

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавр

(ступінь вищої освіти)

на тему СТВОРЕННЯ ПРОГРАМНОЇ СИСТЕМИ НА ОСНОВІ ХМАРНИХ
ОБЧИСЛЕНЬ ДЛЯ ВІЗУАЛІЗАЦІЇ ТА ДИСТРИБУЦІЇ ТОВАРІВ З
ВИКОРИСТАННЯМ ГЕОІНФОРМАЦІЙНИХ СЕРВІСІВ
DESIGN OF A CLOUD-BASED SOFTWARE SYSTEM FOR VISUALIZATION
AND DISTRIBUTION OF GOODS USING GEOINFORMATION SERVICES

Виконав(ла): студент(ка) 4 курсу, групи КНТ-212

Спеціальності 122 Комп'ютерні науки

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Комп'ютерні науки

КУЧЕР І.А.

(ПРИЗВИЩЕ та ініціали)

Керівник ГЛАДКОВА О.М.

(ПРИЗВИЩЕ та ініціали)

Рецензент ГОЛУБ Т.В.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування закладу вищої освіти)

Факультет КНТ
 Кафедра програмних засобів
 Ступінь вищої освіти бакалавр
 Спеціальність 122 Комп'ютерні науки
(код і найменування)
 Освітня програма (спеціалізація) Комп'ютерні науки
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
 “ ” 2026 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

КУЧЕРА Івана Андрійовича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Створення програмної системи на основі хмарних обчислень для візуалізації та дистрибуції товарів з використанням геоінформаційних сервісів. Design of a Cloud-Based Software System for Visualization and Distribution of Goods Using Geoinformation Services

керівник проєкту (роботи) к.т.н., доцент, ГЛАДКОВА Ольга Миколаївна,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “ 07 ” квітня 2026 року № 139

2. Строк подання студентом проєкту (роботи) 10 червня 2026 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Обґрунтування вибору засобів розробки. 3. Розробка вебзастосунку. 4. Аналіз результатів та використання застосунку.

5. Перелік графічного матеріалу (з з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ГЛАДКОВА О.М., доцент		
Нормоконтроль	ДЕЙНЕГА Л.Ю., ст.викладач		

7. Дата видачі завдання « 20 » квітня 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	2 тиждень	Розділ 2
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

(підпис) Іван КУЧЕР
(Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

(підпис) Ольга ГЛАДКОВА
(Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 91 с., 7 табл., 23 рис., 2 дод., 13 джерел.

ВЕБ ЗАСТОСУНОК, ІНТЕГРАЦІЇ, GOOGLE FIREBASE, REACTJS, LEAFLET, ГЕОІНФОРМАЦІЯ, ХМАРНІ ТЕХНОЛОГІЇ.

Об'єкт дослідження – моделі, методи, алгоритми та програмні засоби створення вебзастосунків із використанням геоінформаційних і хмарних технологій.

Предмет дослідження – методи та програмні засоби інтеграції геоінформаційних сервісів і хмарних платформ у вебзастосунках.

Мета роботи – підвищення ефективності управління розподілом товарів та оптимізація їх візуалізації шляхом розробки хмарного вебзастосунку з інтеграцією геоінформаційних сервісів.

Матеріали, методи та технічні засоби: HTML5, CSS3, JavaScript, бібліотека ReactJS. Просторові дані обробляються за допомогою Leaflet (OpenStreetMap), хмарна інфраструктура на основі Google Firebase. Програмно-апаратна база: ОС Windows 11 Pro, процесор AMD Ryzen 5.

Результати. Створено функціональний клієнтський вебзастосунок з використанням бібліотеки ReactJS та екосистеми Node.js.

Висновки. Розроблений клієнтський вебзастосунок забезпечує зручне сортування та перегляд товарів каталогу. Впровадження геоінформаційних сервісів та гнучкої хмарної архітектури підвищує загальну продуктивність системи та зручність її використання.

Галузь використання – сфера логістики, електронна комерція та сервіси управління розподілом для клієнтського користування.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 91 pages, 7 tables, 23 figures, 2 appendixes, 13 sources.

WEB APPLICATIONS, INTEGRATIONS, GOOGLE FIREBASE, REACTJS, LEAFLET, GEOINFORMATION, CLOUD TECHNOLOGIES.

The object of research is models, methods, algorithms and software for creating web applications using geoinformation and cloud technologies.

The subject of research is methods and software for integrating geoinformation services and cloud platforms in web applications.

The purpose of the work is to improve the efficiency of product distribution management and optimize their visualization by developing a cloud web application with the integration of geoinformation services.

Materials, methods and technical means: HTML5, CSS3, JavaScript, ReactJS library. Spatial data is processed using Leaflet (OpenStreetMap), cloud infrastructure based on Google Firebase. Software and hardware base: Windows 11 Pro OS, AMD Ryzen 5 processor.

Results. A functional client web application was created using the ReactJS library and the Node.js ecosystem.

Conclusions. The developed client web application provides convenient sorting and viewing of catalog products. The implementation of geo-information services and flexible cloud architecture increases the overall performance of the system and its ease of use.

The field of use is logistics, e-commerce and distribution management services for client use.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	9
Вступ.....	10
1 Аналіз предметної області.....	11
1.1 Особливості геоінформаційних технологій та хмарних сервісів	11
1.2 Хмарні обчислення.....	11
1.3 Геоінформаційна система Leaflet.....	12
1.4 Хмарний сервіс Google Firebase.....	12
1.5 Бібліотека ReactJS та тенденції веб-розробки.....	13
1.6 Інструмент Vite.....	14
2 Обґрунтування вибору засобів розробки.....	15
2.1 ReactJS та архітектура на його основі.....	15
2.2 Firebase сервіс та його особливості	17
2.3 Інструмент Vite та його особливості.....	19
2.4 Контроль версій Git та GitHub	20
3 Розробка вебзастосунку.....	22
3.1 Вибір та обґрунтування технологічного стека та архітектури	22
3.1.1 Концепція Single Page Application (SPA).....	23
3.1.2 Ініціалізація точки входу.....	23
3.2 Побудова системи навігації та глобального стану.....	24
3.2.1 Маршрутизація за допомогою React Router.....	24
3.2.2 Розробка адаптивної навігаційної панелі.....	25
3.3 Інтеграція хмарної інфраструктури Firebase	25
3.3.1 Конфігурація сервісів.....	26
3.3.2 Управління автентифікацією та безпекою.....	26
3.4 Проєктування бази даних та генерація тестових даних	27
3.4.1 Автоматизація наповнення бази даних (DBSeeder).....	27
3.4.2 Структура бази даних Firestore	28

3.5 Розробка каталогу товарів та системи пошуку.....	28
3.5.1 Патерн Lifting State Up.....	29
3.5.2 Оптимізація рендерингу.....	29
3.6 Персоналізація, управління списком обраного.....	30
3.7 Геоінформаційний модуль та візуалізація логістики.....	31
3.7.1 Ініціалізація карти та позиціонування.....	31
3.7.2 Побудова та розрахунок маршрутів.....	31
3.8 Асинхронна взаємодія та оптимізація потоків даних.....	32
3.8.1 Застосування патерну async / await.....	33
3.8.2 Управління життєвим циклом асинхронних подій у React.....	33
3.8.3 Паралельне виконання мережових запитів.....	34
3.9 Обробка виняткових ситуацій та оптимізація UI/UX.....	35
3.10 Висновки за розділом 3.....	35
4 Аналіз результатів та використання застосунку.....	37
4.1 Результати роботи.....	37
4.2 Методика та стратегія тестування системи.....	40
4.2.1 Функціональне тестування.....	41
4.2.2 Інтеграційне тестування з Firebase.....	41
4.3 Аналіз результатів тестування та продуктивності.....	41
4.3.1 Показники швидкодії.....	41
4.3.2 Валідація геоінформаційної складової.....	42
4.4 Виявлені дефекти та запропоновані технічні рішення.....	42
4.4.1 Оптимізація запитів до Firestore.....	42
4.4.2 Покращення валідації на рівні бази даних.....	43
4.5 Перспективи масштабування та вдосконалення системи.....	43
4.5.1 Технічне масштабування.....	43
4.5.2 Функціональний розвиток.....	44
4.6 Інструкція використання системи.....	44
4.6.1 Налаштування середовища розробки.....	44
4.6.2 Порядок роботи користувача.....	44

4.7 Висновки за розділом 4.....	45
Висновки	46
Перелік джерел посилення	47
Додаток А Текст програми.....	49
Додаток Б Слайди презентації.....	85

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API – Application Programming Interface;
BaaS – Backend-as-a-Service;
CDN – Content Delivery Network;
CSS – Cascading Style Sheets;
DOM – Document Object Model;
ES – ECMAScript;
Git – розподілена система керування версіями;
HMR – Hot Module Replacement;
HTML – HyperText Markup Language;
IaaS – Infrastructure-as-a-Service;
ID – Identifier (ідентифікатор);
IDE – Integrated Development Environment;
JS – JavaScript;
JSON – JavaScript Object Notation;
MPA – Multi-Page Application;
NoSQL – Not Only SQL;
OSRM – Open Source Routing Machine;
PaaS – Platform-as-a-Service;
SaaS – Software-as-a-Service;
SPA – Single Page Application;
UI – User Interface;
UX – User Experience;
VSCode – Visual Studio Code;
БД – база даних;
ГІС – геоінформаційна система.

ВСТУП

Зазвичай при використанні при використанні сайтів для логістики, або для приватних організацій та процесів відстеження. Але у сучасну цифрову епоху співпраця між ГІС, вебзастосунки та бізнесом стала ключовим рушієм інновацій. Географічні інформаційні системи (ГІС) дозволяють візуалізувати бізнес-дані просторово, що робить аналіз та прийняття рішень ефективнішими. Наприклад, компанії можуть відображати дані про продажі або демографічні дані клієнтів на інтерактивних картах, щоб визначати просторові тенденції та моделі ефективності.

Водночас вебзастосунки та цифрові платформи розширюють охоплення ринку та сприяють взаємодії з клієнтами через електронну комерцію, маркетинговий контент та цифрові послуги. У поєднанні ГІС як джерело інформації про місцезнаходження, вебзастосунки як цифрові канали розповсюдження та бізнес-моделі, що базуються на даних, ці елементи значно покращують операційну ефективність та дозволяють реалізовувати більш цілеспрямовані маркетингові стратегії [1].

Сучасний маркетинг також сильно залежить від онлайн-даних. Відгуки клієнтів у пошукових системах та соціальних мережах відіграють вирішальну роль у формуванні довіри та впливі на рішення про покупку. Як результат, ефективне управління вебзастосунком та оптимізація локального пошуку стали важливими бізнес-можливостями. У поєднанні з ГІС, такими як функції пошуку магазинів або інтерактивні карти клієнтів, веб-платформи можуть ще більше підвищити залученість користувачів, забезпечуючи досвід, що залежить від місцезнаходження [1].

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Особливості геоінформаційних технологій та хмарних сервісів

Веб-ГІС – це будь-який метод доступу, аналізу та відображення геопросторових даних за допомогою стандартних веб-технологій. Це включає стандартні клієнтські технології, такі як HTML, CSS та JavaScript. На стороні сервера вона може включати кілька мов програмування, що використовуються для взаємодії з базою даних. Найпоширенішою є PHP, але вона також може включати Java, Python, Microsoft Asp.NET, Perl, Ruby та навіть більше JavaScript за допомогою Node.JS [2].

Веб-розробка охоплює створення та підтримку веб-сайтів і веб-додатків, тоді як хмарні обчислення пропонують масштабовану інфраструктуру для розміщення та управління цифровими послугами. Хмарні обчислення відіграють вирішальну роль у веб-розробці, пропонуючи економічну ефективність та підвищену продуктивність. Наприклад, такі компанії, як Netflix, використовують хмарні сервіси для обробки мільйонів одночасних потоків, динамічно адаптуючи ресурси на основі попиту користувачів, щоб забезпечити безперебійне надання послуг [3].

1.2 Хмарні обчислення

Хмарні обчислення, які часто називають «хмарою», революціонізували спосіб управління, розгортання та доступу до цифрових послуг. Простіше кажучи, хмарні обчислення пропонують різні обчислювальні послуги, такі як сховище, обчислювальна потужність та бази даних, через Інтернет. Завдяки такому підходу організаціям не потрібне фізичне обладнання, оскільки вони можуть використовувати віддалені сервери, що надаються постачальниками хмарних послуг. Ця технологія має вирішальне значення для забезпечення доступності та адаптивності сучасної веб-розробки [3]. Зокрема, використання хмарних платформ дозволяє

реалізувати такі моделі обслуговування, як IaaS (інфраструктура як послуга), PaaS (платформа як послуга) та SaaS (програмне забезпечення як послуга). Це забезпечує автоматичне масштабування обчислювальних ресурсів залежно від поточного навантаження на інформаційну систему.

1.3 Геоінформаційна система Leaflet

Leaflet – це бібліотека JavaScript, яка використовується для створення веб-картографічних програм. Вона дозволяє розробникам відображати плиткові веб-карти, розміщені на публічному сервері, з додатковими плитковими накладаннями. Вона може завантажувати дані про об'єкти з файлів GeoJSON, стилізувати їх та створювати інтерактивні шари, такі як маркери зі спливаючими вікнами при натисканні [4].

Leaflet розроблено з урахуванням простоти, продуктивності та зручності використання. Він ефективно працює на всіх основних платформах для настільних комп'ютерів і мобільних пристроїв, може бути розширений за допомогою безлічі плагінів, має гарний, простий у використанні та добре документований API, а також простий, читабельний вихідний код [5].

Завдяки своїй легковажності та модульній структурі, ця бібліотека безперешкодно інтегрується із сучасними компонентними веб-фреймворками (зокрема, з React через спеціалізовані обгортки), що робить її оптимальним інструментом для розробки геоінформаційних веб-додатків, систем моніторингу логістики та візуалізації процесів розподілу товарів.

1.4 Хмарний сервіс Google Firebase

Firebase – це компанія, яка розробляла програмне забезпечення для серверних систем. Вона була заснована в Сан-Франциско в 2011 році та зареєстрована в штаті Делавер.

У 2014 році Firebase була придбана компанією Google. Її назва продовжується як набір хмарних обчислювальних сервісів серверних систем та платформ розробки додатків, що надаються Google. Вона розміщує бази даних, сервіси, автентифікацію та інтеграцію для різноманітних додатків, включаючи Android, iOS, JavaScript, Node.js, Java, Unity, PHP та C++ [6].

Особливу цінність для побудови сучасних веб-додатків становлять сервіси Firebase Realtime Database та Cloud Firestore, які забезпечують двосторонню синхронізацію даних між клієнтом і сервером у режимі реального часу через протокол WebSockets. Крім того, вбудована підсистема Firebase Authentication дозволяє швидко реалізувати безпечну авторизацію користувачів, а Firebase Hosting забезпечує надійне розгортання статичних ресурсів через глобальну мережу доставки контенту (CDN).

1.5 Бібліотека ReactJS та тенденції веб-розробки

Розробка фронтенду (або розробка на стороні клієнта). Ця галузь розробки включає створення частин веб-сайту, з якими користувачі взаємодіють безпосередньо. Вона використовує такі технології, як HTML, CSS та JavaScript, для відображення всього контенту, який користувач веб-сайту або редактор контенту в системі керування контентом зник бачити – зображення, текст, кнопки, функціональність користувача та навігацію тощо [7].

React (також відомий як React.js або ReactJS) – це безкоштовна бібліотека JavaScript з відкритим вихідним кодом, яка має на меті зробити створення користувацьких інтерфейсів на основі компонентів більш «безшовним». Вона підтримується Meta (раніше Facebook) та спільнотою окремих розробників і компаній. Згідно з опитуванням розробників Stack Overflow за 2025 рік, React є однією з найбільш поширених веб-технологій.

