МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА КРОСПЛАТФОРМЕНОГО ДОДАТКА ДЛЯ ПЛАНУВАННЯ ЗАДАЧ З ВИКОРИСТАННЯМ REACT NATIVE ТА FIREBASE»**

Виконав: студент 4 курсу, групи 6.1219-1пі
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
А.В. Єрохін
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, доцент, к.ф.-м.н. Мильцев О.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент завідувач кафедри фундаментальної та прикладної математики, професор, д.т.н. Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ 07 ” 02 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Єрохіну Артуру Вадимовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка кросплатформеного додатка для планування задач з використанням React Native та Firebase

керівник роботи Мильцев Олександр Михайлович, к.ф.-м.н., доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 26 » 01 2023 року № 102-с

2. Строк подання студентом роботи 07.06.2023 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Розробка кросплатформеного додатка для планування задач з використанням React Native та Firebase.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою докладу

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 07.02.2023 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	08.02.2023	
2.	Збір вихідних даних.	22.02.2023	
3.	Обробка методичних та теоретичних	15.03.2023	
	джерел.		
4.	Розробка першого та другого розділу.	20.04.2023	
5.	Розробка третього розділу.	15.05.2023	
6.	Оформлення та нормоконтроль	01.06.2023	
	кваліфікаційної роботи.		
7.	Захист кваліфікаційної роботи.	21.06.2023	

Студент _____ А.В. Єрохін
(підпис) (ініціали та прізвище)

Керівник роботи _____ О.М. Мильцев
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка кросплатформеного додатка для планування задач з використанням React Native та Firebase»: 63 с., 26 рис., 12 джерел.

КРОСПЛАТФОРМЕННИЙ ДОДАТОК, ANDROID, REACT NATIVE, REACT NAVIGATION, TYPESCRIPT, WEB, ZUSTAND.

Об’єкт дослідження – кросплатформенна система планування та автоматизації задач, пов’язані зі змінами ціни та наявністю товарів на ресурсах продажу (магазини, маркетплейси).

Мета роботи: розробка системи, яка автоматично визначає потрібну ціну товару, враховуючи закупівельну ціну, відсоток ресурсу продажу та відсоток прибутку, а також передає дані про наявність товарів на складі до системи джерел продажу.

Методи дослідження – аналіз, проєктування, реалізація, тестування.

У даній роботі досліджується система автоматизації процесів зміни ціни та наявності товарів на ресурсах продажу, таких як магазини та маркетплейси. Метою розробки є створення системи, що спрощує управління цінами та наявністю товарів, автоматично визначає оптимальну ціну, враховуючи ряд параметрів, а також передає дані про наявність товарів на складах до відповідних джерел продажу.

Розроблена система базується на сучасних технологіях та інструментах, зокрема React Native, TypeScript, Zustand, React Navigation, Tanstack Query, Tanstack Table та .NET.

SUMMARY

Bachelor's qualifying paper «Development of a Cross-platform Application for Tasks Planning using React Native and Firebase»: 63 pages, 26 figures, 12 references.

ANDROID, CROSS-PLATFORM APPLICATION, REACT NATIVE, REACT NAVIGATION, TYPESCRIPT, WEB, ZUSTAND.

The object of the study is a planning and automation system for tasks related to price changes and product availability on sales resources (shops, marketplaces).

The aim of the study is a development of a system that automatically determines the required price of the product, considering the purchase price, the percentage of the sales resource, and the profit percentage. This system also transfers data about the availability of goods on the shelf to the sales system.

Methods of research are analysis, design, implementation, testing.

This paper investigates a system for automating the processes of changing the price and availability of goods on the sales resource. The aim is to create a system that simplifies the management of prices and availability of goods, automatically determining the optimal price based on the purchase price, the percentage of the sales resource, and the profit percentage. This system also handles the transfer of data about the availability of goods on the warehouses to the respective sales systems.

The system is based on modern technologies and tools, including React Native, TypeScript, Zustand, React Navigation, Tanstack Query, Tanstack Table and .NET.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

DOM	Document Object Model
YML	Тип файлу YML (YAML Application Markup Language)
JSON	JavaScript Object Notation
API	Application Programming Interface
JS	Javascript
HTML	HyperText Markup Language
Піксель	Одиниця розмірності елемента у цифровому зображенні

ВСТУП

В сучасному світі, глобалізація та діджиталізація ринку роблять використання технологій все більш актуальним для багатьох галузей, включаючи сферу роздрібної торгівлі. Доступ до товарів, що пропонуються по всьому світу, стає все більш простим і привабливим для споживачів. Однак ця багатоманітність може створити виклики для торгових підприємств, що намагаються конкурувати та вирізнитися серед великої кількості пропозицій.

Одним із способів вирішення цієї проблеми є використання автоматизації для забезпечення найкращого моніторингу цін та доступності продуктів, який може бути забезпечений за допомогою використання спеціалізованих програмних рішень. Саме в цьому контексті з'являється наша мета – розробка автоматизованої системи для моніторингу цін і доступності товарів на ринку.

Актуальність дослідження полягає в потребі роздрібних компаній і магазинів в покращенні ефективності своїх дій, надаючи їм можливість реагувати на зміни цін та доступності продуктів на ринку в режимі реального часу.

На основі цього ми можемо виділити наступні цілі та завдання нашого дослідження:

– сформулювати вимоги до системи;
– спроєктувати та побудувати архітектуру системи;
– реалізувати систему планування задач та автоматизувати зміни цін та доступності товарів;
– протестувати роботу системи.

Об'єкт дослідження: процес зміни цін та доступності товарів на ринку.

Предмет дослідження: створення кросплатформенної системи для планування задач, які автоматизують моніторинг цін та доступності товарів.

Методи дослідження: аналіз, проєктування, реалізація, тестування.

Перший розділ присвячено збору та аналізу вимог до системи, огляду інструментів розробки та опису системи. У другому розділі розглянуто етапи

проєктування системи, наведено детальний опис прецедентів. Третій розділ присвячено реалізації та тестуванню роботи системи, наведено детальне керівництво користувача, яке описує процес роботи у системі.

1 ТЕХНІЧНЕ ЗАВДАННЯ

1.1 Терміни та визначення

Кросплатформеність – це концепція або властивість програмного забезпечення або додатків, яка означає їх здатність працювати на різних платформах або операційних системах з мінімальними змінами в коді або логіці додатку. Це означає, що ви можете розробляти одну версію додатку та запускати її на різних платформах, таких як iOS, Android, веб-браузери тощо, замість того, щоб писати окремий код для кожної платформи.

Нативний – це термін, який використовується в контексті програмного забезпечення для опису компонентів або додатків, які розроблені для конкретної платформи або операційної системи. Коли говорять про нативний код, це означає, що код написаний з використанням мови програмування та інструментів, які є стандартними для даної платформи.

Фреймворк – це набір попередньо створених компонентів, бібліотек, інструментів та правил, які надають основу для розробки програмного забезпечення або додатків. Він визначає структуру, організацію та спосіб взаємодії між компонентами програми, дозволяючи розробникам швидше створювати та підтримувати програмне забезпечення.

React хуки – це спеціальні функції, які дозволяють використовувати стан та інші можливості React в компонентах без потреби використовувати класовий синтаксис. Це новий підхід, який був доданий до React для полегшення роботи зі станом і побудови компонентів.

Рендер – процес відтворення компонентів і створення візуального представлення на екрані на основі стану даних.

VirtualDOM (віртуальний DOM) – це концепція, що використовується в бібліотеках та фреймворках, таких як React, для ефективного оновлення та маніпулювання структурою сторінки або інтерфейсу користувача. У

віртуальному DOM весь інтерфейс користувача представлений у вигляді дерева об'єктів JavaScript. Це абстракція над реальним DOM, яка дозволяє реалізувати швидке оновлення інтерфейсу, маніпулювати елементами та додавати нові компоненти без прямого взаємодії з реальним DOM.

Yarn – це пакетний менеджер для JavaScript, призначений для управління залежностями проєктів і установки пакетів. Він є альтернативою до популярного пакетного менеджера npm.

Перевикористання – це принцип або практика в програмному забезпеченні, коли певні компоненти, модулі, функції або інші елементи коду можуть бути використані повторно в різних частинах програми або навіть у різних програмах.

TypeScript – строго типізована надбудова JavaScript, що додає статичну типізацію та об'єктно-орієнтоване програмування.

API – набір процедур та функцій, які дозволяють створювати додатки, які взаємодіють з іншими додатками або компонентами.

Маркетплейс – онлайн-площадка, де продавці та покупці можуть проводити торгівлю.

YML-файл – формат даних, який використовується для представлення структурованих даних у вигляді ієрархії ключ-значення.

JSON – це формат обміну даними, який використовується для передачі структурованої інформації між різними програмами або системами. Він базується на синтаксисі JavaScript, але може бути використаний в багатьох інших мовах програмування.

React Native – фреймворк для розробки крос мобільних додатків.

.NET, C#, ASP .NET Core – технології Microsoft для розробки серверних додатків.

MSSQL DB – Система управління реляційними базами даних Microsoft SQL Server.

1.2 Функціональні вимоги

1.2.1 Клієнтська частина

Система має бути доступна як додаток у веб-браузері, так і на мобільних пристроях з операційною системою Android. Використовуючи технологію React Native, додаток має бути оптимізований для роботи на обох платформах з мінімальними змінами в коді. Для досягнення підтримки React Native додатку у веб-браузерах, має бути реалізований спосіб, який дозволяє транслювати код, написаний для мобільного додатку React Native, на веб-платформу

Мобільний додаток та веб-додаток мають підтримувати широкий діапазон розмірів екрана, включаючи від 420x900 до 1920x1080 пікселів. Це означає, що інтерфейс користувача має бути гнучким та адаптивним, щоб користувачі мали зручний та ефективний доступ до функціональності системи, незалежно від пристрою, який вони використовують.

Для забезпечення взаємодії між клієнтською (візуальною) частиною додатка та сервером має використовуватися REST API. REST API дозволяє клієнтському додатку відправляти запити на сервер та отримувати від нього відповіді, що включають необхідні дані або статус операції. Це дає можливість реалізувати високий рівень абстракції, забезпечуючи гнучкість та масштабованість системи.

1.2.2 Серверна частина

Серверна частина має відповідати таким вимогам:

- система повинна вміти парсити дані з YML-файлу;
- система повинна вміти парсити дані через API з джерел Rozetka та Prom;
- система повинна мати можливість парсити дані з інших джерел, якщо

вони будуть доступні;

– система повинна вміти парсити дані про складські залишки з JSON або YML-файлу. Система повинна автоматично оновлювати наявність та ціну в кожному з джерел продажу.

1.3 Нефункціональні вимоги

Система має відповідати таким нефункціональним вимогам:

– надійність: система повинна надавати стабільну роботу та відновлюватися після збоїв;
– ефективність: система повинна забезпечувати швидку обробку даних;
– безпека: дані, що передаються між системою та джерелами, повинні бути захищені;
– масштабованість: система повинна легко масштабуватися залежно від кількості джерел та товарів.

1.4 Опис предметної області

Система призначена для автоматизації процесів зміни ціни та наявності у картках товарів на різних ресурсах продажу, таких як магазини та маркетплейси. Це включає обробку та передачу даних про закупівельні ціни, відсотки ресурсів продажу та прибутки з товарних позицій.

1.5 Опис системи

Система складається з двох основних компонентів: серверної та клієнтської частин.

1.5.1 Серверна частина

Серверна частина системи, розроблена на основі .NET, ASP .NET Core та MSSQL DB, відповідає за обробку та управління даними. Вона реалізує основні функції системи, включаючи парсинг даних з різних джерел, визначення оптимальної ціни товарів, управління наявністю товарів, а також передачу даних до систем джерел продажу.

Серверна частина також включає REST API, який забезпечує взаємодію між сервером та клієнтською частиною додатка. API включає набір ендпойнтів, які дозволяють клієнтському додатку виконувати різноманітні операції, включаючи отримання, оновлення та видалення даних.

1.5.2 Клієнтська частина

Клієнтська частина додатка, розроблена за допомогою React Native та React Native Web, представляє собою інтерфейс, через який користувачі взаємодіють з системою. Вона включає в себе набір екранів та компонентів, які надають доступ до функціональності системи, включаючи перегляд та редагування інформації про товари, налаштування автоматичних правил для зміни ціни та наявності товарів, та відстеження статусу певних операцій. Клієнтська частина включає таблицю товарів з можливістю фільтрації по категоріям, кількості товарів тощо. Вона також має вбудований конструктор таблиці, який дозволяє користувачам налаштовувати відображення колонок відповідно до своїх потреб. Ресурси продажу можуть бути додані динамічно в конструкторі, що забезпечує гнучкість управління та відображення даних.

1.6 Огляд інструментів розробки

1.6.1 Інструменти для реалізації клієнтської частини

Для реалізації користувацького інтерфейсу необхідно чітко визначити проєкт та його цілі. Це включає в себе розуміння функціональних вимог, цільової аудиторії, основних функцій додатку та його очікуваних користувацьких сценаріїв.

Для реалізації користувацького інтерфейсу були використані наступні інструменти та технології:

- Typescript – інструмент для забезпечення типізації та поліпшення безпеки коду;
- React Native – фреймворк для розробки мобільного додатку, оскільки він надає можливість створювати кросплатформенні додатки;
- React Native Web – інструмент для адаптації додатку для веб-платформи, що дозволяє забезпечити спільний код для мобільного та веб-інтерфейсів;
- React Navigation – інструмент для реалізації навігації та маршрутизації в додатку;
- Zustand – бібліотека для керування станом додатку, що дозволяє легко організувати та підтримувати стан додатку;
- Tanstack Table – бібліотека для створення таблиць з даними у додатку, що надає можливість сортування, фільтрації та пагінації;
- Tanstack Query – бібліотека для керування асинхронними операціями та взаємодії з сервером;

1.6.2 Інструменти для реалізації серверної частини

Для розробки серверної частини була обрана платформа Firebase, яка надає набір інструментів та сервісів для керування базами даних в мобільному та веб

додатку. Але через зміну цінової політики під час розробки системи, було прийняте рішення міграції системи на платформу Azure з наступними аргументами:

– цінова політика: Azure пропонує гнучкі моделі ціноутворення, дозволяючи ефективно використовувати ресурси та зменшити витрати на інфраструктуру системи, що є важливим фактором під час розробки системи;
– широкий спектр хмарних сервісів: Azure надає розширений набір сервісів, які включають обчислення, зберігання даних, штучний інтелект, аналітику, безпеку та інші, що відповідають потребам серверної частини розробленої системи;
– гнучкість та масштабованість: Azure дозволяє легко розгортати та масштабувати додатки, незалежно від їх розміру та навантаження, що забезпечує ефективне функціонування системи.

