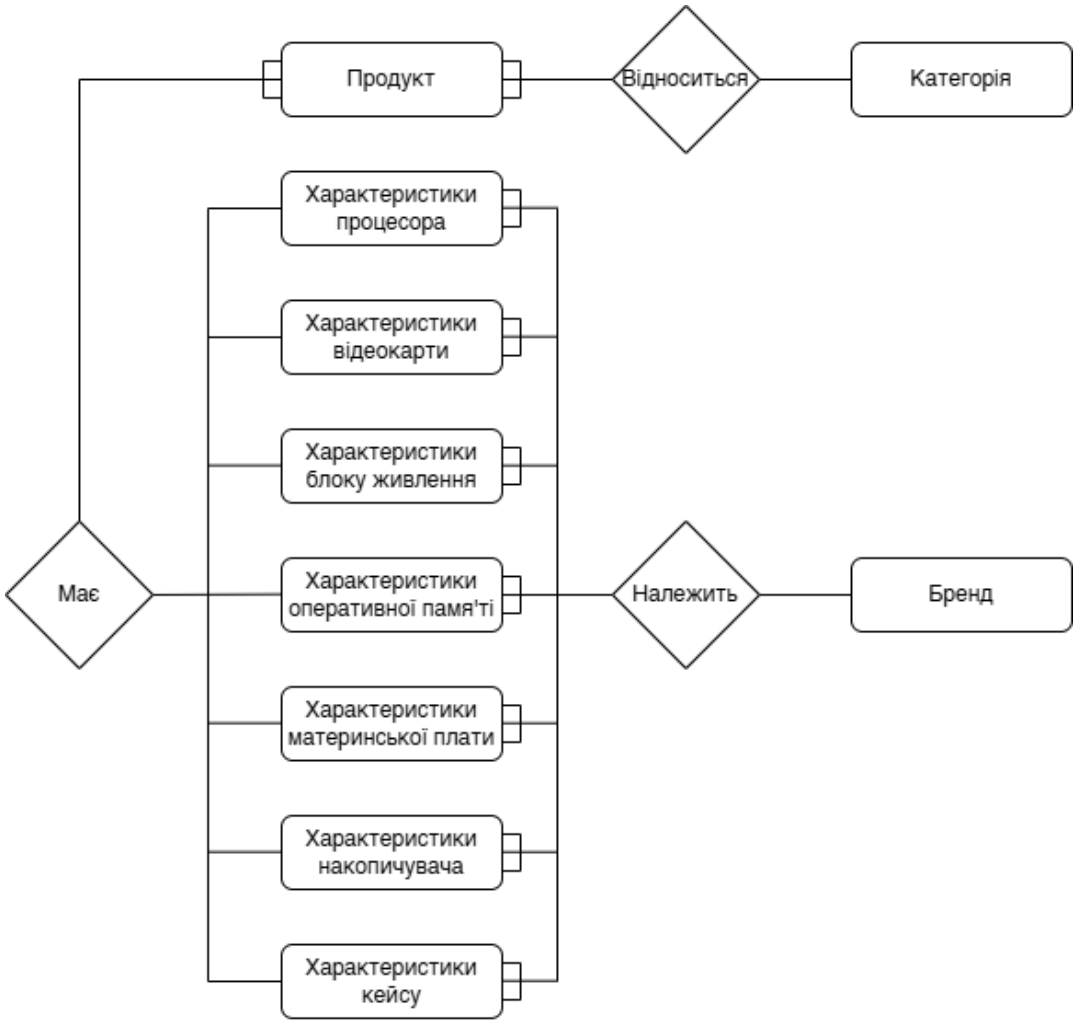

Рисунок 2.4 – ER-модель предметної області «Комп’ютерна техніка»

У результаті було розроблено ER-модель бази даних для інтернет-магазину комп’ютерної техніки, яка включає сутності: товар, категорія, бренд, а також характеристики процесора, відеокарти, оперативної пам’яті, накопичувача, материнської плати, блоку живлення та корпусу. Усі зв’язки між сутностями реалізовано у форматі «один-до-багатьох», що дозволяє гнучко описати відношення між товарами та їх складовими параметрами. Розроблена модель забезпечує структуроване зберігання інформації, полегшує подальшу реалізацію бази даних.

2.5 Розробка схеми алгоритму роботи інтернет-магазину комп’ютерної техніки

Одним із ключових етапів проєктування інтернет-магазину є розробка алгоритмів його роботи. Алгоритми визначають послідовність дій, які виконує система для обробки запитів користувача, забезпечення доступу до товарів, оформлення замовлень та інших функцій.

Ефективна організація алгоритмів дозволяє спростити процес взаємодії користувача з магазином, зробити його більш інтуїтивно зрозумілим і зручним. Крім того, чітка схема алгоритмів спрощує розробку та подальшу підтримку програмного забезпечення.

У цьому підрозділі розглянемо основні випадки використання системи, які охоплюють ключові сценарії роботи інтернет-магазину комп’ютерної техніки.

Зокрема, будуть описані такі процеси:

- Пошук товарів у каталозі;
- Оформлення замовлення;
- Керування списком бажаного.

Кожен із цих випадків буде представлено у вигляді покрокового алгоритму, що дозволяє зрозуміти логіку роботи системи на кожному етапі.

Для опису алгоритму роботи інтернет-магазину комп’ютерної техніки було обрано подання алгоритму у вигляді блок-схеми. Схему алгоритму розробки представлено на рисунку 2.5.

І випадок

Алгоритм пошук товарів у каталозі (рис. 2.5) складається з таких кроків:

Крок 1. Користувач відкриває головну сторінку інтернет-магазину.

Крок 2. Користувач переходить на сторінку каталогу товарів.

Крок 3. Користувач вводить ключове слово у полі пошуку.

Рисунок 2.5 – Схема алгоритму роботи інтернет-магазину комп’ютерної техніки

Крок 4. Користувач обирає додаткові фільтри для звуження результатів.

Крок 5. Користувач натискає кнопку «Пошук» або спрацьовує автоматичне оновлення результатів.

Крок 6. Запит надсилається на сервер.

Крок 7. Перевірка наявності товарів.

Крок 8. Якщо товар не знайдено, система виводить повідомлення про відсутність товару.

Крок 9. Якщо товар знайдено, користувач отримує пошукові результати.

II випадок

Алгоритм керування списком бажаного (рис. 2.5, аркуш 2):

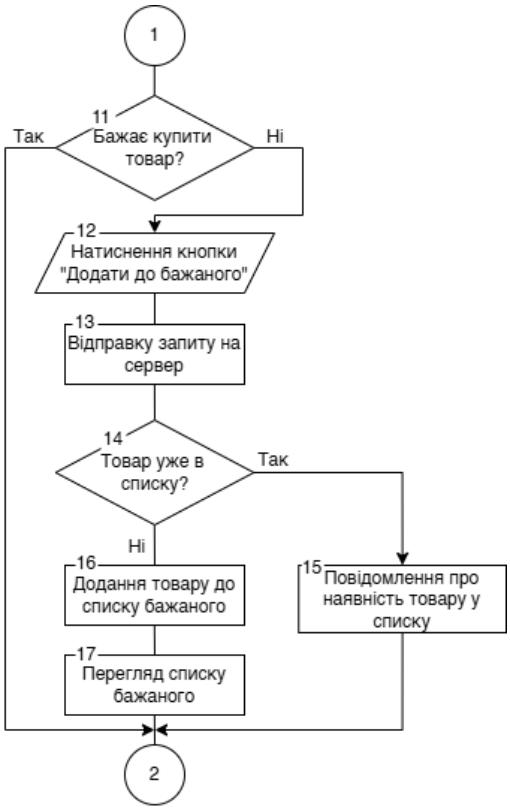

Рисунок 2.5, аркуш 2

Крок 1. Користувач відкриває головну сторінку інтернет-магазину.

Крок 2. Користувач переходить на сторінку каталогу товарів.

Крок 10. Користувач вибирає товар.

Крок 11. Користувач обирає чи бажає купити товар зараз чи добавити в список бажаного.

Крок 12. Якщо користувач бажає добавити товар у список бажаного, натискає кнопку «Додати до списку бажаного».

Крок 13. Відправка запиту на сервер.

Крок 14. Сервер перевіряє, чи товар уже є у списку бажаного.

Крок 15. Якщо так, видається повідомлення, що товар уже доданий.

Крок 16. Якщо ні, товар додається у список бажаного.

Крок 17. Користувач може переглянути список бажаного у відповідному розділі.

III випадок

Алгоритм додавання товарів у кошик (рис. 2.5, аркуш 3):

Рисунок 2.5, аркуш 3

Крок 1. Користувач відкриває головну сторінку інтернет-магазину.

Крок 2. Користувач переходить на сторінку каталогу товарів.

Крок 10. Користувач вибирає товар.

Крок 11. Користувач обирає чи бажає купити товар зараз чи добавити в список бажаного.

Крок 18. Якщо користувач бажає купити товар, користувач додає товар до кошика.

Крок 19. Відправка запиту на сервер.

Крок 20. Користувач переходить до кошика для перегляду вибраних товарів.

Крок 21. Користувач перевіряє вміст кошика.

Крок 22. Користувач натискає кнопку замовлення товару.

2.6 Обґрунтування вибору хмарної платформи для реалізації інтернет-магазину комп'ютерної техніки

Вибір хмарної платформи є важливим рішенням у процесі розробки інтернет-магазину комп'ютерної техніки, оскільки від цього залежить продуктивність, масштабованість та стабільність системи. Основними конкурентами на ринку хмарних обчислень є Amazon Web Services (AWS), Microsoft Azure та Google Cloud Platform (GCP). Кожна з цих платформ має свої переваги та недоліки.

Основним критерієм вибору хмарної платформи була її здатність забезпечити стабільну роботу інтернет-магазину в умовах високого навантаження, особливо під час пікових періодів, таких як акції чи розпродажі. Також важливу роль відігравали можливості платформи для реалізації мікросервісної архітектури, підтримка контейнерних технологій, висока відмовостійкість, а також економічна ефективність використання ресурсів.

Amazon Web Services пропонує широкий спектр сервісів для хмарних обчислень, які дозволяють будувати масштабовані та безпечні WEB-застосунки. У

випадку розроблюваного інтернет-магазину AWS забезпечує ефективне керування мікросервісною архітектурою.

Також, AWS надає технології для запуску коду, управління даними та інтеграції застосунків без необхідності керування серверами. Безсерверні технології забезпечують автоматичне масштабування, вбудовану високу доступність та модель оплати за фактичне використання, що сприяє підвищенню гнучкості та оптимізації витрат. Вони також усувають завдання з адміністрування інфраструктури, такі як резервування потужностей та встановлення оновлень, що дозволяє розробникам зосередитися на написанні коду та створенні цінності для користувачів [15].

Ще одним важливим аспектом є обробка та зберігання даних. AWS надає гнучкі рішення для роботи з базами даних, що ідеально прийнятні для реалізації NoSQL-архітектури, яка використовується в розробці.

Альтернативні хмарні платформи, такі як Microsoft Azure і Google Cloud Platform, також були розглянуті, але вони мають певні обмеження, які зробили їх менш привабливими для реалізації цього проєкту. Microsoft Azure добре інтегрується з корпоративними рішеннями Microsoft, такими як Office 365 та Active Directory, але його можливості масштабування поступаються AWS [9].

Google Cloud Platform є гарним вибором для проєктів, що потребують потужних інструментів для аналізу даних, машинного навчання та штучного інтелекту [10]. Проте для інтернет-магазину ця платформа менш універсальна, оскільки має меншу кількість дата-центрів та менший вибір хмарних сервісів у порівнянні з AWS.

Отже, AWS обрано як хмарну платформу для інтернет-магазину через її надійність, масштабованість і підтримку мікросервісної архітектури. Вона забезпечує безсерверні технології, ефективне зберігання даних, автоматичне масштабування. Це дозволяє створити стабільну, продуктивну та економічно вигідну систему, здатну витримувати високі навантаження.

2.7 Розробка архітектурної діаграми хмарного середовища

Розробку архітектурної діаграми, ще називають архітектурним моделюванням.

Архітектурне моделювання – це процес створення візуальних представлень компонентів програмних систем. У контексті програмного забезпечення термін «архітектура» означає різні функції, їх реалізацію та взаємодію між ними. Оскільки програмне забезпечення є абстрактним за своєю природою, архітектурні діаграми дозволяють візуалізувати рух даних у системі. Вони також підкреслюють, як програмне забезпечення взаємодіє із зовнішнім середовищем, полегшуючи розуміння складних зв’язків між компонентами та інтеграцію з іншими системами [16].

Створення візуального представлення середовища, має кілька переваг [16]:

- допомагають виявляти потенційні ризики у процесі розробки системи, такі як помилки у логіці або недостатнє тестування. Завдяки ранньому виявленню та усуненню ризиків на початкових етапах життєвого циклу розробки програмного забезпечення команди можуть вносити необхідні зміни завчасно, що знижує ймовірність виникнення серйозних проблем на пізніх стадіях;
- значно підвищують співпрацю між розробниками та дизайнерами, створюючи єдине бачення функціональності системи та потенційних проблем. Спільне розуміння системи, додатку чи WEB-ресурсу має низку переваг. Воно сприяє ефективній комунікації під час процесу проєктування, допомагає командам розробляти ефективні програмні компоненти системи та забезпечує досягнення цілей проєкту;
- архітектурні діаграми надають чітке уявлення про компоненти та структуру системи. Це дозволяє зацікавленим сторонам точно ідентифікувати проблеми та швидко їх вирішувати. Крім того, такі діаграми полегшують підтримку та

масштабування систем, що робить подальші зміни більш ефективними та менш трудомісткими.

Таким чином, архітектурна діаграма хмарного середовища є ключовим інструментом для візуалізації структури системи, взаємодії її компонентів та їхнього розподілу в хмарному середовищі. Для інтернет-магазину комп'ютерної техніки ця діаграма базується на використанні сервісів Amazon Web Services (AWS), які забезпечують ефективність, масштабованість та гнучкість системи.

На рисунку 2.6 зображено розроблену діаграму хмарного середовища інтернет-магазину комп'ютерної техніки.

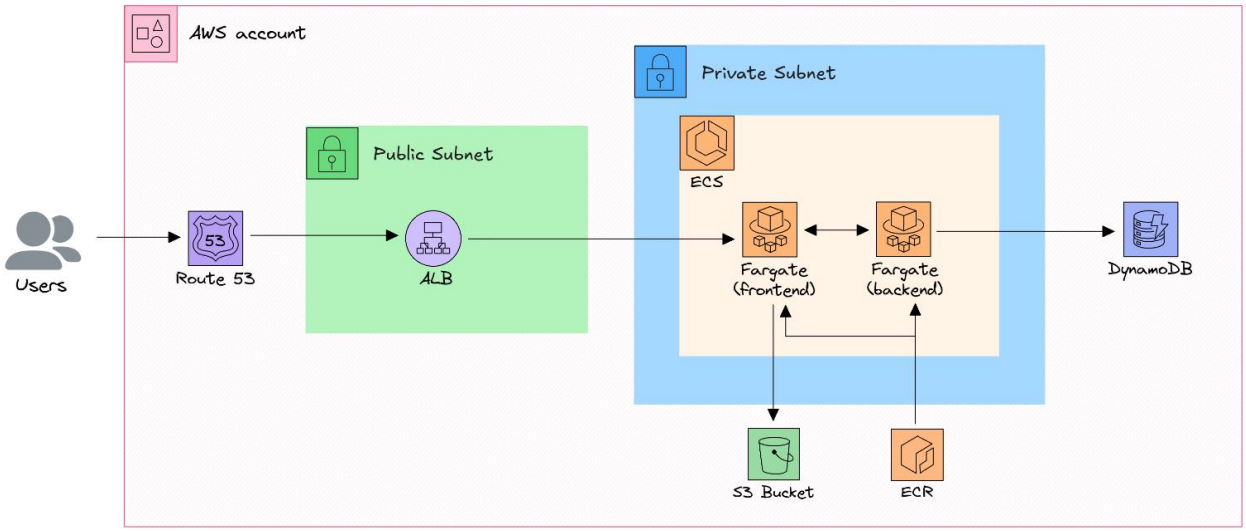

Рисунок 2.6 – Архітектурна діаграма хмарного середовища інтернет-магазину комп'ютерної техніки

Першим важливим компонентом архітектури є система управління доменними іменами. Amazon Route 53 – це високо доступний та масштабований сервіс для роботи з системою доменних імен (DNS), реєстрації доменів та перевірки стану хмарних сервісів. Він надає розробникам і бізнесу надійний та економічно ефективний спосіб маршрутизації кінцевих користувачів до інтернет-додатків,

перетворюючи доменні імена, наприклад, example.com, у числові IP-адреси, такі як 192.0.2.1, які комп'ютери використовують для взаємодії [17].

AWS Route 53 забезпечує перенаправлення запитів користувачів до системи, працюючи як точка входу. Потім запити потрапляють до Application Load Balancer (ALB), розташованого в публічній підмережі.

Application Load Balancer працює на рівні застосунків, що відповідає сьомому рівню моделі OSI. Після отримання запиту балансувальник навантаження аналізує правила обробки запитів (listener rules) у порядку їх пріоритетності, щоб визначити, яке правило застосувати. Потім він обирає цільовий ресурс із відповідної групи цільових ресурсів (target group) згідно з вказаними налаштуваннями [18].

На діаграмі (рис. 2.6) ALB відповідає за розподіл навантаження між мікросервісами фронтенду, що забезпечує рівномірний розподіл трафіку і стабільну роботу системи навіть при високих навантаженнях.

Фронтенд і бекенд-застосунки розгорнуті як окремі контейнери у сервісі AWS Fargate, що працює в рамках Elastic Container Service (ECS).

Elastic Container Service (ECS) – це повністю керований сервіс оркестрації контейнерів, який дозволяє ефективно розгортати, управляти та масштабувати контейнеризовані застосунки. Завдяки глибокій інтеграції з екосистемою AWS, ECS надає зручне рішення для запуску контейнеризованих додатків [19].

Контейнер фронтенд відповідає за взаємодію з користувачами через WEB-інтерфейс, тоді як бекенд контейнер обробляє запити, виконує бізнес-логіку та взаємодіє з базою даних. Розміщення цих компонентів у приватній підмережі забезпечує додатковий рівень захисту, оскільки вони не мають прямого доступу з інтернету.

Amazon DynamoDB – це серверлес NoSQL база даних, яка дозволяє розробляти сучасні програми будь-якого масштабу. Вона оптимізована для роботи з документними та ключ-значенними структурами даних, що робить її ідеальним

рішенням для обробки великих обсягів інформації з високою доступністю. DynamoDB автоматично масштабується для обробки змін у трафіку, забезпечуючи стабільну продуктивність без необхідності ручного управління інфраструктурою. Вона підтримує як документи у форматі JSON, так і прості пари ключ-значення, що дозволяє гнучко зберігати дані, такі як інформація про товари, замовлення та користувачів [20].