React можна використовувати для розробки односторінкових, мобільних або серверних додатків за допомогою фреймворків, таких як

Next.js та React Router. Оскільки React займається лише користувацьким інтерфейсом та візуалізацією компонентів у DOM, додатки React часто покладаються на бібліотеки для маршрутизації та інших функцій на стороні клієнта. Ключовою перевагою React є те, що він повторно візуалізує лише ті частини сторінки, які змінилися, уникаючи непотрібного повторного візуалізації незмінних елементів DOM. React використовується приблизно 6% усіх вебсайтів [8].

1.6 Інструмент Vite

Зі зростанням розміру та складності веб-додатків інструменти, що використовуються для їх створення, ледве встигають за ними. Розробники, що працюють над великими проектами, стикаються з надзвичайно повільними запуском серверів розробки, повільними гарячими оновленнями та тривалим часом збірки у виробничій середі. Кожне покоління інструментів для збірки покращувалося порівняно з попереднім, але ці проблеми залишаються.

Vite був створений для вирішення цієї проблеми. Замість того, щоб поступово вдосконалювати існуючі підходи, він переосмислив, як код має обслуговуватися під час розробки. Відтоді Vite пройшов кілька основних версій, щоразу адаптуючись до нових можливостей екосистеми: від використання власних модулів ES у браузері до впровадження повністю заснованого на Rust інструментарію.

Сьогодні Vite працює на багатьох фреймворках та інструментах. Його архітектура розроблена для розвитку разом з веб-платформою, а не закріплена за якимось одним підходом, що робить її основою, на якій можна будувати в довгостроковій перспективі [9].

2 ОБГРУНТУВАННЯ ВИБОРУ ЗАСОБІВ РОЗРОБКИ

2.1 ReactJS та архітектура на його основі

ReactJS є сучасною JavaScript-бібліотекою для створення користувацьких інтерфейсів вебзастосунків. Основною особливістю React є компонентно-орієнтований підхід, відповідно до якого інтерфейс поділяється на незалежні компоненти, що можуть повторно використовуватись у різних частинах системи (рис. 2.1). Такий підхід значно спрощує підтримку, масштабування та подальший розвиток програмного забезпечення. ReactJS забезпечує декларативний спосіб опису інтерфейсу, завдяки чому розробник визначає кінцевий стан UI, а бібліотека автоматично оновлює необхідні елементи сторінки.

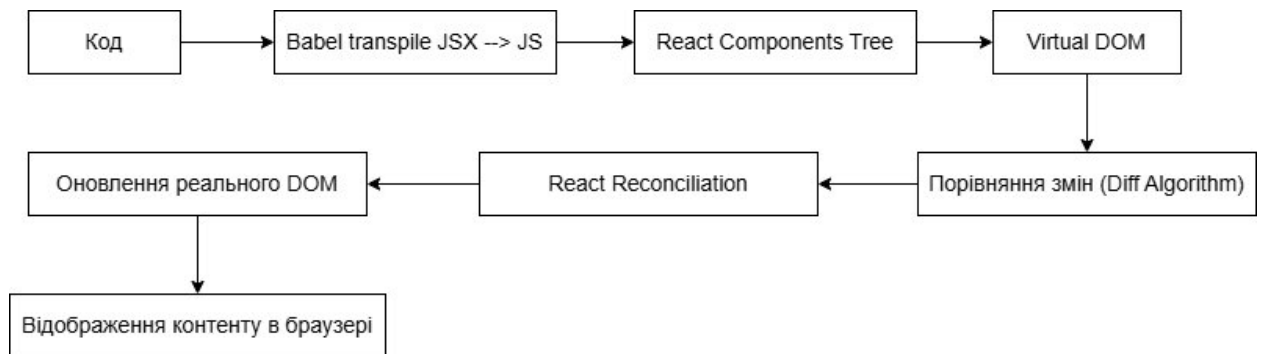


Рисунок 2.1 – Діаграма роботи бібліотеки React

Односторінковий додаток (SPA) – це вебзастосунок, який взаємодіє з користувачем, динамічно переписуючи поточну веб-сторінку з новими даними з веб-сервера, замість стандартного методу завантаження цілих нових сторінок. Метою є швидші переходи, які роблять веб-сайт більш схожим на нативний додаток.

Приклад діаграми структури бібліотеки Router зображено на рисунку 2.2.

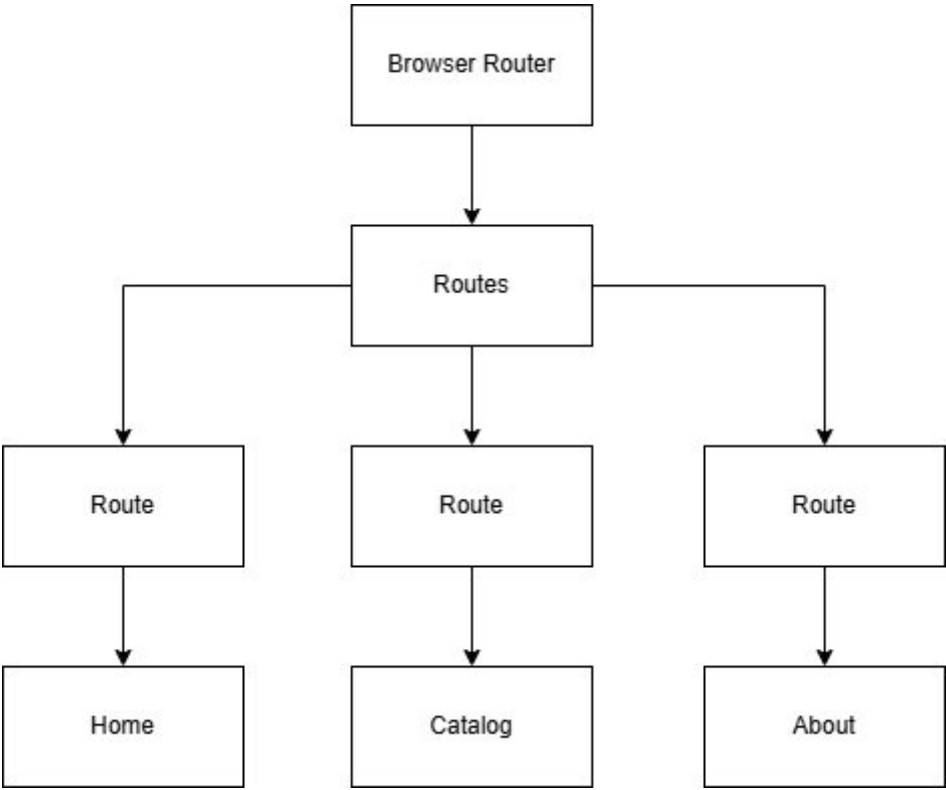


Рисунок 2.2 – Діаграма структури бібліотеки Router

В таблиці 2.1 наведено порівняння архітектур побудови веб-сторінок. В SPA оновлення сторінки ніколи не відбувається; натомість весь необхідний код HTML, JavaScript та CSS або отримується браузером під час одного завантаження сторінки, або відповідні ресурси динамічно завантажуються та додаються на сторінку за потреби, зазвичай у відповідь на дії користувача [10].

Таблиця 2.1 – Порівняння архітектури побудови веб-сторінок

Критерій	SPA (React)	МРА
Перезавантаження сторінки	Ні	Так
Швидкість UX	Висока	Нижча
Routing	Client-side	Server-side
Оновлення контенту	Динамічне	Повне
React Router	Використовується	Необов’язково

React Hooks – це спеціальні функції, які дозволяють вам «підключатися» до можливостей React у функціональних компонентах (табл. 2.2). Наприклад, хук `useState` дозволяє додавати стан, тоді як `useEffect` дозволяє виконувати побічні ефекти. Раніше побічні ефекти реалізовувалися за допомогою методів життєвого циклу. З хуками це більше не потрібно.

Таблиця 2.2 – Таблиця Hooks у ReactJS

Hook	Призначення
<code>useState</code>	Зберігання стану
<code>useEffect</code>	Побічні ефекти
<code>useContext</code>	Передача даних
<code>useMemo</code>	Оптимізація
<code>useCallback</code>	Кешування функцій
<code>useRef</code>	Виключення із рендерінгу
<code>useReducer</code>	Чистота коду

Хуки – це революційна функція, яка спростить ваш код, зробивши його легким для читання, підтримки, тестування окремо та повторного використання у ваших проектах. Вам знадобиться лише година, щоб ознайомитися з ними, але це змусить вас по-іншому поглянути на те, як ви пишете код React [11].

2.2 Firebase сервіс та його особливості

Firebase забезпечує інтеграцію баз даних, авторизації користувачів, хмарного зберігання файлів, хостингу та аналітики в межах єдиної платформи на основі платформи BaaS (Backend as a Service).

В таблиці 2.3 наведено порівняння з іншими хмарними сервісами. Однією з основних переваг Firebase є використання хмарної NoSQL бази даних Cloud Firestore, що забезпечує зберігання та синхронізацію даних у

режимі реального часу. Архітектура Firestore базується на колекціях та документах, що забезпечує гнучкість структури даних та спрощує масштабування застосунку. Синхронізація даних між клієнтом та сервером виконується автоматично, що дозволяє швидко оновлювати інформацію в інтерфейсі користувача.

Таблиця 2.3 – Порівняння хмарних сервісів

Характеристика	Firebase	Supabase	Appwrite
Тип платформи	Backend-as-a-Service	Open Source BaaS	Open Source BaaS
Тип бази даних	NoSQL (Firestore)	PostgreSQL (SQL)	MariaDB / SQL
Realtime синхронізація	Так	Так	Так
Offline підтримка	Повна	Обмежена	Обмежена
Авторизація	Email, Google, OAuth	Email, OAuth, JWT	Email, OAuth, JWT
Open Source	Ні	Так	Так
Self-hosting	Ні	Так	Так
Масштабованість	Висока	Висока	Середня/Висока
Vendor Lock-in	Високий	Низький	Низький
SQL-запити	Ні	Так	Так
REST API	Частково	Так	Так
Cloud Functions	Так	Так	Так
Підходить для React	Так	Так	Так
Найкраще застосування	Mobile та realtime apps	Web/SaaS системи	Self-hosted проєкти

Firebase також надає систему Authentication, яка забезпечує автентифікацію користувачів за допомогою електронної пошти, пароля або

сторонніх сервісів, зокрема Google Sign-In. Це дозволяє реалізувати безпечний механізм доступу до функціоналу застосунку без необхідності створення власної системи авторизації.

Для розробки сучасного ReactJS-застосунку найбільш доцільним є використання Firebase або Supabase. Firebase забезпечує швидку інтеграцію, realtime-функціональність та готову інфраструктуру, що значно пришвидшує процес розробки. Supabase, у свою чергу, є оптимальним рішенням для систем із складною структурою даних та необхідністю використання SQL. Appwrite доцільно використовувати у випадках, коли необхідний повний контроль над серверною інфраструктурою та self-hosting.

2.3 Інструмент Vite та його особливості

Vite – це сучасний інструмент для збірки, який дозволяє швидко та легко розпочати новий проект. На відміну від традиційних пакетів (порівняння наведено в таблиці 2.4), Vite розроблений для швидкої роботи. Він забезпечує миттєвий зворотний зв'язок під час написання коду, що робить розробку більш плавною. Vite також пропонує швидку збірку у продакшені, що робить його чудовим вибором для сучасних веб-проектів. Vite підтримує багато популярних фреймворків, таких як React, Vue та Svelte. Він гнучкий і добре працює з більшістю фронтенд-технологій [12].

Таблиця 2.4 – Порівняння інструментів Vite and Webpack

Характеристика	Vite	Webpack
Швидкість запуску	Висока	Нижча
Hot Reload	Миттєвий	Повільніший
Конфігурація	Мінімальна	Складніша
Підтримка ES Modules	Так	Частково
Production Build	Rollup	Webpack
Підходить для React	Так	Так

На відміну від традиційних інструментів збірки, таких як Webpack, Vite виконує попередню обробку лише необхідних модулів, що значно скорочує час запуску проєкту та прискорює процес розробки. Для production-збірки Vite використовує Rollup, який забезпечує оптимізацію JavaScript-коду та зменшення розміру фінального застосунку.

Важливою перевагою Vite є проста інтеграція з ReactJS та TypeScript, підтримка сучасних стандартів ECMAScript, а також мінімальна конфігурація проєкту. Це дозволяє швидко створювати масштабовані вебзастосунки з високою продуктивністю та зручним процесом розробки.

2.4 Контроль версій Git та GitHub

Безумовно, найпоширенішою сучасною системою контролю версій у світі є Git. Git – це зрілий, активно підтримуваний проєкт з відкритим кодом, спочатку розроблений у 2005 році Лінусом Торвальдсом, відомим творцем ядра операційної системи Linux.

Вражаюча кількість програмних проєктів покладається на Git для контролю версій, включаючи комерційні проєкти, а також проєкти з відкритим кодом. Розробники, які працювали з Git, добре представлені серед доступних талантів розробників програмного забезпечення, і він добре працює на широкому спектрі операційних систем та IDE (інтегрованих середовищ розробки).

Маючи розподілену архітектуру, Git є прикладом DVCS (Розподілена Система Контролю Версій). Замість того, щоб мати лише одне місце для повної історії версій програмного забезпечення, як це поширено в колись популярних системах контролю версій, таких як CVS або Subversion (також відомих як SVN), у Git робоча копія коду кожного розробника також є репозиторієм, який може містити повну історію всіх змін.

Окрім розподіленості, Git був розроблений з урахуванням продуктивності, безпеки та гнучкості [13].

Характеристики продуктивності Git дуже сильні порівняно з багатьма альтернативами. Запис нових змін, розгалуження, об'єднання та порівняння попередніх версій оптимізовано для підвищення продуктивності. Алгоритми, реалізовані в Git, використовують глибокі знання про загальні атрибути реальних дерев файлів вихідного коду, про те, як вони зазвичай змінюються з часом та які шаблони доступу.

На відміну від деяких програм для контролю версій, Git не обманюється назвами файлів, визначаючи, яким має бути сховище та історія версій дерева файлів, натомість Git зосереджується на самому вмісті файлу. Зрештою, файли вихідного коду часто перейменовуються, розділяються та перевпорядковуються. Об'єктний формат файлів репозиторію Git використовує комбінацію дельта-кодування (зберігання відмінностей вмісту), стиснення та явно зберігає вміст каталогів та об'єкти метаданих версій [13].

Для зберігання віддаленого репозиторію та організації спільної роботи використовується сервіс GitHub, який надає хмарну платформу для управління Git-репозиторіями. GitHub забезпечує підтримку branch-моделі розробки, pull request, code review та системи резервного зберігання коду (табл.2.5). Використання Git та GitHub дозволяє ефективно організувати процес розробки вебзастосунку.

Таблиця 2.5 – Можливості Git

Технологія	Призначення
Git	Контроль версій
GitHub	Хмарне зберігання репозиторіїв
Branch	Окрема гілка розробки
Commit	Збереження змін
Pull Request	Перевірка та об'єднання коду
Clone	Копіювання репозиторію
Push/Pull	Синхронізація змін

3 РОЗРОБКА ВЕБЗАСТОСУНКУ

Даний розділ присвячено етапам практичної реалізації хмарної інформаційної системи для візуалізації та розподілу товарів із використанням геоінформаційних сервісів. Розробка програмного продукту вимагає побудови надійної клієнт-серверної архітектури, здатної обробляти динамічні дані в режимі реального часу, забезпечувати безпечну автентифікацію та візуалізувати просторову інформацію.

Діаграма будови вебзастосунку зображена на рисунку 3.1.