Серверна частина складається з таких інструментів:

– C# – це об'єктно-орієнтована мова програмування від Microsoft, що базується на .NET, і призначена для створення надійних та високопродуктивних додатків. C# підтримує ряд сучасних функцій, таких як типи-значення, делегати, асинхронне програмування та LINQ;
– .NET – це платформа розробки від Microsoft, яка включає в себе бібліотеки класів, ряд сервісів та інструментів для розробки, випуску та супроводу додатків на різних мовах програмування;
– ASP .NET Core – це кросплатформенний фреймворк для розробки веб-додатків та веб-служб на основі .NET. ASP .NET Core пропонує високу продуктивність, масштабованість та безпеку, а також розширені можливості для тестування та розгортання додатків;
– Microsoft SQL Server – це система управління реляційними базами даних, розроблена Microsoft, яка використовується для зберігання, обробки та аналізу великих обсягів даних. SQL Server пропонує розширені можливості для роботи з даними, включаючи підтримку транзакцій, індексування, збережені процедури та ін.

Узагальнюючи, перенос інфраструктури усієї системи до платформи Azure, дозволило розгортати мобільний та веб додаток, та сам сервер в централізованій платформі.

2 ПРОЄКТУВАННЯ

2.1 Проєктування за допомогою UML

У розробці систем використовуються різні методології, а одним з найпоширеніших інструментів моделювання є уніфікована мова моделювання UML (Unified Modeling Language). UML призначена для моделювання, уявлення, проєктування та документування програмних систем.

Завдяки використанню UML, розробники програмного забезпечення можуть спілкуватися за допомогою єдиного мовного інструментарію, що сприяє зменшенню ризиків непорозумінь та покращенню співпраці. Візуалізація концепцій і структур системи у вигляді діаграм допомагає зрозуміти їх взаємозв'язок та логіку роботи. Використання UML в розробці систем дозволяє зосередитися на проєктуванні та архітектурі, створюючи чітке уявлення про систему до початку реалізації.

2.1.1 Проєктування за допомогою діаграми прецедентів

Для проєктування користувацького інтерфейсу була застосована діаграма прецедентів (Use Case Diagram), яка допомагає ідентифікувати та візуалізувати різні сценарії використання системи, що дозволяє зрозуміти, як користувачі будуть взаємодіяти з додатком.

Діаграма прецедентів складається з акторів та прецедентів (use cases). Актори – це зовнішні сутності, які взаємодіють з системою, такі як користувачі, зовнішні системи або інші сторонні сервіси. Прецеденти – це конкретні сценарії або функціональні можливості системи.

Діаграма прецедентів дозволяє:

– виявити основні функціональні можливості системи та потреби

користувачів;

- встановити взаємозв'язки між акторами та прецедентами;
- визначити ролі акторів та їх взаємодію з системою;
- встановити послідовність дій, які користувачі виконують для досягнення певних цілей;
- ідентифікувати ролі та відповідальності акторів у виконанні прецедентів.

На рисунку 2.1 зображена діаграма прецедентів проєкту.

На діаграмі представлений лише один актор «Користувач». У даному контексті «користувач» цієї системи це універсальне позначення для двох головних ролей: менеджерів з продажу та менеджерів з закупівель. Це означає, що як і менеджери з продажу, так і менеджери з закупівель, мають доступ до системи та можуть використовувати її функціонал. Обидві ці ролі виконують різні дії в системі, але об'єднані спільною метою – оптимізацією процесу продажу та закупівлі товарів. Менеджери з продажу використовують систему для автоматичного оновлення цін та наявності товарів на різних торгових площадках, відстеження динаміки продажів та аналізу ринку. Менеджери з закупівель використовують систему для відстеження закупівельних цін, наявності товарів на складах, планування закупівель та оптимізації запасів.

В діаграмі виділені три основні прецеденти: «Робота з джерелами продажу», «Робота зі складами» та «Робота з товарами». Після того, як користувач авторизується у системі, він буде мати доступ до керування інформацією про товари, їх наявність на складах та синхронізування даних про товари з різними джерелами продажу.

Загалом в діаграмі описані наступні прецеденти.

Вхід до системи: дозволяє користувачу отримати доступ до функціоналу системи. Процес зазвичай включає введення імені користувача та пароля в відповідні поля на екрані входу. Після успішної автентифікації користувача система відкриває доступ до основного інтерфейсу.

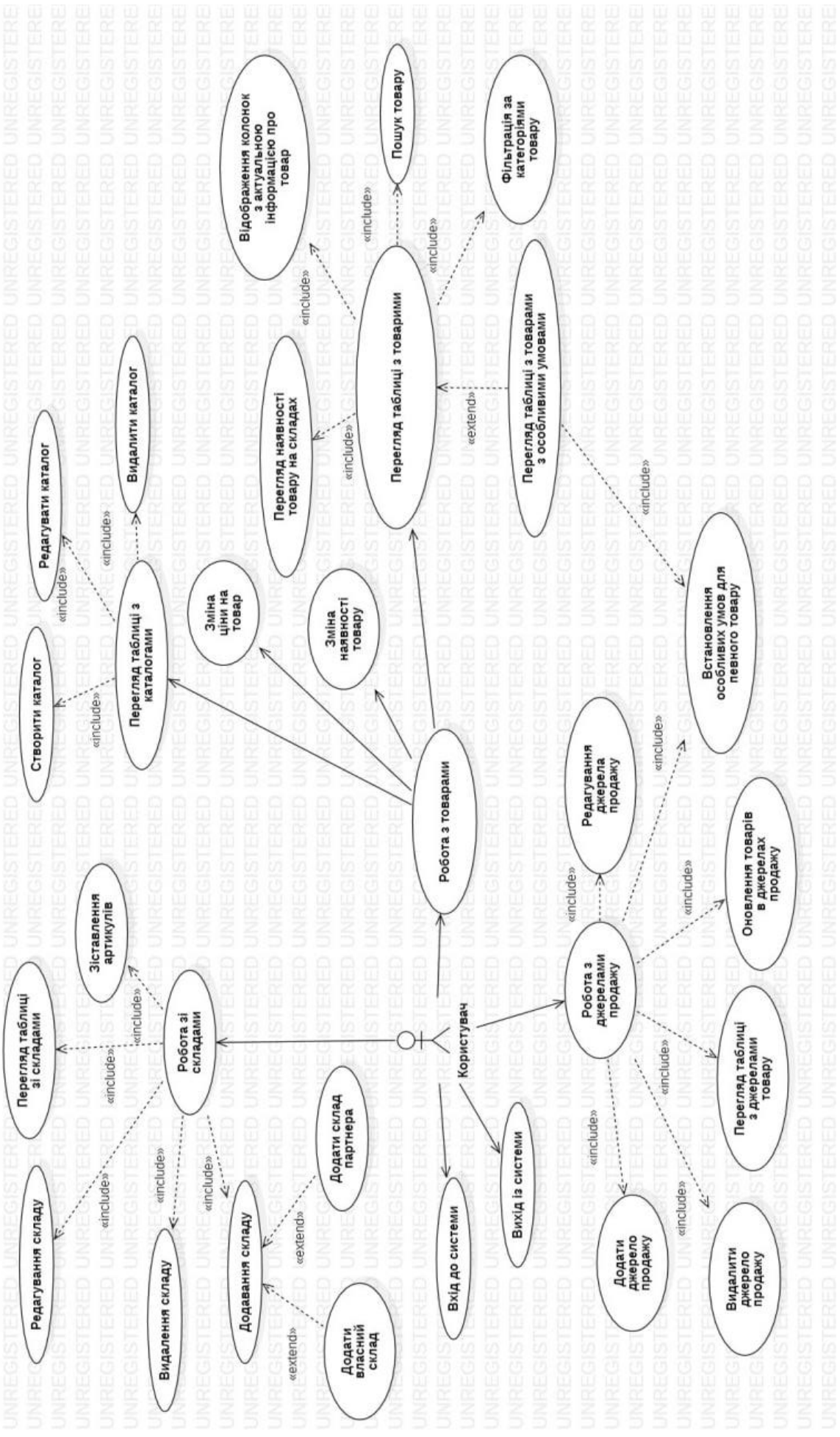

Рисунок 2.1 – Діаграма прецедентів

Вихід із системи: дозволяє користувачу завершити свою сеанс роботи в системі. Після вибору опції «вийти», система закриває доступ до своїх функцій і повертає користувача до екрану входу. Це важливий прецедент з точки зору безпеки, оскільки він допомагає запобігти несанкціонованому доступу до системи.

Робота з товарами: включає в себе керування переліком товарів, відстеження їх статусу та розміщення на різних торгових площадках. Користувач може встановлювати ціни на товари, редагувати інформацію про них, синхронізувати дані між складом і торговими площадками, а також аналізувати динаміку продажів окремих товарів.

Перегляд таблиці з каталогами: цей прецедент дозволяє користувачеві переглянути таблицю, в якій відображено каталоги товарів. Тут користувач може бачити різні атрибути каталогів, такі як їх назви, описи, і т.д.

Зміна ціни на товар: дозволяє користувачу вносити зміни в ціну конкретного товару. Це може бути необхідно для адаптації до змін у ринкових умовах, обмежених пропозиціях або спеціальних акціях.

Зміна наявності товару: надає користувачу можливість оновлювати статус наявності конкретного товару. Це може бути важливо для відображення реальної кількості товарів на складі або вказівки на те, що товар тимчасово недоступний.

Створити каталог: створення нових каталогів товарів. Користувач вносить необхідну інформацію про каталог, включаючи його назву, опис, і, якщо необхідно, асоціює його з відповідними товарами.

Редагувати каталог: можливість внесення зміни в існуючий каталог. Це може включати оновлення назви, опису або асоціацій з товарами. Це забезпечує гнучкість у керуванні каталогами, оскільки вони можуть змінюватися з часом.

Видалити каталог: Видалення існуючого каталогу. Це може бути необхідно, коли каталог більше не актуальний або коли товари, які в ньому містяться, переходять до іншого каталогу.

Перегляд наявності товару на складах: важливий сценарій для ведення

обліку товару та планування закупівель.

Відображення колонок з актуальною інформацією про товар: відображає інформацію про товар в таблиці, включаючи такі дані, як назва, опис, ціна, наявність на складах тощо.

Пошук товару: можливість пошуку конкретного товару за різними параметрами, такими як назва, категорія або інші характеристики.

Фільтрація за категоріями товару: застосування фільтрів для відображення лише тих товарів, які відповідають вибраним категоріям.

Перегляд таблиці з товарами з особливими умовами: це розширення основного прецеденту «Перегляд таблиці з товарами», яке дозволяє користувачу використовувати специфічні фільтри або параметри сортування, які залежать від конкретних джерел продажу. Наприклад, користувач може переглянути товари, які продаються в конкретному інтернет-магазині або на конкретному маркетплейсі. Або вони можуть шукати товари, які виконують певні критерії продажу в залежності від політики або особливостей конкретного джерела продажу. Також, це включає можливість перегляду товарів за певними параметрами, які важливі для конкретних джерел продажу – наприклад, рейтинги продавців, відгуки клієнтів, статистика продажів тощо.

Робота з джерелами продажу: описує процес взаємодії користувача (менеджера з продажу або менеджера з закупівель) з різними торговими площадками, такими як магазини або маркетплейси. Це включає в себе налаштування підключення до цих площадок, синхронізацію інформації про товари, включаючи ціни та наявність, а також отримання статистики та аналітики продажів.

Редагування джерела продажу: внесення змін в інформацію про джерело продажу. Така інформація, яка включає в себе адресу веб-сайту магазину, особливі вимоги та інше.

Встановлення особливих умов для певного товару: цей прецедент дозволяє користувачу встановлювати специфічні умови для певних товарів або категорій товарів такі які забороняють автоматичне оновлення ціни або кількості

товару в наявності. Зокрема, користувач може вибрати артикул конкретного товару або вказати цілу категорію товарів, для яких будуть застосовуватися ці особливі умови.

Робота зі складами: у рамках цього прецеденту користувач має змогу керувати інформацією про склади, включаючи перелік доступних товарів, їх кількість, закупівельну ціну та інші параметри. Це також включає в себе ведення обліку залишків товарів, планування закупівель.

Зіставлення артикулів товарів: цей прецедент дає користувачу зв'язати артикул товару з власного складу із артикулом того самого товару на партнерському складі. Це необхідно для того, щоб система могла правильно оновлювати інформацію про товар на джерелах продажу, використовуючи дані як з власного, так і з партнерського складів.

Додавання складу: користувач може додавати нові склади в систему. Це включає можливість вводити деталі складу, включаючи його назву, адресу, тип та вказівку на те, чи є це власний склад, або склад партнера.

Створення власного складу: розширений прецедент «Додавання складу», яка дозволяє користувачу створити власний склад в системі. Користувач може додавати туди товари, встановлювати їх кількість, а також редагувати іншу інформацію про них.

Створення партнерського складу: розширений прецедент «Додавання складу», яка дає можливість користувачу додати до системи партнерський склад. Це може бути склад постачальника або іншого партнера, який надає свої товари для продажу. Після додавання партнерського складу, користувач може здійснювати зіставлення артикулів товарів між власним та партнерським складами для правильного оновлення інформації про товари на джерелах продажу.

Перегляд таблиці зі складами: перегляд список всіх складів, які він використовує. Він може бачити деталі кожного складу, включаючи його назву, адресу, тип та кількість наявних товарів.

Редагування складу: внесення змін в інформацію про склад, включаючи

його деталі, такі як назва, адреса, а також тип складу.

Видалення складу: якщо склад більше не використовується, користувач може видалити його з системи.

2.1.2 Проєктування за допомогою діаграми компонентів

Діаграма компонентів надає візуальне представлення структури системи та взаємодій між її компонентами, допомагаючи краще зрозуміти архітектуру додатку і спрощувати комунікацію між розробниками та зацікавленими сторонами. Компоненти можуть бути представлені у вигляді блоків або прямокутників, які містять назву компонента. Кожен компонент зазвичай має свою назву, яка вказує на його функціональне призначення або роль у системі. Назва компонента розміщується всередині блоку або прямокутника. На діаграмі компонентів також можуть бути зображені залежності між компонентами, що показують, як вони взаємодіють між собою. Це можуть бути напрямлені стрілки або лінії, що з'єднують компоненти. Напрямок стрілок вказує напрямок залежності або комунікації між компонентами.