Безпека DynamoDB забезпечується за рахунок інтеграції з AWS Identity and Access Management, що дозволяє точно контролювати доступ до даних. База даних може бути розгорнута в приватній підмережі, обмежуючи зовнішній доступ та дозволяючи з'єднання лише через бекенд-сервіси. Додаткові функції безпеки, такі як шифрування даних при зберіганні та передачі, автоматичне резервне копіювання та точка відновлення, забезпечують надійний захист конфіденційної інформації.

Статичні ресурси, такі як зображення товарів, зберігаються у сховищі Amazon S3. Amazon Simple Storage Service (Amazon S3) – це сервіс об'єктного зберігання, який забезпечує високу масштабованість, доступність даних, безпеку та продуктивність. Його використовують мільйони клієнтів різного масштабу та галузей для зберігання, управління, аналізу та захисту будь-якої кількості даних для різних сценаріїв використання [21]. Це рішення забезпечує швидкий доступ до файлів і їх надійність завдяки вбудованим механізмам резервного копіювання.

Для централізованого управління контейнерними образами використовується Elastic Container Registry (ECR). Приватний реєстр Elastic Container Registry зберігає образи контейнерів у високо доступній та масштабованій інфраструктурі. Використовуючи приватний реєстр, можна керувати власними сховищами образів Docker та Open Container Initiative (OCI), забезпечуючи безпечне розгортання контейнеризованих застосунків [22]. Для розроблюваного середовища це дозволяє зберігати, оновлювати та масштабувати образи для розгортання в середовищі Fargate.

Запропонована архітектура відповідає сучасним вимогам до безпеки, продуктивності та гнучкості. Її структура легко адаптується до майбутніх змін у функціональності чи масштабах розробки. Завдяки використанню хмарних сервісів AWS забезпечується стабільна, швидка та безпечна робота інтернет-магазину, що робить її оптимальним вибором для реалізації розробки.

2.8 Висновок

У другому розділі було проведено всебічне проектування інтернет-магазину комп’ютерної техніки на основі хмарних технологій. Розділ містить детальний аналіз ключових аспектів, необхідних для успішної реалізації проєкту та охоплює всі етапи від вимог до побудови архітектурної моделі.

На етапі аналізу функціональних і нефункціональних вимог були сформульовані основні завдання системи, включаючи обробку запитів користувачів, управління замовленнями та забезпечення безпеки. Нефункціональні вимоги, такі як масштабованість, швидкодія та стабільність, визначили ключові критерії вибору архітектурних рішень.

Вибір архітектурного підходу зупинився на мікросервісній архітектурі як найбільш ефективній для цього проєкту. Такий підхід забезпечує гнучкість у розробці та масштабуванні, дозволяє незалежно розгортати й оновлювати окремі компоненти системи. Це важливо для сучасного електронного магазину, що працює в умовах постійно змінюваного середовища.

У проектуванні логічної структури системи було визначено основні компоненти, їх функції та взаємодію. Архітектура включає модулі: головної сторінки, рекомендованих товарів, каталогу товарів, вподобаних товарів та корзини. Кожен модуль виконує чітко визначену роль і забезпечує ізольовану функціональність.

Розроблено детальну ER-модель бази даних, яка охоплює всі сутності предметної області – продукти, характеристики, бренди та категорії. Це дозволило структурувати дані відповідно до реальних потреб інтернет-торгівлі технікою.

Також, було розроблено алгоритм роботи інтернет-магазину комп'ютерної техніки, який охоплює основні сценарії взаємодії користувача із системою: пошук товарів у каталозі, додавання товарів до кошика та керування списком бажаного. Запропонований алгоритм враховує всі ключові етапи роботи користувача з магазином – від пошуку продукції до її замовлення та управління вибраними товарами.

Обґрунтування вибору хмарної платформи показало, що Amazon Web Services (AWS) є оптимальним рішенням для проєкту. Завдяки широкому набору інструментів для оркестрації контейнерів, управління даними та забезпечення безпеки, AWS повністю відповідає вимогам масштабованого та стабільного інтернет-магазину.

Розробка архітектурної діаграми хмарного середовища дозволила наочно представити взаємодію компонентів системи, їх розташування в хмарному середовищі та принципи безпеки. Було продемонстровано, як сервіси Amazon Route 53, ALB, Amazon ECS, S3, DocumentDB та ECR інтегруються для забезпечення стабільності та продуктивності системи.

Таким чином, розділ заклав основи для технічної реалізації інтернет-магазину, детально обґрунтувавши вибір технологій, архітектурних рішень та хмарної інфраструктури. Було враховано всі ключові аспекти, починаючи від функціональних і нефункціональних вимог до системи, до вибору оптимальної архітектури та технологічної платформи, що найкраще відповідає потребам проєкту.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ІНТЕРНЕТ-МАГАЗИНУ КОМП'ЮТЕРНОЇ ТЕХНІКИ НА ОСНОВІ ХМАРНИХ ТЕХНОЛОГІЙ

3.1 Обґрунтування вибору мови програмування для управління базою даних

При розробці інтернет-магазину комп’ютерної техніки важливим етапом є вибір типу бази даних для зберігання даних. У сучасній розробці використовуються два основних підходи: реляційні бази даних (SQL) та нереляційні бази даних (NoSQL).

Реляційна база даних складається з таблиць, які представляють реальні об'єкти або поняття, які часто називають сутностями. Кожен стовпець таблиці містить певний тип даних, відомий як атрибут, а поле зберігає фактичне значення атрибута. Рядки і стовпці таблиці представляють набір пов'язаних значень одного об'єкта або сутності. Така таблична структура бази даних дозволяє легко встановлювати взаємозв'язки, так що доступ до інформації можна отримати різними способами без реорганізації самих даних [23].

Головна перевага реляційної моделі даних полягає в тому, що вона дозволяє отримувати корисну інформацію з даних. Наприклад, можна підраховувати, сортувати і групувати дані в таблицях, виконувати різні обчислення і створювати детальні звіти. Завдяки цьому реляційні бази даних залишаються найпоширенішими для зберігання даних сьогодні.

Іншою перевагою реляційних баз даних є те, що вони забезпечують цілісність даних, тобто узгодженість, точність і надійність даних. Реляційні бази даних застосовують правила та обмеження, які запобігають аномаліям даних, таким як дублювання, неузгодженість або пошкодження. Наприклад, можна використовувати первинні й зовнішні ключі, унікальні обмеження та правила посилань, щоб

переконатися, що записи мають унікальні ідентифікатори, правильно пов'язані між собою, і що некоректні чи порожні дані не будуть додані або видалені [24].

Однак реляційні бази даних мають і недоліки. Одним із головних недоліків реляційних баз даних є те, що вони не розраховані для роботи з великими обсягами даних або дуже швидкими потоками даних. Вони розраховані на зберігання чітко структурованих даних у фіксованих схемах, тому їх важко змінювати чи адаптувати, коли дані збільшуються або змінюються. Для масштабування такої бази доводиться оновлювати обладнання або розподіляти дані між кількома серверами, що може бути складно, дорого та ризиковано [24].

Ще один недолік реляційних баз даних полягає в їхній складності, що потребує більше зусиль для проєктування, обслуговування та адміністрування порівняно з іншими типами баз даних. Щоб забезпечити правильність і ефективність роботи, необхідно ретельно планувати структуру таблиць, стовпців, ключів і зв'язків. Однак це також ускладнює внесення змін, оскільки будь-яке коригування схеми чи даних може вплинути на функціональність та продуктивність бази. Наприклад, при зміні структури доводиться оновлювати кілька таблиць, індексів або тригерів, що займає багато часу [24].

Нереляційні бази даних розроблені для роботи з різними типами даних і використовують гнучкі схеми, що легко адаптуються під потреби сучасних додатків. Вони забезпечують високу продуктивність і економічність, особливо для великих обсягів даних. Нереляційні бази також відомі своєю простотою у використанні, масштабованістю і високою швидкістю роботи [25].

Головною перевагою NoSQL баз даних є їх здатність працювати з різними форматами великих обсягів неструктурованої інформації. Вони підтримують кілька типів даних, кожен з яких прийнятний для різних завдань [26]:

- Документні бази даних: зберігають дані у вигляді документів, наприклад, JSON або XML. Це дозволяє зберігати складні об’єкти з різними полями, що зручно для роботи з гнучкими структурами даних.

- Бази даних ключ-значення: дані організовані у пари «ключ-значення». Цей тип доречний для швидкого доступу до простих даних, наприклад, у кешуванні.

- Колонкові бази даних: зберігають дані у стовпцях замість рядків, що доречно для аналітики та обробки великих обсягів табличних даних.

- Графові бази даних: використовуються для роботи зі складними зв’язками між даними, наприклад, у соціальних мережах або рекомендаційних системах, де важливі взаємозв’язки між об'єктами.

Завдяки цій різноманітності, NoSQL бази даних легко адаптуються до потреб сучасних додатків.

Ще однією перевагою нереляційних баз даних є висока масштабованість, що досягається через горизонтальне масштабування. Це робить NoSQL бази ідеальними для проєктів, які мають справу з великими обсягами даних або потребують високої швидкодії [26].

Водночас NoSQL бази даних мають і певні недоліки. Одним із них є слабша підтримка транзакцій у порівнянні з реляційними базами. Крім того, робота з великими обсягами даних у нереляційній структурі може потребувати ретельного проектування для уникнення дублювання даних.

З огляду на специфіку програмного забезпечення, NoSQL база даних є оптимальним вибором. Розроблюваний інтернет-магазин потребує гнучкої системи, здатної обробляти великі обсяги даних про товари, замовлення та користувачів, а також швидко адаптуватися до змін. Крім того, її масштабованість забезпечить стабільну роботу системи навіть під час пікових навантажень, наприклад, у періоди акцій чи розпродажів.

Таким чином, використання NoSQL бази даних є найкращим вибором для реалізації проєкту, оскільки вона забезпечує необхідну гнучкість, продуктивність і масштабованість, відповідаючи вимогам сучасних WEB-застосунків.

3.2 Обґрунтування використаних технологій серверної частини

Розробка серверної частини інтернет-магазину комп'ютерної техніки потребувала ретельного вибору архітектурного підходу, що забезпечив би масштабованість, модульність та ефективну інтеграцію з хмарною інфраструктурою. З огляду на вимоги до гнучкості системи та простоти супроводу, було обрано архітектуру на основі RESTful API, організовану у вигляді багаторівневої структури. REST-архітектура дозволяє створювати логічно розділені маршрути для кожного ресурсу системи, зберігаючи при цьому передбачувану структуру взаємодії з клієнтською частиною.

В якості основного технологічного стеку для серверної частини обрано мову програмування Python та мікрофреймворк Flask.

Мова програмування Python є інтерпретованою, об'єктно-орієнтованою та високорівневою мовою з динамічною семантикою. Завдяки вбудованим високорівневим структурам даних, поєднаним із динамічною типізацією та динамічним зв'язуванням, Python є надзвичайно привабливою для швидкої розробки застосунків. Крім того, вона широко застосовується як скриптова мова або як «клейова» мова для поєднання окремих компонентів системи.

Однією з ключових переваг Python є проста та легко засвоювана синтаксична структура, яка робить код читабельним і зменшує витрати на підтримку програмного забезпечення. Python підтримує використання модулів і пакетів, що сприяє модульності програм і повторному використанню коду [27].

Для порівняння, альтернативним вибором могла б стати мова JavaScript у зв'язці з середовищем Node.js та фреймворком Express.js. Express – це мінімалістичний серверний фреймворк, який дозволяє швидко налаштувати API та має широку підтримку в спільноті розробників. Його перевагою є робота в одному стеку – як серверна, так і клієнтська частини можуть бути написані однією мовою, що спрощує командну розробку та повторне використання компонентів [28].

Однак у порівнянні з Express, Flask згідно з даними таблиці 3.1 має низку переваг, зокрема:

- простішу структуру та вищу читабельність коду;
- зручну інтеграцію з хмарними сервісами AWS (через boto3);
- можливість багатопотокової обробки запитів;
- більшу гнучкість у налаштуванні production-середовища (через Gunicorn, Nginx);
-

Таблиця 3.1 – Порівняння фреймворків Express та Flask [29]

Аспект	**Express**	**Flask**
Мова програмування	JavaScript, динамічна типізація	Python
Фреймворк	WEB-фреймворк JavaScript	WEB-фреймворк Python
Платформа	Працює на V8-движку (Chrome)	CPython
Архітектура	Події-керована, неблокуюче введення/виведення	WSGI (інтерфейс шлюзу WEB-серверів)
Паралелізм	Однопоточна модель	Багатопотокова модель
Сильні сторони	Висока масштабованість, реальний час, серверний JavaScript	Простота, легкість у використанні, велика екосистема Python
Інтеграція WEB-серверу	Вбудований HTTP-сервер	Потребує окремого WEB-сервера
Крива навчання	Відносно нестрімка, прийнятна для початківців	Відносно проста, особливо для тих, хто вже працює з Python

На додаток до цього, Flask дозволяє гнучко налаштовувати маршрути, обробляти винятки та масштабувати застосунок з використанням Gunicorn, uWSGI чи інших WSGI-серверів, що відповідає вимогам до виробничих середовищ.

Для зберігання даних про товари, користувачів та замовлення використано Amazon DynamoDB – NoSQL-рішення від AWS, яке забезпечує масштабованість, низьку затримку доступу та високу надійність. Це дозволяє уникнути складних налаштувань класичних SQL-баз і зосередитися на гнучкому зберіганні документів у форматі key-value.

Таким чином, обраний стек технологій Python + Flask + DynamoDB забезпечує оптимальний баланс між простотою реалізації, розширюваністю, безпекою та готовністю до інтеграції з хмарними сервісами, залишаючись більш контрольованим і передбачуваним, ніж JavaScript-рішення на базі Express.js.

3.3 Розробка серверної частини інтернет-магазину

Стартовою точкою застосунку є файл app.py, у якому створюється екземпляр Flask-застосунку, підключаються конфігураційні параметри та маршрути API. Для організації коду було використано концепцію Blueprint, яка дозволяє винести маршрути в окремий модуль, зробивши основний файл менш перевантаженим. Таким чином, структура проєкту стала більш логічною, а підтримка та масштабування – зручнішими. Нижче наведено контролер керуванням застосунку.

Реалізація класу FlaskAppController має такий зміст:

```
class FlaskAppController:
    """
    Controller class for managing the Flask application.
    """
    def __init__(self):
        """Initialize the Flask application."""
```

```
        self.app = Flask(__name__)
        self.cors = CORS(self.app, resources={r"/api/*": {"origins": "*"}})

        self.register_blueprints()

    def register_blueprints(self):
        """Register blueprints for the application."""
        self.app.register_blueprint(api_blueprint, url_prefix="/api")

    def run(self, host="0.0.0.0", port=5000, debug=False):
        self.app.run(host=host, port=port, debug=debug)
```

У серверній частині реалізація REST API здійснюється не у вигляді окремих функцій у файлі routes.py, а через спеціально створений клас ProductAPI, який інкапсулює логіку маршрутизації та обробки запитів, пов’язаних із товарами. Замість прямого визначення маршрутів у Flask-декораторах, маршрути реєструються динамічно через метод register_routes() у конструкторі класу. Це дозволяє централізовано контролювати маршрути та забезпечує гнучкість при масштабуванні API.

Реалізація класу ProductAPI має такий зміст:

```
class ProductAPI:
    def __init__(self, blueprint):
        self.blueprint = blueprint
        self.db = db_instance  # Use the Database instance
        self.register_routes()

    def register_routes(self):
        self.blueprint.route("/health", methods=["GET"])(self.api_status)
        self.blueprint.route("/products", methods=["GET"])(self.get_all_products)
        self.blueprint.route("/products/<string:product_id>", methods=["GET"])(self.get_product)
```

Кожен маршрут викликає відповідний метод класу. Наприклад, GET /products обробляється методом get_all_products(), який звертається до бази даних через інкапсульований об’єкт db_instance. Подібно, GET /products/<product_id> дозволяє отримати дані про конкретний товар за його ідентифікатором. Всі запити повертають дані у форматі JSON, а обробка винятків забезпечує захищене виконання запитів.

Особливістю реалізації є чітке відокремлення доступу до даних у вигляді окремого модуля db_instance, створеного на основі класу бази даних (розміщеного у файлі dynamodb.py). Весь доступ до Amazon DynamoDB відбувається через цей інтерфейс, що значно спрощує супровід та можливу міграцію на інше сховище у майбутньому.

У якості бази даних використовується Amazon DynamoDB – високопродуктивне NoSQL-рішення, яке дозволяє зберігати записи у вигляді пар «ключ-значення». У файлі dynamodb.py реалізовано метод get_all_items(), що виконує повне сканування таблиці через команду scan(), а також метод get_item(product_id), який дозволяє знайти конкретний запис за унікальним ідентифікатором. Доступ до цих методів здійснюється через екземпляр db_instance, що імпортується у ProductAPI.

Реалізація класу Database має такий зміст:

```
class Database:
    """Singleton class to manage DynamoDB connection."""
    _instance = None  # Store a single instance

    def __new__(cls, *args, **kwargs):
        """Ensure only one instance of Database is created."""
        if not cls._instance:
            cls._instance = super(Database, cls).__new__(cls)
            cls._instance._init_db()
        return cls._instance
```

```
def _init_db(self):
    """Initialize the database connection for DynamoDB."""
    self.dynamodb = boto3.resource("dynamodb", region_name=Config.AWS_REGION)
   self.dynamo_table = self.dynamodb.Table(Config.DYNAMO_TABLE)  # Define table once

def get_item(self, key):
    """Retrieve a single item by primary key."""
    response = self.dynamo_table.get_item(Key={"name": key})
    return response.get("Item")  # Return the item if found, else None

def get_all_items(self):
    """Retrieve all items from the DynamoDB table."""
    response = self.dynamo_table.scan()
     return response.get("Items", [])
```

Щоб привести отримані з бази дані до уніфікованого вигляду, використовується модель Product, яка реалізована на базі бібліотеки pydantic. Вона забезпечує автоматичну валідацію полів і серіалізацію у формат JSON. Це дозволяє обробляти навіть складні структури товарів і повертати їх у клієнтський застосунок у стабільному форматі.

Реалізація класу Product(BaseModel) має такий зміст:

```
class Product(BaseModel):
    id: str = Field(alias="_id")  # Map MongoDB's `_id` to `id`
    name: str
    description: str
    price: float
    imageUrl: Optional[str] = ""
    category: str
    characteristics: Dict[str, Union[str, int, float, bool]]
```

3.4 Обґрунтування вибору інструментів реалізації клієнтської частини

Під час створення клієнтської частини інтернет-магазину комп'ютерної техніки особливу увагу було приділено вибору технологій, які забезпечують високу швидкодію, простоту у використанні, легку інтеграцію з серверною частиною та можливість масштабування. Нижче наведено порівняльний аналіз основних інструментів, з яких формувався технологічний стек фронтенду, а також обґрунтування вибору кожного з них для конкретного проєкту.

React є бібліотекою JavaScript для побудови інтерфейсів користувача, яка базується на концепції компонентів і віртуального DOM. Вона дозволяє створювати багаторазові та незалежні частини інтерфейсу, що значно спрощує розробку складних застосунків [30].