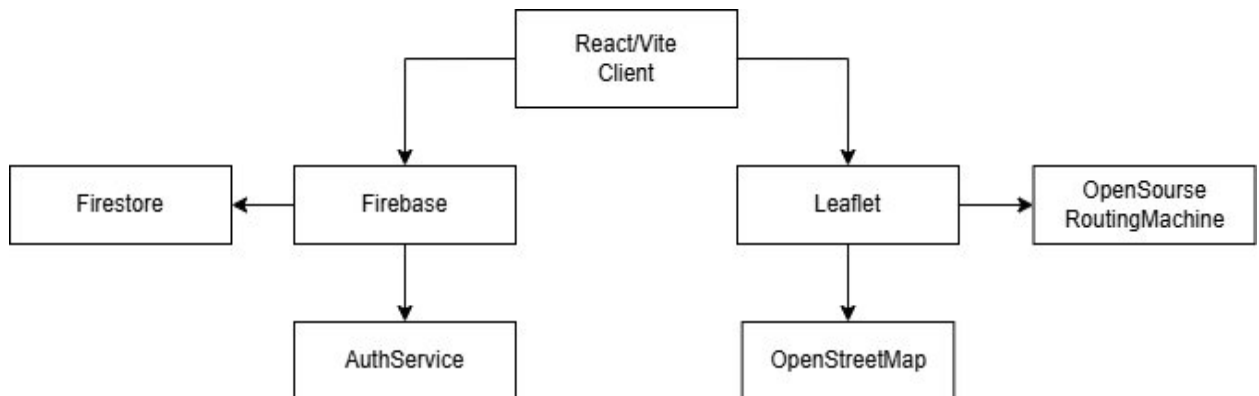


Рисунок 3.1 – Діаграма будови вебзастосунку

Процес розробки розділено на логічні етапи: від конфігурації базового середовища та побудови навігації до інтеграції геоінформаційних модулів та алгоритмів фільтрації великих масивів даних.

3.1 Вибір та обґрунтування технологічного стека та архітектури

Сучасні вимоги до вебзастосунків диктують необхідність використання інструментів, що забезпечують високу швидкість роботи, масштабованість та зручність підтримки коду. Для реалізації системи було обрано архітектурний підхід Single Page Application (SPA).

3.1.1 Концепція Single Page Application (SPA)

На відміну від традиційних багатосторінкових сайтів, де кожна дія користувача призводить до повного перезавантаження сторінки із сервера, SPA завантажує єдиний HTML-документ. Усі подальші зміни інтерфейсу відбуваються динамічно за допомогою JavaScript, який маніпулює об'єктною моделлю документа (DOM).

Цей підхід має низку переваг:

- зниження навантаження на сервер: сервер передає лише дані (зазвичай у форматі JSON), а не готову HTML-розмітку;
- швидкодія інтерфейсу: переходи між розділами каталогу чи картою відбуваються миттєво;
- покращений користувацький досвід (UX): можливість реалізації плавних анімацій та збереження стану застосунку під час навігації.

Основним інструментом для реалізації цієї архітектури було обрано бібліотеку React, а для швидкого збирання проєкту – технологія Vite, яка забезпечує перезавантаження модулів (HMR) під час розробки.

3.1.2 Ініціалізація точки входу

Робота системи починається з файлу `index.html`, який містить мінімальну розмітку та кореневий контейнер `<div id="root"></div>`. Компонент `main.jsx` виконує роль ініціалізатора, який монтує віртуальний DOM React-застосунку в цей контейнер:

```
import { createRoot } from 'react-dom/client'
import App from './App.jsx'
import 'leaflet/dist/leaflet.css';
createRoot(document.getElementById('root')).render( <App />)
```

Підключення стилів `leaflet.css` на цьому рівні є критично важливим, оскільки це гарантує коректне відображення картографічних тайлів та

маркерів на всіх сторінках застосунку, де буде викликано геоінформаційний модуль.

3.2 Побудова системи навігації та глобального стану

Серцем клієнтської частини є компонент `App.jsx`, який виконує роль головного контролера. Він відповідає за ініціалізацію маршрутизатора, управління глобальним станом авторизації та визначення структури сторінок.

3.2.1 Маршрутизація за допомогою React Router

Для імітації багатосторінковості в межах SPA використовується бібліотека `react-router-dom`. Компонент `<BrowserRouter>` (імпортований як `Router`) відстежує зміни URL-адреси в браузері та синхронізує їх з відображеним інтерфейсом.

```
return (
  <Router>
    <div className="app-container">
      <Scroll />
      <Navbar user={user} />
      <main className="main-content">
        <Routes>
          <Route path="/" element={<Home />} />
          <Route path="/catalog" element={<Catalog user={user} />} />
          <Route path="/about" element={<About />} />
          <Route path="/favorites" element={<Favorites user={user} />} />
          <Route path="*" element={<NotFound />} />
        </Routes>
      </main>
      <Footer />
    </div>
  </Router>
)
```

```
</div>
```

```
</Router> );
```

Така структура гарантує, що компоненти Navbar та Footer залишатимуться незмінними під час навігації. Компонент `<NotFound />` виконує роль обробника помилок: якщо користувач вводить неіснуючу адресу (маршрут *), система коректно повідомляє про відсутність сторінки, зберігаючи загальну стилістику сайту.

3.2.2 Розробка адаптивної навігаційної панелі

Компонент `NavBar.jsx` відповідає за основну навігацію. Для забезпечення зручності використання на мобільних пристроях було реалізовано адаптивний дизайн із використанням "бургер-меню". Для цього компонент відстежує розмір вікна браузера за допомогою хука `useEffect`:

```
useEffect(() => {
  const handleResize = () => {
    setIsMobile(window.innerWidth < 400);  };
    window.addEventListener('resize', handleResize);
    return () => window.removeEventListener('resize', handleResize); }, []);
```

Використання функції очищення (`removeEventListener`) під час розмонтування компонента є обов'язковим патерном розробки, що запобігає витокам пам'яті (`memory leaks`) у клієнтському браузері.

3.3 Інтеграція хмарної інфраструктури Firebase

Оскільки система передбачає роботу з розподілом товарів та даними користувачів, архітектура потребує надійного бекенду. Для зменшення витрат на розробку власного сервера було обрано платформу Google Firebase, що працює за моделлю Backend-as-a-Service (BaaS).

3.3.1 Конфігурація сервісів

У файлі `firebase.js` створюється екземпляр з'єднання з хмарою. Система ініціалізує три основні модулі:

- `getAuth` – для управління сесіями та ідентифікацією;
- `getFirestore` – для роботи з NoSQL базою даних реального часу;
- `GoogleAuthProvider` – для забезпечення можливості швидкого входу через обліковий запис Google (OAuth 2.0).

3.3.2 Управління автентифікацією та безпекою

Глобальний стан користувача (`user`) зберігається в `App.jsx`. За допомогою слухача `onAuthStateChanged`, застосунок автоматично реагує на зміну токена (наприклад, коли термін дії сесії закінчується або користувач здійснює вхід).

Компонент `AuthModule.jsx` інкапсулює логіку реєстрації та авторизації. Важливою архітектурною особливістю є синхронізація сервісу авторизації з базою даних. Під час успішної реєстрації нового користувача система не лише створює обліковий запис, але й генерує відповідний документ у колекції `users` бази даних `Firestore`:

```
const userCredential = await createUserWithEmailAndPassword(auth,
email, password);

const currentUser = userCredential.user;
await setDoc(doc(db, "users", currentUser.uid), {
  name: name,
  email: currentUser.email,
  favorites: [], role: "user", createdAt: new Date().toISOString(), });
```

Створення порожнього масиву `favorites` на етапі реєстрації є підготовчим кроком для подальшої роботи бізнес-логіки збереження вподобань.

3.4 Проєктування бази даних та генерація тестових даних

Документо-орієнтована база даних Firestore зберігає інформацію у вигляді колекцій та документів. Для тестування алгоритмів розподілу товарів та візуалізації складів система потребувала великого масиву даних. Для цього було розроблено спеціалізований компонент-сідер (DBSeeder).

3.4.1 Автоматизація наповнення бази даних (DBSeeder)

Компонент DBSeeder.jsx – це інструмент розробника, який генерує 100 унікальних документів товарів та завантажує їх у хмару. Процес включає генерацію випадкових назв, присвоєння категорій, цін та, для геоінформаційної системи – точних географічних координат складів (табл.3.1). Для забезпечення цілісності інтерфейсу, під час асинхронного завантаження даних `await addDoc(productsCollection, newProduct)`, система блокує кнопку та відображає прогрес виконання операції, запобігаючи повторним клікам.

Таблиця 3.1 – Згенеровані локації складів (Warehouses)

Назва складу	Широта	Довгота
Copenhagen Central Hub	55.6761	12.5683
Nørrebro Storage	55.6867	12.5701
Vesterbro Depot	55.6621	12.5415
Østerbro Facility	55.6984	12.5794
Amager Distribution	55.6541	12.605
Frederiksberg Center	55.6795	12.5298
Herlev Stock	55.7204	12.4746
Glostrup Warehouse	55.6558	12.3855
Ballerup Electronics Unit	55.7371	12.3965
Hvidovre Dispatch	55.6262	12.4781

3.4.2 Структура бази даних Firestore

У Firestore база даних представлена як вкладені об'єкти у JavaScript. Наприклад у таблиці users є колекція, яка представляє певного користувача. У цій колекції є createdAt, email, масив favorites, name, role. У колекції продукту є category, createdAt, description, image, location, name, price.

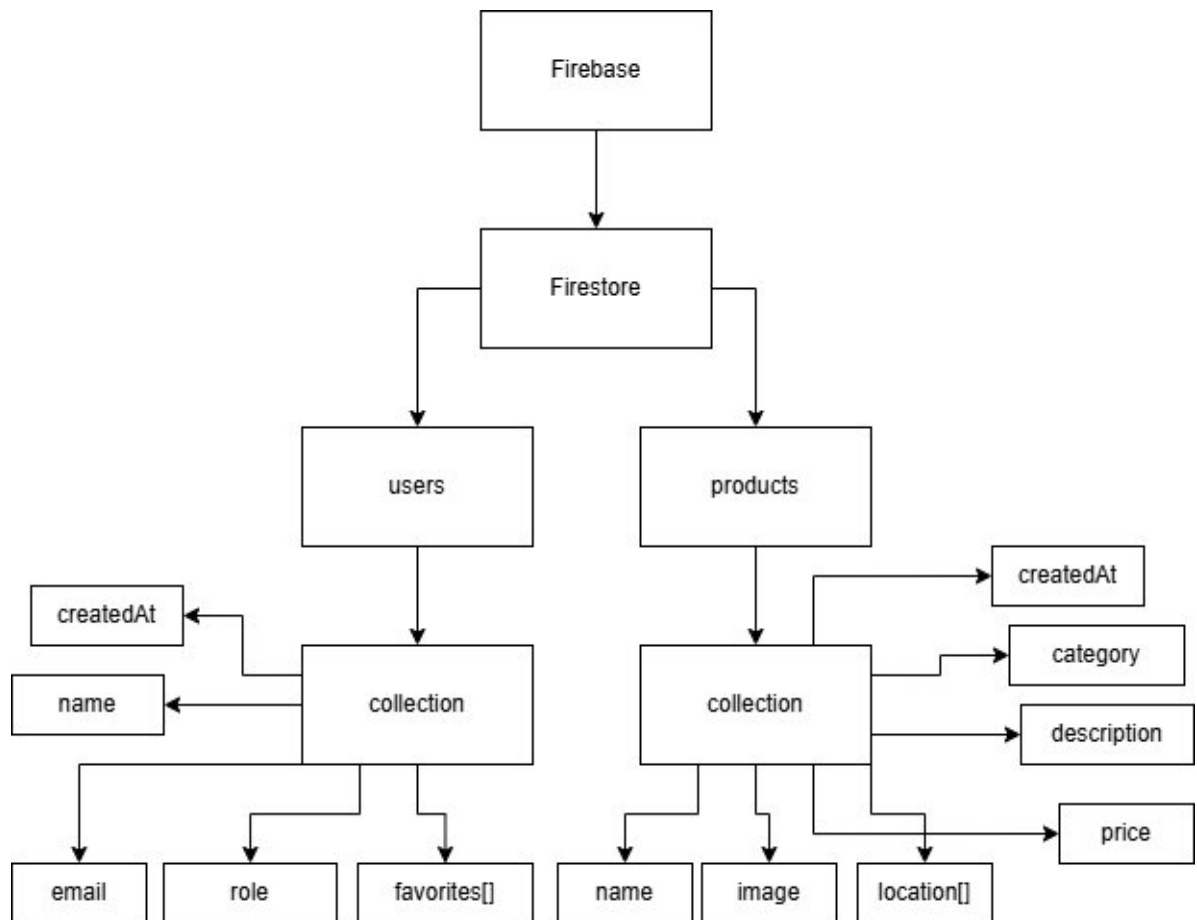


Рисунок 3.2 – Структура бази даних Firestore

3.5 Розробка каталогу товарів та системи пошуку

Основний бізнес-функціонал системи зосереджено на сторінці каталогу (Catalog.jsx), яка інтегрує в собі компоненти пошуку (SearchBar.jsx), фільтрації (SideBar.jsx) та безпосереднього відображення карток товарів (ProductCard.jsx).

3.5.1 Патерн Lifting State Up

Для узгодженої роботи пошуку та бічної панелі фільтрів було застосовано патерн "підняття стану". Глобальний об'єкт фільтрів `filters` зберігається в батьківському компоненті `Catalog`:

```
const [filters, setFilters] = useState({
  searchQuery: "",
  sortBy: "default",
  category: "All",
  warehouse: "All",
  minPrice: "",
  maxPrice: "",});
```

Функції для зміни цього стану передаються як пропси до `SearchBar` (для оновлення `searchQuery`) та `SideBar` (для оновлення категорій, цін і локацій складів). Це забезпечує єдине джерело істини (single source of truth) для алгоритмів фільтрації.

3.5.2 Оптимізація рендерингу

Каталог може містити тисячі записів. Щоб уникнути перерахунку всього масиву під час кожного оновлення інтерфейсу (наприклад, під час введення тексту в поле пошуку), алгоритм фільтрації обгорнуто в хук `useMemo`. Він кешує результат і перераховує масив `filteredProducts` лише тоді, коли змінюються залежності – базовий масив `products` або об'єкт `filters`:

```
const filteredProducts = useMemo(() => {
  let result = [...products];
  // 1. Пошук за назвою та категорією
  if (filters.searchQuery.trim() !== "") {
    const query = filters.searchQuery.toLowerCase();
    result = result.filter(p =>
```

```

    p.name.toLowerCase().includes(query) ||
    p.category.toLowerCase().includes(query) ); }
// 2. Фільтрація за категорією та складом
if (filters.category !== "All") result = result.filter(p => p.category ===
filters.category);
if (filters.warehouse !== "All") result = result.filter(p =>
p.location?.warehouseName === filters.warehouse);
// 3. Сортування
if (filters.sortBy === "priceAsc") result.sort((a, b) => a.price - b.price);
else if (filters.sortBy === "priceDesc") result.sort((a, b) => b.price -
a.price);
return result;
}, [products, filters]);

```

3.6 Персоналізація, управління списком обраного

Взаємодія з обраними товарами відбувається на рівні компонента `ProductCard.jsx`. Під час натискання іконки "Favorites" спрацьовує метод `toggleFavorite`.

Замість того, щоб завантажувати весь масив обраного користувача, модифікувати його на клієнті і відправляти назад, система використовує атомарні операції `Firestore arrayUnion` та `arrayRemove`. Це унеможливорює втрату даних при одночасних запитах з різних пристроїв.