У контексті створення клієнтської частини додатку, діаграми компонентів дозволили описати, які компоненти складають додаток і як вони взаємодіють між собою (див. рис. 2.2).

Проєкт складається з двох основних частин – «серверна» та «клієнтська».

Серверна частина відповідає за обробку та зберігання всієї необхідної інформації, яка відображається на клієнтській частині.

Клієнтська частина включає в себе всі компоненти, інструменти і ресурси, які відповідають за взаємодію з користувачем. Це включає в себе графічний інтерфейс користувача, який забезпечує візуальне відображення даних, взаємодію з елементами управління і передачу даних між користувачем та сервером. Клієнтська частина також включає логіку, пов'язану з перевіркою та валідацією даних, обробкою подій та навігацією у додатку.

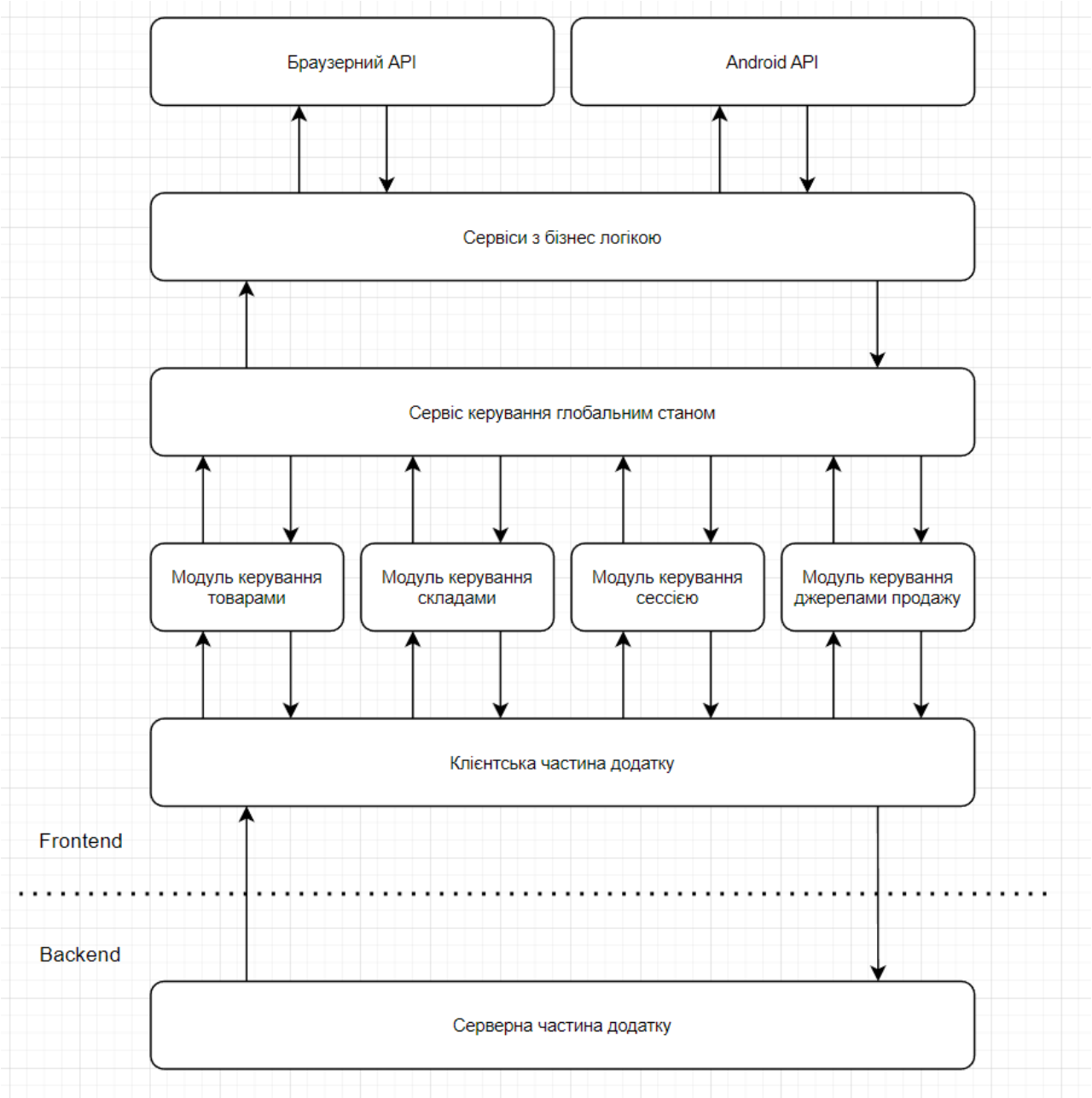

Рисунок 2.2 – Діаграма компонентів

Клієнтська частина складається з 4 основних модулів які відображають бізнес логіку:

– модуль «Керування товарами» – компонент, який відповідає за функціонал, пов'язаний з управлінням товарами в додатку. Він може містити логіку, пов'язану з додаванням, редагуванням та видаленням товарів, а також відображенням їх деталей;
– модуль «Керування складами» – компонент, який відповідає за управління запасами товарів на складах. Він може містити функції,

пов'язані з додаванням та зменшенням кількості товарів на складах, відстеженням наявності товарів та керуванням їх розподілом;

– модуль «Керування сесією» – компонент, відповідає за управління сесіями користувачів в додатку. Він може включати функціональність, пов'язану з аутентифікацією, авторизацією та збереженням стану сесій користувачів;
– модуль «Керування джерелами продажу» – компонент, який відповідає за управління джерелами, через які здійснюється продаж товарів. Він може містити функції, пов'язані зі збором, обробкою та аналізом інформації про різні джерела продажу.

Для того щоб певний модуль мав доступ до стану іншого модуля, був розроблений Сервіс керування глобальним станом. Цей компонент зберігає стан усієї бізнес-логіки додатку. Він відповідає за управління та збереження даних, які використовуються і обробляються різними модулями та сервісами в додатку.

Клієнтська частина також складається ще з трьох компонентів, які є складовими компоненту «Сервіс керування глобальним станом»:

– компонент «Сервіси з бізнес логікою» – компонент, який містить набір перевикористовуваних сервісів, які мають бізнес-логіку і можуть бути використані у різних частинах додатку. Ці сервіси можуть включати функціонал, пов'язаний з обробкою даних, валідацією, розрахунками тощо;
– компонент «Браузерний API» – компонент, який відповідає за взаємодію з браузерним середовищем. Він може містити функції та методи, які дозволяють взаємодіяти зі стандартними можливостями браузера, такими як робота з DOM-деревом, виконання запитів до сервера або маніпуляції зі станом сторінки;
– компонент «Android API» – компонент, який забезпечує доступ до функцій операційної системи Android для розробки мобільних додатків, специфічних для цієї платформи.

2.2 Архітектура

Для розробки клієнтської частини було застосовано Flux архітектуру (див. рис. 2.3). Flux є архітектурним шаблоном, спеціально розробленим для управління станом додатку у клієнтській частині. Він забезпечує односторонній потік даних і допомагає підтримувати чистоту та передбачуваність управління станом.

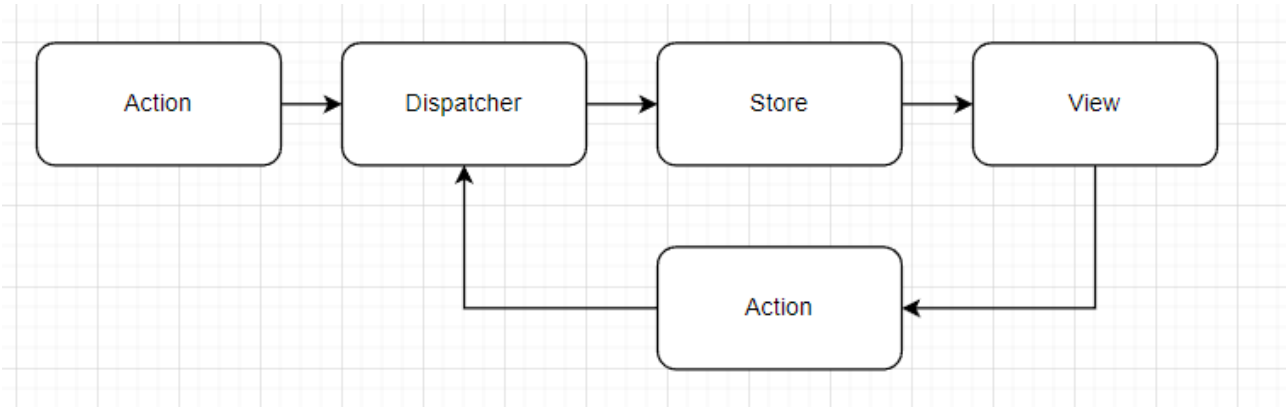

Рисунок 2.3 – Flux архітектура

Розглянемо основні складові Flux архітектури.

«Store (сховище)»: виступає як централізований сховище даних додатку. Воно зберігає усі дані, які можуть змінюватися у процесі роботи додатку. Кожне сховище в Flux відповідає за конкретну частину стану додатку і надає методи для зміни даних у сховищі.

«Action (дія)»: в Flux архітектурі представляє подію або команду, яку виконує користувач або система. Вона спричиняє зміни в стані додатку. Дії є єдиним джерелом зміни даних у Flux і вони передаються до сховища за допомогою виклику певних методів.

«Dispatcher (диспетчер)»: виступає як посередник між діями та сховищами. Він отримує дію і розподіляє її до всіх відповідних сховищ. Диспетчер допомагає забезпечити послідовність та унікальність змін даних у сховищах.

«View (представлення)»: в Flux архітектурі представляють користувацький інтерфейс, який відображає дані з сховищ та реагує на зміни стану. Вони

отримують дані з сховища та відображають їх, а також можуть виконувати дії, які відправляються до диспетчера.

Процес роботи Flux архітектури включає наступні кроки:

а) користувач взаємодіє з додатком та виконує дію;

б) дія передається до диспетчера;

в) диспетчер розподіляє дію до всіх відповідних сховищ;

г) сховища оновлюють свої дані відповідно до отриманої дії;

д) після оновлення даних у сховищах, вони повідомляють своїх підписників (точніше «Предствалення») про зміни;

е) представлення отримують нові дані з сховищ та оновлюють свій інтерфейс відповідно до змін.

Цей односторонній потік даних та чітко визначені ролі дозволило підтримувати чистоту та передбачуваність управління станом додатку в Flux архітектурі. Вона забезпечує простий та масштабований спосіб організації коду та управління станом у клієнтській частині додатку.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1 Опис інструментів для розробки клієнтської частини

3.1.1 React Native

React Native – це відкритий фреймворк для розробки мобільних додатків, який дозволяє використовувати JavaScript та React для побудови нативних інтерфейсів користувача для платформ iOS та Android [1].

React Native базується на архітектурі «однієї кодової бази, багатьох платформ». Це означає, що весь основний код додатка пишеться на JavaScript, а компоненти і API React Native перетворюють його на нативний код для кожної платформи. Для iOS використовується Objective-C або Swift, а для Android – Java або Kotlin. Це дозволяє розробникам використовувати спільну кодову базу для обох платформ і зменшує зусилля, потрібні для розробки й підтримки мобільних додатків. React Native надає розробникам доступ до широкого набору можливостей для побудови мобільних додатків [1].

Розглянемо основні можливості React Native.

Компоненти: React Native надає багато вбудованих компонентів, таких як кнопки, тексти, списки, зображення тощо, які можна використовувати для побудови інтерфейсу користувача. Також можна створювати власні компоненти для перевикористання і зручної організації коду.

Гнучкість: розробники можуть використовувати стандартні CSS-стилі для оформлення компонентів React Native або визначати власні стилі. Інструмент StyleSheet дозволяє задавати стилізацію компонентів з використанням простого синтаксису.

Доступ до пристроїв: За допомогою React Native можна отримати доступ до функціональності пристрою, такої як камера, геолокація, акселерометр і т.д., використовуючи вбудовані API.

Розробники можуть швидко переглядати результати своєї роботи за допомогою гарячого перезавантаження, що дозволяє відразу бачити зміни без необхідності перекомпіляції або перезапуску додатка.

Структура проєкту може варіюватися в залежності від ваших вимог та вибраної організації коду. Структура React Native проєкту складається з таких директорій та файлів [1]:

– директорія «android/» – директорія, яка містить код для платформи Android;
– директорія «src/» – директорія, де зберігаються власні компоненти, стилі, роутери та інші файли додатку;
– директорія «node_modules/» – директорія, в якій зберігаються JavaScript залежності проєкту, установлені за допомогою yarn;
– файл «package.json» – файл, що містить інформацію про проєкт, включаючи залежності та налаштування скриптів;
– файл «App.js» – головний файл, в якому відбувається ініціалізація додатка та відображення головного компонента.

Проєкт може, також, містити конфігураційні файли, ресурси (зображення, шрифти і т.д.), тести та інші файли, які використовуються у додатку.

3.1.2 Мова програмування Typescript

TypeScript – це надбудова над JavaScript, розроблена Microsoft, яка додає статичну типізацію та додаткові об'єктно-орієнтовані можливості до мови. TypeScript спрощує написання більш надійного та безпечного коду та полегшує його налагодження та підтримку [3].

Typescript має такі особливості, як:

– статична типізація: TypeScript надає можливість використовувати статичну типізацію – це означає, що типи перевіряються на етапі компіляції, а не на етапі виконання, як це відбувається в JavaScript (це

допомагає виявити та виправити помилки ще до запуску коду);

- зручність розробки: TypeScript покращує продуктивність розробки, надаючи інструменти для рефакторингу коду, автозаповнення та навігації по коду;
- інтеграція з JavaScript: оскільки TypeScript є надбудовою для JavaScript, він сумісний з усім існуючим JavaScript кодом (ви можете включити JavaScript файл в TypeScript проєкт без будь-яких змін);
- підтримка модулів та просторів імен: TypeScript підтримує модулі та простори імен, що допомагає організовувати код більш ефективно;
- засоби діагностики: TypeScript має потужні інструменти діагностики для перевірки типів, включаючи тонку настройку статичного аналізу коду, що може допомогти виявити помилки, які можуть бути ускладненими в JavaScript.

3.1.3 Tanstack Query

Tanstack Query – це інструмент для управління станом даних в React додатку. Він дозволяє, безпосередньо, взаємодіяти з асинхронними даними, надаючи можливості кешування, синхронізації та відновлення стану.