Основними перевагами React є висока продуктивність, підтримка великою спільнотою, активна екосистема додаткових бібліотек та можливість легкої інтеграції з іншими технологіями. Важливою перевагою є також гнучкість: розробник самостійно обирає структуру проєкту, спосіб керування станом та бібліотеки для допоміжних задач.

До недоліків React можна віднести порівняно високу складність керування станом у великих застосунках (що часто потребує підключення додаткових інструментів) та необхідність використання JSX, що може бути неочевидним для новачків.

Angular – це повноцінний фреймворк для створення односторінкових WEB-застосунків, який розробляється Google. Angular забезпечує високий рівень організації коду, надає вбудовану підтримку сервісів, форм, роутингу та валідації. Його головною перевагою є комплексність: усі інструменти вже включені в ядро фреймворку, що дозволяє швидко створювати масштабовані застосунки. Проте така складність є й недоліком: поріг входу для Angular суттєво вищий, ніж для React,

структура проєкту жорстко регламентована, а надлишкова декларативність ускладнює розробку простих інтерфейсів. Angular більше розрахований для великих команд та корпоративних застосунків, де є чітко визначена архітектура та багато взаємопов’язаних модулів [31].

З огляду на вимоги до проєкту – швидка розробка, мінімалізм інтерфейсу, інтеграція з бекендом через REST API та можливість масштабування – вибір було зроблено на користь React. Ця бібліотека забезпечує високу продуктивність, гарно відповідає компонентній архітектурі інтернет-магазину, а також дозволяє гнучко організувати взаємодію з мікросервісами.

Для типізації в проєкті було використано TypeScript, який є надбудовою над JavaScript і вводить статичну типізацію. Основна ідея TypeScript полягає у запобіганні помилкам ще на етапі компіляції, а також у підвищенні читабельності коду та його передбачуваності. Завдяки цьому підхід до розробки стає надійнішим, а документація коду – зрозумілішою [32].

Перевагами TypeScript є інтеграція з сучасними IDE, підтримка складних структур типів, контроль за відповідністю структури даних. Серед недоліків варто назвати складніший синтаксис, вищу вартість початкового навчання та необхідність налаштування додаткових інструментів компіляції.

У якості альтернативи розглядався чистий JavaScript, який не потребує компіляції, є простішим у використанні та достатнім для невеликих застосунків. Його перевагою є гнучкість та універсальність, однак відсутність типізації ускладнює виявлення помилок та призводить до появи нестабільного коду, особливо в складних проєктах. JavaScript вимагає більшої уваги до ручної перевірки типів і не забезпечує гарантій надійності API-викликів.

Враховуючи потребу у масштабованості, модульності й зниженні кількості помилок при взаємодії з бекендом, було прийнято рішення використовувати

TypeScript. Його впровадження дозволяє зменшити технічний борг та забезпечити вищу якість коду на довготривалій перспективі.

Для організації стилів інтерфейсу обрано Tailwind CSS – утилітарний CSS-фреймворк, що дозволяє будувати адаптивний дизайн без створення окремих CSS-файлів. Tailwind працює за принципом готових класів, які вказуються безпосередньо в HTML-елементах, забезпечуючи високу швидкість верстки, передбачуваність стилів та зручність у підтримці. Його перевагами є швидкість розробки, адаптивність «із коробки» та можливість гнучкого налаштування [33].

Основним недоліком Tailwind є початковий обсяг класів, що може знижувати читабельність HTML-розмітки та ускладнювати підтримку у дуже великих проєктах без правильного структурування компонентів.

У якості альтернативи міг використовуватись Bootstrap – класичний CSS-фреймворк із набором готових компонентів. Bootstrap зручний для швидкого створення базового інтерфейсу, але має обмежені можливості кастомізації без переписування стилів. У проєктах із унікальним дизайном він часто створює конфлікти між системними стилями та індивідуальними рішеннями, що ускладнює підтримку [34].

Зважаючи на необхідність створення сучасного, легкого та кастомізованого інтерфейсу, а також швидкість реалізації – Tailwind CSS став оптимальним вибором. Він гарно інтегрується з React/Next.js і дозволяє створювати адаптивні, уніфіковані за стилем компоненти з мінімальними витратами часу.

Таким чином, обрані інструменти React, TypeScript та Tailwind CSS у поєднанні з архітектурою Next.js забезпечили реалізацію клієнтської частини інтернет-магазину, що відповідає всім сучасним вимогам: масштабованість, гнучкість, модульність, адаптивність і зручність підтримки.

3.5 Реалізація інфраструктурної частини системи за допомогою Terraform

У межах проєкту створення інтернет-магазину комп'ютерної техніки на базі хмарних технологій важливу роль відіграє правильне та автоматизоване розгортання інфраструктури. Для цього було обрано підхід «інфраструктура як код» (Infrastructure as Code, IaC) із використанням інструменту Terraform. Завдяки цьому стало можливим описати всю хмарну інфраструктуру у вигляді коду, що забезпечує повторюваність, масштабованість і просте обслуговування середовища.

Для реалізації інфраструктури були задіяні основні сервіси AWS, зокрема: балансувальник навантаження Application Load Balancer (ALB), контейнерна платформа Amazon ECS (Fargate), сховище контейнерів Elastic Container Registry (ECR), серверна NoSQL база даних DynamoDB і служба доменних імен Amazon Route 53. Далі наведено опис кожного компонента з відповідним фрагментом конфігурації Terraform.

Для розподілу трафіку між компонентами системи (наприклад, frontend і backend) використовується ALB, який автоматично розподіляє навантаження та підтримує HTTPS-з'єднання. Це покращує відмовостійкість і забезпечує безпечний доступ до додатку з Інтернету.

Реалізація балансувальника навантаження ALB має такий зміст:

```
module "alb" {
  source  = "terraform-aws-modules/alb/aws"
  version = "9.11.0"

  name        = format("%s-alb", var.project.name)
  vpc_id      = module.vpc.vpc_id
  subnets     = module.vpc.public_subnets

  security_group_ingress_rules = {
    all_https = {
```

```
      from_port   = 443
      to_port     = 443
      ip_protocol = "tcp"
      description = "HTTPS web traffic"
      cidr_ipv4   = "0.0.0.0/0"
    }
  }
  listeners = {
    http = {
      port     = 80
      protocol = "HTTP"
      redirect = {
        port        = "443"
        protocol    = "HTTPS"
        status_code = "HTTP_301"
      }
    }
  }
}
```

Цей фрагмент коду конфігурації Terraform описує створення модуля Application Load Balancer (ALB) за допомогою офіційного модуля terraform-aws-modules/alb/aws. Основна мета цього ресурсу – забезпечити розподіл вхідного HTTP(S) трафіку до окремих компонентів інтернет-магазину, таких як frontend і backend. Це критично важливо для забезпечення високої доступності, масштабованості та безпечного доступу до сервісу.

У параметрі vpc_id вказується ідентифікатор VPC, у якому розгортається балансувальник, а поле subnets визначає список публічних підмереж, до яких прив’язується ALB. Таким чином, ALB стає доступним із зовнішньої мережі Інтернет, що дозволяє обробляти запити клієнтів.

Дуже важливим елементом є конфігурація правил безпеки (Security Groups), які вказані в секціях security_group_ingress_rules та security_group_egress_rules. У цьому випадку дозволяється весь вхідний HTTPS-трафік (порт 443) з будь-яких IP-адрес (cidr_ipv4 = "0.0.0.0/0"), що робить додаток доступним для всіх користувачів. Водночас вихідний трафік дозволено повністю (ip_protocol = "-1"), що забезпечує взаємодію з іншими сервісами.

Окрему увагу приділено секції listeners, яка описує налаштування для портів, що обробляються ALB. У наведеному прикладі всі запити, що надходять на порт 80 (HTTP), автоматично перенаправляються на порт 443 (HTTPS) за допомогою редіректу з кодом HTTP_301. Це забезпечує шифрування трафіку між клієнтом і сервером, що є важливою умовою для сучасних WEB-сервісів, які обробляють персональні або платіжні дані.

Для розміщення frontend- і backend-компонентів використовується Amazon ECS у режимі Fargate, що дозволяє запускати контейнери без управління EC2-серверами. Це спрощує масштабування та оновлення.

Реалізація ECS має такий зміст:

```
module "ecs" {
  source  = "terraform-aws-modules/ecs/aws"
  version = "5.12.0"
  cluster_name = format("%s-cluster", var.project.name)
  fargate_capacity_providers = {
    FARGATE = {
      default_capacity_provider_strategy = {
        weight = 100
        base   = 3
      }
    }
  }
  services = {
```

```
    frontend = {
      cpu    = 512
      memory = 1024
      container_definitions = {
        frontend = {
          cpu       = 512
          memory    = 1024
            image     = format("%s:latest", data.aws_ecr_repository.frontend.repository_url)
          port_mappings = [{
            name          = "frontend-port"
            containerPort = 3000
            hostPort      = 3000
          }]
        }
      }
    }
  }
```

У цьому фрагменті коду здійснюється налаштування кластеру ECS (Elastic Container Service) із використанням обчислювальної моделі Fargate. Це дозволяє запускати та масштабувати контейнери без необхідності вручну керувати віртуальними машинами (наприклад, EC2). Такий підхід значно спрощує адміністрування інфраструктури, оскільки AWS бере на себе усі задачі з розподілу ресурсів, оновлення безпеки, балансування навантаження тощо.

Визначення cluster_name формує унікальну назву кластеру, прив’язану до назви проєкту, що робить розгортання більш зрозумілим і структурованим. У блоці fargate_capacity_providers задається політика масштабування, яка вказує, що ECS має використовувати провайдера FARGATE як основний. Атрибут base = 3 означає, що завжди має бути запущено щонайменше три екземпляри завдань (tasks), а weight

= 100 вказує на те, що весь трафік спрямовується саме через цей провайдер потужностей.

У розділі services конфігурується окремий ECS-сервіс — у цьому випадку frontend, який є ключовим елементом WEB-інтерфейсу інтернет-магазину. Параметри cpu = 512 і memory = 1024 визначають ресурси, виділені кожному контейнеру: 0.5 vCPU та 1 ГБ оперативної пам’яті. Це дозволяє досягти оптимального балансу між продуктивністю і вартістю використання ресурсів.

Секція container_definitions описує окремі контейнери, які мають запускатися в межах цього сервісу. Для кожного контейнера вказується:

- image: шлях до Docker-образу, який береться з приватного репозиторію ECR (Elastic Container Registry), за допомогою конструкції data.aws_ecr_repository.frontend.repository_url. Це гарантує, що при кожному оновленні контейнер буде запускатися з актуальною версією програмного коду;
- port_mappings: описує, на яких портах контейнер прийматиме запити (у цьому випадку порт 3000), як на стороні контейнера (containerPort), так і на стороні хоста (hostPort).

Як базу даних у проєкті використано Amazon DynamoDB – це повністю керована NoSQL база, яка забезпечує швидкий і надійний доступ до даних без необхідності в адмініструванні серверів. Вона ідеально розрахована для зберігання замовлень, списків бажаного, токенів сесій тощо.

Реалізація DynamoDB має такий зміст:

```
module "dynamodb_table" {
  source   = "terraform-aws-modules/dynamodb-table/aws"
  name     = format("%s-table", var.project.name)
  hash_key = "_id"
  attributes = [
    {
      name = "_id"
      type = "S"
```

```
    }
  ]
  tags = var.project.tags
}
```

Для зберігання образів Docker, що використовуються у ECS, було застосовано ECR – безпечне, масштабоване сховище контейнерів.

Реалізація ECR має такий зміст:

```
data "aws_ecr_repository" "frontend" {
  name = "tech-savvy-fe"
}
data "aws_ecr_repository" "backend" {
  name = "tech-savvy-be"
}
```

Для керування DNS-записами та підключення доменів використовується Route 53 – сервіс, який забезпечує масштабованість, надійність та інтеграцію з ALB.

Реалізація Route 53 має такий зміст:

```
resource "aws_route53_record" "frontend" {
  zone_id = var.route53_zone_id
  name    = "frontend.${var.domain_name}"
  type    = "A"
  alias {
    name                   = module.alb.this_lb_dns_name
    zone_id                = module.alb.this_lb_zone_id
    evaluate_target_health = true
  }
}
```

Завдяки використанню Terraform та сервісів AWS було реалізовано повністю автоматизоване розгортання хмарної інфраструктури. Всі ключові компоненти – балансування трафіку, контейнеризація, зберігання даних, DNS – інтегровані в єдине кероване середовище. Це забезпечує не лише надійність і масштабованість, а

й спрощує оновлення та підтримку системи. Інфраструктура є гнучкою до змін і готова до збільшення навантаження в умовах збільшення кількості користувачів та транзакцій.

3.6 Тестування роботи програми та аналіз результатів

Тестування програмного забезпечення є важливою складовою розробки, що дозволяє перевірити правильність функціонування системи, її зручність у використанні, стабільність та ефективність. Це дає змогу виявити та усунути помилки на ранніх етапах, забезпечити відповідність функціональним і нефункціональним вимогам, а також покращити загальну якість користувацького досвіду. Розроблений програмний продукт був ретельно протестований, із проведенням понад 100 тестових запусків.

Після запуску з’являється головна сторінка з закріпленим продуктом. На рисунку 3.1 зображено головну сторінку зі закріпленим продуктом.

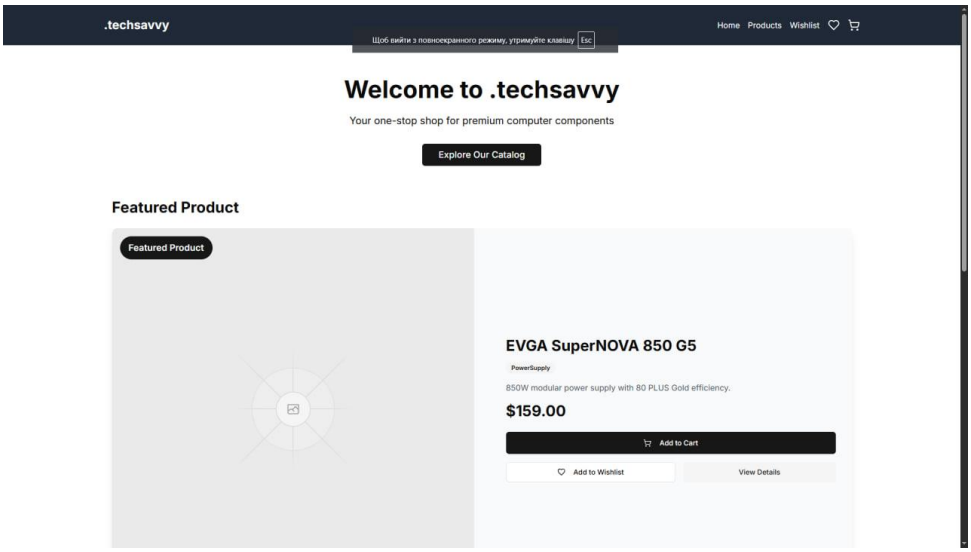

Рисунок 3.1 – Загальний вигляд інтерфейсу головної сторінки розробки

Прогорнувши нижче також можна побачити рекомендовані товари та секцію «Чому варто обрати цей магазин» (рис. 3.2).

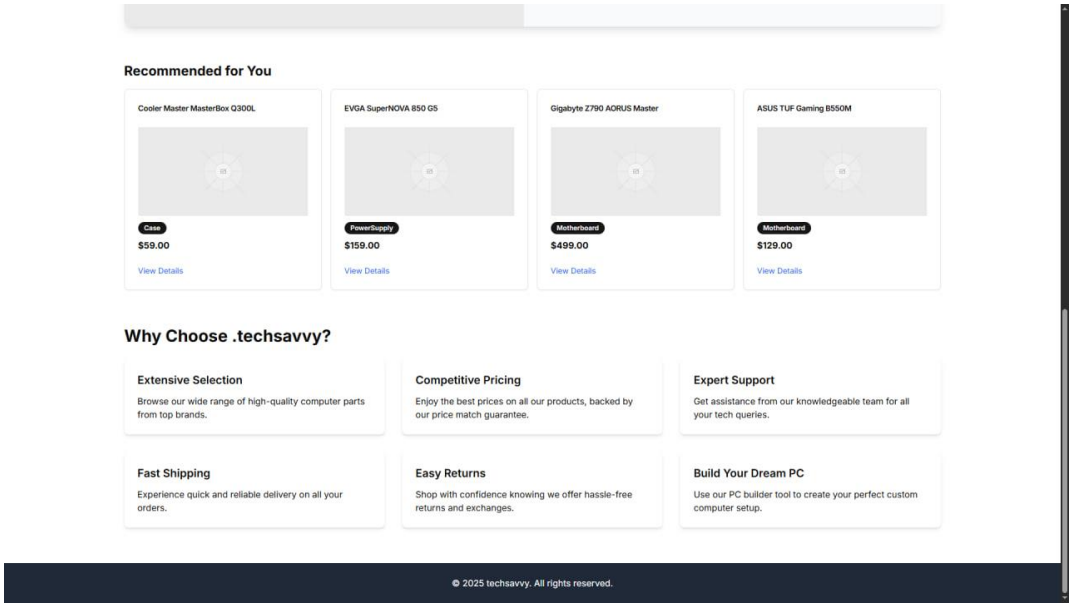

Рисунок 3.2 – Загальний вигляд інтерфейсу рекомендованих товарів

Наступним кроком необхідно перевірити сторінку з продуктами магазину та її працездатність. Перейдемо на сторінку «Products», яка відповідає за виведення всіх доступних товарів. Як видно з рисунка 3.3, WEB-ресурс коректно завантажує сторінку з усіма продуктами – контент зображується чітко, інтерфейс адаптивний і не спостерігається жодних збоїв у відображенні товарів.

Після цього перевіримо функціональність системи фільтрації товарів. Одним із важливих інструментів пошуку є фільтр за категорією продукту. Застосуємо фільтр із вибраною категорією «Storage», що відображено на рисунку 3.4. У результаті система успішно оновлює вміст каталогу відповідно до заданої категорії, тобто виводяться лише ті товари, що належать до групи накопичувачів.