На сторінці `Favorites.jsx` завантаження збережених товарів оптимізовано завдяки паралельному виконанню запитів. Замість послідовного звернення до бази даних (що викликало б затримки типу "waterfall"), система формує масив промісів (Promises) і чекає їхнього одночасного виконання:

```

const productPromises = favoriteIds.map((id) =>
  getDoc(doc(db, "products", id)));

```

```
const productDocs = await Promise.all(productPromises);
```

3.7 Геоінформаційний модуль та візуалізація логістики

Найважливішим інноваційним елементом застосунку є інтеграція ГІС-технологій для відображення локацій товарів і розрахунку логістики. За це відповідає компонент `Map.jsx`, побудований на базі `Leaflet`.

3.7.1 Ініціалізація карти та позиціонування

Карта генерується за допомогою компонента `<MapContainer>`, який відмальовує тайли (графічні фрагменти карти) із відкритого джерела `OpenStreetMap`. На карті автоматично встановлюється маркер (`Marker`), координати якого відповідають параметрам `lat` та `lng` складу, на якому зберігається обраний товар.

Для забезпечення можливості розрахунку маршруту система використовує `HTML5 Geolocation API`.

Метод `navigator.geolocation.getCurrentPosition` дозволяє вебзастосунку, за умови дозволу користувача, отримати координати його поточного місцезнаходження.

3.7.2 Побудова та розрахунок маршрутів

Після отримання координат клієнта активується субкомпонент `RoutingMachine`. Він використовує плагін `leaflet-routing-machine`, який здійснює запит до маршрутизатора (`OSRM`) для розрахунку найкоротшого шляху між двома точками дорожньою мережею.

```
const routingControl = L.Routing.control({
  waypoints: [
```

```

        L.latLng(userPos[0], userPos[1]),           // Точка відправлення
        (Користувач)
        L.latLng(warehousePos[0], warehousePos[1]), // Точка призначення
        (Склад) ],
        lineOptions: {
            styles: [{ color: "#4caf50", weight: 5 }], },
        addWaypoints: false,
        fitSelectedRoutes: true,
        show: false,
    }).addTo(map);

```

Завдяки налаштуванню `fitSelectedRoutes: true`, камера карти автоматично масштабується таким чином, щоб весь побудований маршрут вміщувався в область видимості екрана. Щоб карта не ламала загальну верстку карток товарів, її рендеринг реалізовано через механізм React Portals (`createPortal`).

Це дозволяє логічно прив'язати модальне вікно до компонента `ProductCard`, але фізично відобразити його на найвищому рівні DOM-дерева безпосередньо перед тегом `</body>`.

3.8 Асинхронна взаємодія та оптимізація потоків даних

Оскільки розроблена інформаційна система побудована на хмарній архітектурі, вона постійно взаємодіє з віддаленими серверами бази даних та сервісами авторизації. JavaScript, на якому базується React, є однопотоковою мовою програмування.

Це означає, що виконання важких або тривалих мережевих запитів у синхронному режимі призвело б до блокування основного потоку та інтерфейсу користувача (UI). Для вирішення цієї проблеми у системі широко застосовується асинхронне програмування.

3.8.1 Застосування патерну `async / await`

Для обробки асинхронних подій замість застарілого підходу з використанням функцій зворотного виклику (callbacks) або складних ланцюжків промісів (`.then().catch()`), у проєкті імплементовано конструкції `async` та `await`.

Використання ключового слова `async` перед оголошенням функції автоматично перетворює її результат на об'єкт `Promise`. У свою чергу, оператор `await` зупиняє виконання коду всередині цієї функції до моменту завершення мережевого запиту, звільняючи при цьому основний потік для рендерингу інших компонентів застосунку.

Типовий приклад такої реалізації застосовано у модулі автентифікації (`AuthModule.jsx`) під час реєстрації нового користувача. Такий підхід робить асинхронний код візуально лінійним (схожим на синхронний), що значно спрощує його читання, налагодження (debugging) та дозволяє інкапсулювати всю логіку обробки помилок в єдиний блок `try...catch...finally`.

3.8.2 Управління життєвим циклом асинхронних подій у React

Специфіка роботи React-хука `useEffect`, який відповідає за виконання побічних ефектів (наприклад, завантаження каталогу товарів під час відкриття сторінки), не дозволяє передавати йому асинхронну функцію безпосередньо. `useEffect` має повертати або функцію очищення (cleanup function), або нічого (undefined).

Тому у компонентах `Catalog.jsx` та `Favorites.jsx` застосовано архітектурний патерн створення внутрішньої асинхронної функції з її подальшим негайним викликом (рис. 3.3). Це дозволяє безпечно використовувати `async/await` під час монтування компонента.

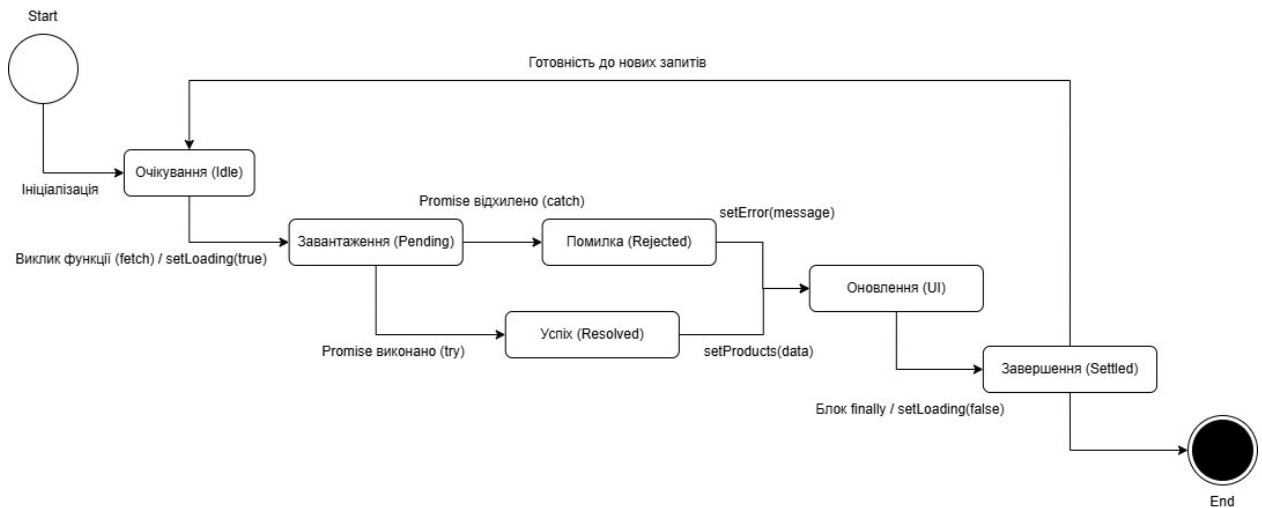


Рисунок 3.3 – Діаграма станів асинхронного запиту до бази даних Firestore

3.8.3 Паралельне виконання мережевих запитів

Окремої уваги заслуговує оптимізація швидкодії на сторінці обраних товарів (Favorites.jsx). Архітектура бази даних передбачає, що в документі користувача зберігається лише масив ідентифікаторів товарів (ID). Відповідно, щоб відобразити список, системі потрібно зробити окремий запит до колекції products для кожного ID з масиву.

Якби ці запити виконувалися за допомогою класичного циклу for або for...of із застосуванням await перед кожною ітерацією, це призвело б до проблеми, що кожен наступний запит чекав би завершення попереднього. Час завантаження сторінки зростає би пропорційно кількості обраних товарів.

Для вирішення цієї проблеми використано метод статичного класу Promise.all(), який дозволяє виконувати масив промісів паралельно. Завдяки цьому архітектурному рішення, загальний час завантаження списку з десятків товарів зменшується до тривалості найповільнішого одиничного запиту, що критично важливо для збереження високих показників швидкодії SPA-застосунку.

3.9 Обробка виняткових ситуацій та оптимізація UI/UX

Останнім, але не менш важливим етапом розробки, є забезпечення відмовостійкості застосунку. Оскільки система постійно взаємодіє з мережевими запитами (Firebase, картографічні тайли), у всіх асинхронних функціях впроваджено блоки `try...catch...finally`.

Наприклад, якщо геолокація користувача недоступна через відсутність прав доступу в браузері, система перехоплює помилку в `navigator.geolocation` та виводить кастомне повідомлення: `alert("Unable to retrieve your location. Check permissions.")`, замість того, щоб спричинити збій всього компонента карти.

Також усюди реалізовано локальні стани `loading`. Доки дані не будуть повністю отримані із сервера, користувач бачить індикатор завантаження, що є стандартом сучасного дизайну користувацьких інтерфейсів.

3.10 Висновки за розділом 3

У третьому розділі було детально розглянуто всі етапи практичної реалізації хмарної інформаційної системи для візуалізації та розподілу товарів. Процес розробки охопив повний цикл створення сучасного вебзастосунку: від проєктування базової архітектури до інтеграції складних геоінформаційних сервісів.

Обґрунтовано вибір та успішно застосовано технологічний стек на базі бібліотеки `React` та інструменту збирання `Vite`. Побудова системи за принципом `Single Page Application` із використанням `react-router-dom` забезпечила високу швидкодію, плавність навігації та відсутність зайвих перезавантажень сторінок, що суттєво покращило користувацький досвід.

Замість розробки власного сервера використано платформу `Google Firebase` (модель `BaaS`). Це дозволило швидко та безпечно впровадити комплексну систему автентифікації користувачів (електронна пошта, пароль,

OAuth через Google) та налаштувати документо-орієнтовану базу даних Firestore для зберігання персональних даних і списків товарів у режимі реального часу.

Створено модульну систему відображення товарів із просунутими алгоритмами фільтрації, сортування та пошуку. Завдяки застосуванню патерну «підняття стану» та хука useMemo досягнуто високої продуктивності інтерфейсу навіть під час обробки великих масивів даних. Також реалізовано ефективний механізм управління списком обраного за допомогою атомарних операцій Firestore.

Ключовим функціональним досягненням стала успішна інтеграція бібліотеки Leaflet та плагіна маршрутизації OSRM. Система здатна динамічно визначати геолокацію користувача (через HTML5 Geolocation API) та візуально прокладати оптимальні логістичні маршрути до складів на інтерактивній карті, що повністю відповідає поставленій меті дипломного проєкту.

Проведено комплексну роботу з обробки виняткових ситуацій (перехоплення помилок авторизації, відсутності доступу до геоданих тощо). Впровадження глобальних та локальних станів завантаження (loading states) зробило процес очікування відповіді від сервера прозорим та зрозумілим для кінцевого користувача.

4 АНАЛІЗ РЕЗУЛЬТАТІВ ТА ВИКОРИСТАННЯ ЗАСТОСУНКУ

Після завершення етапу програмної реалізації було проведено комплексне випробування хмарної інформаційної системи. Метою даного розділу є оцінка працездатності розробленого функціоналу, аналіз технічних характеристик продуктивності, опис знайдених дефектів та визначення векторів подальшого розвитку проекту.

4.1 Результати роботи

На рисунку 4.1 зображена головна сторінка вебзастосунку.

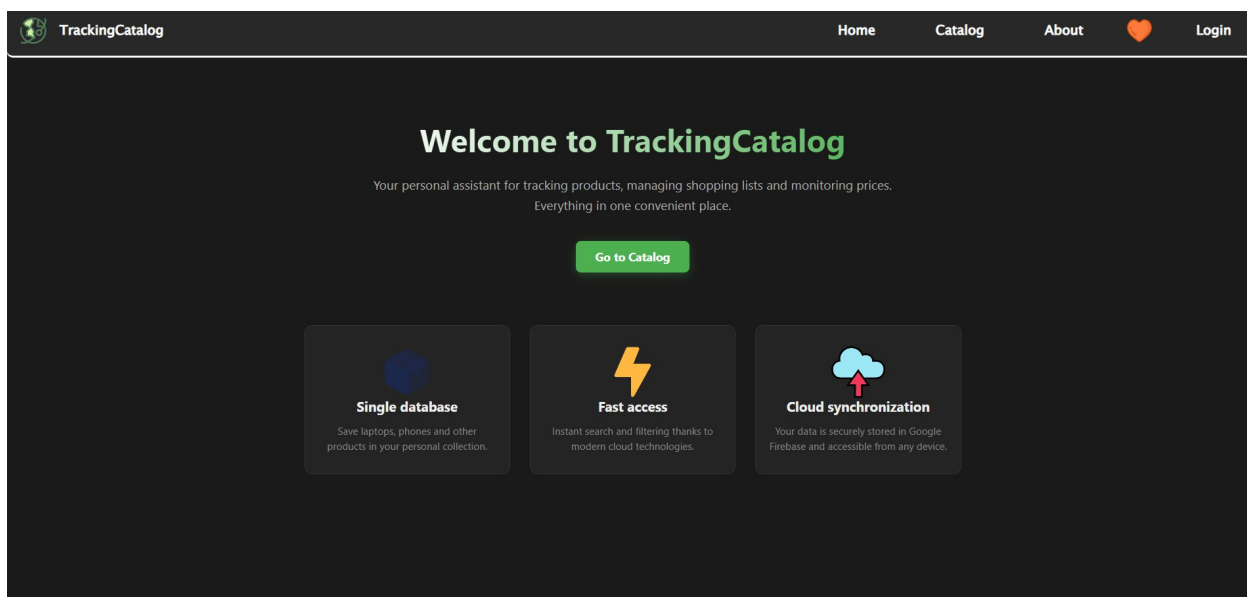


Рисунок 4.1 – Головна сторінка Home

На рисунках 4.2 зображено відно реєстрації та авторизації, для входу в систему можна використовувати Google акаунт.

На рисунку 4.3 наведено приклад профілю користувача. На рисунку 4.4 можна побачити основну сторінку пошуку та фільтрації товарів.

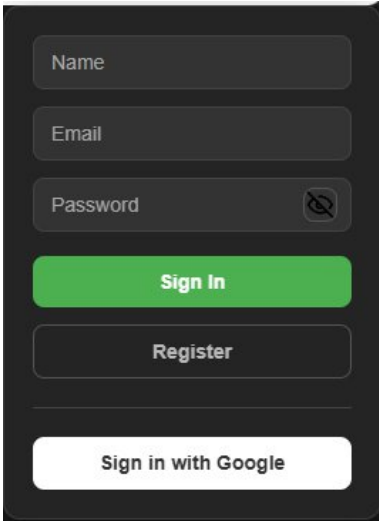


Рисунок 4.2 – Вікно реєстрацію та авторизації користувача

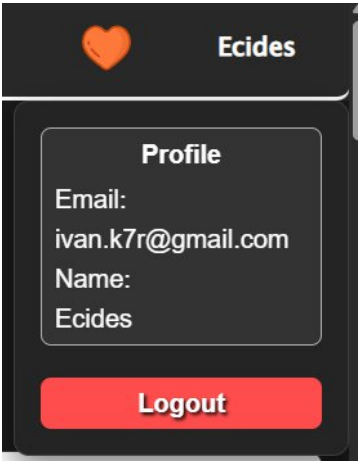


Рисунок 4.3 – Профіль авторизованого користувача

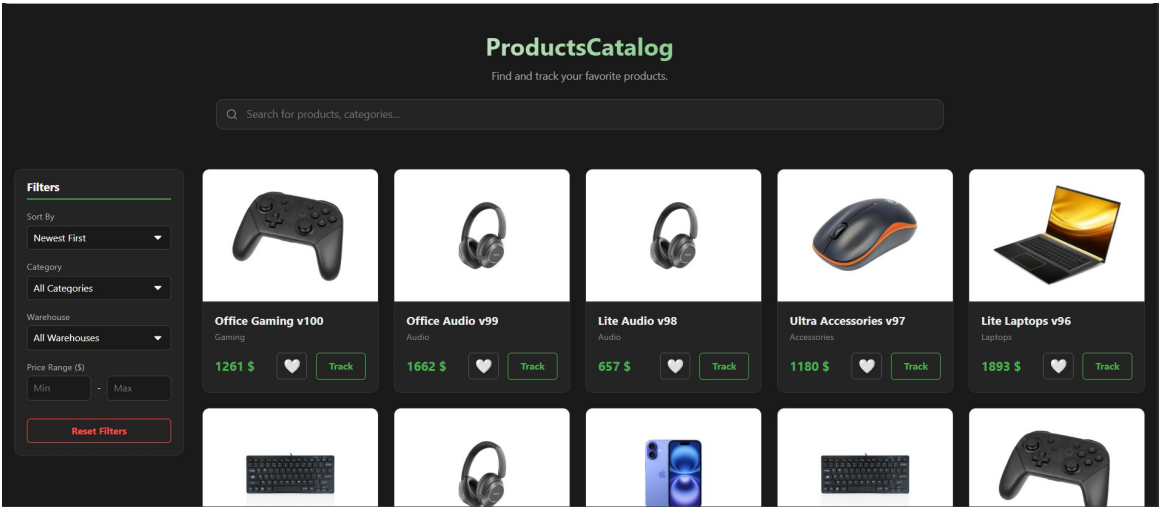


Рисунок 4.4 – Сторінка товарів пошуку та фільтрів

На рисунку 4.5 наведено список обраних товарів користувачем. А на рисунку 4.6 відображається приклад мапи розміщення складу з товаром та рисунку 4.7 прокладений маршрут до складу з товаром.