Ключові особлиовсті інструменту Tanstack Query [6]:

- автоматичне кешування: інструмент автоматично кешує запити та їх результати, що значно покращує продуктивність додатку;
- автоматичне відновлення: інструмент автоматично відновлює кешовані дані після мутацій, які можуть змінити ці дані;
- підтримка паралельних запитів: Tanstack Query дозволяє виконувати паралельні запити до декількох ресурсів одночасно;
- повторні спроби або відкладені запити: інструмент має вбудовану підтримку повторних спроб та відкладання запитів;
- миттєві оновлення: інструмент підтримує миттєві оновлення, що

дозволяє оновлювати інтерфейс користувача одразу, після завершення асинхронних операцій.

3.1.4 Tanstack Table

Tanstack Table – це інструмент для побудови та відображення складних, багатофункціональних таблиць в React додатку. Ця бібліотека дозволяє розробникам створювати налаштовані таблиці з підтримкою сортування, фільтрації, пагінації, групування та багатьох інших функцій.

Основна ідея полягає в тому, що інструмент Tanstack Table не відображає таблицю напряму, а надає гнучкі React хуки, які дозволяють вам самостійно створити таблицю за власними потребами [5].

3.1.5 Zustand

Zustand – це інструмент, який створений з метою надати прості та зрозумілі абстракції для управління станом в React додатках. Вона забезпечує легкість і простоту внесення змін у стан, та інтегрується в React додаток через механізм React хуків [7].

Особливості та переваги Zustand:

– простота використання: Zustand дозволяє знизити кількість шаблонного коду, що необхідний для управління станом (це означає, що можна працювати, як зі звичайним об'єктом або масивом);
– React хуки: використовуючи Zustand, створюються власні хуки для читання та зміни стану (це підсилює інтеграцію з React і сприяє більш чистому та зрозумілому коду);
– глобальний та локальний стан: інструмент Zustand може управляти як глобальним станом, доступним у всьому додатку, так і локальним

станом, який обмежений конкретним компонентом;

– асинхронні дії: інструмент Zustand дозволяє легко реалізувати асинхронні дії, що впливають на стан, такі як завантаження даних з API;
– безпека типів з TypeScript: інструмент Zustand має підтримку TypeScript, що дозволяє безпечно використовувати типи в коді, що покращує якість розробки.

3.2 Процес створення інтерфейсу користувача

3.2.1 Підтримка веб-браузерів за допомогою React Native Web

Для підтримки React Native додатка у веб-середовищі існує інструмент під назвою React Native Web. Цей інструмент дозволяє використовувати React Native компоненти та логіку в веб-додатках, щоб забезпечити перевикористання коду між мобільними та веб-платформами [8].

Основна ідея React Native Web полягає в тому, щоб дати змогу використовувати React Native код в веб-проєктах, залишаючи його якомога ближче до оригінального React Native проєкту. Це означає, що ви можете перевикористовувати компоненти та логіку з React Native, щоб швидко розробляти веб-інтерфейси знайомим способом. React Native Web забезпечує веб-специфічну реалізацію React Native компонентів. Замість використання нативних компонентів платформи Android, він використовує веб-специфічні HTML елементи, такі як div, span, input тощо, для відображення компонентів в браузері.

Для налаштування React Native Web в існуючому додатку на React Native необхідно виконати наступні кроки:

– встановлення необхідних пакетів: за допомогою пакетного менеджера yarn встановлюються пакети react-native-web та react-dom, які

забезпечать підтримку React Native Web в вашому проєкті;

- конфігурація Babel: оновлюється файл конфігурації Babel (.babelrc або babel.config.js) для включення плагінів react-native-web та expo-web (це дозволить правильно транспілювати та перетворити код React Native для веб-платформи);
- конфігурація Webpack: для підтримки веб платформи використовується модульний збірник Webpack, та оновлюється файл конфігурації (webpack.config.js) для забезпечення підтримки React Native Web (додається налаштування, які дозволяють коректно обробляти та збирати код для веб-платформи);
- зміна вхідного файлу для вебу: оновлюється вхідний файл для запуску додатку на вебі (зазвичай index.ts або App.ts);
- використання спільного коду та компонентів: перевіряється та розподіляється спільний код між React Native та React Native Web (використовуються спільні компоненти та логіка, щоб забезпечити однакову поведінку функціональності на обох платформах).

3.2.2 Організація структури проєкту

Для побудови структури, була використана методологія Feature based design. Методологія Feature based design використовується для організації файлів та коду проєкту за функціональними блоками або «так званими фічами». Кожна функціональна частина додатку відповідає окремому функціональному блоку, що дозволяє зберігати, розробляти та тестувати ці частини незалежно одна від одної. Це допомагає полегшити розробку, підтримку та масштабування проєкту [10].

Основні особливості підходу до створення функціональних блоків:

- кожен функціональний блок має визначену область відповідальності та

функціональність;

– кожен функціональний блок може мати свою власну структуру файлів та компонентів, що відповідає його потребам;
– компоненти в межах функціонального блоку можуть бути розділені на менші компоненти для ефективного перевикористання та підтримки.

Організація файлів та каталогів:

– створення кореневої папки проєкту, де будуть зберігатись всі файлі проєкту;
– в кореневій директорії створюємо папку для кожного функціонального блоку;
– кожна папка функціонального блоку містить необхідні файли, такі як компоненти, стилі, зображення та інші пов'язані файли;
– для кожного компонента створюємо окрему папку, де будуть знаходитись файли, пов'язані з цим компонентом, наприклад, файл компонента, стилі, зображення тощо;
– в директорії функціонального блоку можуть бути додаткові директорії для додаткових рівнів організації, які відповідають логічним розділам функціональності.

3.2.3 Створення компонентів за допомогою React Native

Для розробки користувацького інтерфейсу, створюються базові компоненти, такі як кнопки, поля вводу, заголовки, списки тощо. Створення базового компонента забезпечує вам основну будівельну одиницю для вашого додатку.

На рисунку 3.1 продемонстрований приклад створення базової кнопки з власною стилізацією.

```tsx
import React from 'react';
import { TouchableOpacity, Text, StyleSheet } from 'react-native';

type ButtonProps = {
  title: string;
  onPress: () => void;
};

const Button: React.FC<ButtonProps> = ({ title, onPress }) => {
  return (
    <TouchableOpacity style={styles.button} onPress={onPress}>
      <Text style={styles.buttonText}>{title}</Text>
    </TouchableOpacity>
  );
};

const styles = StyleSheet.create({
  button: {
    backgroundColor: '#2196F3',
    padding: 10,
    borderRadius: 5,
  },
  buttonText: {
    color: '#FFFFFF',
    fontSize: 16,
    fontWeight: 'bold',
    textAlign: 'center',
  },
});

export default Button;
```

Рисунок 3.1 – Створення базового React Native компоненту

На основі базових компонентів створюються складніші компоненти, які включають більше функціональності та взаємодію з користувачем.

Наприклад, форми, списки з можливістю сортування або фільтрації, картки з деталями тощо.

Для стилізації компонентів використовуються CSS-подібні стилі, які надає React Native (див. рис. 3.2).

Стилізація додає зовнішній вигляд і макет компонента, щоб він виглядав належним чином, а типізація допомагає контролювати передачу правильних властивостей та забезпечує зручну розробку з підказками у редакторі коду.

Це підвищує якість та надійність коду та спрощує подальшу розробку та підтримку додатку.

```
const styles = StyleSheet.create({
  button: {
    backgroundColor: '#2196F3',
    padding: 10,
    borderRadius: 5,
  },
  buttonText: {
    color: '#FFFFFF',
    fontSize: 16,
    fontWeight: 'bold',
    textAlign: 'center',
  },
});
```

Рисунок 3.2 – Стилізація React Native компоненту

3.2.4 Навігація за допомогою React Navigation

Для створення навігації, була використана бібліотека React Navigation, яка надає зручний спосіб створення та управління навігацією мобільного додатку. Вона дозволяє створювати різноманітні види навігацій, які відповідають певній платформі, для якої створюється React Native додаток [4].

Розглянемо основні аспекти React Navigation.

Стек навігації (Stack Navigation): стек навігації дозволяє створювати стеки екранів, де кожен новий екран додається на вершину стеку. Користувач може переходити до попередніх екранів за допомогою кнопки «Назад». Це корисно для створення сторінок деталей, форм та інших сценаріїв, де важливий контекст сторінки.

Таб-навігація (Tab Navigation): дозволяє створювати набір вкладок (табів), де кожен таб відповідає за відкриття певного екрану. Користувач може переходити між вкладками шляхом натискання на них. Це добре підходить для організації головного меню, навігації між основними секціями додатку та інших сценаріїв, де розділення функціональності на різні вкладки зручне.

Бокова панель навігації (Drawer Navigation): дозволяє створювати бічне меню, яке може відкриватися або закриватися з боку, щоб показати приховану навігацію. Це зручний спосіб розмістити меню навігації та додаткові

налаштування, які доступні з будь-якої частини додатку.

Маршрутизація параметрів (Route Parameters): React Navigation дозволяє передавати параметри між екранами, щоб передавати дані або контекст з одного екрана на інший. Це дозволяє створювати динамічну навігацію, де екрани можуть отримувати та використовувати дані з попереднього екрану.

Вкладена навігація (Nested Navigation): React Navigation дозволяє вкладати навігацію одну в одну, створюючи більш складну структуру навігації. Наприклад, ви можете мати стеки екранів всередині табів, або таби всередині бокової панелі.

Анімація навігації (Navigation Animation): React Navigation надає можливість налаштовувати анімацію переходів між екранами. Ви можете використовувати різні види анімації, такі як перелистування, затемнення та інші. Це допомагає покращити взаємодію та досвід використання додатку.

Конфігурація навігації (Navigation Configuration): можливість налаштувати поведінку та вигляд навігації за допомогою різних параметрів конфігурації. За допомогою React Navigation, можна налаштувати заголовки екранів, стилі, анімацію, навігаційні опції та інше.

За дизайном додатку, основна навігація між екранами, відбувається за допомогою верхньої панелі з з іконками, на які можна натискати, та переходити до інших екранів. Для того щоб її створити, виконуються такі кроки:

- імпортуються необхідні компоненти з пакетів «@react-navigation/native» та «@react-navigation/stack» для створення навігаційного заголовку, такі як «useNavigation» та «createStackNavigator»;
- для того щоб кнопки навігації розуміли, до яких екранів потрібно посилатися, для кожного екрана створюється компонент який відображає контент сторінки;
- створюється окремий компонент «так званий Header» який відображає навігаційний заголовок у стековому навігаторі. Встановіть опцію header для кожного екрана, в якому ви задаєте власний компонент заголовку.

3.2.5 Керування станом додатку за допомогою Zustand

Основна ідея Zustand полягає в тому, що весь стан додатку може бути організований в «стори» – окремі контейнери, які містять дані та функції для їх зміни. Кожен стор представляє собою окремий модуль, який відповідає за свій власний шматок стану та пов'язану з ним функціональність.

Створити стор можна двома способами.

Перший спосіб. Функція «create()»: створює React хук, який автоматично посилається на стор, і цей хук можна використовувати в компонентах, що спрощує доступ до стану та функцій для зміни стану. Зазвичай такий стор доступний глобально.

Другий спосіб. Функція «createStore()»: створює сам стор без прив'язки до життєвого циклу React. Використовується для гнучкого керування станом, коли якщо потрібно створювати стор під час монтування React компоненту. Тому, для використання такого стора, потрібно використовувати React контекст або передавати через пропси певного компоненту, який потребує доступ до стану стору. Для того щоб React розумів зміни стану, створений стор, має бути прокинутий в спеціальний хук «useStore()» який підписується до життєвого циклу React компоненту.

За допомогою Zustand підходить для застосування принципу Feature Based Design, де кожен функціональний блок додатка має свій власний стор та пов'язану з ним функціональність. Це дозволило легко організовувати та керувати станом в межах кожної функціональної частини додатка.

3.2.6 Асинхронні операції за допомогою Tanstack Query

TanStack Query – це бібліотека для керування асинхронними операціями та кешування даних у React-додатках. Основні компоненти бібліотеки TanStack Query які застосовуються у додатку, наведено нижче.

Об’єкт «QueryClient» – це центральний об'єкт, який управляє всіма

запитами та кешуванням даних. Він створюється на рівні додатка та надає доступ до функціоналу TanStack Query.

Функція «useQuery» – це React хук, який дозволяє компонентам отримувати дані з кешу або виконувати запити до сервера. Він приймає ключ запиту та функцію-запит, а повертає об'єкт з результатами запиту, такими як дані, статус, помилка тощо. Під час виконання запиту, «useQuery()» повертає об'єкт зі статусом запиту, такими як «isLoading», «isError», а також збереженими даними data. Залежно від статусу запиту, компонент може відображати відповідні повідомлення або відображати отримані дані.

Функція «useMutation» – це React хук, який дозволяє виконувати мутації на сервері. Він приймає функцію-мутацію, яка визначає логіку зміни даних, і повертає функцію, яку можна викликати для виконання мутації.

На рисунку 3.3 продемонстрований приклад, де виконується запит до сервера для отримання списку задач. Результат запиту доступний в змінній data. Змінні «isLoading» та «isError» допомагають відслідковувати стан завантаження та помилок запиту. Якщо дані ще завантажуються, відображається повідомлення «Loading...», а якщо виникає помилка – повідомлення про помилку. Якщо дані успішно отримані, вони відображаються у вигляді списку задач.

```
interface Product {
  name: string;
  id: string;
}

const fetchProducts = async (): Promise<Product[]> => {
  const response = await fetch('https://api.example.com/products');
  const data = await response.json();
  return data;
};

export const ProductsComponent: React.FC = () => {
  const { data, isLoading, isError } = useQuery('users', fetchProducts);

  if (isLoading) {
    return <Text>Loading...</Text>;
  }

  if (isError) {
    return <Text>Error occurred while fetching data.</Text>;
  }

  return (
    <View>
      <Text>Товари:</Text>
      {data.map((product) => (
        <Text key={product.id}>{product.name}</Text>
      ))}
    </View>
  );
};
```

Рисунок 3.3 – Приклад використання React хуку «useQuery()»

На рисунку 3.4 продемонстрований приклад, де використовуються компоненти React Native для побудови інтерфейсу, такі як «View», «Text» та «Button». Механізм роботи «useMutation» майже схожий на «useQuery» Функціональний компонент «CreateProductButton» містить логіку використання «useMutation()» для створення нового продукту. При натисканні на кнопку «Create Product», викликається функція «handleCreateProduct()», яка створює новий продукт, передає його в «mutate()» і запускає мутацію (оновлення даних) на сервері. Якщо мутація все ще завантажується, відображається індикатор очікування, а якщо виникає помилка – повідомлення про помилку.

```
export const CreateProductButton = () => {
  const { mutate, isLoading, isError } = useMutation((newProduct) =>
    createProduct(newProduct)
  );

  const handleCreateProduct = () => {
    const newProduct = { name: 'New Product', price: 10 };
    mutate(newProduct);
  };

  if (isLoading) {
    return <Text>Loading...</Text>;
  }

  if (isError) {
    return <Text>Error creating product</Text>;
  }

  return (
    <View>
      <Button title="Create Product" onPress={handleCreateProduct} />
    </View>
  );
};
```

Рисунок 3.4 – Приклад використання React хуку «useMutation()»

3.2.7 Робота з таблицями за допомогою Tanstack Table

За допомогою інструменту Tanstack Table можна визначити структуру таблиці, включаючи заголовки, рядки та комірки. Для визначення структури встановлюються параметри відображення, такі як сортування, фільтрація та пагінація. Таблиця відображається з використанням методів Tanstack Table, що дозволяє легко змінювати верстку таблиці та оновлювати дані в таблиці.