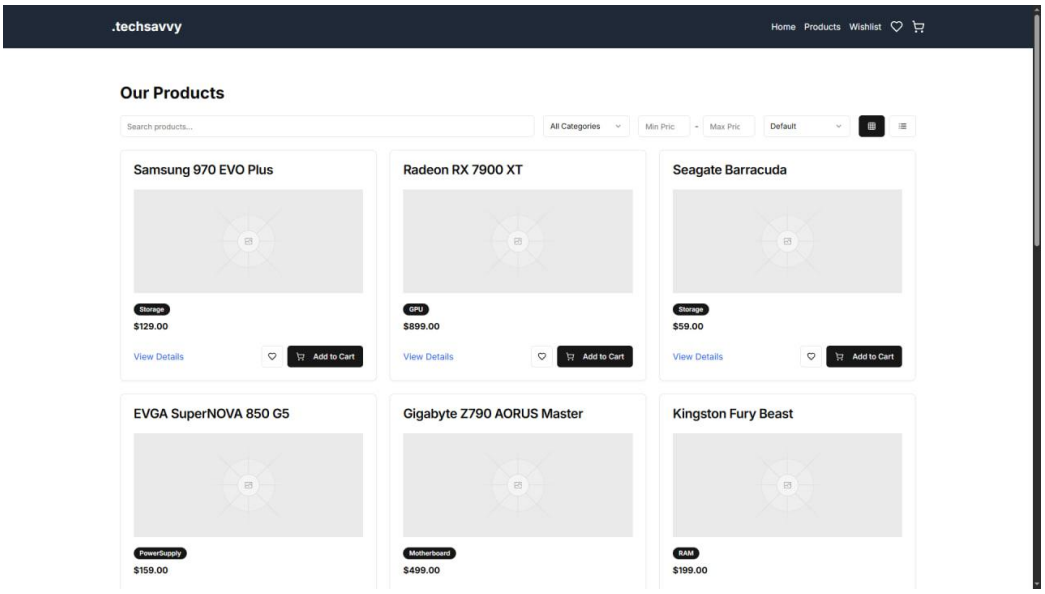

Рисунок 3.3 – Загальний вигляд інтерфейсу сторінки продуктів

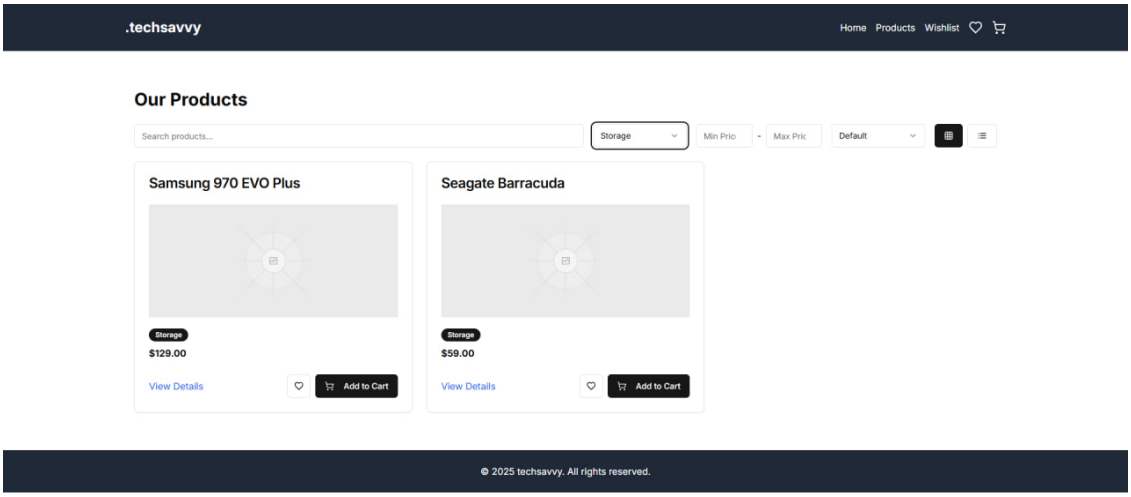

Рисунок 3.4 – Загальний вигляд інтерфейсу застосування категорій на сторінці продуктів

Це свідчить про коректну реалізацію фільтрації, зокрема обробку запитів до бази даних і динамічне оновлення інтерфейсу без потреби перезавантаження сторінки.

Також було протестовано зміну режиму відображення товарів на сторінці каталогу. Інтерфейс передбачає два варіанти подання: у вигляді плитки та у вигляді

списку. Перемикання між цими режимами здійснюється натисканням відповідних іконок у правій частині панелі фільтрації.

У процесі тестування підтверджено, що зміна режиму відбувається миттєво, без перезавантаження сторінки та без втрати заданих параметрів фільтрації. У режимі плитки товари відображаються компактно, що зручно для швидкого перегляду, тоді як режим списку дозволяє користувачеві бачити розширену інформацію про кожен продукт.

Зображення на рисунку 3.5 демонструє інтерфейс сторінки з активним режимом списку, застосованим фільтром за категорією Storage.

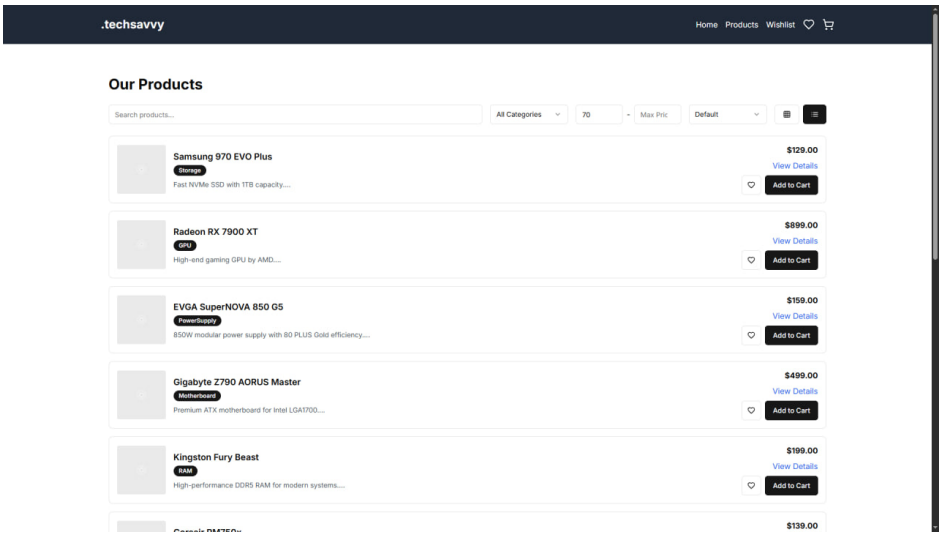

Рисунок 3.5 – Загальний вигляд режиму відображення «список»

Також було протестовано функціональність додавання товарів до списку бажаного. Ця можливість дозволяє користувачу зберігати вподобані товари для подальшого перегляду або покупки. На сторінці каталогу біля кожного товару передбачено іконку у вигляді серця. При натисканні на неї товар автоматично додається до списку бажаного.

На рисунку 3.6 показано, як виглядає сторінка продуктів у момент, коли деякі товари вже були позначені як бажані.

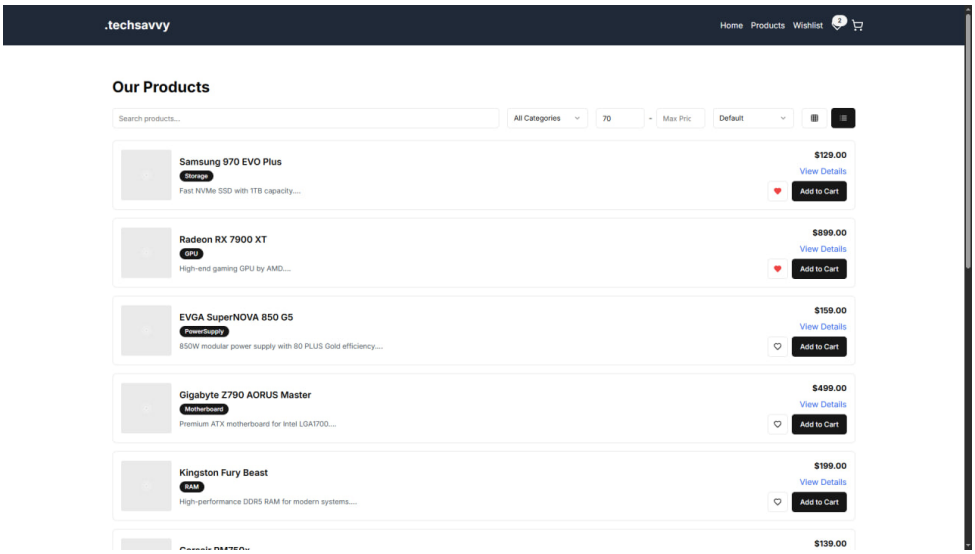

Рисунок 3.6 – Загальний вигляд інтерфейсу додавання товарів до списку бажаного

Перехід до розділу Wishlist дозволяє переглядати всі додані товари у зручному форматі. На рисунку 3.7 представлено вигляд цієї сторінки. Тестування показало, що всі дії виконуються без затримок, динамічно оновлюючи інтерфейс без перезавантаження сторінки.

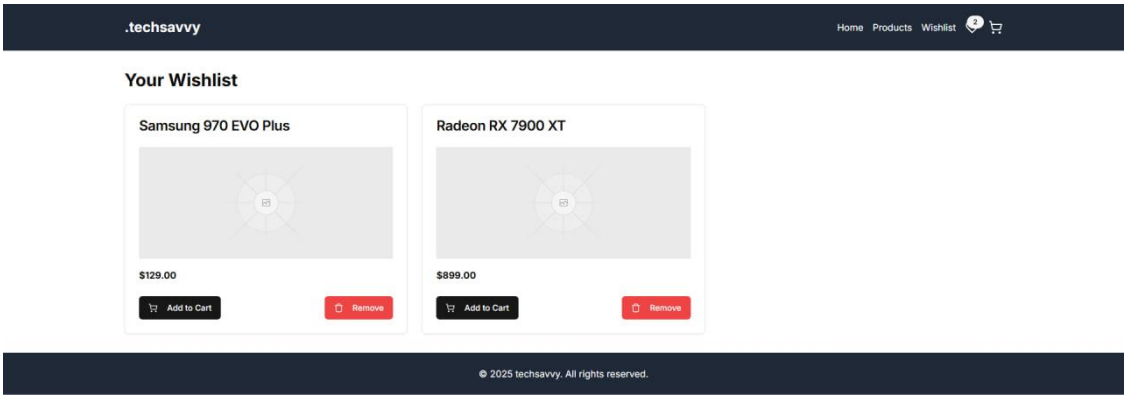

Рисунок 3.7 – Загальний вигляд інтерфейсу сторінки списку бажаного

Таким чином, функція wishlist повністю виконує свою роль: користувач може легко додавати й видаляти товари зі списку, зберігаючи при цьому швидкий доступ

до вподобаних позицій. Реалізація цієї функції відповідає сучасним стандартам електронної комерції.

Наступним етапом тестування стала перевірка роботи кошика, який є важливою частиною інтернет-магазину. Основними функціями кошика є: додавання товарів, перегляд вмісту, зміна кількості, видалення окремих позицій або повне очищення, а також перехід до оформлення замовлення.

Після додавання користувач має можливість перейти до кошика, де відображається повний список товарів з вказанням їх назви, категорії, ціни, кількості та загальної суми.

На рисунку 3.8 показано вигляд сторінки кошика після додавання трьох позицій. Загальна сума покупки виводиться автоматично та оновлюється при зміні вмісту кошика.

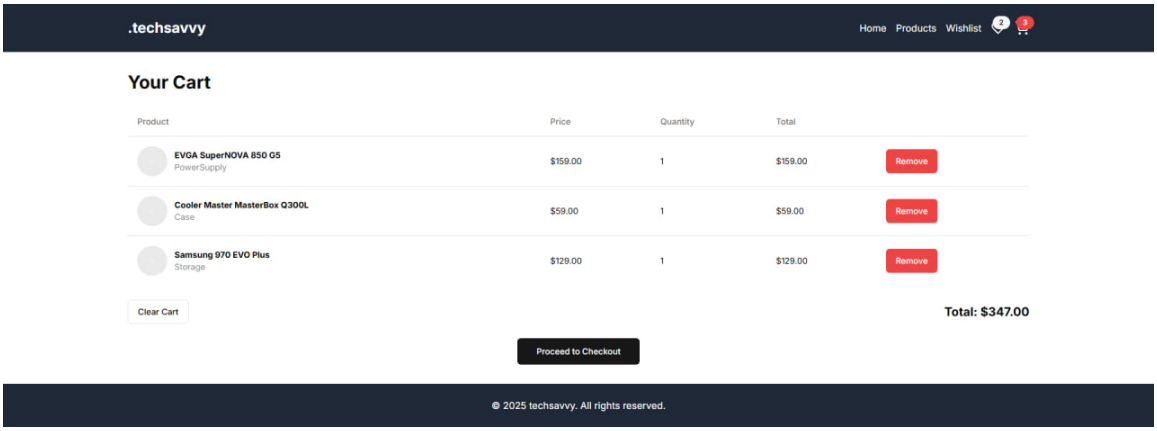

Рисунок 3.8 – Загальний вигляд інтерфейсу сторінки перегляду кошика

Під час тестування всі основні функції кошика працювали стабільно. Додавання та видалення товарів, а також підрахунок суми виконувалися миттєво, без затримок чи збоїв. Дані зберігалися впродовж сесії, а користувач мав повний контроль над вмістом кошика перед переходом до оформлення замовлення.

Окрему увагу під час тестування було приділено перевірці сторінки окремого товару. Ця сторінка надає користувачу розширену інформацію про конкретну

позицію, включаючи назву, опис, ціну, технічні характеристики, а також кнопки для взаємодії – додавання до кошика або видалення зі списку бажаного.

На рисунку 3.9 показано приклад сторінки для товару Radeon RX 7900 XT.

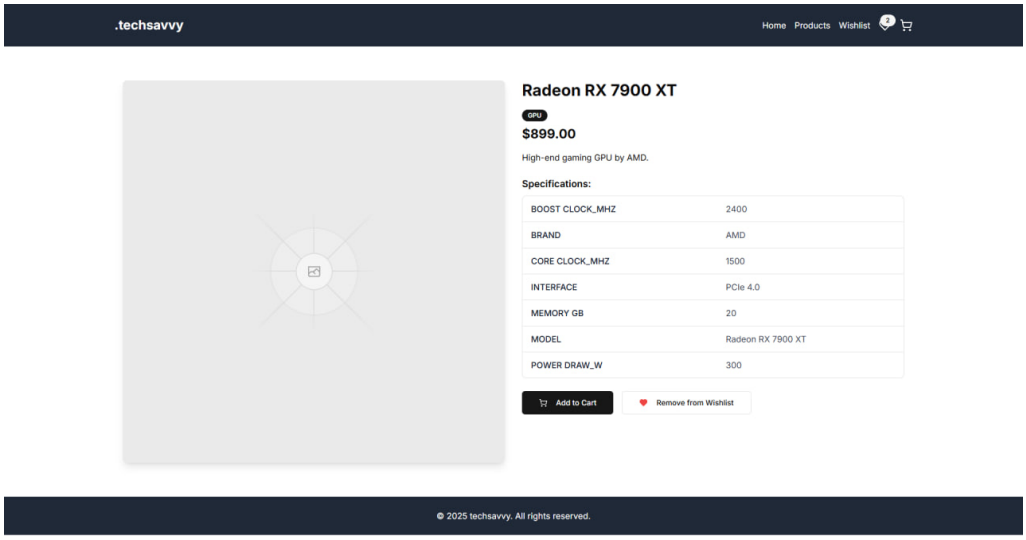

Рисунок 3.9 – Загальний вигляд інтерфейсу перегляду продукту

Інтерфейс складається з великого блоку для зображення товару, блоку з коротким описом і структурованої таблиці специфікацій. У таблиці (рис. 3.9) наведено ключові характеристики. У нижній частині сторінки передбачено зручні кнопки, які дозволяють швидко виконати відповідні дії без переходу на інші сторінки. В процесі тестування підтверджено, що всі елементи працюють стабільно, зміна стану кнопок відбувається миттєво, а інформація про товар відображається без затримок.

У результаті проведеного тестування підтверджено коректну роботу всіх ключових функцій інтернет-магазину: перегляду каталогу, фільтрації, перемикання режимів відображення, роботи зі списком бажаного, кошиком та сторінкою окремого товару. Інтерфейс працює стабільно, динамічні дії виконуються без перезавантаження сторінок, а функціональність відповідає заявленим вимогам. Отримані результати свідчать про готовність системи до практичного використання.

У таблиці 3.2 подано порівняння результатів тестування розробленого програмного продукту та аналогів.

Таблиця 3.2 – Результати тестування розробленого програмного продукту та аналогів

	Розроблений сайт	Artline	Telemart
Список бажаного	+	–	+
Широкий список характеристик	+	–	+
Збереження фільтрів	+	–	–
Персоналізовані рекомендації	+	+	–
Інтеграція із сучасними хмарними технологіями	+	±	±
Використання serverless технологій	+	–	–

Отже, проаналізувавши та протестувавши програмний продукт, можна дійти висновку, що поставлених цілей було досягнуто.

З таблиці 3.2 можна побачити, що функціонал розробленого WEB-ресурсу комп'ютерної техніки було покращено у порівнянні з системами-аналогами на 3 можливості: збереження фільтрів, інтеграція з сучасними хмарними технологіями, використання serverless технологій.

3.7 Висновок

У третьому розділі було здійснено повну програмну реалізацію інтернет-магазину комп'ютерної техніки, що охоплює серверну, клієнтську та інфраструктурну частини системи. В основу реалізації покладено сучасні технології, які забезпечують масштабованість, стабільність і високу продуктивність WEB-застосунку.

Для обробки запитів до бази даних було обґрунтовано вибір мови програмування Python та використання фреймворку Flask. Серверна частина

реалізована у вигляді RESTful API з логічною багаторівневою структурою, що підвищує її гнучкість та дозволяє легко масштабувати застосунок. База даних побудована на основі NoSQL-рішення Amazon DynamoDB, що забезпечує високу швидкодію та надійність зберігання інформації.

Клієнтська частина реалізована з використанням бібліотеки React, яка забезпечує побудову компонентного інтерфейсу з високою продуктивністю. Запроваджено управління кошиком, списком бажаного, фільтрацією та сортуванням товарів, що дозволяє користувачам комфортно взаємодіяти з системою.

Інфраструктурна частина системи розгорнута за допомогою інструменту Terraform, що забезпечує декларативне управління ресурсами AWS. Це включає автоматизацію створення контейнерів, їх розміщення в сервісі ECS, використання S3 для зберігання зображень, ECR для зберігання Docker-образів та використання сервісу Route 53 і ALB для маршрутизації трафіку.

На завершальному етапі було проведено тестування розробленої системи, результати якого підтвердили її працездатність, відповідність функціональним вимогам та стабільність при обробці запитів. Розроблений програмний продукт демонструє покращений функціонал порівняно з аналогами на 3 можливості, а саме, збереження фільтрів, інтеграція з сучасними хмарними технологіями, використання serverless технологій.

Таким чином, програмна реалізація інтернет-магазину повністю відповідає вимогам до сучасного WEB-застосунку і є надійною основою для подальшого розвитку функціоналу.

ВИСНОВКИ

Всі задачі, поставлені для реалізації інтернет-магазину комп'ютерної техніки на основі хмарних технологій були виконані в повному обсязі, а саме: обґрунтовано доцільність розробки інтернет-магазину комп'ютерної техніки на основі хмарних технологій, спроектовано архітектуру інтернет-магазину комп'ютерної техніки на основі хмарних технологій, програмно реалізовано інтернет-магазин комп'ютерної техніки на основі хмарних технологій, виконано тестування системи та проведено аналіз отриманих результатів, розроблено інструкцію користувача.