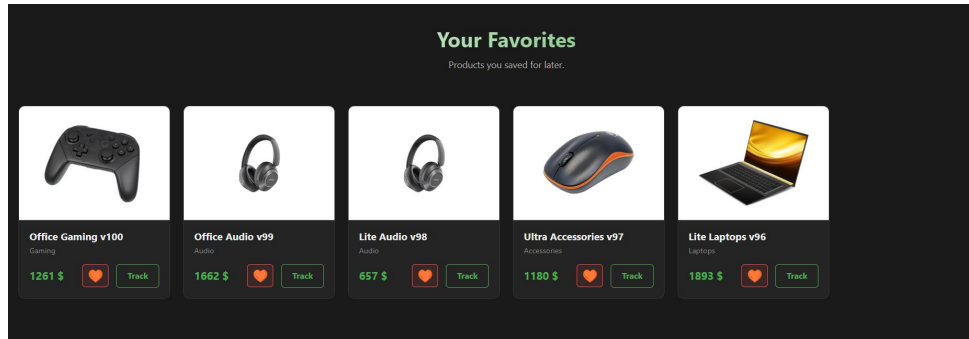


Рисунок 4.5 – Список обраного для авторизованого користувача

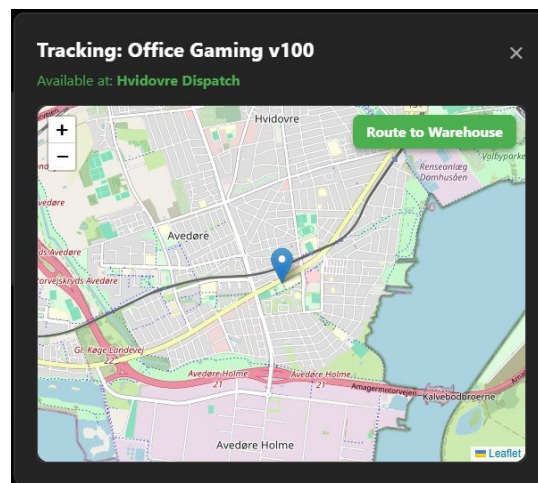


Рисунок 4.6 – Мапа показує розміщення товару на складі

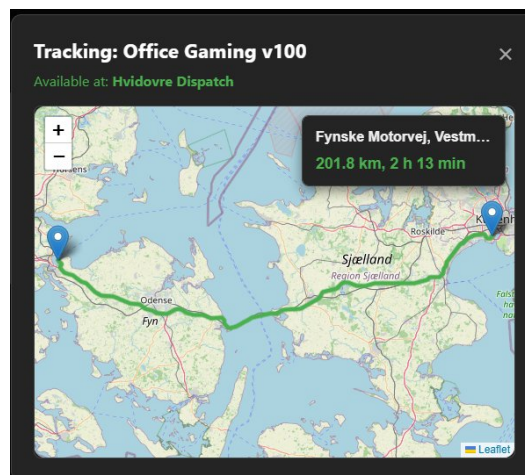


Рисунок 4.7 – Маршрут від користувача до складу

На рисунку 4.8 зображено послідовність дій користувача, які зображені на рисунках 4.1 – 4.7.



Рисунок 4.8 – Діаграма послідовності дій користувача

Основний функціонал розробленого вебзастосунку, весь функціонал та usability сайту відповідає нормам та доволі зручний у користуванні.

4.2 Методика та стратегія тестування системи

Для підтвердження надійності вебзастосунку було застосовано комбінований підхід до тестування, що включав функціональну перевірку, тестування інтерфейсу (UI/UX) та інтеграційне тестування взаємодії з хмарними сервісами.

4.2.1 Функціональне тестування

Модуль автентифікації: перевірка коректності створення облікових записів, валідація вхідних даних та безпека перенаправлення авторизованих користувачів.

Модуль каталогу: тестування точності роботи фільтрів, сортування та динамічного пошуку за ключовими словами.

Геоінформаційний модуль: перевірка коректності визначення координат за допомогою Geolocation API та точності прокладання маршрутів через сервіс Leaflet Routing Machine.

4.2.2 Інтеграційне тестування з Firebase

Оскільки система побудована на безсерверній архітектурі, критично важливим було перевірити швидкість синхронізації даних між клієнтом та Firestore. Тестування проводилося при симуляції низької швидкості інтернет-з'єднання для оцінки поведінки станів завантаження (loading states).

4.3 Аналіз результатів тестування та продуктивності

У результаті серії запусків було отримано кількісні та якісні показники роботи системи. Загальна кількість успішно виконаних тест-кейсів склала 100%, що свідчить про готовність базової версії продукту до використання.

4.3.1 Показники швидкодії

Однією з переваг використання Vite та React є висока швидкість рендерингу компонентів. Було проведено замір часу відповіді системи на основні дії користувача (таблиця 4.1).

Таблиця 4.1 – Показники швидкості

Операція	Середній час виконання	Очікуваний результат
Початкове завантаження SPA	< 800 мс	Успішно
Авторизація через Google OAuth	1.2 – 1.8 с	Успішно
Фільтрація каталогу (100 товарів)	< 50 мс	Успішно
Побудова маршруту на карті	0.5 – 1.0 с	Успішно

4.3.2 Валідація геоінформаційної складової

Під час тестування модуля карт було підтверджено, що система коректно обробляє випадки відмови користувача від надання доступу до геолокації. У такому випадку маркер користувача не відображається, проте карта залишається функціональною, дозволяючи візуально знайти склад товару за встановленим маркером.

4.4 Виявлені дефекти та запропоновані технічні рішення

Незважаючи на успішне проходження основних тестів, у процесі розробки та фінальної перевірки було виявлено кілька технічних нюансів, що потребують уваги при подальшій підтримці.

4.4.1 Оптимізація запитів до Firestore

Було помічено, що при частому оновленні списку обраних товарів створюється велика кількість запитів до бази даних.

Виявлена проблема: Кожне натискання на кнопку "Track" або "Favorite" ініціює миттєвий запит.

Рішення: Впровадження механізму `debounce` або `throttle` для обмеження частоти запитів до хмари, що дозволить знизити навантаження на API та зменшити вартість використання Firebase.

4.4.2 Покращення валідації на рівні бази даних

На даному етапі основна валідація відбувається на клієнтській стороні.

Пропозиція: Налаштувати `Firestore Security Rules` на стороні сервера Firebase, щоб гарантувати неможливість запису некоректних даних у БД навіть при спробі прямого доступу до API.

4.5 Перспективи масштабування та вдосконалення системи

Розроблена система «TrackingCatalog» має високий потенціал для масштабування завдяки модульній архітектурі React та гнучкості хмарної інфраструктури Google.

4.5.1 Технічне масштабування

Для розширення можливостей системи пропонується:

Впровадження PWA (Progressive Web App): Налаштування `Service Workers` дозволить користувачам працювати з каталогом обраних товарів навіть за відсутності доступу до мережі Інтернет.

Реалізація SSR (Server Side Rendering): Перехід на Next.js для покращення SEO-показників, що важливо для публічних каталогів товарів.

Розширення ГІС-аналітики: Додавання можливості розрахунку приблизного часу доставки (ETA) та інтеграція з API реальних кур'єрських служб.

4.5.2 Функціональний розвиток

З точки зору користувацького досвіду, перспективним є додавання системи повідомлень (Push Notifications). Користувач зможе отримувати сповіщення про зміну ціни на товари зі списку обраного або появу нових позицій на найближчому до нього складі.

4.6 Інструкція використання системи

Для забезпечення коректного розгортання та експлуатації системи розроблено коротку інструкцію для розробників та кінцевих користувачів.

4.6.1 Налаштування середовища розробки

Встановити Node.js версії 18.0 або вище.

Встановити IDE для перегляду проекту.

Перейти до репозиторію GitHub.

Склонувати або завантажити проєкт.

Відкрити проєкт та термінал.

Запустити сервер розробки командою `npm run dev` у терміналі VSCode (Якщо використовується саме ця IDE).

Відкрити посилання – `localhost:5173`

4.6.2 Порядок роботи користувача

Головна сторінка: перегляд загальної інформації про сервіс.

Авторизація: вхід через пошту або Google для доступу до персональних функцій.

Каталог: використання пошуку та бічної панелі для вибору товарів.

Відстеження: натискання кнопки «Track» для відкриття карти та побудови маршруту до складу.

Обране: перегляд збережених товарів у розділі «Favorites» для швидкого доступу.

4.7 Висновки за розділом 4

Четвертий розділ присвячений критичній оцінці результатів розробки хмарної системи «TrackingCatalog». Проведене тестування підтвердило високу ефективність обраної архітектури:

Використання SPA на базі React забезпечило відгук інтерфейсу в межах 50-800 мс.

Інтеграція з Firebase Firestore дозволила реалізувати синхронізацію даних у реальному часі без створення складного серверного коду.

Геоінформаційний модуль успішно вирішує задачу візуалізації складів та побудови маршрутів, що було однією з пріоритетних цілей роботи.

Виявлені під час аналізу недоліки (необхідність додаткової валідації на бекенді та оптимізації частоти запитів) не є критичними та враховані в плані подальшої модернізації системи. Створений програмний продукт є повністю функціональним, масштабованим та готовим до впровадження як інструмент для автоматизації каталогів товарів із логістичною складовою.

ВИСНОВКИ

У результаті виконання дипломної роботи було проведено комплексне дослідження методів проєктування, оптимізації та програмної реалізації хмарних інформаційних систем для візуалізації та розподілу товарів. Проведений аналіз архітектурних рішень, алгоритмів обробки даних та інтеграції геоінформаційних сервісів дозволив зробити такі науково-практичні висновки.

Аналіз сучасних патернів проєктування вебзастосунків підтвердив високу ефективність концепції Single Page Application (SPA) для систем із великим масивом динамічних даних. Дослідження продуктивності показало, що використання бібліотеки React забезпечує оптимізований рендеринг інтерфейсу завдяки механізму віртуального DOM. Визначено, що застосування алгоритмів мемоїзації під час фільтрації каталогу дозволяє виконувати миттєвий пошук та сортування даних на стороні клієнта без додаткових звернень до сервера, що критично знижує навантаження на мережу та покращує показники швидкодії.

Аналіз існуючих картографічних рішень виявив, що інтеграція відкритих геоінформаційних сервісів є оптимальною альтернативою комерційним API. Дослідження логістичного модуля підтвердило, що використання стандарту HTML5 Geolocation API дозволяє з високою точністю візуалізувати просторовий розподіл товарів на складах та динамічно будувати оптимальні маршрути для користувача в режимі реального часу.

Проведене дослідження та подальша програмна реалізація доводять, що розроблений вебзастосунок є стабільним, масштабованим та технологічно обґрунтованим рішенням.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Integrating GIS Websites and Business in the Digital Era. URL: <https://www.linkedin.com/pulse/integrating-gis-websites-business-digital-era-mapidseeit-muijc> (date of access: 12.05.2026).
2. Why your organization needs a web GIS strategy. URL: <https://millermountain.com/geospatialblog/2017/05/02/web-gis-strategy/> (date of access: 12.05.2026).
3. The Role of Cloud Computing in Web Development. URL: <https://www.ironhack.com/us/blog/the-role-of-cloud-computing-in-web-development> (date of access: 12.05.2026).
4. Wikipedia Leaflet software. URL: [https://en.wikipedia.org/wiki/Leaflet_\(software\)](https://en.wikipedia.org/wiki/Leaflet_(software)) (date of access: 13.05.2026).
5. Leaflet – an open-source JavaScript library for mobile-friendly interactive maps. URL: <https://leafletjs.com/> (date of access: 13.05.2026).
6. Google Firebase. URL: <https://en.wikipedia.org/wiki/Firebase> (date of access: 13.05.2026).
7. Technologies and frameworks used in web development. URL: <https://online.wlv.ac.uk/technologies-and-frameworks-used-in-web-development-an-explainer/> (date of access: 13.05.2026).
8. React (software). URL: [https://en.wikipedia.org/wiki/React_\(software\)](https://en.wikipedia.org/wiki/React_(software)) (date of access: 13.05.2026).
9. Why Vite. URL: <https://vite.dev/guide/why> (date of access: 13.05.2026).
10. Single Page Application. URL: https://en.wikipedia.org/wiki/Single-page_application (date of access: 13.05.2026).
11. React Hooks: How to Get Started. URL: <https://www.sitepoint.com/react-hooks/> (date of access: 13.05.2026).
12. Getting Started with Vite. URL: <https://www.geeksforgeeks.org/javascript/getting-started-with-vite/> (date of access: 13.05.2026).

13.05.2026).