Процес використання Tanstack Table складається з таких етапів:

– створення або отримання даних, які будуть відображені в таблиці (запевняємося, що дані мають необхідну структуру та формат);
– використання React хуку «useTable» для створення таблиці та налаштування її параметрів (до цієї функції передаються дані та опції, такі як стовпці, сортування, фільтрація тощо);
– опис колонок таблиці: основний параметр, який приймає React хук «useTable» – це «columns», за допомогою якого описується структура колонки, та її можливості (сортування, фільтрація і т.п.);
– відображення даних у таблиці: використовуються отримані значення з хука «useTable» для відображення даних у таблиці (використовуються потрібні компоненти та властивості для створення таблиці згідно з вашими потребами).

3.2.8 Адаптація верстки під різні розміри екранів

React Native використовує систему розмітки «Flexbox» для розмітки інтерфейсу. Вона дозволяє розташовувати елементи на сторінці в найбільш ефективний спосіб, особливо коли розміри сторінки або перегляду елементів невідомі заздалегідь [1].

Основні властивості Flexbox включають:

– «flexDirection» – визначення напрямку потоку flex елементів. За замовчуванням в React Native використовується значення «column», що відрізняється від «row», що використовується за замовчуванням в CSS на веб-сторінках;
– «justifyContent» – визначення розташування flex елементів вздовж основної осі контейнера (може приймати значення flex-start, flex-end, center, space-between та space-around);
– «alignItems» – визначення розташування flex елементів вздовж поперечної осі контейнера (може приймати значення flex-start, flex-end, center, stretch та baseline);

– «flex» – визначення співвідношення між розміром flex елементу і рештою доступного простору в контейнері.

Flexbox складається з таких ключових концепцій (див. рис. 3.5):

– «Container» – блок, що містить елементи Flexbox. У React Native, це зазвичай компонент View, в якому ви встановлюєте властивість «flex» або «flexDirection»;

– «Items» – елементи, що містяться в контейнері Flexbox. У React Native, це зазвичай компоненти в середині View.

На рисунку 3.5, продемонстрований приклад застосування Flexbox розмітки.

```
import React from 'react';
import { View, Text } from 'react-native';

export default function ListGroup() {
  return (
    <View style={styles.container}>
      <Text style={styles.flexItem}>Item 1</Text>
      <Text style={styles.flexItem}>Item 2</Text>
      <Text style={styles.flexItem}>Item 3</Text>
    </View>
  );
}

const styles = StyleSheet.create({
  // стилі для Flex Container
  container: {
    flexDirection: 'row',
    justifyContent: 'space-between',
  },
  // стилі для Flex Items
  flexItem: {
    flex: 1,
  },
});
```

Рисунок 3.5 – Приклад застосування «Flexbox» контейнеру

Компонент View, як контейнер Flexbox, використовує властивість «flexDirection: 'row'», щоб розмістити елементи горизонтально. Властивість «justifyContent: 'space-between'» розташовує елементи так, щоб вони рівномірно були розподілені по всьому простору контейнера, з рівними проміжками між

ними. Властивість «flex: 1» в Text компонентах робить їх розміри однаковими.

Для визначення стилей за розмірами екрану React Native надає API для визначення розмірів екрану через об'єкт «Dimensions».

Ви можете отримати ширину та висоту екрану таким чином (див. рис 3.6):

```
import { Dimensions } from 'react-native';

const windowWidth = Dimensions.get('window').width;
const windowHeight = Dimensions.get('window').height;
```

Рисунок 3.6 – Визначення ширини та висоти екрану пристрою

Статичний метод «Dimensions.get('window')» повертає об'єкт, який містить інформацію про ширину та висоту вікна. Метод get приймає рядок, який визначає, які розміри ми хочемо отримати. На рисунку 3.6, ми отримуємо розміри «вікна». Потім ми отримуємо ширину та висоту, посилаючись до властивостей «width» та «height» цього об'єкту.

Для адаптивного дизайну з урахуванням отриманих розмірів екрану, створюються стилі, які змінюються в залежності від розміру екрану (див. рис. 3.7)

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: 'aliceblue',
    alignItems: 'center',
    justifyContent: 'center',
  },
  box: {
    width: windowWidth > 500 ? 500 : windowWidth * 0.9,
    height: windowHeight > 500 ? 500 : windowHeight * 0.9,
    backgroundColor: 'tomato',
  },
});
```

Рисунок 3.7 – Визначення значення властивостей в залежності від розміру екрана

На рисунку 3.7, ми задаємо ширину та висоту коробки на основі розмірів вікна. Якщо ширина вікна менша за 500 пікселів, ми задаємо ширину коробки як 90% ширини вікна. Якщо висота вікна менша за 500 пікселів, ми задаємо висоту коробки як 90% висоти вікна. Це приклад того, як ви можете зробити адаптивний дизайн, використовуючи конкретні розміри екрану.

Крім того, у випадку коли розміри екрану можуть змінюватися під час виконання програми (наприклад, при зміні орієнтації екрану), використовується обробник подій зміни розміру (див. рис. 3.8).

```
import { useEffect, useState } from 'react';
import { Dimensions } from 'react-native';

const useScreenDimensions = () => {
  const [dimensions, setDimensions] = useState({ window });

  useEffect(() => {
    const onChange = ({ window }) => {
      setDimensions({ window });
    };

    Dimensions.addEventListener('change', onChange);

    return () => {
      Dimensions.removeEventListener('change', onChange);
    };
  }, []);