Для клієнтської частини використано TypeScript з React, що забезпечує роботу в WEB-браузерах. Серверну частину реалізовано на Python з використанням Flask, а для зберігання даних застосовано NoSQL-базу Amazon DynamoDB. Це дозволяє створити масштабовану, швидкодіючу та гнучку систему з розділенням логіки між клієнтською та серверною частинами, забезпечити ефективну обробку великої кількості запитів, а також зручно керувати динамічними та неструктурованими даними в умовах реального комерційного навантаження.

Для реалізації була розроблена архітектура інтернет-магазину комп’ютерної техніки, побудовано UML-діаграми класів для клієнтської та серверної частин, сформовано ER-модель бази даних із урахуванням специфіки NoSQL, а також розроблено алгоритмічну схему роботи основних модулів системи, що сприяє підвищенню структурованості, зрозумілості та ефективності розробки програмного забезпечення. Крім того, обґрунтовано вибір хмарної платформи AWS як найоптимальнішого рішення для забезпечення стабільної, безпечної та масштабованої роботи системи в хмарному середовищі.

Розроблений програмний продукт успішно пройшов тестування та демонструє переваги над існуючими аналогами, надаючи користувачам

щонайменше 3 додаткові функціональні можливості, що значно підвищує рівень зручності та задоволеності від взаємодії із системою.

Мета роботи, яка полягала в розширенні функціональних можливостей інтернет-магазину комп'ютерної техніки на основі хмарних технологій досягнута за рахунок введення таких 3 додаткових можливостей: збереження фільтрів, інтеграція з сучасними хмарними технологіями, використання serverless технологій.

Робота виконана у повному обсязі відповідно до поставлених цілей, вимог та методичних рекомендацій щодо підготовки бакалаврських кваліфікаційних робіт. Усі етапи були реалізовані належним чином, що підтверджує якість, завершеність і практичну цінність розробленого рішення [35].

У порівнянні з наявними аналогами, розроблений продукт вирізняється зручною системою пошуку, персоналізованими рекомендаціями та застосування сучасних хмарних технологій.

Отже, створений інтернет-магазин повністю відповідає поставленим вимогам.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Дидь К. В., Крилик Л. В. Перспективи використання хмарних технологій у розробці інтернет-магазину. *LIV Всеукраїнська науково-технічна конференція факультету інтелектуальних інформаційних технологій та автоматизації.* URL: https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2025/paper/view/23116 (дата звернення: 12.05.2025).

2. Revenue share of the e-commerce market worldwide from 2019 to 2029, by sales channel. URL: https://www.statista.com/statistics/534123/e-commerce-share-of-retail-sales-worldwide/ (дата звернення: 12.05.2025).

3. E-commerce в Україні: цифри, факти, перспективи розвитку онлайн-торгівлі. URL: https://www.site2b.ua/ua/web-blog-ua/e-commerce-v-ukraini-cifri-fakti-perspektivi-rozvitku-onlajn-torgivli.html (дата звернення: 12.05.2025).

4. Telemart.ua. URL: https://telemart.ua/ua/content/about.html (дата звернення: 12.05.2025).

5. Artline.ua. URL: https://artline.ua/uk/o-nas (дата звернення: 12.05.2025).

6. Hotline.ua. URL: https://hotline.ua/about/ (дата звернення: 12.05.2025).

7. AWS vs Azure vs Google Cloud. URL: https://cloudfresh.com/en/blog/aws-vs-azure-vs-google-cloud/ (дата звернення: 12.05.2025).

8. AWS. URL: https://aws.amazon.com/ (дата звернення: 12.05.2025).

9. Azure. URL: https://azure.microsoft.com/en-us/ (дата звернення: 12.05.2025).

10. Google Cloud. URL: https://cloud.google.com/ (дата звернення: 12.05.2025).

11. Функціональні та нефункціональні вимоги. URL: https://visuresolutions.com/uk/посібник-з-відстеження-управління-вимогами/функціональні-та-нефункціональні-вимоги/ (дата звернення: 12.05.2025).

12. Монолітна архітектура проти архітектури мікросервісів. URL: https://javascript.org.ua/monolitna-arhitektura-proti-arhitekturi-mikroservisiv/ (дата звернення: 12.05.2025).

13. Вступ до REST API–RESTful вебсервіси. URL: https://robotdreams.cc/uk/blog/466-vstup-do-rest-api-restful-vebservisi (дата звернення: 12.05.2025).

14. Поняття ER-моделі. Поняття сутності (entity). Атрибути. Види атрибутів. URL: https://www.bestprog.net/uk/2019/01/24/the-concept-of-er-model-the-concept-of-essence-and-communication-attributes-attribute-types-ua/ (дата звернення: 12.05.2025).

15. Serverless on AWS. URL: https://aws.amazon.com/serverless/ (дата звернення: 12.05.2025).

16. What is Architecture Diagramming? URL: https://aws.amazon.com/what-is/architecture-diagramming/ (дата звернення: 12.05.2025).

17. Amazon Route 53. URL: https://aws.amazon.com/route53/ (дата звернення: 12.05.2025).

18. What is an Application Load Balancer? URL: https://docs.aws.amazon.com/elasticloadbalancing/latest/application/introduction.html (дата звернення: 12.05.2025).

19. Amazon Elastic Container Service. URL: https://aws.amazon.com/ecs/ (дата звернення: 12.05.2025).

20. Amazon DynamoDB. URL: https://aws.amazon.com/dynamodb/ (дата звернення: 12.05.2025).

21. Amazon S3. URL: https://aws.amazon.com/s3/ (дата звернення: 12.05.2025).

22. Amazon ECR private registry. URL: https://docs.aws.amazon.com/AmazonECR/latest/userguide/Registries.html (дата звернення: 12.05.2025).

23. Relational Database. URL: https://aws.amazon.com/relational-database/ (дата звернення: 12.05.2025).

24. What are the benefits and drawbacks of using a relational database? URL: https://www.linkedin.com/advice/3/what-benefits-drawbacks-using-relational-database-skills-databases (дата звернення: 12.05.2025).

25. What is NoSQL? URL: https://aws.amazon.com/nosql/ (дата звернення: 12.05.2025).

26. Relational vs. Non-Relational Databases. URL: https://www.mongodb.com/resources/compare/relational-vs-non-relational-databases (дата звернення: 12.05.2025).

27. What is Python? Executive Summary. URL: https://www.python.org/doc/essays/blurb/ (дата звернення: 12.05.2025).

28. ExpressJS. URL: https://expressjs.com/ (дата звернення: 12.05.2025).

29. Python Flask vs Node.js Express. URL: https://medium.com/@roelljr/python-flask-vs-node-js-express-4662b6f97b28 (дата звернення: 12.05.2025).

30. React. URL: https://react.dev/ (дата звернення: 12.05.2025).

31. Angular. URL: https://angular.dev/ (дата звернення: 12.05.2025).

32. TypeScript. URL: https://www.typescriptlang.org/ (дата звернення: 12.05.2025).

33. Tailwindcss. URL: https://tailwindcss.com/ (дата звернення: 12.05.2025).

34. Bootstrap. URL: https://getbootstrap.com/ (дата звернення: 12.05.2025).

35. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 122 «Комп’ютерні науки» [Електронний ресурс] / уклад.: А. А. Яровий, О. В. Сілагін, І. В. Богач. Вінниця : ВНТУ, 2023. (PDF, 54 с.).

ДОДАТКИ

ДОДАТОК А (обов'язковий)
ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Розробка інтернет-магазину комп'ютерної техніки на основі хмарних технологій

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра комп'ютерних наук
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 4,23 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

✓ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

Яровий А. А., зав. каф. КН _____
(прізвище, ініціали, посада) (підпис)

Колесницький О. К., проф. каф КН _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Озеранський В. С.
(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Крилик Л. В.
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Дидь К. В.
(підпис) (прізвище, ініціали)

ДОДАТОК Б (обов'язковий)

Лістинг програми

```
# === alb.tf ===
module "alb" {
  source  = "terraform-aws-modules/alb/aws"
  version = "9.11.0"

  name                       = format("%s-alb", var.project.name)
  vpc_id                     = module.vpc.vpc_id
  subnets                    = module.vpc.public_subnets
  enable_deletion_protection = false

  security_group_ingress_rules = {
    all_http = {
      from_port   = 80
      to_port     = 80
      ip_protocol = "tcp"
      description = "HTTP web traffic"
      cidr_ipv4   = "0.0.0.0/0"
    }
    all_https = {
      from_port   = 443
      to_port     = 443
      ip_protocol = "tcp"
      description = "HTTPS web traffic"
      cidr_ipv4   = "0.0.0.0/0"
    }
  }

  security_group_egress_rules = {
    all = {
      ip_protocol = "-1"
      cidr_ipv4   = "0.0.0.0/0"
    }
  }

  listeners = {
    http = {
      port     = 80
      protocol = "HTTP"
      redirect = {
        port        = "443"
        protocol    = "HTTPS"
        status_code = "HTTP_301"
      }
    }
    https = {
      port     = 443
      protocol = "HTTPS"

      certificate_arn = aws_acm_certificate.this.arn

      forward = {
        target_group_key = "frontend"
      }

      rules = {
        frontend = {
          priority = 1
```

```
        conditions = [{
          path_pattern = {
            values = ["/"]
          },
          host_header = {
            values = [aws_route53_record.this.name]
          }
        }]
        actions = [
          {
            type             = "forward"
            target_group_key = "frontend"
          }
        ]
      }
      backend = {
        priority = 2
        conditions = [{
          path_pattern = {
            values = ["/api/*"]
          },
          host_header = {
            values = [aws_route53_record.this.name]
          }
        }]
        actions = [
          {
            type             = "forward"
            target_group_key = "backend"
          }
        ]
      }
    }
  }
}

target_groups = {
  frontend = {
    create_attachment = false
    vpc_id            = module.vpc.vpc_id
    name_prefix      = "front"
    protocol         = "HTTP"
    port             = 3000
    target_type      = "ip"
    health_check = {
      path                = "/"
      interval            = 30
      timeout             = 10
      healthy_threshold   = 5
      unhealthy_threshold = 2
      matcher             = "200"
    }
  }
  backend = {
    create_attachment = false
    vpc_id            = module.vpc.vpc_id
    name_prefix      = "back"
    protocol         = "HTTP"
    port             = 8080
    target_type      = "ip"
    health_check = {
      path                = "/api/health"
      interval            = 30
      timeout             = 10
      healthy_threshold   = 5
```

```
        unhealthy_threshold = 2
        matcher             = "200"
      }
    }
  }

  tags = var.project.tags
}

# === dynamodb.tf ===
module "dynamodb_table" {
  source   = "terraform-aws-modules/dynamodb-table/aws"

  name     = format("%s-table", var.project.name)
  hash_key = "_id"

  attributes = [
    {
      name = "_id"
      type = "S"
    }
  ]

  tags = var.project.tags
}

# === ecr.tf ===
data "aws_ecr_repository" "frontend" {
  name = "tech-savvy-fe"
}

data "aws_ecr_repository" "backend" {
  name = "tech-savvy-be"
}

# === ecs.tf ===
module "ecs" {
  source = "terraform-aws-modules/ecs/aws"
  version = "5.12.0"

  cluster_name = format("%s-cluster", var.project.name)

  # Configuration for ECS Exec (optional)
  cluster_configuration = {
    execute_command_configuration = {
      logging = "OVERRIDE"
      log_configuration = {
        cloud_watch_log_group_name = "/aws/ecs/aws-ec2"
      }
    }
  }

  # Capacity Providers
  fargate_capacity_providers = {
    FARGATE = {
      default_capacity_provider_strategy = {
        weight = 100
        base   = 3
      }
    }
  }

  services = {
    frontend = {
```

```
  cpu    = 512
  memory = 1024

  container_definitions = {
    frontend = {
      cpu       = 512
      memory    = 1024
      essential = true
      image     = format("%s:latest", data.aws_ecr_repository.frontend.repository_url)
      port_mappings = [
        {
          name          = "frontend-port"
          containerPort = 3000
          hostPort      = 3000
          protocol      = "tcp"
        }
      ]
    }
  }

  enable_cloudwatch_logging = false

  load_balancer = {
    service = {
      target_group_arn = module.alb.target_groups["frontend"].arn
      container_name   = "frontend"
      container_port   = 3000
    }
  }

  subnet_ids = module.vpc.private_subnets
  security_group_ids = [ aws_security_group.frontend.id ]

}

backend = {

  cpu    = 512
  memory = 1024

  container_definitions = {
    backend = {
      cpu       = 512
      memory    = 1024
      essential = true
      image     = format("%s:latest", data.aws_ecr_repository.backend.repository_url)
      environment = [
        {
          name  = "DYNAMO_TABLE"
          value = format("%s-table", var.project.name)
        },
        {
          name  = "AWS_REGION"
          value = var.project.region
        }

      ]
      port_mappings = [
        {
          name          = "backend-port"
          containerPort = 8080
          hostPort      = 8080
          protocol      = "tcp"
        }
      ]
```

```
        }
      }

      enable_cloudwatch_logging = false

      load_balancer = {
        service = {
          target_group_arn = module.alb.target_groups["backend"].arn
          container_name   = "backend"
          container_port   = 8080
        }
      }

      tasks_iam_role_name        = "backend-tasks"
      tasks_iam_role_description = "Backend tasks IAM role"
      tasks_iam_role_policies = {
        ReadOnlyAccess = "arn:aws:iam::aws:policy/AmazonDynamoDBFullAccess"
      }
      subnet_ids = module.vpc.private_subnets
      security_group_ids = [ aws_security_group.backend.id ]

    }

  }
  depends_on = [aws_security_group.frontend, aws_security_group.backend]
  tags = var.project.tags
}

resource "aws_security_group" "frontend" {
  name        = "frontend-sg"
  description = "Security group for Frontend service"
  vpc_id      = module.vpc.vpc_id

  ingress {
    from_port   = 3000
    to_port     = 3000
    protocol    = "tcp"
    security_groups = [module.alb.security_group_id]
  }

  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }

  depends_on = [ module.alb ]
  tags = var.project.tags
}

resource "aws_security_group" "backend" {
  name        = "backend-sg"
  description = "Security group for Backend service"
  vpc_id      = module.vpc.vpc_id

  ingress {
    from_port   = 8080
    to_port     = 8080
    protocol    = "tcp"
    security_groups = [module.alb.security_group_id]
  }

  egress {
    from_port   = 0
```

```
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }

  depends_on = [ module.alb ]
  tags = var.project.tags
}

# === locals.tf ===
locals {
  name   = "ex-${basename(path.cwd)}"
  region = "eu-central-1"

  vpc_cidr = "10.0.0.0/16"
  azs      = slice(data.aws_availability_zones.available.names, 0, 3)

  tags = {
    Example    = local.name
    GithubRepo = "terraform-aws-vpc"
    GithubOrg  = "terraform-aws-modules"
  }
}

# === outputs.tf ===
output "private_subnets" {
  value = module.vpc.private_subnets
}

output "vpc_id" {
  value = module.vpc.vpc_id
}

# === providers.tf ===
terraform {
  cloud {
    organization = "**********"

    workspaces {
      name = "**********"
    }
  }
}

provider "aws" {}

data "aws_availability_zones" "available" {}

# === route53.tf ===
data "aws_route53_zone" "selected" {
  name         = "kostycloud.com"
  private_zone = false
}

resource "aws_route53_record" "this" {
  zone_id = data.aws_route53_zone.selected.zone_id
  name    = "techsavvy.kostycloud.com"
  type    = "A"
  allow_overwrite = true

  alias {
    name                   = module.alb.dns_name
    zone_id                = module.alb.zone_id
    evaluate_target_health = true
  }
```

```
}

resource "aws_acm_certificate" "this" {
  domain_name       = "*.kostycloud.com"
  validation_method = "DNS"
}

resource "aws_route53_record" "cert_validation" {
  for_each = {
    for dvo in aws_acm_certificate.this.domain_validation_options : dvo.domain_name => {
      name  = dvo.resource_record_name
      type  = dvo.resource_record_type
      value = dvo.resource_record_value
    }
  }

  zone_id = data.aws_route53_zone.selected.zone_id
  name    = each.value.name
  type    = each.value.type
  ttl     = 60
  records = [each.value.value]
}

resource "aws_acm_certificate_validation" "this" {
  certificate_arn         = aws_acm_certificate.this.arn
  validation_record_fqdns = [for record in aws_route53_record.cert_validation : record.name]
}

# === variables.tf ===
variable "project" {
  type = object(
    {
      name        = string
      region      = string
      vpc_cidr    = string
      tags        = map(any)
    }
  )
  description = "Base setting for creeating project"
  default = {
    name        = "tech-savvy"
    region      = "eu-central-1"
    vpc_cidr    = "10.0.0.0/16"
    tags = {
        Terraform   = "true"
        Environment = "dev"
    }
  }
}

# === vpc.tf ===
module "vpc" {
  source = "terraform-aws-modules/vpc/aws"
  name = format("%s-vpc", var.project.name)
  cidr = var.project.vpc_cidr
  azs = slice(data.aws_availability_zones.available.names, 0, 3)
  private_subnets  = [for k, v in local.azs : cidrsubnet(local.vpc_cidr, 8, k)]
  public_subnets   = [for k, v in local.azs : cidrsubnet(local.vpc_cidr, 8, k + 4)]
  map_public_ip_on_launch = true
  enable_nat_gateway = true
  single_nat_gateway = true
  enable_vpn_gateway = true
  tags = var.project.tags
}
```

ДОДАТОК В (обов'язковий)

ГРАФІЧНА ЧАСТИНА

«РОЗРОБКА ІНТЕРНЕТ-МАГАЗИНУ КОМП'ЮТЕРНОЇ ТЕХНІКИ НА ОСНОВІ ХМАРНИХ ТЕХНОЛОГІЙ»

Виконав: студент 4 курсу, групи 4КН-21б

спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)

Дидь К. В.
(прізвище та ініціали)

Керівник: к.т.н., доцент каф. КН

Крилик Л. В.
(прізвище та ініціали)

«03» червня 2025 р.