13. What is Git. URL: <https://www.atlassian.com/git/tutorials/what-is-git?>
(date of access: 13.05.2026).

ДОДАТОК А
Текст програми

A.1 Текст файла main.jsx

```
import { createRoot } from "react-dom/client";
import App from "./App.jsx";
import "leaflet/dist/leaflet.css";

createRoot(document.getElementById("root")).render(<App />);
```

A.2 Текст файла App.jsx

```
// React imports
import { BrowserRouter as Router, Routes, Route } from "react-router-dom";
import { useState, useEffect } from "react";

// Components
import Navbar from "./components/Navbar.jsx";
import Footer from "./components/Footer.jsx";
import Scroll from "./components/Scroll.jsx";

// Pages
import Home from "./pages/Home.jsx";
import Catalog from "./pages/Catalog.jsx";
import About from "./pages/About.jsx";
import Favorites from "./pages/Favorites.jsx";
import NotFound from "./pages/NotFound.jsx";

// Styles
import "modern-normalize/modern-normalize.css";
import "./styles.css";

// Firebase
import { auth } from "./firebase.js";
import { onAuthStateChanged } from "firebase/auth";

// Component
function App() {
  const [user, setUser] = useState(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    const unsubscribe = onAuthStateChanged(auth, (currentUser) => {
      setUser(currentUser);
      setLoading(false);
    });
  });
```

```

    return () => unsubscribe();
  }, []);

  if (loading) {
    return (
      <div
        style={{
          fontSize: "30px",
          color: "white",
          fontWeight: "bold",
          textAlign: "center",
          marginTop: "150px",
        }}
      >
        Loading...
      </div>
    );
  }

  return (
    <Router>
      <div className="app-container">
        <Scroll />
        <Navbar user={user} />
        <main className="main-content">
          <Routes>
            <Route path="/" element={<Home />} />
            <Route path="/catalog" element={<Catalog user={user} />} />
            <Route path="/about" element={<About />} />
            <Route path="/favorites" element={<Favorites user={user} />} />
            <Route path="*" element={<NotFound />} />
          </Routes>
        </main>
        <Footer />
      </div>
    </Router>
  );
}

export default App;

```

A.3 Текст файла firebase.js

```

import { initializeApp } from "firebase/app";
import { getAuth, GoogleAuthProvider } from "firebase/auth";

```

```
import { getFirestore } from "firebase/firestore";

const firebaseConfig = {
  apiKey: "AIzaSyC2oOKGAB6iLuVjBtUpb4eHnwFfiv9J510",
  authDomain: "diplom-77427.firebaseio.com",
  projectId: "diplom-77427",
  storageBucket: "diplom-77427.firebaseio.com",
  messagingSenderId: "560631965709",
  appId: "1:560631965709:web:bb5dcebf650fe3baf44d44"
};

const app = initializeApp(firebaseConfig);

export const auth = getAuth(app);
export const db = getFirestore(app);
export const googleProvider = new GoogleAuthProvider();
```

A.4 Текст файла Home.jsx

```
// Imports
import { Link } from "react-router-dom";
import box from "../assets/box.svg";
import energy from "../assets/energy.svg";
import cloud from "../assets/cloud.svg";
import DBSeeder from "../components/DBSeeder.jsx";
import "../styles/homeStyles.css";

// Component
function Home() {
  return (
    <div className="home-container">
      <section className="hero-section">
        <h1 className="hero-title">Welcome to TrackingCatalog</h1>
        <p className="hero-subtitle">
          Your personal assistant for tracking products, managing shopping lists
          and monitoring prices. Everything in one convenient place.
        </p>
        <Link to="/catalog" className="hero-btn">
          Go to Catalog
        </Link>
      </section>

      <section className="features-section">
        <div className="feature-card">
```

```

    <div className="feature-icon"> <img src={box} alt="Single database" />
  </div>
    <h3>Single database</h3>
    <p>
      Save laptops, phones and other products in your personal collection.
    </p>
  </div>

  <div className="feature-card">
    <div className="feature-icon"> <img src={energy} alt="Fast access" />
  </div>
    <h3>Fast access</h3>
    <p>
      Instant search and filtering thanks to modern cloud technologies.
    </p>
  </div>

  <div className="feature-card">
    <div className="feature-icon"> <img src={cloud} alt="Cloud
synchronization" /> </div>
    <h3>Cloud synchronization</h3>
    <p>
      Your data is securely stored in Google Firebase and accessible from
      any device.
    </p>
  </div>
</section>
</div>
);
}

```

```
export default Home;
```

A.5 Текст файла About.jsx

```

// Styles
import "../styles/aboutStyles.css";

// Component
function About() {
  return (
    <div className="about-container">
      <section className="about-hero">
        <h1 className="about-title">About TrackingCatalog</h1>
        <p className="about-subtitle">

```

TrackingCatalog — modern web application for managing your collection of favorite products and convenient tracking.

</p>

</section>

<div className="about-content">

<div className="about-card">

<h2>Our goal</h2>

<p>

We strive to make the process of selecting, saving, and tracking products as simple and convenient as possible. No more keeping dozens of tabs open in your browser or using inconvenient notes - save and organize everything in one place!

</p>

</div>

<div className="about-card">

<h2>Technologies that was used</h2>

<p>

The project is built on the foundation of modern client-server web technologies:

</p>

<ul className="tech-list">

React (Vite) - Provides fast rendering and instant UI responsiveness.

Google Firebase - Reliable cloud architecture for database and authentication system.

Modern CSS - Adaptive design, Flexbox, and smooth animations without heavy libraries.


```

    </div>

    <div className="about-card">
      <h2>On Development</h2>
      <p>
        This application is being developed as part of a project in the
        field of "Web Development and Cloud Technologies". It demonstrates
        the practical application of modern approaches to creating scalable
        and secure web services.
      </p>
    </div>
  </div>
</div>
);
}

export default About;

```

A.6 Текст файла Catalog.jsx

```

// Import react and firebase
import { useState, useEffect, useMemo } from "react";
import { db } from "../firebase.js";
import {
  collection,
  getDocs,
  query,
  orderBy,
  doc,
  getDoc,
} from "firebase/firestore";

// Import components and styles
import ProductCard from "../components/ProductCard";
import Sidebar from "../components/Sidebar";
import SearchBar from "../components/SearchBar";
import "../styles/catalogStyles.css";

// Component
function Catalog({ user }) {
  const [products, setProducts] = useState([]);
  const [loading, setLoading] = useState(true);
  const [userFavorites, setUserFavorites] = useState([]);

  const [filters, setFilters] = useState({

```

```

searchQuery: "",
sortBy: "default",
category: "All",
warehouse: "All",
minPrice: "",
maxPrice: "",
});

useEffect(() => {
  const fetchProducts = async () => {
    try {
      const productsRef = collection(db, "products");
      const q = query(productsRef, orderBy("createdAt", "desc"));

      const productSnapshot = await getDocs(q);

      const productList = productSnapshot.docs.map((doc) => ({
        id: doc.id,
        ...doc.data(),
      }));

      setProducts(productList);
    } catch (error) {
      console.error("Error fetching products from Firebase:", error);
    } finally {
      setLoading(false);
    }
  };

  fetchProducts();
}, []);

useEffect(() => {
  const fetchFavorites = async () => {
    if (user) {
      try {
        const userRef = doc(db, "users", user.uid);
        const userSnap = await getDoc(userRef);

        if (userSnap.exists()) {
          setUserFavorites(userSnap.data().favorites || []);
        }
      } catch (error) {
        console.error("Error fetching user favorites:", error);
      }
    }
  };

```

```

    } else {
      setUserFavorites([]);
    }
  };

  fetchFavorites();
}, [user]);

const categories = useMemo(() => {
  return [...new Set(products.map((p) => p.category))];
}, [products]);

const warehouses = useMemo(() => {
  return [
    ...new Set(
      products.map((p) => p.location?.warehouseName).filter(Boolean),
    ),
  ];
}, [products]);

const filteredProducts = useMemo(() => {
  let result = [...products];

  if (filters.searchQuery.trim() !== "") {
    const query = filters.searchQuery.toLowerCase();
    result = result.filter(
      (p) =>
        p.name.toLowerCase().includes(query) ||
        p.category.toLowerCase().includes(query),
    );
  }

  if (filters.category !== "All") {
    result = result.filter((p) => p.category === filters.category);
  }

  if (filters.warehouse !== "All") {
    result = result.filter(
      (p) => p.location?.warehouseName === filters.warehouse,
    );
  }

  if (filters.minPrice !== "") {
    result = result.filter((p) => p.price >= Number(filters.minPrice));
  }
}

```

```

if (filters.maxPrice !== "") {
  result = result.filter((p) => p.price <= Number(filters.maxPrice));
}

if (filters.sortBy === "nameAsc") {
  result.sort((a, b) => a.name.localeCompare(b.name));
} else if (filters.sortBy === "nameDesc") {
  result.sort((a, b) => b.name.localeCompare(a.name));
} else if (filters.sortBy === "priceAsc") {
  result.sort((a, b) => a.price - b.price);
} else if (filters.sortBy === "priceDesc") {
  result.sort((a, b) => b.price - a.price);
}

return result;
}, [products, filters]);

if (loading) {
  return (
    <div
      className="catalog-container"
      style={{ justifyContent: "center", minHeight: "60vh" }}
    >
      <h2>Loading products...</h2>
    </div>
  );
}

return (
  <div className="catalog-container">
    <div className="catalog-header">
      <h1>ProductsCatalog</h1>
      <p>Find and track your favorite products.</p>
      <SearchBar
        searchQuery={filters.searchQuery}
        onChange={(value) =>
          setFilters((prev) => ({ ...prev, searchQuery: value })))
        >
        <onClear={() => setFilters((prev) => ({ ...prev, searchQuery: "" })))}
      />
    </div>

    <div
      className="catalog-content"
      style={{

```

```

        display: "flex",
        gap: "30px",
        width: "100%",
      }}
    >
    <Sidebar
      filters={filters}
      setFilters={setFilters}
      categories={categories}
      warehouses={warehouses}
    />

    <div className="products-grid" style={{ flexGrow: 1 }}>
      {filteredProducts.length === 0 ? (
        <p style={{ textAlign: "center", width: "100%", color: "#aaa" }}>
          No products match your filters.
        </p>
      ) : (
        filteredProducts.map((product) => (
          <ProductCard
            key={product.id}
            product={product}
            user={user}
            userFavorites={userFavorites}
          />
        ))
      )}
    </div>
  </div>
);
}

```

```
export default Catalog;
```

A.7 Текст файла NotFound.jsx

```

function NotFound() {
  return (
    <div className="not-found">
      <h1>404 Page Not Found</h1>
      <p>Sorry, the page you are looking for does not exist.</p>
    </div>
  );
}

```

```
export default NotFound;
```

A.8 Текст файла Favorites.jsx

```
// Import react and firebase
import { useState, useEffect } from "react";
import { db } from "../firebase";
import { doc, getDoc } from "firebase/firestore";

// Import components and styles
import ProductCard from "../components/ProductCard";
import "../styles/catalogStyles.css";

// Component
function Favorites({ user }) {
  const [favoriteProducts, setFavoriteProducts] = useState([]);
  const [loading, setLoading] = useState(true);
  const [favIds, setFavIds] = useState([]);

  const handleRemoveFavorite = (productId) => {
    setFavoriteProducts((prevProducts) =>
      prevProducts.filter((p) => p.id !== productId),
    );
    setFavIds((prevIds) => prevIds.filter((id) => id !== productId));
  };

  useEffect(() => {
    const fetchFavorites = async () => {
      if (!user) {
        setLoading(false);
        return;
      }

      try {
        const userRef = doc(db, "users", user.uid);
        const userSnap = await getDoc(userRef);

        if (userSnap.exists()) {
          const userData = userSnap.data();
          const favoriteIds = userData.favorites || [];

          setFavIds(favoriteIds);

          if (favoriteIds.length === 0) {
```

```

    setFavoriteProducts([]);
    setLoading(false);
    return;
  }

  const productPromises = favoriteIds.map((id) =>
    getDoc(doc(db, "products", id)),
  );

  const productDocs = await Promise.all(productPromises);

  const loadedProducts = productDocs
    .filter((docSnap) => docSnap.exists())
    .map((docSnap) => ({
      id: docSnap.id,
      ...docSnap.data(),
    }));

  setFavoriteProducts(loadedProducts);
}
} catch (error) {
  console.error("Error fetching favorite products:", error);
} finally {
  setLoading(false);
}
};

fetchFavorites();
}, [user]);

if (loading) {
  return (
    <div
      className="catalog-container"
      style={{ justifyContent: "center", minHeight: "60vh" }}
    >
      <h2>Loading favorites...</h2>
    </div>
  );
}

if (!user) {
  return (
    <div
      className="catalog-container"

```

```

    style={{ justifyContent: "center", minHeight: "60vh" }}
  >
    <h2>Please sign in to see your favorites.</h2>
  </div>
);
}

return (
  <div className="catalog-container">
    <div className="catalog-header">
      <h1>Your Favorites</h1>
      <p>Products you saved for later.</p>
    </div>

    {favoriteProducts.length === 0 ? (
      <p
        style={{
          textAlign: "center",
          color: "#aaa",
          fontSize: "1.2rem",
          marginTop: "40px",
        }}
      >
        You haven't added any products to favorites yet
      </p>
    ) : (
      <div className="products-grid">
        {favoriteProducts.map((product) => (
          <ProductCard
            key={product.id}
            product={product}
            user={user}
            userFavorites={favIds}
            onRemove={handleRemoveFavorite}
          />
        ))}
      </div>
    )}
  </div>
);
}

export default Favorites;

```

A.9 Текст файла AuthModule.jsx

```
// Imports
import { useState } from "react";
import {
  createUserWithEmailAndPassword,
  signInWithPopup,
  signInWithEmailAndPassword,
  updateProfile,
  signOut,
} from "firebase/auth";

import { doc, setDoc, getDoc } from "firebase/firestore";
import { auth, db, googleProvider } from "../firebase.js";

import "../styles/authStyles.css";
import eyeopen from "../assets/eyeopen.svg";
import eyeclose from "../assets/eyeclose.svg";

// Component
function AuthModule({ user, closeMenu }) {
  const [name, setName] = useState("");
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [showPassword, setShowPassword] = useState(false);

  const [error, setError] = useState("");

  const handleRegister = async () => {
    try {
      setError("");
      console.log("Begin registration...");

      const userCredential = await createUserWithEmailAndPassword(
        auth,
        email,
        password,
      );
      const currentUser = userCredential.user;

      await updateProfile(currentUser, { displayName: name });

      await setDoc(doc(db, "users", currentUser.uid), {
        name: name,
        email: currentUser.email,
```

```

    favorites: [],
    role: "user",
    createdAt: new Date().toISOString(),
  });

  console.log("Success! User registered and added to the database.");
  closeMenu();
} catch (err) {
  console.error("Firebase error:", err);
  setError("Registration error: " + err.message);
}
};

const handleLogin = async (e) => {
  e.preventDefault();
  try {
    setError("");
    console.log("Trying to log in as:", email);

    const userCredential = await signInWithEmailAndPassword(
      auth,
      email,
      password,
    );

    console.log("Success! Welcome back,", userCredential.user.email);

    setEmail("");
    setPassword("");
  } catch (err) {
    console.error("Login error:", err);

    if (err.code === "auth/invalid-credential") {
      setError("Invalid email or password.");
    } else {
      setError("Error: " + err.message);
    }
  }
};

const handleGoogleSignIn = async () => {
  try {
    setError("");

    const result = await signInWithPopup(auth, googleProvider);

```

```

const currentUser = result.user;

const userRef = doc(db, "users", currentUser.uid);
const userSnap = await getDoc(userRef);

if (!userSnap.exists()) {
  await setDoc(userRef, {
    email: currentUser.email,
    name: currentUser.displayName,
    favorites: [],
    role: "user",
    createdAt: new Date().toISOString(),
  });
}

console.log("Success! Logged in with Google:", currentUser.displayName);
} catch (err) {
  console.error(err);
  setError("Google authentication error: " + err.message);
}
};

const handleLogout = async () => {
  try {
    await signOut(auth);
    console.log("Success! Logged out.");
  } catch (err) {
    console.error("Error during logout:", err);
  }
};

return (
  <div className="dropdown" onClick={(e) => e.stopPropagation()}>
    {user ? (
      <div style={{ display: "flex", flexDirection: "column", gap: "10px" }}>
        <div className="profile">
          Profile
          <div className="flex-profile">
            <p>Email: </p>
            <p>{user.email}</p>
            <p>Name: </p>
            <p>{user.displayName}</p>
          </div>
        </div>
        <div className="logout-btn" onClick={handleLogout}>

```

```

    Logout
  </div>
</div>
) : (
  <form className="auth-form" onSubmit={handleLogin}>
    {error && (
      <p style={{ color: "#ff4d4d", fontSize: "12px", margin: 0 }}>
        {error}
      </p>
    )}
  </form>

  <input
    name="name"
    type="text"
    placeholder="Name"
    value={name}
    onChange={(e) => setName(e.target.value)}
    maxLength={10}
  />

  <input
    name="email"
    type="email"
    placeholder="Email"
    value={email}
    onChange={(e) => setEmail(e.target.value)}
    required
  />

  <div className="password-wrapper">
    <input
      name="password"
      type={showPassword ? "text" : "password"}
      placeholder="Password"
      value={password}
      onChange={(e) => setPassword(e.target.value)}
      required
      maxLength={10}
    />
    <button
      type="button"
      className="toggle-password-btn"
      onClick={() => setShowPassword(!showPassword)}
    >