  return dimensions;
};
```

Рисунок 3.8 – Динамічна обробка зміни розміру екрану

На рисунку 3.8, демонструється створення React хука, який допомагає відслідковувати зміни розмірів екрану у React Native додатку. Всередині цього хука використовується стандартний React хук «useState» для створення стану dimensions і функції оновлення «setDimensions». Початкове значення цього стану – це об'єкт, який містить поточні розміри вікна, отримані за допомогою «Dimensions.get('window')». Далі використовується React хук «useEffect», щоб виконати код після рендеру компонента.

React хук useEffect приймає два аргументи [2]:

– функція, яку слід виконати;
– масив залежностей.

На рисунку 3.8, масив залежностей порожній, що означає, що код в useEffect виконується тільки один раз, і тільки після першого рендеру компонента. Всередині React хуку useEffect створюється функція «onChange», яка приймає об'єкт з новими розмірами вікна та оновлює стан dimensions, викликаючи «setDimensions». Далі використовується статичний метод Dimensions «addEventListener» для додавання обробника події «change», який буде викликати функцію «onChange», коли розміри вікна змінюються. В кінці React хук useEffect повертає функцію, яка буде виконана при «очищенні» ефекту, коли компонент буде видалено з VirtualDOM. Ця функція видаляє обробник події, який був створений, за допомогою статичного методу «removeEventListener» класу Dimensions.

3.3 Тестування системи

Тестування системи, включаючи як сервер, так і клієнт, є необхідною частиною процесу розробки програмного забезпечення. Воно дозволяє перевірити, чи відповідає система вимогам, працює правильно і не має помилок або проблем. Тестування допомагає виявити та усунути потенційні проблеми, забезпечуючи якість та надійність системи перед її впровадженням. Це включає проведення різних видів тестів, таких як функціональні, регресійні, навантажувальні та інші, для повного оцінювання функціональності та продуктивності системи. Тестування є важливим етапом, що допомагає забезпечити якість та надійність програмного забезпечення перед його випуском у продуктивне середовище.

3.3.1 Запуск додатку на операційній системі Android

На рисунках 3.9 – 3.10 продемонстрована робота додатку через операційну систему Android.

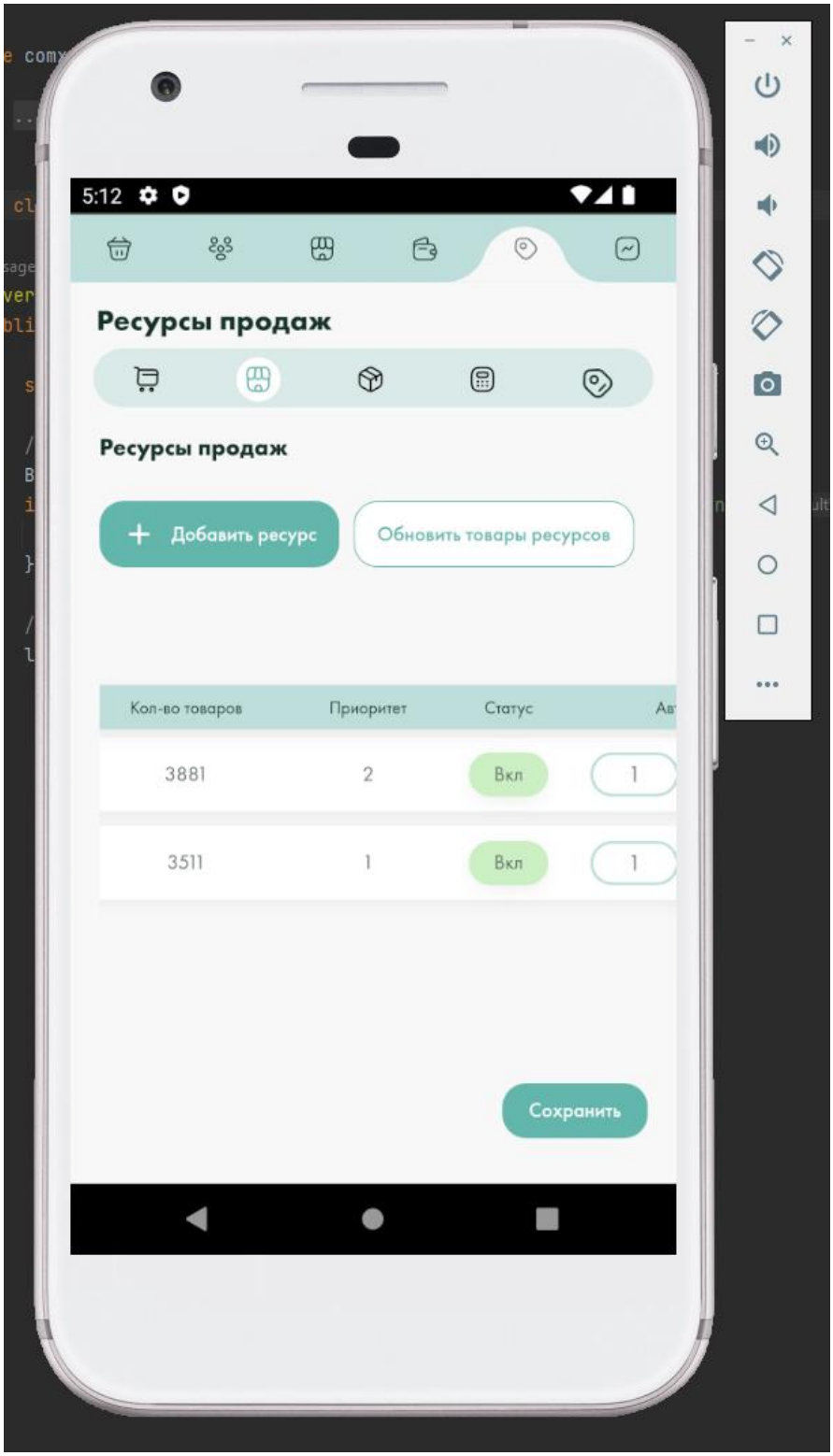

Рисунок 3.9 – Робота додатку на емуляторі Android

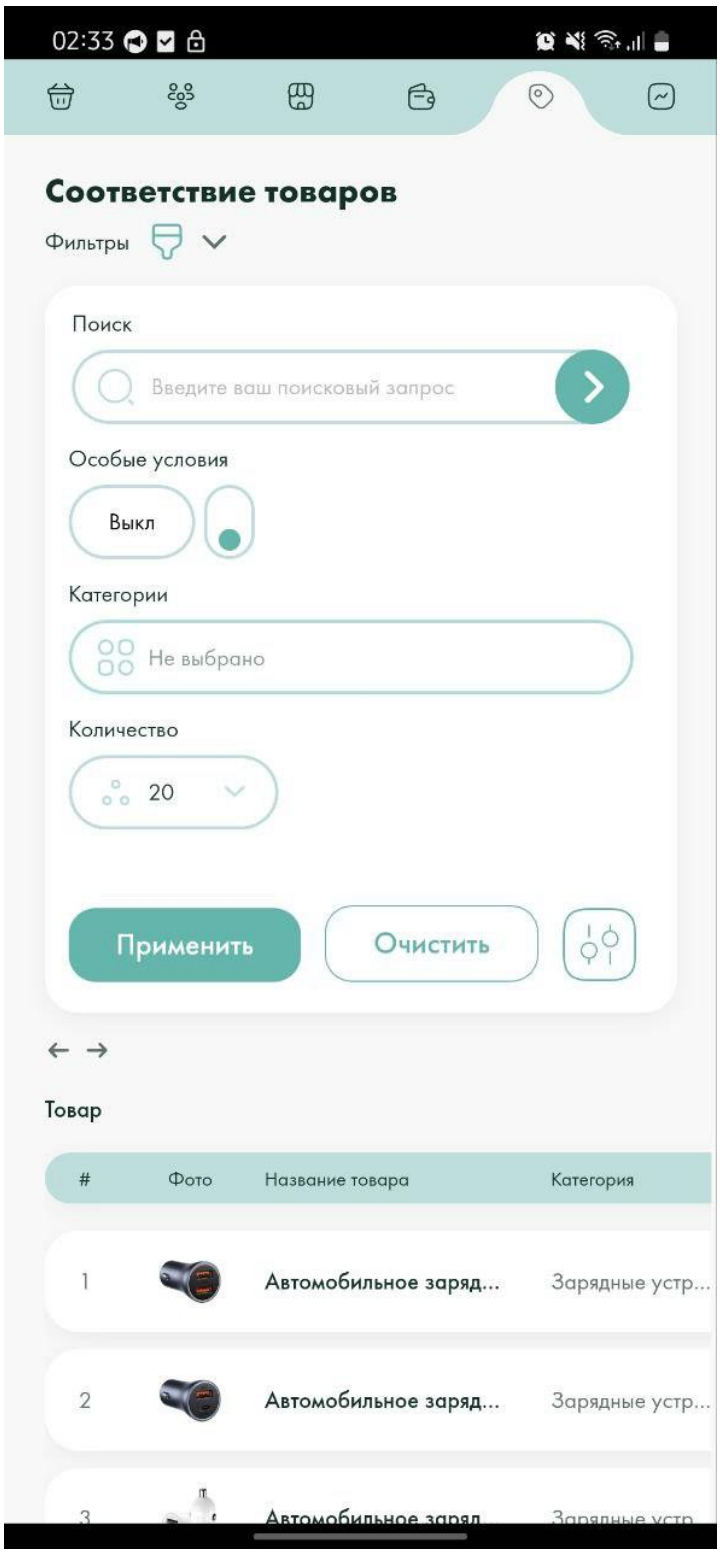

Рисунок 3.10 – Робота додатку на фізичному пристрої з операційною системою Android

Система успішно запущена та протестована на операційній системі Android. Це було досягнуто шляхом використання кросплатформених технологій, що дозволило адаптувати додаток під специфіку Android.

3.3.2 Запуск додатку у веб-браузері

Система була також оптимізована для роботи у веб-браузерах. Під час розробки було враховано особливості найбільш популярних браузерів для забезпечення найкращого користувацького досвіду. Протестовано і підтверджено стабільну роботу системи в основних веб-браузерах, таких як Chrome, Firefox, Safari та Edge.

На рисунках 3.11 – 3.13 продемонстрована робота системи у веб-браузерах Chrome та Microsoft Edge.

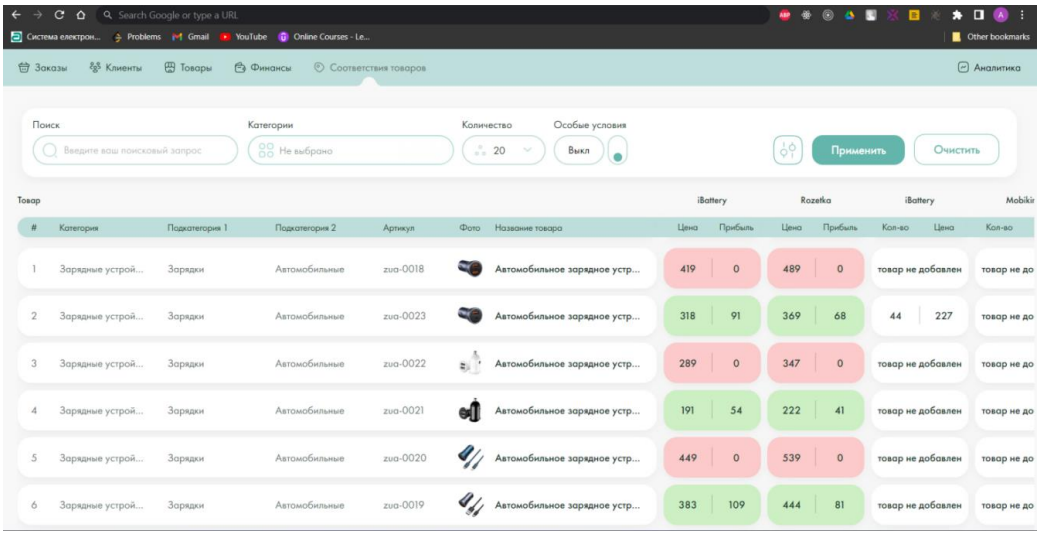

Рисунок 3.11 – Вхід до системи через веб-браузер Chrome

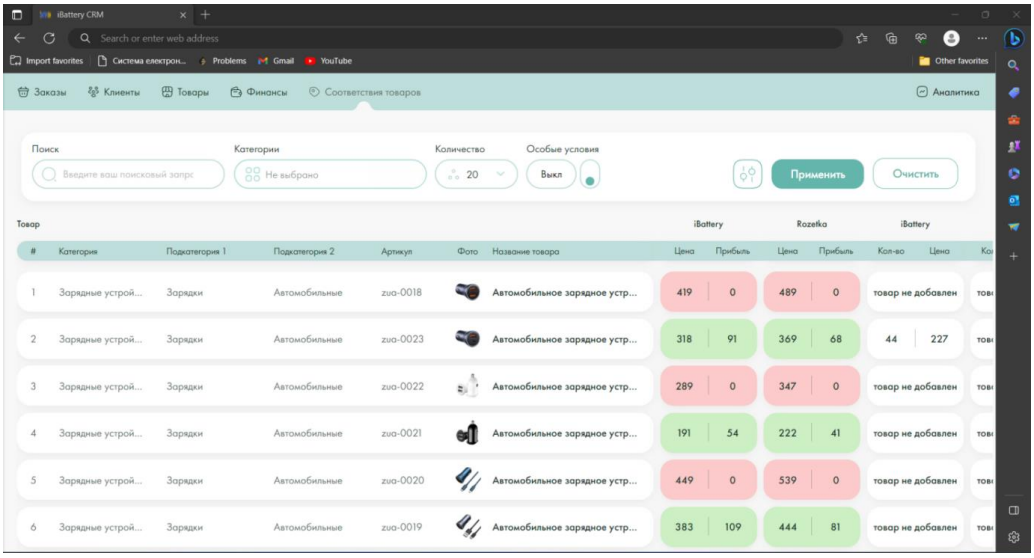

Рисунок 3.12 – Вхід до системи через веб-браузер Microsoft Edge

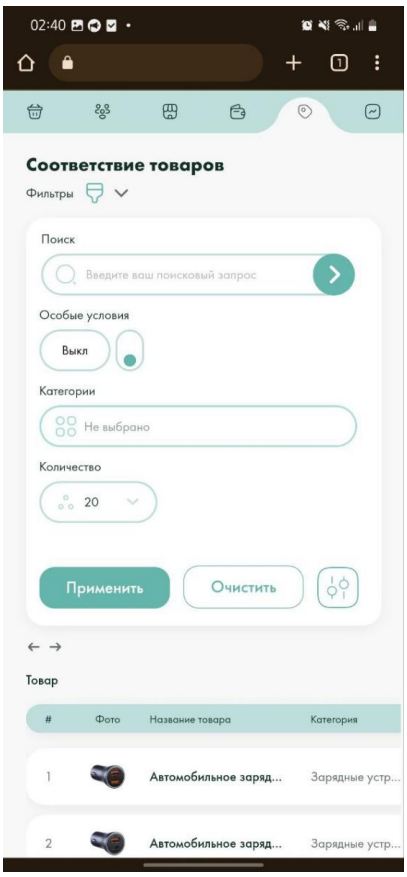

Рисунок 3.13 – Вхід до системи через мобільний веб-браузер Chrome

3.3.3 Тестування адаптації верстки

На рисунках 3.14 – 3.16 продемонстрована адаптація верстки під різні розміри екрану.

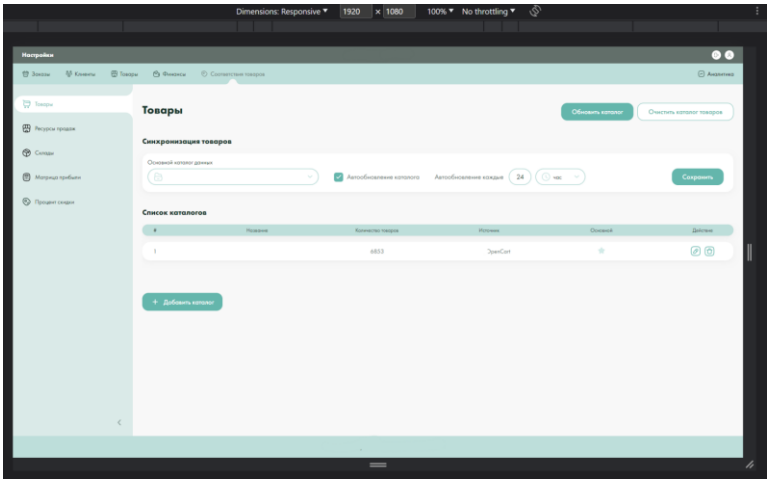

Рисунок 3.14 – Демонстрація додатку з розмірами екрану 1920x1080

Для забезпечення відповідного відображення інтерфейсу на різних екранах, було проведено тестування адаптації верстки. Процес включав перевірку на різних розмірах екранів та орієнтаціях, що дозволило переконатися, що система коректно адаптується під різні умови використання. Результати тестування показали, що верстка відповідає всім сучасним вимогам адаптивності.

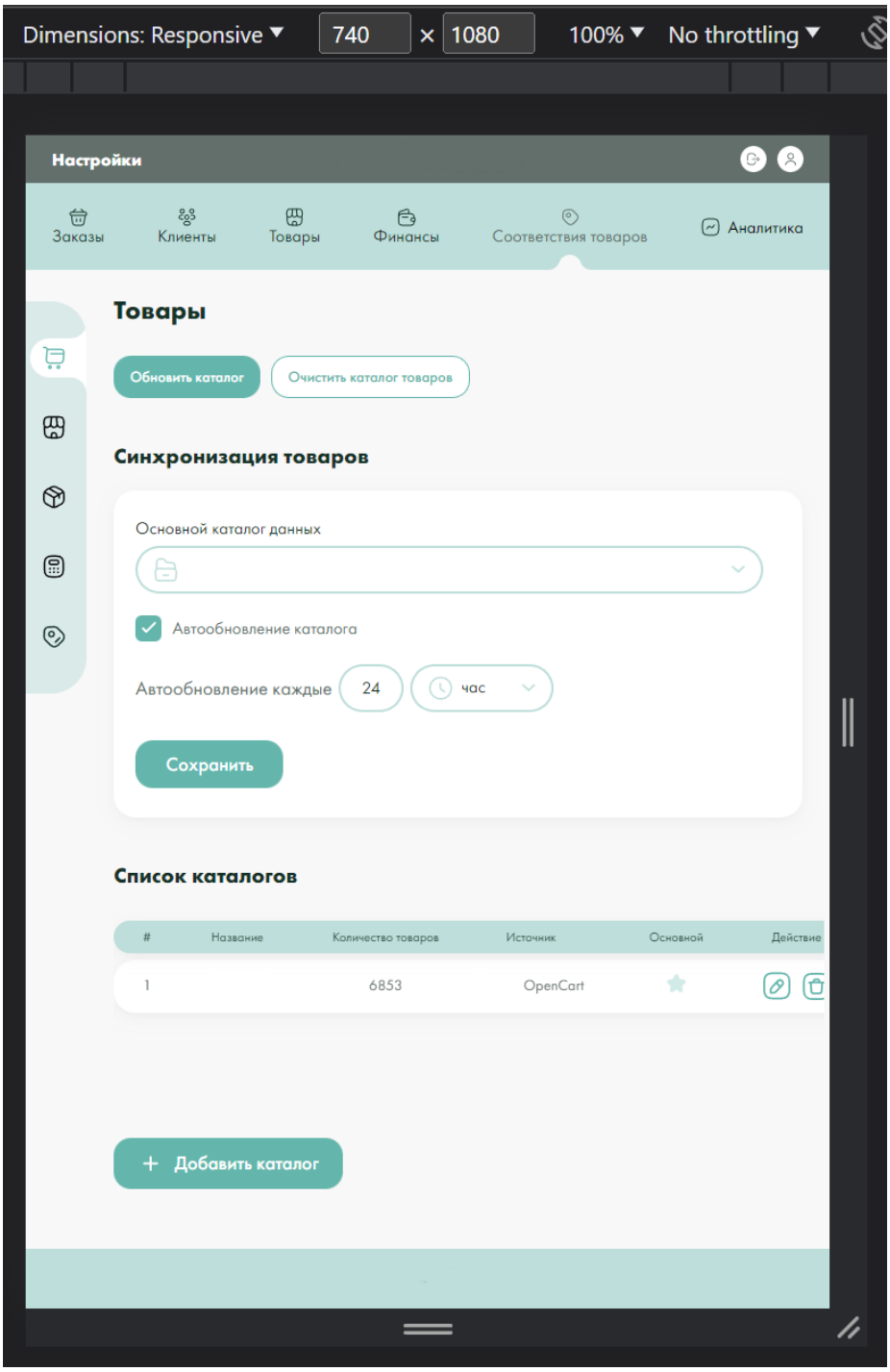

Рисунок 3.15 – Демонстрація додатку з розмірами екрану 740x1080

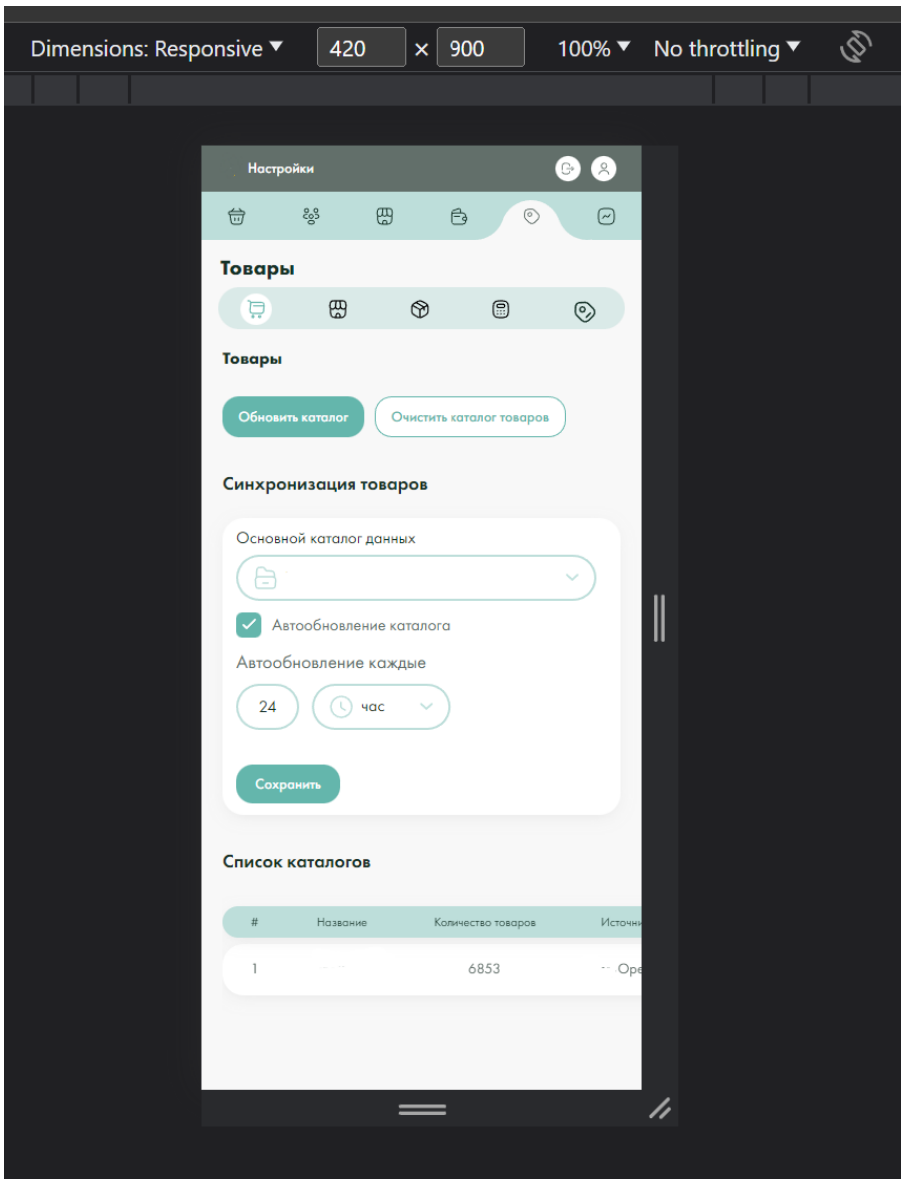

Рисунок 3.16 – Демонстрація додатку на моніторі з розмірами екрану 420x900

3.4 Керівництво для користувача

Ефективне використання даного додатку вимагає від користувача базових технічних навичок, як-то вміння працювати з комп'ютером, веб-браузером або мобільною операційною системою. Користувач повинен впевнено орієнтуватися у використанні операційної системи, встановлюванні програмного забезпечення, роботі з файлами, а також у розумінні основних концепцій браузера. Вивчення наявних інструкцій користувача допоможе відкрити всі можливості сайту і навчитися використовувати його згідно з рекомендаціями.

3.4.1 Підготовчий процес

Щоб розпочати роботу з системою, користувачу потрібен доступ до Інтернету і пристрій в якій встановлена операційна система Android, для доступу до встановленого додатку, або веб-браузер. Після запуску додатку, користувача перенаправляють на сторінку авторизації. Додаток визначає два типи користувачів: неавторизованих відвідувачів, що не мають облікового запису або не ввійшли в систему, та авторизованих користувачів, які зареєстровані та мають доступ до користувацьких функцій додатку.