Вінниця ВНТУ - 2025 рік

Рисунок В.1 − Загальна структурна схема функціональних модулів

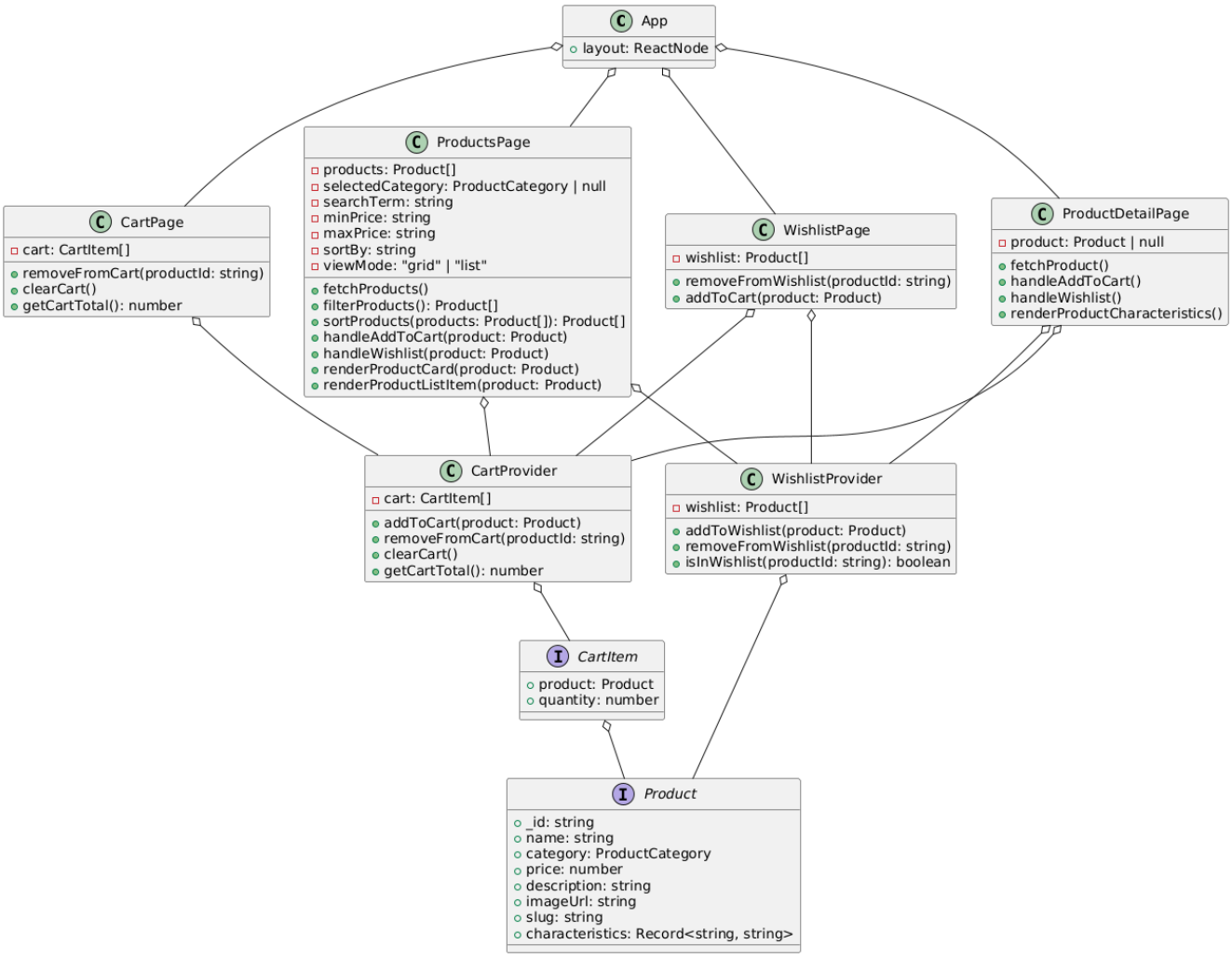

Рисунок В.2 – UML-діаграма класів фронтенд-частини

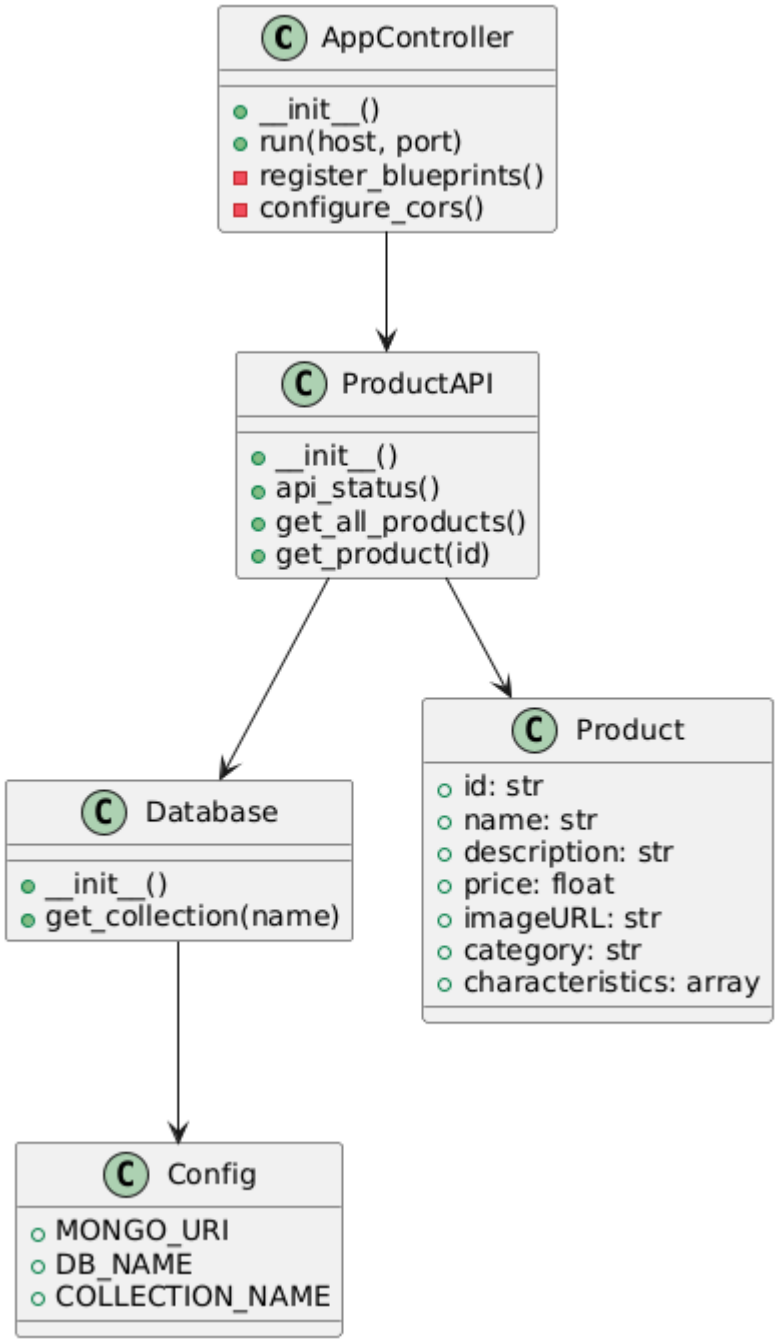

Рисунок В.3 – UML-діаграма класів мікросервісу API

Рисунок В.4 – Схема алгоритму роботи інтернет-магазину комп’ютерної техніки

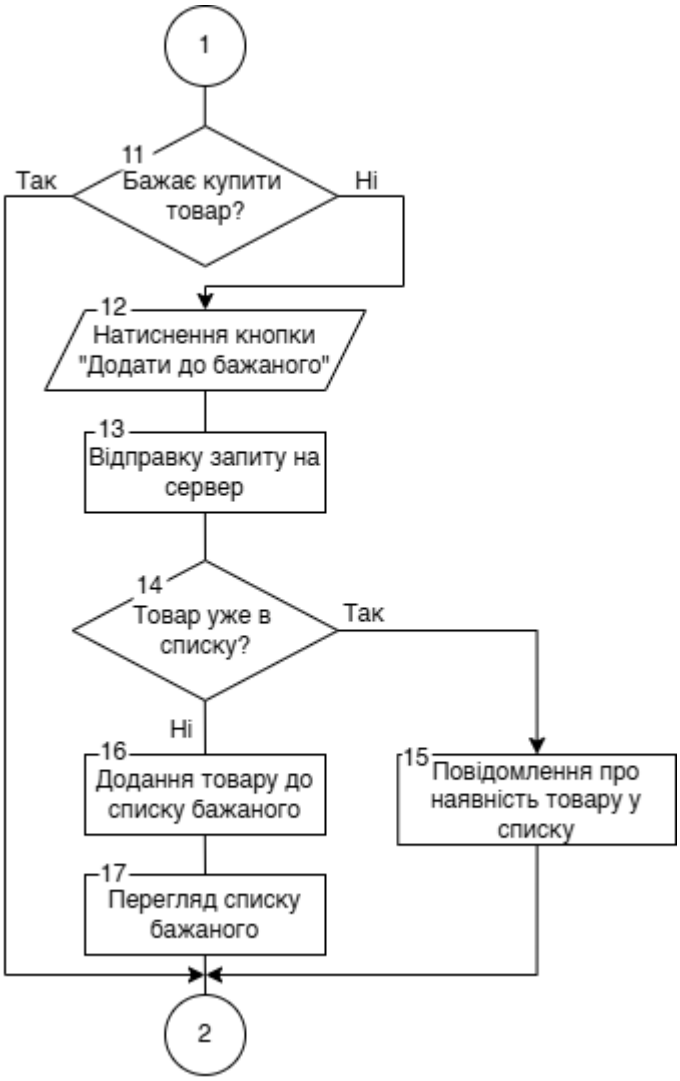

Рисунок В.4, аркуш 2

Рисунок В.4, аркуш 3

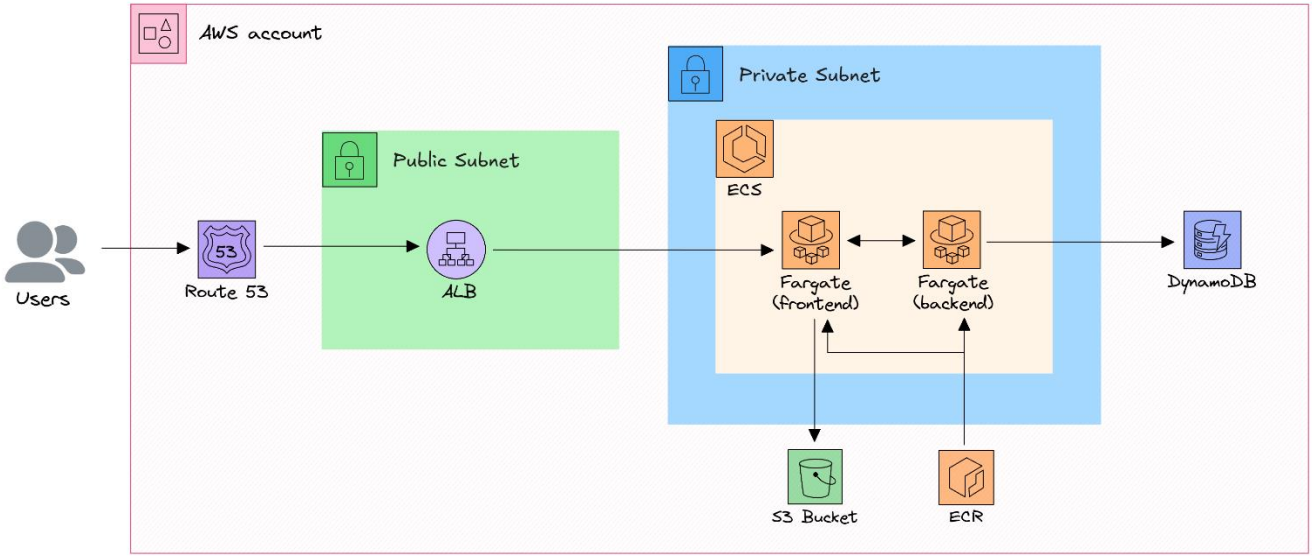

Рисунок В.5 – Архітектурна діаграма хмарного середовища інтернет-магазину комп’ютерної техніки

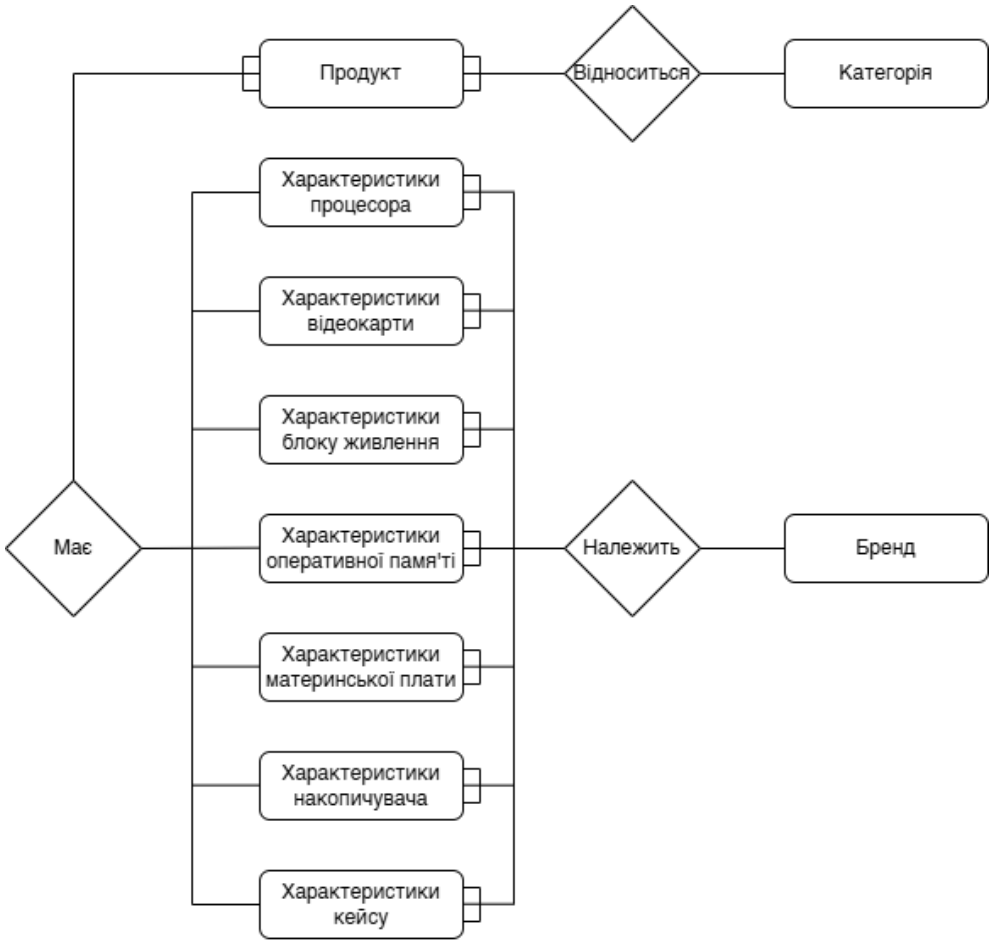

Рисунок В.6 – ER-модель предметної області «Комп’ютерна техніка»