```

```

      {showPassword ? <img style={{width: "20px", height: "20px"}}
src={eyeclose} alt="Hide password" /> : <img style={{width: "20px", height:
"20px"}} src={eyeopen} alt="Show password" />}
    </button>
  </div>

  <button type="submit">Sign In</button>

  <button type="button" onClick={handleRegister}>
    Register
  </button>

  <hr />

  <button
    type="button"
    className="google-btn"
    onClick={handleGoogleSignIn}
  >
    Sign in with Google
  </button>
</form>
))
</div>
);
}

```

```
export default AuthModule;
```

A.10 Текст файла ProductCard.jsx

```

// Imports
import { useState, useEffect } from "react";
import { createPortal } from "react-dom";
import { db } from "../firebase";
import { doc, setDoc, arrayUnion, arrayRemove } from "firebase/firestore";
import Map from "../Map.jsx";
import fav from "../assets/fav.svg";
import notFav from "../assets/notFav.svg";
import "../styles/catalogStyles.css";

// Component
function ProductCard({ product, user, userFavorites = [], onRemove }) {
  const [isFavorite, setIsFavorite] = useState(
    userFavorites.includes(product.id),

```

```

);
const [isProcessing, setIsProcessing] = useState(false);
const [isTracking, setIsTracking] = useState(false);

useEffect(() => {
  setIsFavorite(userFavorites.includes(product.id));
}, [userFavorites, product.id]);

const toggleFavorite = async () => {
  if (!user) {
    alert("Sign in to manage favorites!");
    return;
  }

  if (isProcessing) return;
  setIsProcessing(true);

  const userRef = doc(db, "users", user.uid);
  const previousState = isFavorite;

  try {
    setIsFavorite(!previousState);

    if (previousState) {
      await setDoc(
        userRef,
        {
          favorites: arrayRemove(product.id),
        },
        { merge: true },
      );

      if (onRemove) {
        onRemove(product.id);
      }
    } else {
      await setDoc(
        userRef,
        {
          favorites: arrayUnion(product.id),
        },
        { merge: true },
      );
    }
  } catch (error) {

```

```

    console.error("Error while updating favorites:", error);
    setIsFavorite(previousState);
  } finally {
    setIsProcessing(false);
  }
};

return (
  <div className="product-card">
    <div className="product-image-container">
      <img src={product.image} alt={product.name} className="product-img" />
      <p className="product-description">{product.description}</p>
    </div>

    <div className="product-info">
      <h3 className="product-title">{product.name}</h3>
      <p className="product-category">{product.category}</p>

      <div className="product-bottom">
        <span className="product-price">{product.price} $</span>

        <div className="action-buttons">
          <button
            className={`fav-btn ${isFavorite ? "active" : ""}`}
            onClick={toggleFavorite}
            title="Add to favorites"
          >
            {isFavorite ? (
              <img src={fav} alt="Unfavorite" />
            ) : (
              <img src={notFav} alt="Favorite" />
            )}
          </button>
          <button className="track-btn" onClick={() => setIsTracking(true)}>
            Track
          </button>
        </div>
      </div>
    </div>

    {isTracking &&
      createPortal(
        <div className="modal-overlay" onClick={() => setIsTracking(false)}>
          <div className="modal-content" onClick={(e) => e.stopPropagation()}>
            <div className="modal-header">

```

```

    <h3>Tracking: {product.name}</h3>
    <button
      className="close-modal-btn"
      onClick={() => setIsTracking(false)}
    >
      &times;
    </button>
  </div>

  <p className="modal-warehouse">
    Available at: {" "}
    <strong>
      {product.location?.warehouseName || "Main Central Hub"}
    </strong>
  </p>

  <div className="modal-map-wrapper">
    <Map location={product.location} productName={product.name} />
  </div>
</div>
document.body,
  )}
</div>
);
}

export default ProductCard;

```

A.11 Текст файла NavBar.jsx

```

// Imports
import { useState, useEffect } from "react";
import { Link } from "react-router-dom";
import { useNavigate } from "react-router";
import AuthModule from "../AuthModule";
import Favorites from "../pages/Favorites";
import "../styles/navStyles.css";
import favIcon from "../assets/fav.svg";
import logo from "../assets/logo.svg";

// Component
function Navbar({ user }) {
  const items = ["Home", "Catalog", "About"];
  const logoItems = ["TrackingCatalog", "TR"]

```

```

let navigate = useNavigate();

const [isAuthOpen, setIsAuthOpen] = useState(false);
const [isMobileMenuOpen, setIsMobileMenuOpen] = useState(false);
const [isMobile, setIsMobile] = useState(window.innerWidth < 400);

const handleFavoritesClick = () => {
  if (!user) {
    alert("Sign in to manage favorites!");
  } else {
    navigate("/favorites");
  }
};

useEffect(() => {
  const handleResize = () => {
    setIsMobile(window.innerWidth < 400);
  };

  window.addEventListener('resize', handleResize);

  return () => window.removeEventListener('resize', handleResize);
}, []);

return (
  <nav>
    <div className="flex-container">
      <div
        className="logo"
        style={{ cursor: "pointer" }}
        onClick={() => {
          navigate("/");
        }}
      >
        {" "}
        <img src={logo} alt="Logo" /> <p>{isMobile ? logoItems[1] :
logoItems[0]}</p>{" "}
      </div>

      <div className="mobile-controls">
        <img
          src={favIcon}
          alt="Favorites"
          className="block-fav"
          onClick={handleFavoritesClick}

```

```

/>

<div
  className={`burger-icon ${isMobileMenuOpen ? "active" : ""}`}
  onClick={() => setIsMobileMenuOpen(!isMobileMenuOpen)}
>
  <span></span>
  <span></span>
  <span></span>
</div>
</div>

<div className="container">
  <ul className={`nav-links ${isMobileMenuOpen ? "active" : ""}`}>
    {items.map((item) => (
      <li key={item} onClick={() => setIsMobileMenuOpen(false)}>
        <Link to={item === "Home" ? "/" : `/${item.toLowerCase()}`}>
          {item}
        </Link>
      </li>
    ))}

    <li
      className="list-fav"
      style={{
        backgroundColor: "transparent",
        width: "50px",
        height: "50px",
        padding: "5px",
        flexShrink: 0,
      }}
      onClick={handleFavoritesClick}
    >
      <img
        src={favIcon}
        alt="Favorites"
        style={{ width: "100%", height: "100%", objectFit: "contain" }}
      />
    </li>

    <li onClick={() => setIsAuthOpen(!isAuthOpen)}>
      {" "}
      {user ? user.displayName : "Login"}
    </li>
  </ul>

```

```

    {isAuthOpen && (
      <AuthModule user={user} closeMenu={() => setIsAuthOpen(false)} />
    )}
  </div>
</div>
</nav>
);
}

export default Navbar;

```

A.12 Текст файла Footer.jsx

```

// Imports
import { Link } from "react-router-dom";
import "../styles/footerStyles.css";

// Component
function Footer() {
  const currentYear = new Date().getFullYear();

  return (
    <footer className="footer-container">
      <div className="footer-content">
        <div className="footer-section brand">
          <h2 className="footer-logo">TrackingCatalog</h2>
          <p>
            Your ultimate destination to find, track, and manage your favorite
            tech products. Built with React and Firebase.
          </p>
        </div>

        <div className="footer-section links">
          <h3>Quick Links</h3>
          <ul>
            <li>
              <Link to="/">Home</Link>
            </li>
            <li>
              <Link to="/catalog">Catalog</Link>
            </li>
            <li>
              <Link to="/favorites">Favorites</Link>
            </li>
            <li>

```

```

    <Link to="/about">About Us</Link>
  </li>
</ul>
</div>

<div className="footer-section contact">
  <h3>Contact Us</h3>
  <p>Email: ivan.k7r@gmail.com</p>
  <p>Phone: +45 71 89 18 08</p>
  <div className="social-links">
    <a href="https://github.com/" target="_blank" rel="noreferrer">
      GitHub
    </a>
    <a href="https://linkedin.com/" target="_blank" rel="noreferrer">
      LinkedIn
    </a>
  </div>
</div>
</div>

<div className="footer-bottom">
  <p>&copy; {currentYear} TrackingCatalog. All rights reserved.</p>
</div>
</footer>
);
}

export default Footer;

```

A.13 Текст файла Map.jsx

```

// Imports
import { useState } from "react";
import { MapContainer, TileLayer, Marker, Popup, useMap } from "react-leaflet";
import L from "leaflet";
import "leaflet-routing-machine";
import { useEffect } from "react";

// Extra routing component
function RoutingMachine({ userPos, warehousePos }) {
  const map = useMap();

  useEffect(() => {
    if (!map || !userPos || !warehousePos) return;

```

```

const routingControl = L.Routing.control({
  waypoints: [
    L.latLng(userPos[0], userPos[1]),
    L.latLng(warehousePos[0], warehousePos[1]),
  ],
  lineOptions: {
    styles: [{ color: "#4caf50", weight: 5 }],
  },
  addWaypoints: false,
  draggableWaypoints: false,
  fitSelectedRoutes: true,
  show: false,
}).addTo(map);

return () => map.removeControl(routingControl);
}, [map, userPos, warehousePos]);

return null;
}

// Map Component
function Map({ location, productName }) {
  const [userLocation, setUserLocation] = useState(null);
  const [showRoute, setShowRoute] = useState(false);

  const warehousePos = [location?.lat || 55.6761, location?.lng || 12.5683];

  const findMyLocation = () => {
    if (!navigator.geolocation) {
      alert("Geolocation is not supported by your browser");
      return;
    }

    navigator.geolocation.getCurrentPosition(
      (position) => {
        const { latitude, longitude } = position.coords;
        setUserLocation([latitude, longitude]);
        setShowRoute(true);
      },
      () => {
        alert("Unable to retrieve your location. Check permissions.");
      },
    );
  };
}

```

```

return (
  <div style={{ position: "relative", height: "400px", width: "100%" }}>
    <button onClick={findMyLocation} style={mapButtonStyle}>
      {showRoute ? "Showing Route" : "Route to Warehouse"}
    </button>

    <MapContainer
      center={warehousePos}
      zoom={13}
      style={{ height: "100%", width: "100%", borderRadius: "12px" }}
    >
      <TileLayer url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png" />

      <Marker position={warehousePos}>
        <Popup>Warehouse: {productName}</Popup>
      </Marker>

      {userLocation && (
        <Marker position={userLocation}>
          <Popup>You are here</Popup>
        </Marker>
      )}

      {showRoute && userLocation && (
        <RoutingMachine userPos={userLocation} warehousePos={warehousePos}
      />
      )}
    </MapContainer>
  </div>
);
}

// Button styles
const mapButtonStyle = {
  position: "absolute",
  top: "10px",
  right: "10px",
  zIndex: 1000,
  backgroundColor: "#4caf50",
  color: "white",
  border: "none",
  padding: "10px 15px",
  borderRadius: "8px",
  fontWeight: "bold",
  cursor: "pointer",

```

```
    boxShadow: "0 4px 6px rgba(0,0,0,0.3)",
  };
```

```
export default Map;
```

A.14 Текст файла DBSeeder.jsx

```
// Imports
import { useState } from "react";
import { db } from "../firebase.js";
import { collection, addDoc } from "firebase/firestore";

// Component
function DbSeeder() {
  const [isUploading, setIsUploading] = useState(false);
  const [progress, setProgress] = useState(0);

  const seedDatabase = async () => {
    if (
      !window.confirm(
        "Are you sure you want to add 100 products to the database?",
      )
    )
      return;

    setIsUploading(true);
    const productsCollection = collection(db, "products");

    const names = [
      "Ultra",
      "Pro",
      "Max",
      "Lite",
      "Elite",
      "Master",
      "Gamer",
      "Office",
    ];
    const categories = [
      "Laptops",
      "Phones",
      "Tablets",
      "Audio",
      "Gaming",
      "Accessories",
```

```

];

// Warehouses locations
const warehouses = [
  { lat: 55.6761, lng: 12.5683, name: "Copenhagen Central Hub" },
  { lat: 55.6867, lng: 12.5701, name: "Nørrebro Storage" },
  { lat: 55.6621, lng: 12.5415, name: "Vesterbro Depot" },
  { lat: 55.6984, lng: 12.5794, name: "Østerbro Facility" },
  { lat: 55.6541, lng: 12.605, name: "Amager Distribution" },
  { lat: 55.6795, lng: 12.5298, name: "Frederiksberg Center" },
  { lat: 55.7204, lng: 12.4746, name: "Herlev Stock" },
  { lat: 55.6558, lng: 12.3855, name: "Glostrup Warehouse" },
  { lat: 55.7371, lng: 12.3965, name: "Ballerup Electronics Unit" },
  { lat: 55.6262, lng: 12.4781, name: "Hvidovre Dispatch" },
];

const getImageForCategory = (category) => {
  switch (category) {
    case "Laptops":
      return "/laptop.jpg";
    case "Phones":
      return "/iphone.jpg";
    case "Tablets":
      return "/tablet.jpg";
    case "Audio":
      return "/headphones.jpg";
    case "Gaming":
      return "/gamepad.jpg";
    case "Accessories":
      return Math.random() > 0.5 ? "/mouse.jpg" : "/keyboard.jpg";
    default:
      return "/laptop.jpg";
  }
};

try {
  for (let i = 1; i <= 100; i++) {
    const randomCategory =
      categories[Math.floor(Math.random() * categories.length)];
    const randomName = `${names[Math.floor(Math.random() * names.length)]}
    ${randomCategory} v${i}`;

    const randomWarehouse =
      warehouses[Math.floor(Math.random() * warehouses.length)];
  }
}

```

```

const newProduct = {
  name: randomName,
  price: Math.floor(Math.random() * (2000 - 50 + 1)) + 50,
  category: randomCategory,
  image: getImageForCategory(randomCategory),
  description: `Perfect choice in the ${randomCategory} category. The
${randomName} model combines performance and style.`,
  createdAt: new Date(),
  location: {
    lat: randomWarehouse.lat,
    lng: randomWarehouse.lng,
    warehouseName: randomWarehouse.name,
  },
};

await addDoc(productsCollection, newProduct);
setProgress(i);
}
alert("Done! 100 products successfully added with locations and images.");
} catch (error) {
  console.error("Error during seeding:", error);
  alert("An error occurred while uploading data.");
} finally {
  setIsUploading(false);
  setProgress(0);
}
};

return (
  <div style={seederStyle}>
    <h3>Developer Tool</h3>
    <p>
      Click the button below to automatically fill Firestore with products.
    </p>
    <button onClick={seedDatabase} disabled={isUploading}
style={buttonStyle}>
      {isUploading ? `Uploading: ${progress}%` : "Upload 100 Products"}
    </button>
    {isUploading && (
      <p style={{ color: "#4caf50" }}>Please do not close the page...</p>
    )}
  </div>
);
}

```

```
const seederStyle = {
  backgroundColor: "#242424",
  border: "2px dashed #4caf50",
  borderRadius: "12px",
  padding: "20px",
  margin: "40px auto",
  maxWidth: "400px",
  textAlign: "center",
};
```

```
const buttonStyle = {
  backgroundColor: "#4caf50",
  color: "white",
  border: "none",
  padding: "12px 24px",
  borderRadius: "8px",
  fontSize: "1rem",
  fontWeight: "bold",
  cursor: "pointer",
  marginTop: "10px",
  transition: "opacity 0.3s",
};
```

```
export default DbSeeder;
```