3.4.2 Авторизація

Авторизація відіграє критичну роль у забезпеченні безпеки та захисту конфіденційних даних.

На рисунку 3.17 продемонстрована форма авторизації.

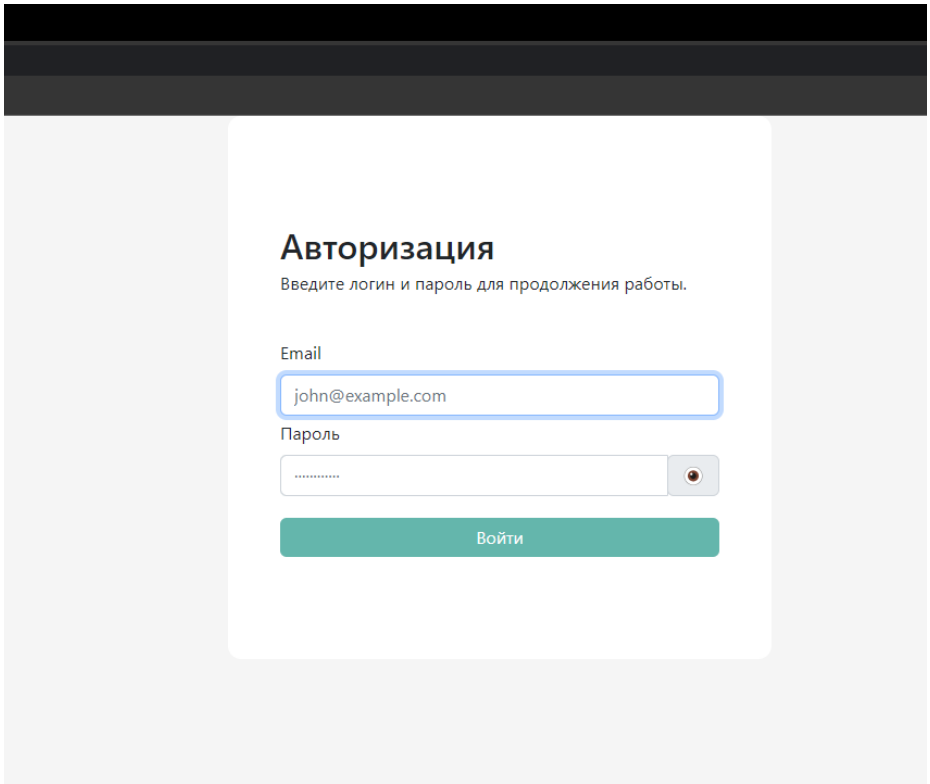

Рисунок 3.17 – Вхід до системи

Процедура входу в систему:

– на сторінці входу в систему користувач вводить свої облікові дані: електронну адресу та пароль;
– система проводить перевірку введених облікових даних і їх відповідності зареєстрованому користувачу;
– після успішної авторизації користувач отримує доступ до відповідних ресурсів та функціоналу системи.

Процес відновлення паролю:

– якщо користувач забув свій пароль, він може використовувати опцію відновлення паролю;
– введення електронної адреси, яка прив’язана до аккаунту;
– на зазначену електронну адресу система відправляє інструкції для відновлення паролю;
– користувач виконує ці інструкції для встановлення нового паролю.

Механізми захисту від несанкціонованого доступу:

– система використовує різноманітні методи для забезпечення безпеки, включаючи шифрування паролів та методи аутентифікації;
– система обмежує кількість невдалих спроб входу до системи, після яких аккаунт тимчасово блокується.

Рекомендується, щоб користувачі використовували надійні паролі та забезпечували їх конфіденційність.

3.4.3 Робота з товарами

Цей розділ відображає каталог товарів з релевантними характеристиками, що використовуються для автоматизованого управління змінами цін та наявності товарів у джерелах продажу.

Він містить наступні елементи:

– ідентифікаційний номер – кожен товар отримує унікальний номер при завантаженні даних в систему;
– категорія: основна класифікаційна одиниця, до якої належить кожен товар (вона виступає батьківським елементом для Підкатегорії-1);
– підкатегорія-1 – додаткова класифікаційна одиниця, яка не є обов'язковою для кожного товару;
– підкатегорія-2 – ще одна додаткова класифікаційна одиниця, яка також не є обов'язковою для кожного товару;
– фото – це поле, яке відображає зображення товару, отримане за посиланням з YML-файлу;
– артикул – унікальний код, що служить для ідентифікації кожного товару;
– назва – це назва товару, зазначена в YML-файлі, яка буде використовуватися в подальших операціях.

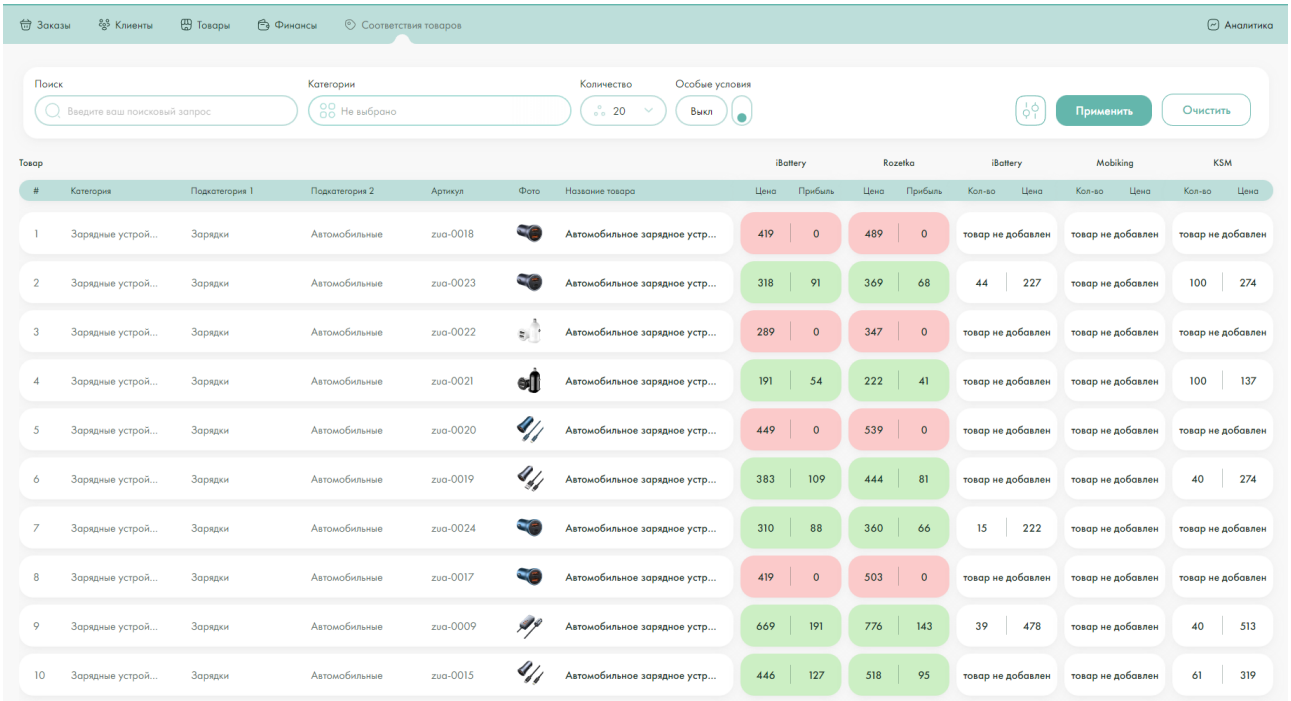

Рисунок 3.18 – Сторінка з керуванням товарами

3.4.5 Робота зі складами

Розділ «Налаштування» -> «Відповідності товарів» -> «Склади» відкриває інтерфейс для роботи зі складами (див. рис. 3.19).

Таблиця «Перелік складів» на верхній частині сторінки включає:

– ідентифікатор;
– найменування;
– власник («власний» або «партнерський»);
– джерело даних («YML-файл або JSON»);
– кількість одиниць – загальна кількість товарів у джерелі;
– додано – число доданих до системи товарів з джерела (стосується партнерських складів, адже не всі товари можуть мати відповідність за артикулами);
– пріоритет: позиція виведення складу у таблиці з товарами.
– статус – статус складу (активний або неактивний). Неактивні склади не відображаються в основній таблиці.

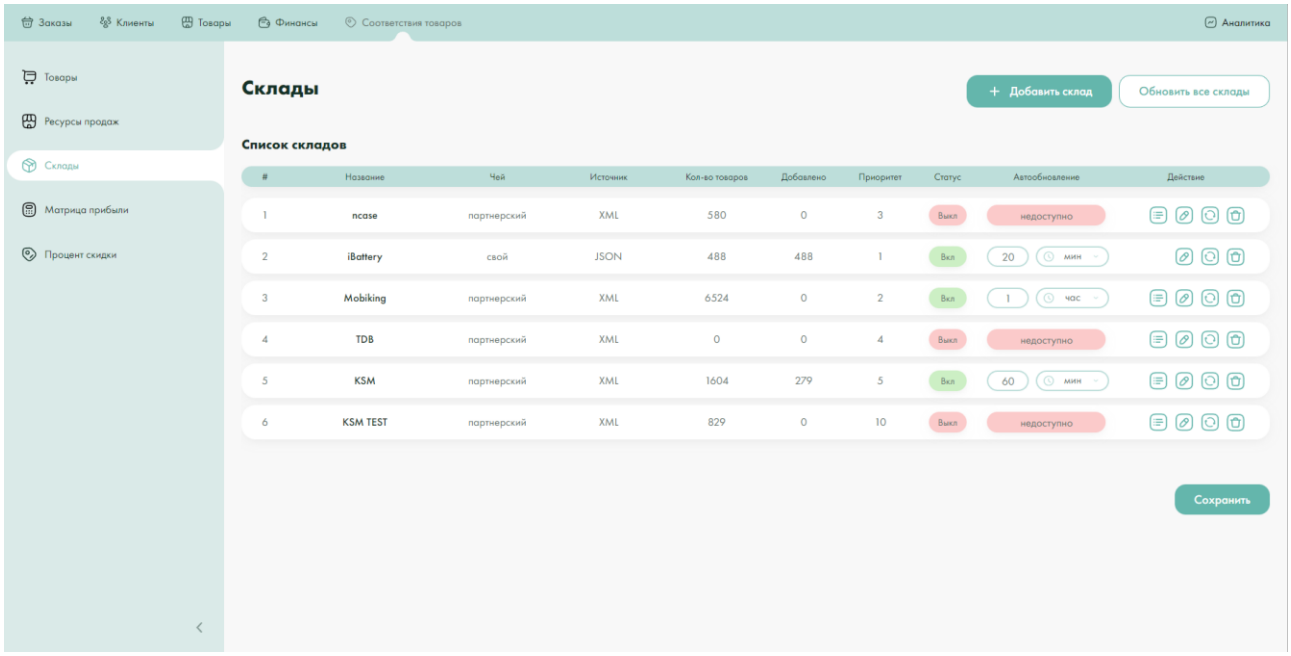

Рисунок 3.19 – Сторінка з керуванням складами

На останній колонці таблиці приведені такі дії (див. рис. 3.20):

– іконка «Відповідність артикулів»: доступне тільки для партнерських

складів, служить для встановлення відповідності артикулів;

– іконка «Змінити»: відкриває форму редагування складу;
– іконка «Оновити»: отримання актуальної інформації про кількість товарів на складі;
– іконка «Видалити»: видалення складу.

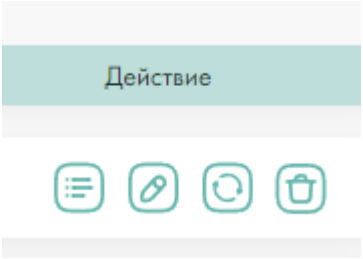

Рисунок 3.20 – Дії над складами

Над таблицею розміщена кнопка «Додати склад», що викликає форму додавання складу (див. рис. 3.21). В цій формі вказується назва та вибирається джерело даних: «YML-файл» або «JSON», задається посилання та встановлюються відповідності артикулів. В розділі «Додаткові» вказується тип складу: «власний” або «партнерський”. Для партнерського складу вказується валюта («грн», «долар», «євро») та обирається тип курсу: «НБУ (Національний Банк України)» або «власний». Для власного курсу потрібно надати посилання, з якого буде отримуватися значення курсу.

Партнерські склади використовують свої артикули товарів, через що вони не збігаються з базовим каталогом. Це вимагає можливості прив'язки артикулів з партнерських складів до власних. При натисканні на іконку Відповідність артикулів, відкривається розділ, яка відображає дві таблиці, вгорі та внизу, що представляють товари партнерського складу і базовий каталог відповідно. При виборі товару з партнерського складу, система, враховуючи лексичні риси назв товарів, виявляє найбільш подібні за назвою товари і сортує їх за ступенем подібності (чим вище в списку – тим вища ймовірність того, що товар є одним і тим же). Крім того, система виділяє однакові слова в назвах товарів з базового каталогу, базуючись на назві вибраного товару з таблиці партнерського складу.