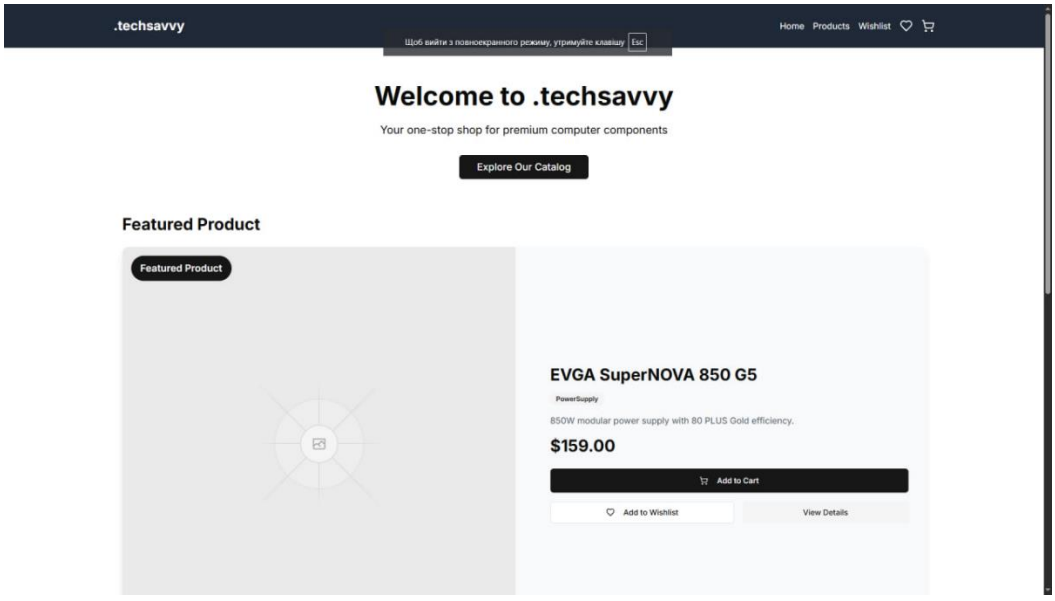

Рисунок В.7 – Загальний вигляд інтерфейсу головної сторінки розробки

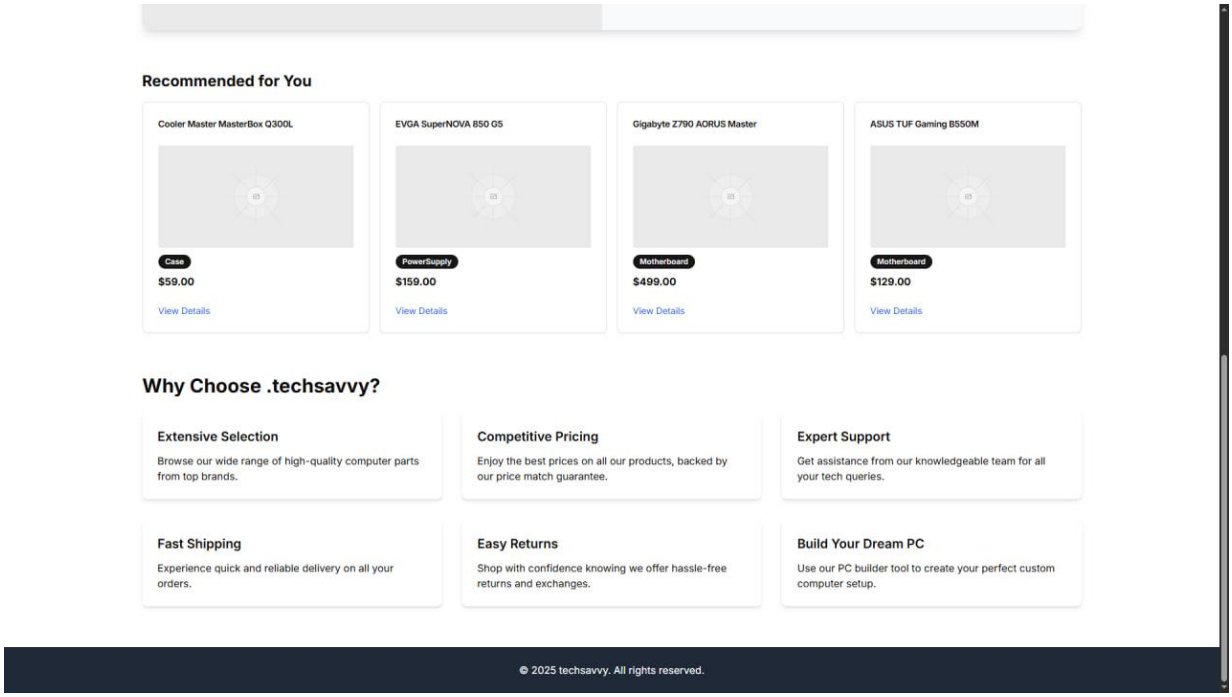

Рисунок В.8 – Загальний вигляд інтерфейсу рекомендованих товарів

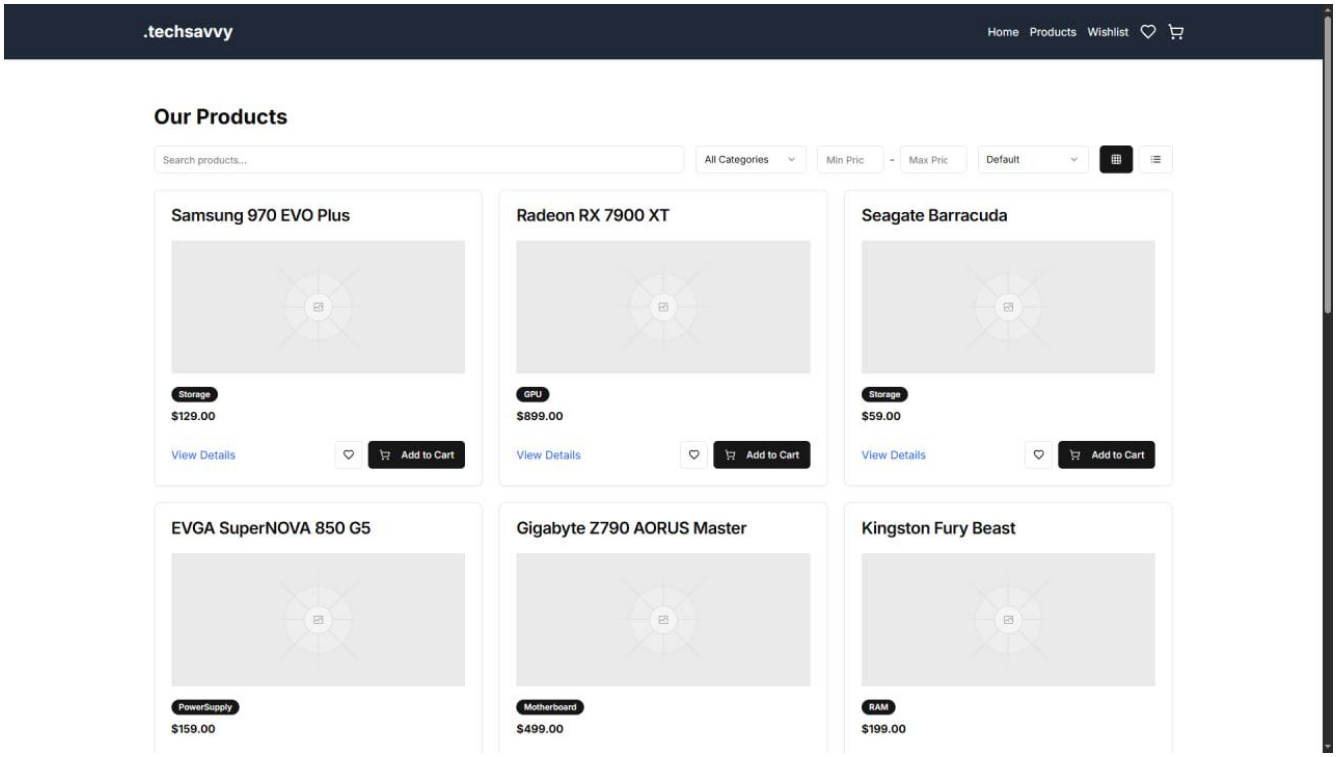

Рисунок В.9 – Загальний вигляд інтерфейсу сторінки продуктів

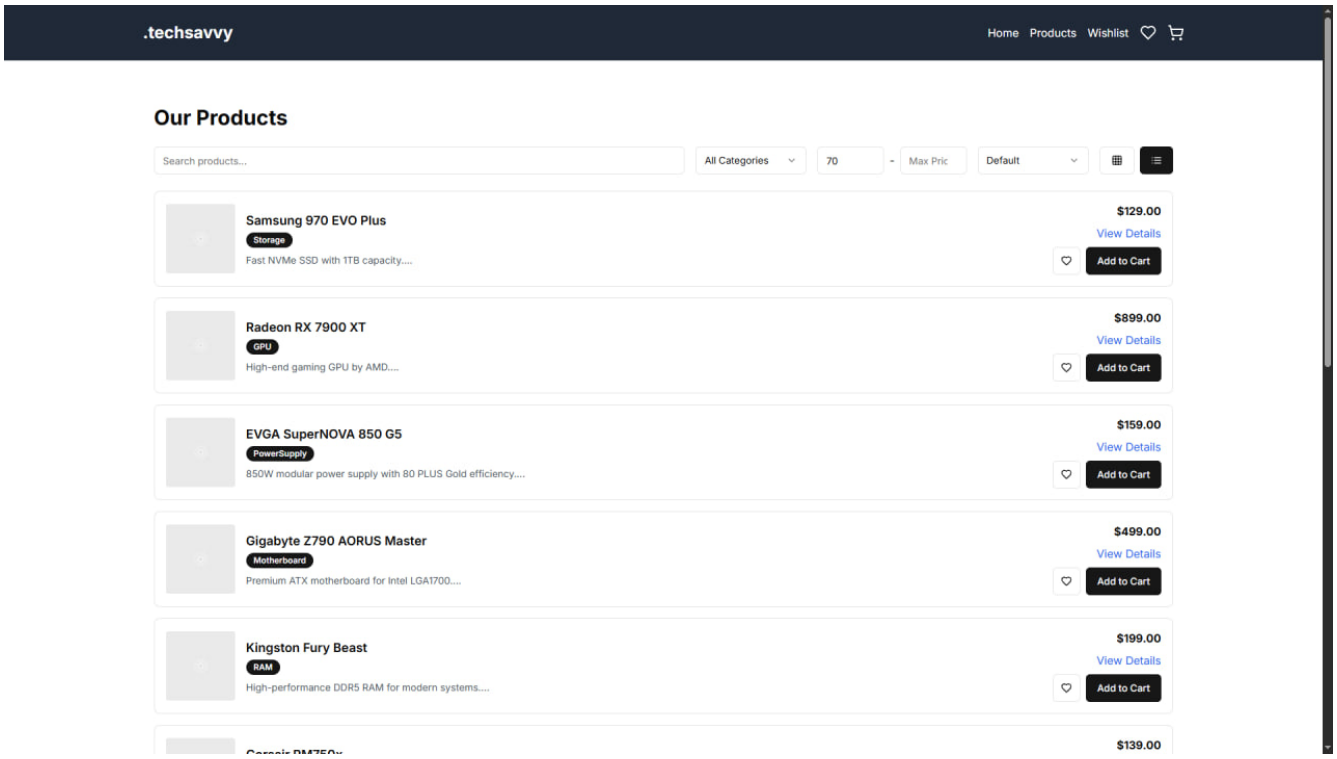

Рисунок В.10 – Загальний вигляд режиму відображення «список»

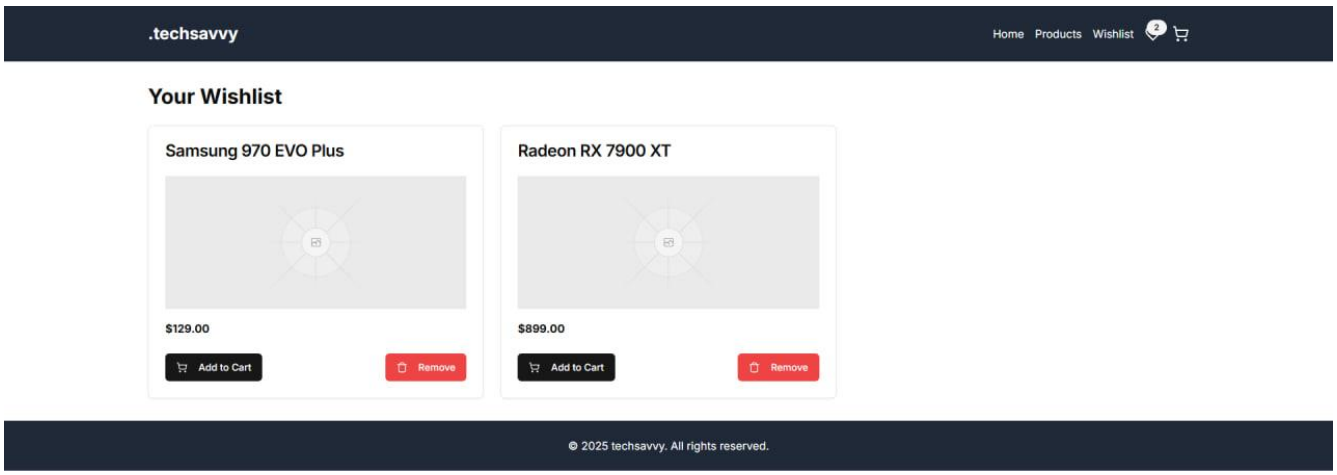

Рисунок В.11 – Загальний вигляд інтерфейсу сторінки списку бажаного

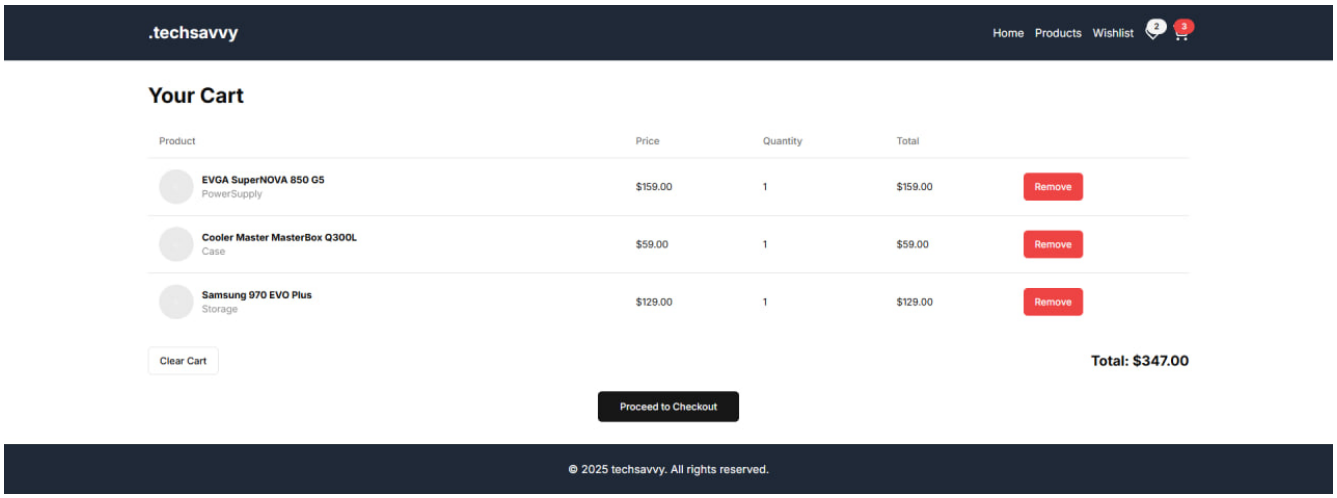

Рисунок В.12 – Загальний вигляд інтерфейсу сторінки перегляду кошика

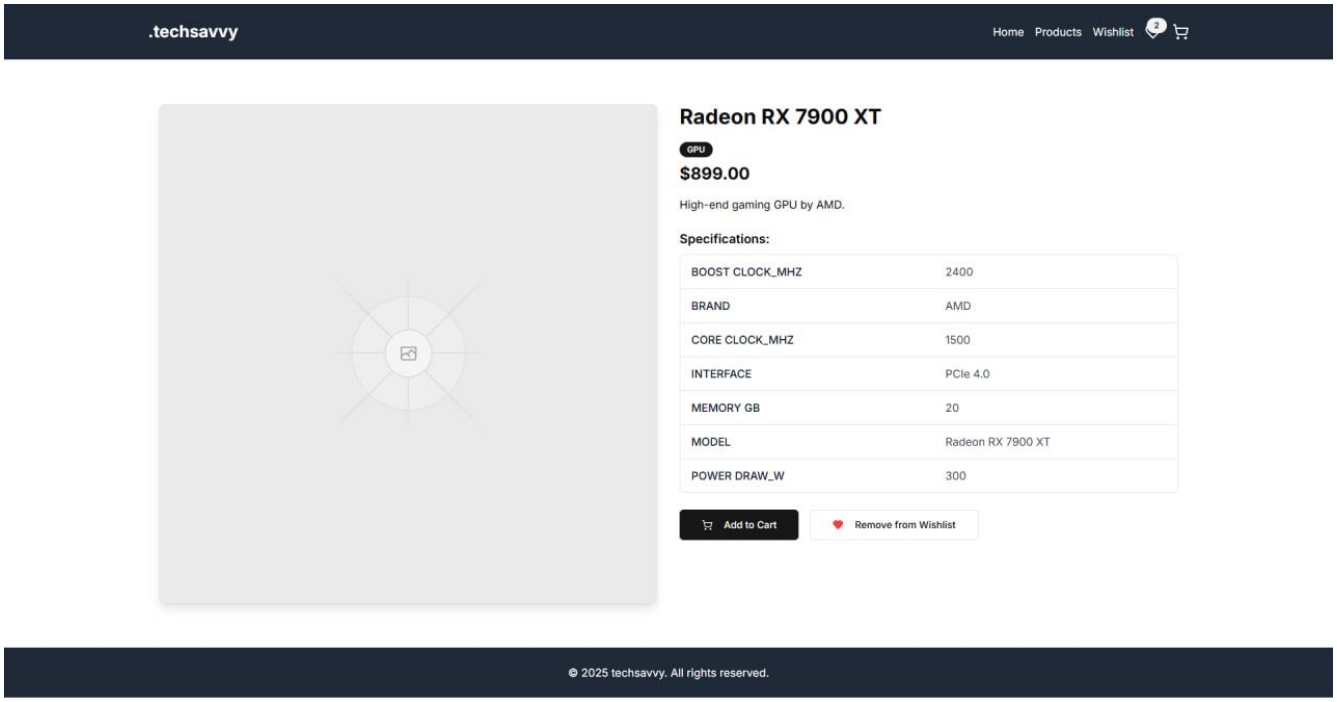

Рисунок В.13 – Загальний вигляд інтерфейсу перегляду продукту

ДОДАТОК Г (довідниковий)
Інструкція користувача

Після запуску WEB-застосунку користувач потрапляє на головну сторінку інтернет-магазину, де відображаються рекомендовані товари, а також меню навігації по сайту. Загальний вигляд головної сторінки показано на рисунку Г.1.

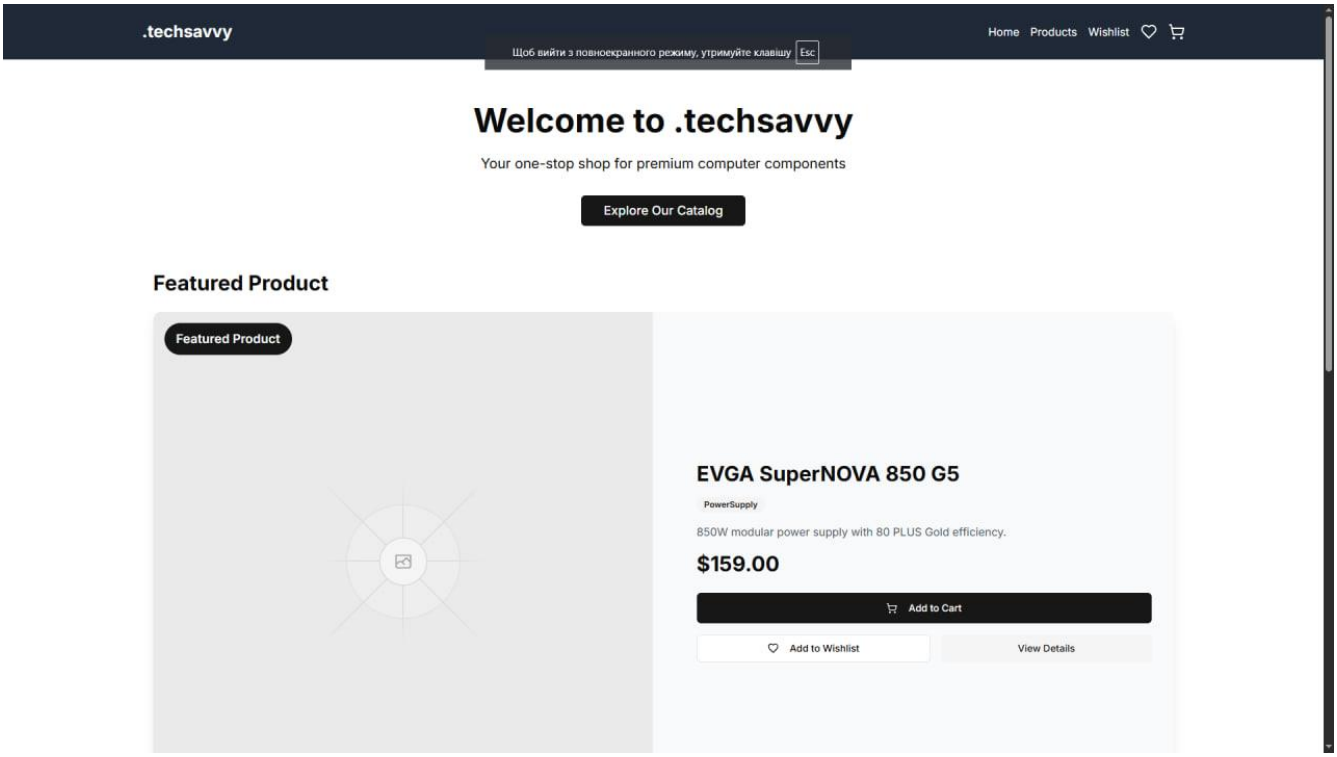

Рисунок Г.1 – Загальний вигляд головної сторінки інтернет-магазину

Перейшовши до розділу «Products», користувач отримує доступ до повного каталогу товарів. На сторінці реалізовано пошук, фільтрацію за категоріями, цінами та брендами, а також можливість перемикання між двома режимами перегляду: у вигляді плитки або списку. На рисунку Г.2 зображено сторінку з результатами фільтрації за мінімальною ціною $70.

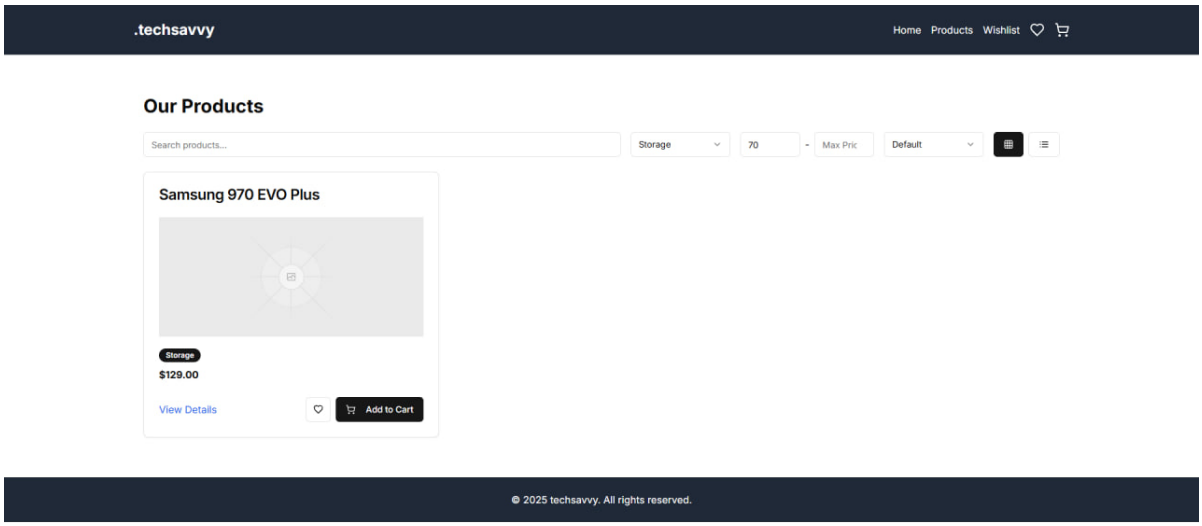

Рисунок Г.2 – Сторінка каталогу товарів після фільтрації за ціною

Для зручності користувача передбачено два режими перегляду товарів – «плитка» та «список». На рисунку Г.3 зображено режим перегляду у форматі плитки, а на рисунку Г.4 – у форматі списку.