A.15 Текст файла SearchBar.jsx

```
// Imports
import "../styles/searchStyles.css";
import x from "../assets/x.svg";

// Component
function SearchBar({ searchQuery, onSearchChange, onClear }) {
  return (
    <div className="search-bar-container">
      <svg
        className="search-icon"
        xmlns="http://www.w3.org/2000/svg"
        viewBox="0 0 24 24"
        fill="none"
        stroke="currentColor"
        strokeWidth="2"
        strokeLinecap="round"
        strokeLinejoin="round"
      >
```

```

    <circle cx="11" cy="11" r="8"></circle>
    <line x1="21" y1="21" x2="16.65" y2="16.65"></line>
  </svg>

  <input
    id="searchID"
    type="text"
    className="search-input"
    placeholder="Search for products, categories..."
    value={searchQuery}
    onChange={(e) => onSearchChange(e.target.value)}
  />

  {searchQuery.length > 0 && (
    <button
      className="clear-search-btn"
      onClick={onClear}
      title="Clear search"
    >
      <img src={x} alt="X" />
    </button>
  )}
</div>
);
}

```

```
export default SearchBar;
```

A.16 Текст файла SideBar.jsx

```

// Imports
import "../styles/sidebarStyles.css";

//Component
function Sidebar({ filters, setFilters, categories, warehouses }) {
  const handleChange = (e) => {
    const { name, value } = e.target;
    setFilters((prev) => ({
      ...prev,
      [name]: value,
    }));
  };

  return (
    <aside className="sidebar">

```

```
<h3>Filters</h3>
```

```
<div className="filter-group">
  <label htmlFor="sortBy">Sort By</label>
  <select
    id="sortBy"
    name="sortBy"
    value={filters.sortBy}
    onChange={handleChange}
  >
    <option value="default">Newest First</option>
    <option value="nameAsc">Name (A-Z)</option>
    <option value="nameDesc">Name (Z-A)</option>
    <option value="priceAsc">Price (Low to High)</option>
    <option value="priceDesc">Price (High to Low)</option>
  </select>
</div>
```

```
<div className="filter-group">
  <label htmlFor="category">Category</label>
  <select
    id="category"
    name="category"
    value={filters.category}
    onChange={handleChange}
  >
    <option value="All">All Categories</option>
    {categories.map((cat) => (
      <option key={cat} value={cat}>
        {cat}
      </option>
    ))}
  </select>
</div>
```

```
<div className="filter-group">
  <label htmlFor="warehouse">Warehouse</label>
  <select
    id="warehouse"
    name="warehouse"
    value={filters.warehouse}
    onChange={handleChange}
  >
    <option value="All">All Warehouses</option>
    {warehouses.map((wh) => (
```

```

    <option key={wh} value={wh}>
      {wh}
    </option>
  )))
</select>
</div>

<div className="filter-group price-filter">
  <label htmlFor="minPrice">Price Range ($)</label>
  <div className="price-inputs">
    <input
      id="minPrice"
      type="number"
      name="minPrice"
      value={filters.minPrice}
      onChange={handleChange}
      placeholder="Min"
      min="0"
    />
    <span>-</span>
    <input
      id="maxPrice"
      type="number"
      name="maxPrice"
      value={filters.maxPrice}
      onChange={handleChange}
      placeholder="Max"
      min="0"
      aria-label="Maximum Price"
    />
  </div>
</div>

<button
  className="reset-btn"
  onClick={() =>
    setFilters({
      sortBy: "default",
      category: "All",
      warehouse: "All",
      minPrice: "",
      maxPrice: "",
    })
  }
/>

```

```
        Reset Filters
      </button>
    </aside>
  );
}

export default Sidebar;
```

ДОДАТОК Б
Слайди презентації

Міністерство освіти і науки України
Національний університет "Запорізька політехніка"

Дипломна кваліфікаційна робота бакалавра
на тему: "Створення програмної системи на основі
хмарних обчислень для візуалізації та дистрибуції
товарів з використанням геоінформаційних сервісів"

Виконав : ст. гр. КНТ - 212

Іван КУЧЕР

Керівник к.т.н., доц. каф. ПЗ

Ольга ГЛАДКОВА

Рисунок Б.1 – Слайд 1

Предмет та мета роботи

Об'єкт дослідження – моделі, методи, алгоритми та програмні засоби створення вебзастосунків із використанням геоінформаційних і хмарних технологій.

Предмет дослідження – методи та програмні засоби інтеграції геоінформаційних сервісів і хмарних платформ у вебзастосунках.

Мета роботи – підвищення ефективності управління розподілом товарів та оптимізація їх візуалізації шляхом розробки хмарного вебзастосунку з інтеграцією геоінформаційних сервісів.

Рисунок Б.2 – Слайд 2

Взаємодія технологій у вебзастосунку

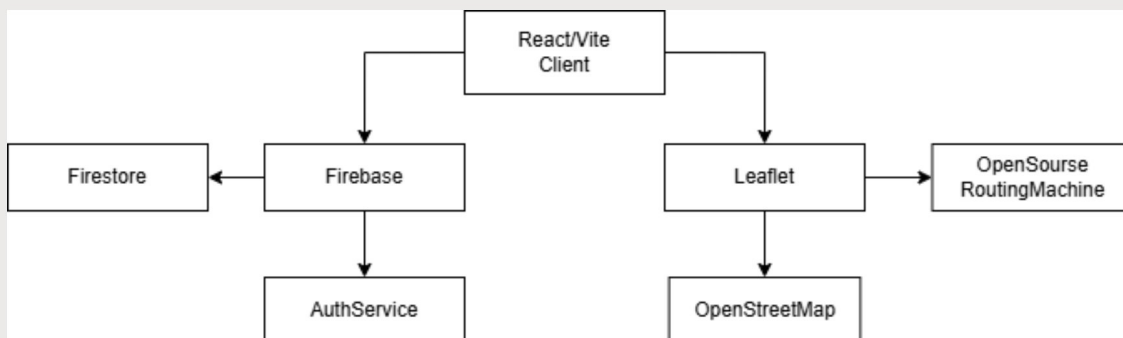


Рисунок Б.3 – Слайд 3

Робота бібліотеки React

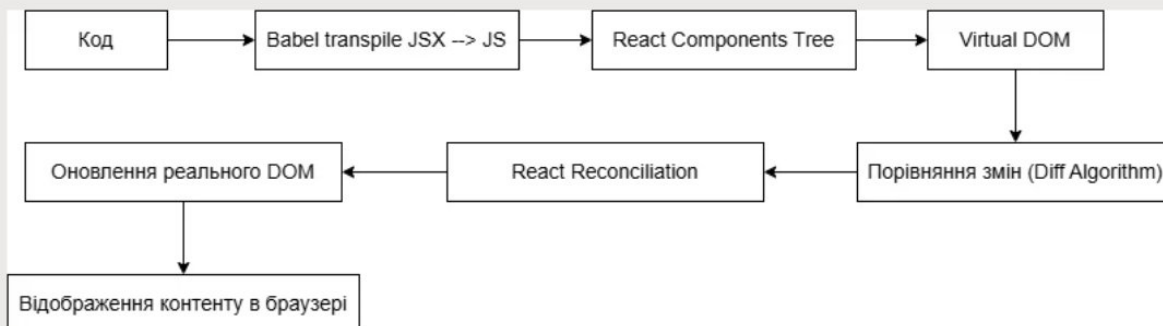


Рисунок Б.4 – Слайд 4

Структура бібліотеки router

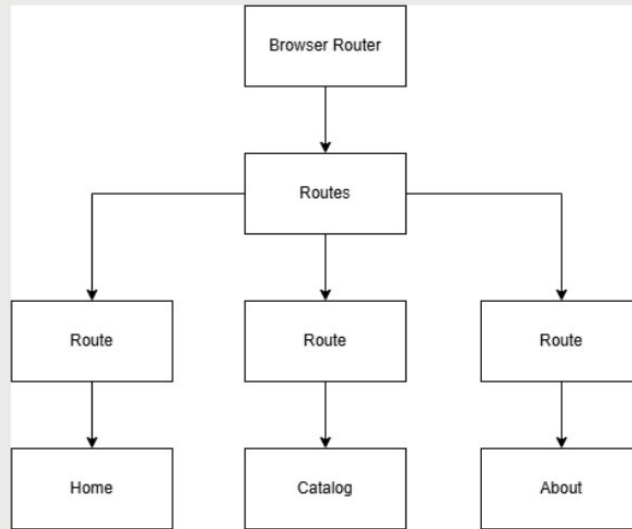


Рисунок Б.5 – Слайд 5

Структура бази даних Firestore

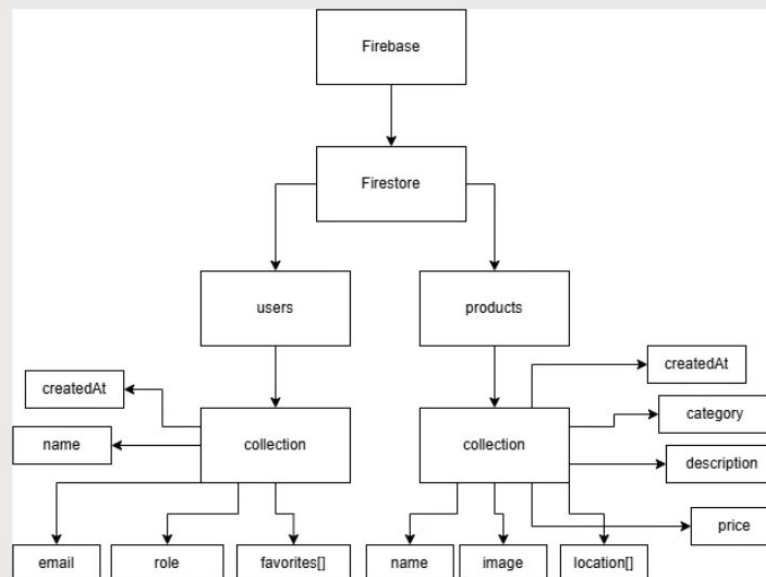


Рисунок Б.6 – Слайд 6

Діаграма послідовності дій користувача у вебзастосунку



Рисунок Б.7 – Слайд 7

Діаграма станів асинхронного запиту до бази даних Firestore

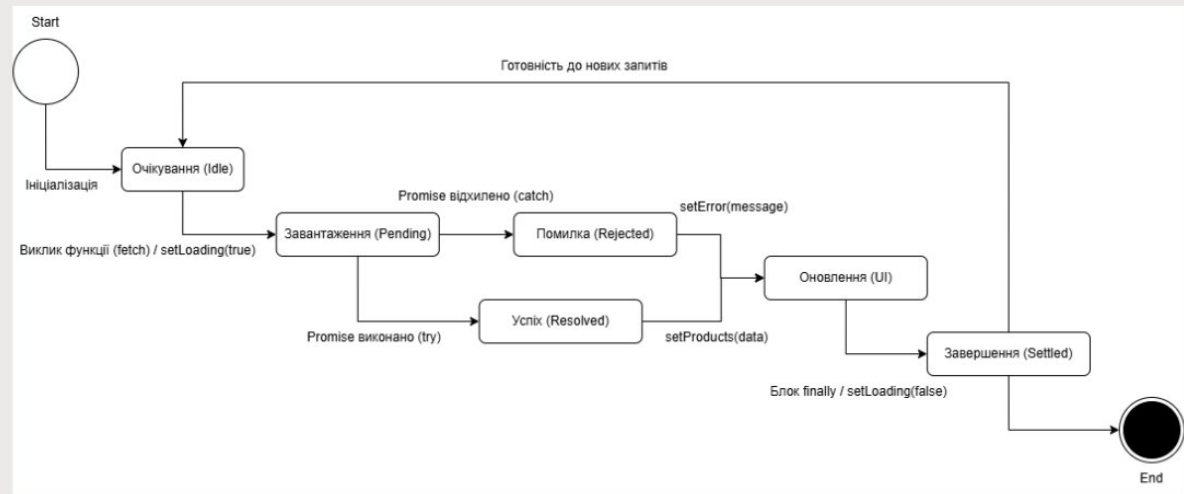


Рисунок Б.8 – Слайд 8

Основний функціонал вебзастосунку

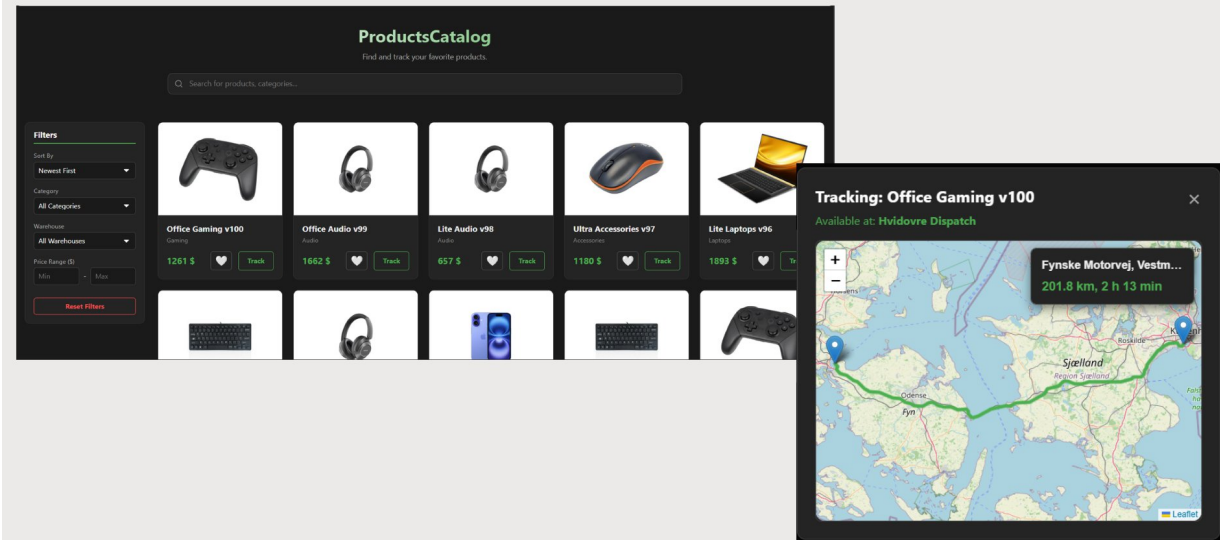


Рисунок Б.9 – Слайд 9

Основного функціонал вебзастосунку

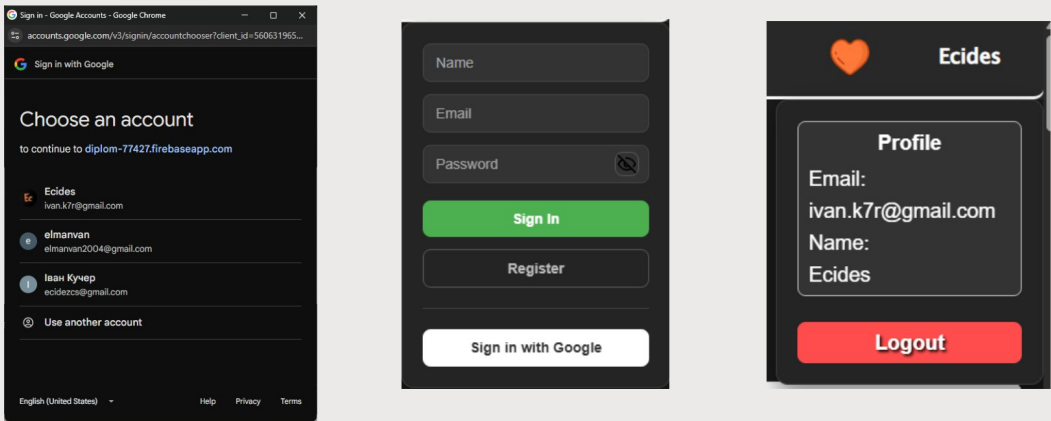


Рисунок Б.10 – Слайд 10

ВИСНОВКИ

У результаті виконання дипломної кваліфікаційної роботи було проведено комплексне дослідження методів проєктування, оптимізації та програмної реалізації хмарних інформаційних систем для візуалізації та розподілу товарів. Розроблено клієнтський вебзастосунок – каталог для користувачів, які можуть сортувати та переглядати представлені товари.

Рисунок Б.11 – Слайд 11