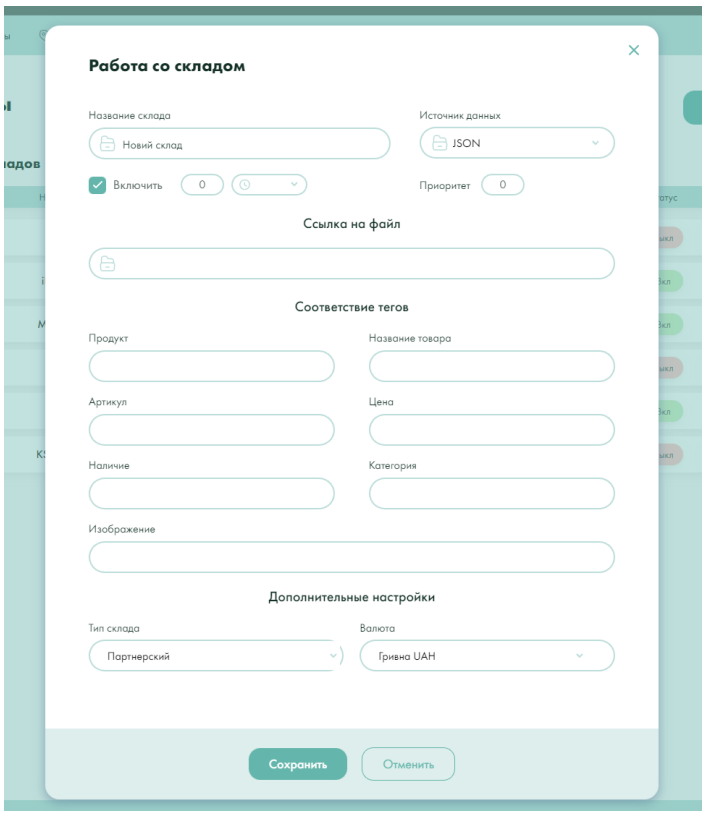

Рисунок 3.21 – Форма додавання нового складу

На рисунку 3.22 продемонстрована робота зіставлення артикулів.

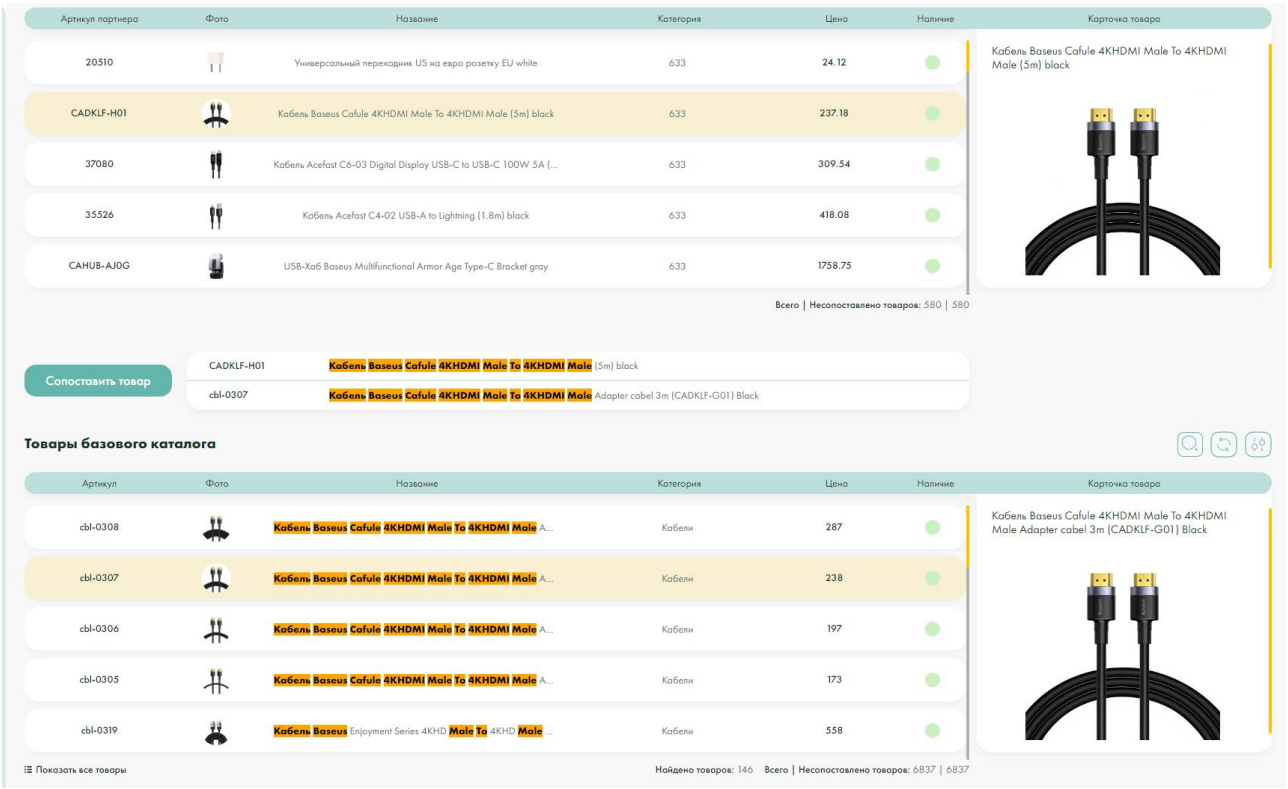

Рисунок 3.22 – Зіставлення артикулів між складом та базовим каталогом

3.4.6 Робота з джерелами продажу

На рисунку 3.23 продемонстрована панель управління, що включає таблицю джерел, де реалізуються товари.

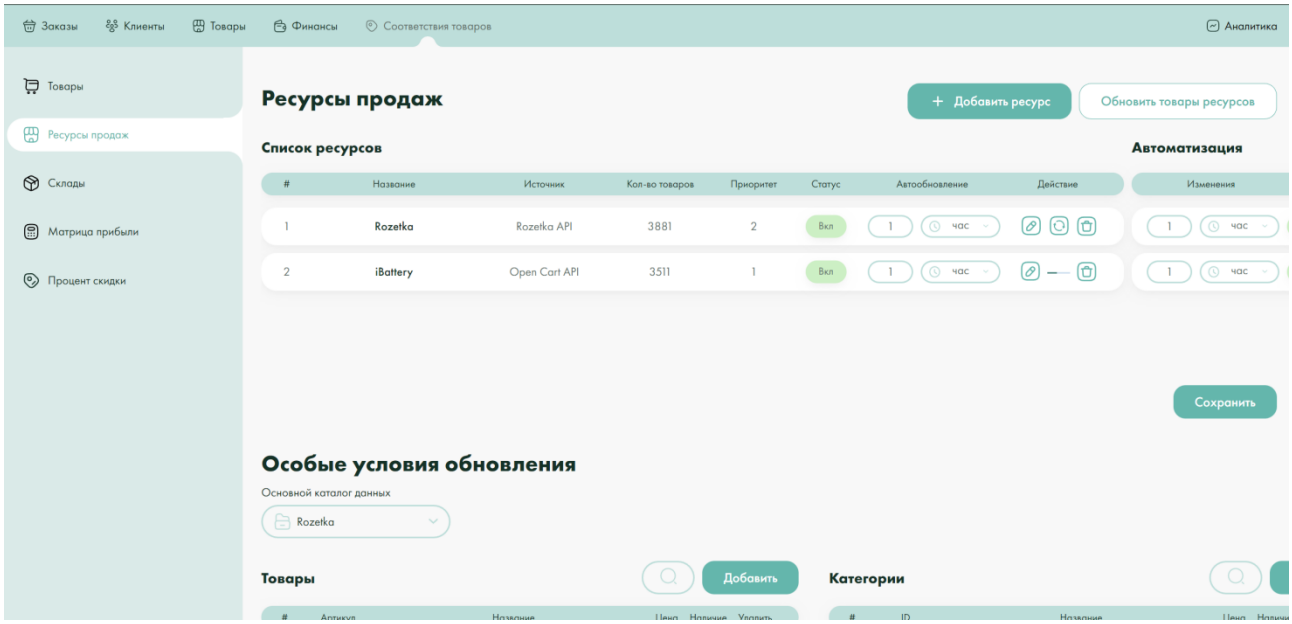

Рисунок 3.23 – Сторінка з керуванням джерелами продажу

В таблиці джерел продажу зображуються такі деталі:

– назва джерела – ім'я платформи чи місця, де товари продаються;
– джерело – це може бути онлайн-магазин, маркетплейс і т.п.;
– статус – це поле, яке вказує, чи активно джерело на даний момент;
– кількість товарів – кількість одиниць товару в наявності;
– останнє оновлення – дата та час останнього оновлення інформації про джерело;
– пріоритет – поле, яке відображає порядок джерел продажу за важливістю. Чим вище пріоритет, тим важливіше джерело продажу для системи;
– статус – це поле, яке відображає поточний стан джерела продажу. Статус може бути активний (коли дані джерела продажу використовуються) або неактивний (коли дані джерела продажу не

використовуються);

– автооновлення – дія, за допомогою якої можна встановити періодичність автоматичного оновлення кількості товарів. Встановлюючи інтервал, ви задаєте частоту, з якою система буде оновлювати дані про наявність товару;
– дії: набір дій, які можна виконати із джерелом продажу.

Колонка з набором дій включає такі кнопки:

– оновлення даних про джерело продажу: дозволяє вручну оновити інформацію про джерело продажу;
– оновлення кількості товарів вручну: ця дія дає можливість вручну запустити автоматичне оновлення кількості товарів та їхні ціни в джерелі продажу;
– видалення джерела продажу: якщо джерело продажу більше не потрібне, його можна видалити за допомогою цієї дії.

ВИСНОВКИ

В результаті виконаної роботи було підготовлено технічне завдання для створення системи, що автоматизує планування задач, що пов’язані з процесами зміни цін та доступності товарів на торгових платформах (магазини, маркетплейси). Для створення клієнтської частини системи були обрані такі технології:

– React Native та Visual Studio Code для розробки клієнтської частини;
– Typescript для реалізації API;
– Zustand для зберігання даних під час сесії;
– Tanstack Query для роботи з асинхронними даними;
– React Navigation для навігації між сторінками;
– React Native Web для підтримки веб-платформ.

За допомогою React Native Web ми можемо переосмислити наш мобільний додаток і адаптувати його для веб-платформи з використанням відомих інструментів та компонентів. Це дозволяє забезпечити кросплатформенний досвід для користувачів, які використовують наш додаток як на мобільних пристроях, так і у веб-браузерах.

В ході виконання роботи, були підготовлені наступні етапи роботи:

– визначення вимог до системи, включаючи функціональні та нефункціональні аспекти (інтерфейс, кросплатформеність, безпека, продуктивність) (додатково було зроблено огляд предметної області та інструментів для розробки);
– проєктування та конструювання структури системи (були розроблені діаграми прецедентів та компонентів з детальним описом);
– реалізація системи для автоматизації процесу зміни цін та наявності товарів на платформах продажу (було надано інструкції щодо створення компонентів системи та опис структури проєкту).

ПЕРЕЛІК ПОСИЛАНЬ

1. Meta Platforms, Inc. React Native Developer Documentation. URL: https://reactnative.dev/ (дата звернення: 05.03.2023).
2. Meta Platforms, Inc. React Developer Documentation. URL: https://react.dev/ (дата звернення: 07.03.2023).
3. Microsoft Corporation. Typescript Developer Documentation. URL: https://www.typescriptlang.org/docs/ (дата звернення: 13.03.2023).
4. React Navigation. Developer Documentation. URL: https://reactnavigation.org/ (дата звернення: 20.03.2023).
5. Tanner Linsley. Tanstack Table Developer Documentation. URL: https://tanstack.com/table/v8/docs/guide/introduction (дата звернення: 02.04.2023).
6. Tanner Linsley. Tanstack Query Developer Documentation. URL: https://tanstack.com/query/latest/docs/react/overview (дата звернення: 14.04.2023).
7. Zustand. Developer Documentation. URL: https://github.com/pmndrs/zustand (дата звернення: 18.04.2023).
8. React Native Web. Developer Documentation. URL: https://necolas.github.io/react-native-web/docs/ (дата звернення: 21.04.2023).
9. Salcescu C. An Introduction to the Flux Architectural Pattern // *Medium*. 2019. https://medium.com/programming-essentials/an-introduction-to-the-flux-architectural-pattern-674ea74775c9 (дата звернення: 25.04.2023).
10. Johannes K. Evolution of a React folder structure and why to group by features right away // *Profy*. 2023. https://profy.dev/article/react-folder-structure (дата звернення: 27.04.2023).
11. Роберт С. М. Чиста архітектура / пер. з англ. І. Бондар-Терещенко. Харків : Фабула, 2019. 368 с.
12. Nader D. React Native in Action: Developing iOS and Android apps with JavaScript. Shelter Island : Manning Publications Co., 2019. 320 p.