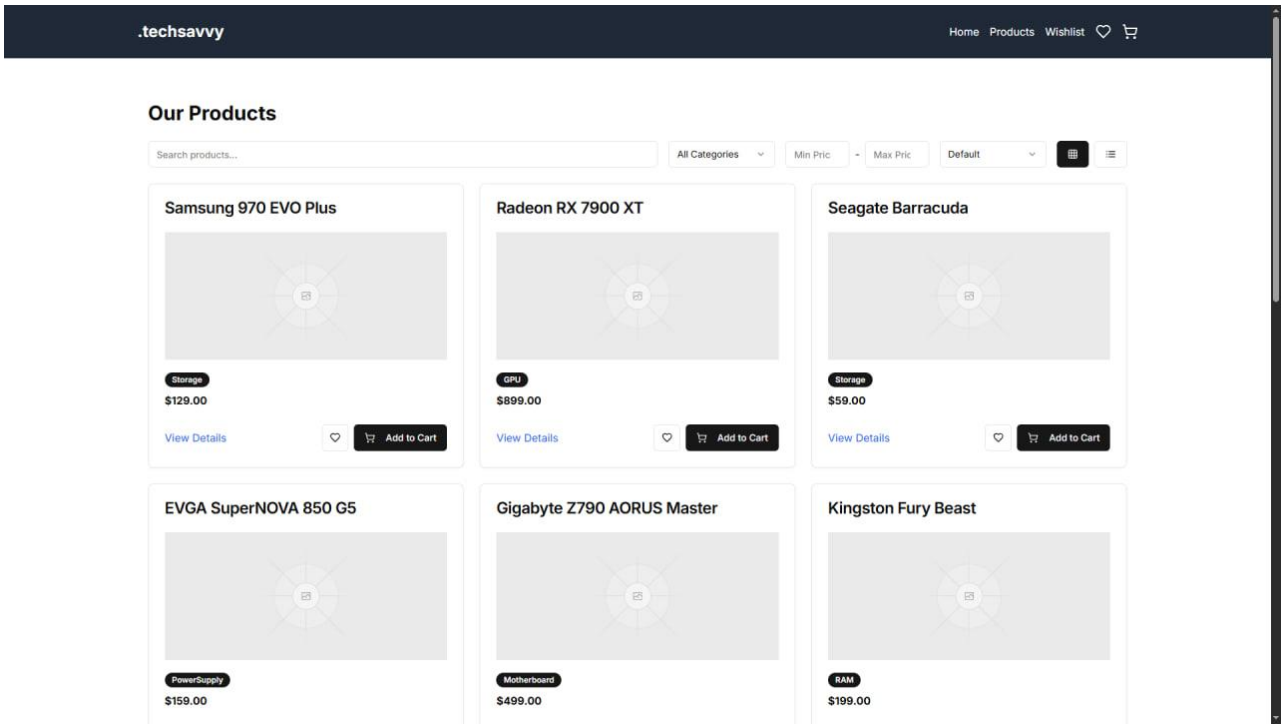

Рисунок Г.3 – Відображення товарів у вигляді плитки

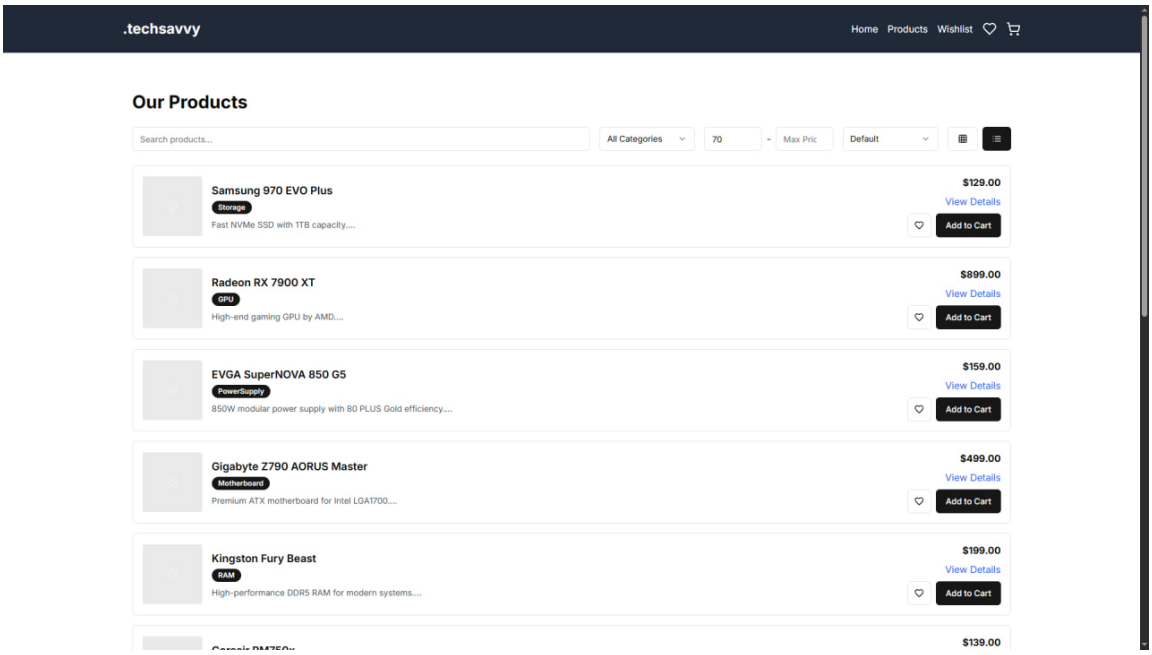

Рисунок Г.4 – Відображення товарів у вигляді списку

При натисканні кнопки «View Details» на картці товару відкривається сторінка з повною інформацією про продукт: назва, опис, технічні характеристики, ціна, категорія. Користувач також має можливість додати товар до кошика або списку бажаного. Приклад такої сторінки показано на рисунку Г.5.

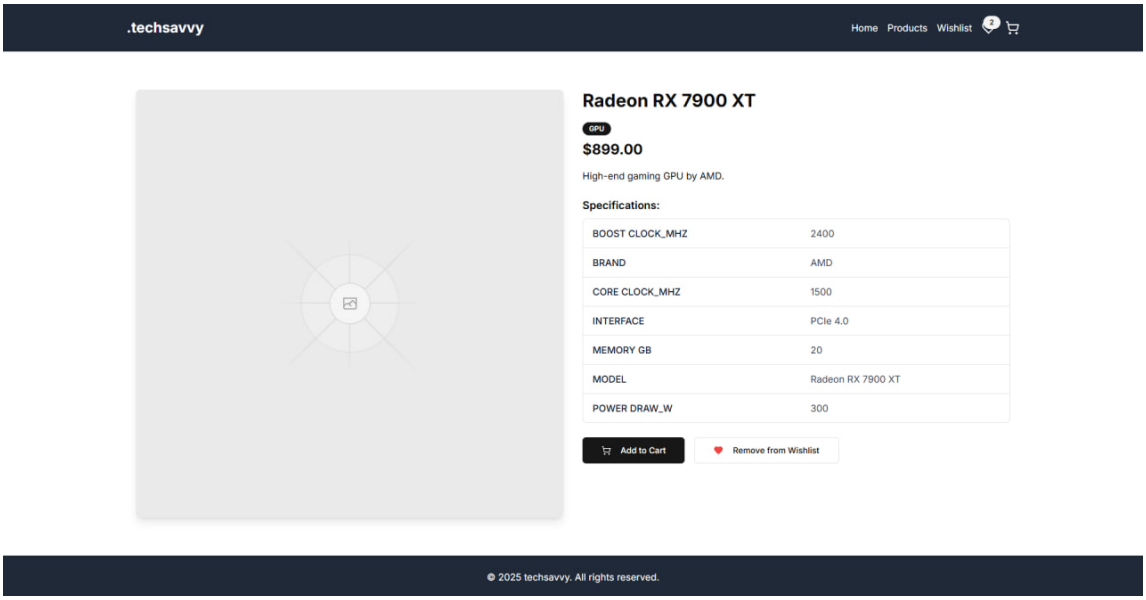

Рисунок Г.5 – Сторінка з детальною інформацією про товар

На сторінці каталогу або продукту користувач може додавати товари до списку бажаного за допомогою іконки «серце». Додані товари автоматично зберігаються в локальному сховищі браузера. На рисунку Г.6 показано процес додавання до списку бажаного, а на рисунку Г.7 – вигляд списку бажаного з товарами.

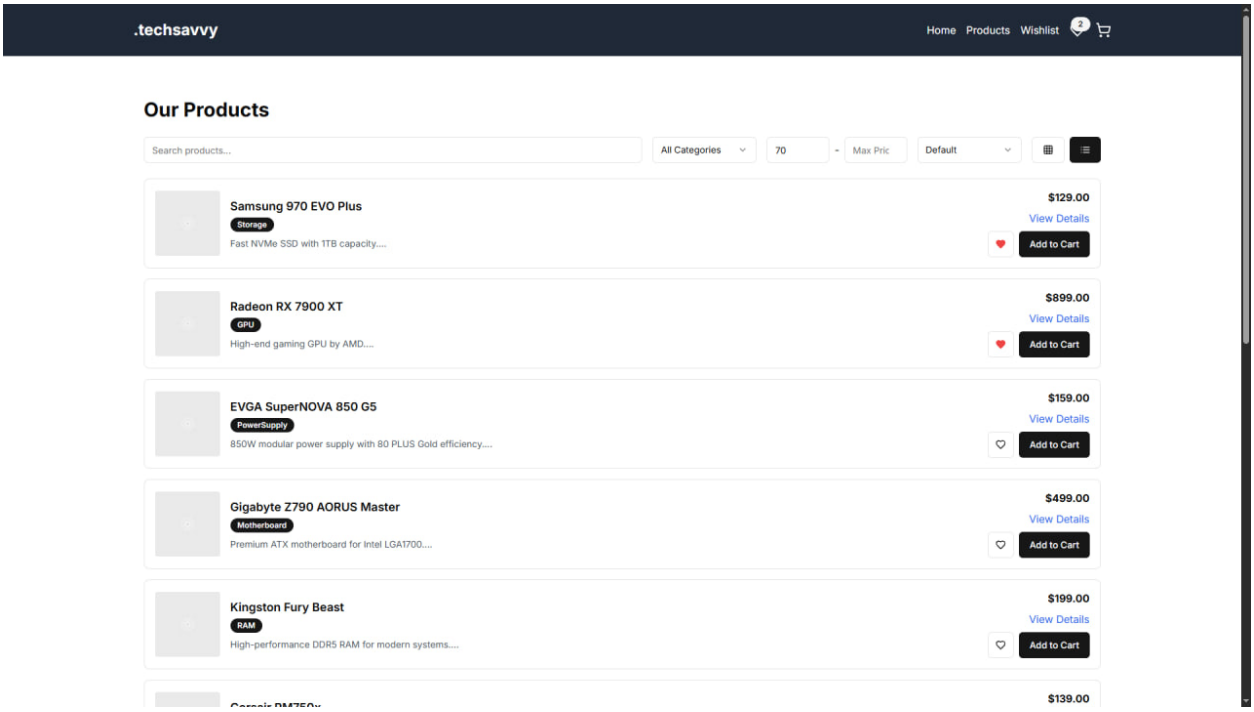

Рисунок Г.6 – Додавання товарів у список бажаного

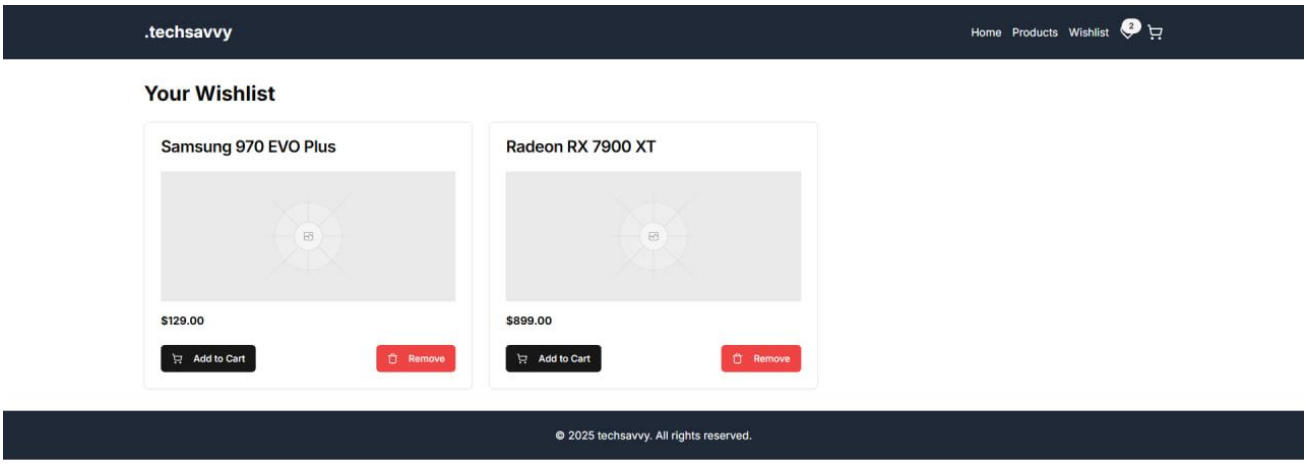

Рисунок Г.7 – Список бажаного з доданими товарами

Користувач може додавати товари до кошика, натиснувши кнопку «Add to Cart». Усі вибрані товари зберігаються у розділі «Cart», де доступні функції редагування кількості, видалення товару або очищення всього кошика. На рисунку Г.8 показано процес додавання товару до кошика, а на рисунку Г.9 – вигляд самого кошика із підсумковою вартістю.

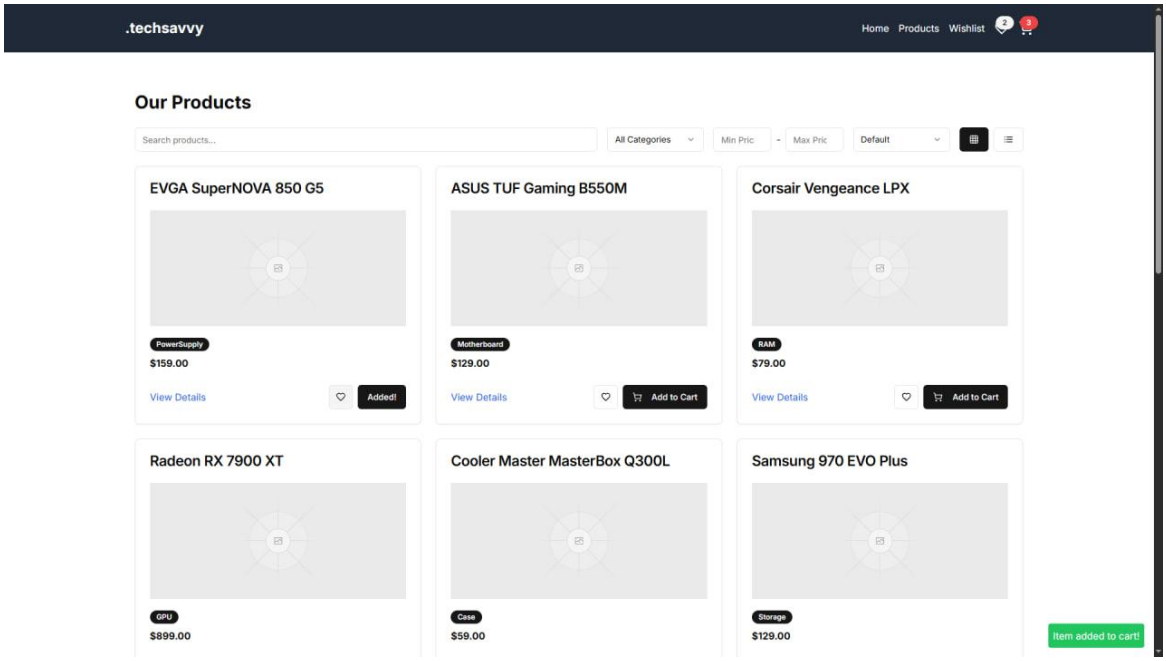

Рисунок Г.8 – Додавання товарів до кошика зі сторінки каталогу

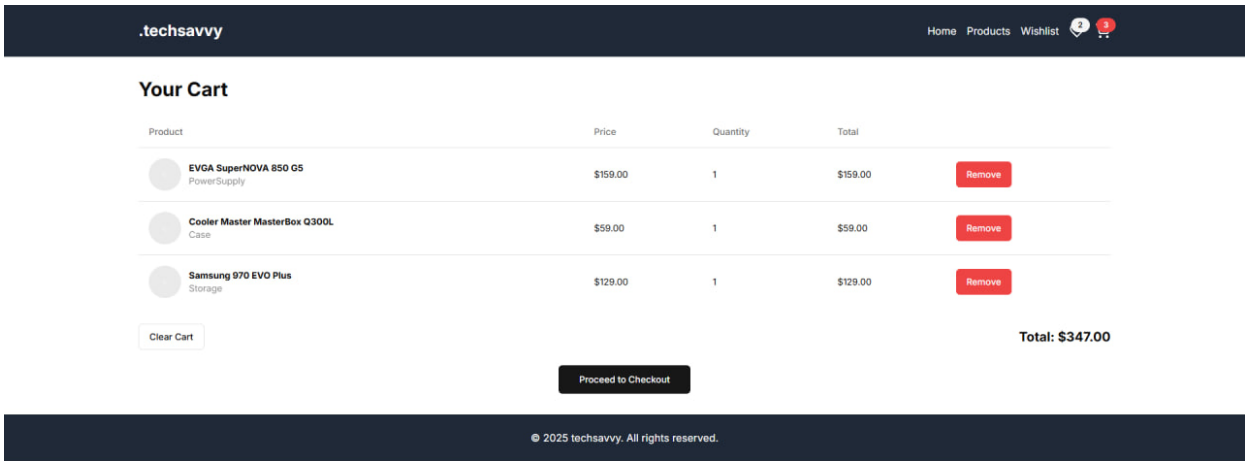

Рисунок Г.9 – Вигляд кошика з доданими товарами