Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Надтоки Богдана Віталійовича*
(ПІБ)

академічної групи *121-18ск-1*
(шифр)

спеціальності *121 Інженерія програмного забезпечення*
(код і назва спеціальності)

освітньої програми *Інженерія програмного забезпечення*
(назва освітньої програми)

на тему: *Розробка мобільного додатку під керуванням ОС Android для платформи Market Dynamics Analyzer для моніторингу динаміки цін.*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	*доц. Приходченко С.Д.*			
розділів:				
спеціальний	*доц. Приходченко С.Д.*			
економічний	*проф. Вагонова О.Г.*			
Рецензент				
Нормоконтролер	*доц. Гуліна І.Г.*			

Дніпро
2021

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

_____ І.М. Удовик
(підпис) (прізвище, ініціали)

« » _____ 2021 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *121-18ск-1* *Надтоки Богдана Віталійовича*
(група) (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка мобільного додатку під керуванням ОС Android для платформи Market Dynamics Analyzer для моніторингу динаміки цін.*

затверджена наказом ректора НТУ «ДП» від *07.06.2021 р.* № *317-с*

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів виробничої практики та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі.*** ***Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***13.05.2021 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПО й тривалості його розробки***	***27.05.2021 р.***

Завдання видав _____ *доц. Приходченко С.Д.*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Надтока Б.В.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2021 р.*

Термін подання кваліфікаційної роботи до ЕК: *11.06.2021 р.*

РЕФЕРАТ

Пояснювальна записка: 89 с., 23 рис., 3 дод., 31 джерело.

Об'єкт розробки: мобільний додаток платформи Market Dynamics Analyzer.

Мета кваліфікаційної роботи: розробка мобільного додатку платформи Market Dynamics Analyzer, що дасть можливість користувачам аналізувати ціни на товари у різних продавців та обирати найкращу пропозицію для заощадження власних коштів.

У вступі виконується аналіз сучасного стану проблеми, уточнюється постановка завдання, мета кваліфікаційної роботи та галузь її застосування, обґрунтовується актуальність теми.

У першому розділі проводиться дослідження предметної галузі та існуючих рішень, визначається актуальність завдання та призначення розробки, розроблюється постановка завдання.

У другому розділі обирається платформа для розробки, виконується проектування програми і її розробка, наводиться опис алгоритму і структури функціонування системи, визначаються вхідні та вихідні дані, наводяться характеристики складу параметрів технічних засобів, опис роботи програми.

В економічному розділі визначається трудомісткість розробленого програмного продукту, проводиться підрахунок вартості роботи по створенню застосунку та розраховується час на його створення.

Практичне значення полягає у розробці мобільного додатку платформи Market Dynamics Analyzer, що дозволяє переглядати ціни на товари та робити вибір в сторону кращої пропозиції.

Актуальність програмного продукту визначається великою популярністю Інтернет-торгівлі та малим ринком мобільних додатків для аналізування цін в Україні.

Список ключових слів: МОБІЛЬНИЙ ДОДАТОК, АНАЛІЗ, ЦІНИ, КОРИСТУВАЧ, ПОШУК, ТОВАР, ПРОДАВЦІ, ФРЕЙМВОРК, КЛІЄНТ, СЕРВЕР.

ABSTRACT

Explanatory note: 89 p., 23 figs., 3 appx., 31 sources.

Object of development: mobile application of the Market Dynamics Analyzer platform.

Purpose of the qualification work: development of a mobile application of the Market Dynamics Analyzer platform, which will allow the user to analyze the prices of goods from different sellers and choose the best offer to save their own money.

In the introduction it is analyzed the current state of the problem, clarified the problem, the purpose of the qualification work and the scope of its application, substantiated the relevance of the topic.

In the first section the research of the subject area and existing decisions is carried out, the urgency of the task and purpose of development is defined, the statement of the task is developed.

In the second section the platform for development is chosen, the program design and its development is carried out, the description of algorithm and structure of functioning of system is given, input and output data are defined, characteristics of structure of parameters of technical means are given, work of the program is described.

In the economic section it is determined the complexity of the developed software product, calculated the cost of work to create an application and calculated the time to create it.

The practical significance lies in the development of a mobile application of the Market Dynamics Analyzer platform, which allows you to view the prices of goods and choose the best offer.

The relevance of the software product is determined by the great popularity of e-commerce and small market of mobile applications for price analysis in Ukraine.

The relevance of the software product is determined by the great popularity of startups and funding them by a large number of people through specialized platforms.

Keywords: MOBILE APPLICATION, ANALYSIS, PRICES, USER, SEARCH, GOODS, SELLERS, FRAMEWORK, CLIENT, SERVER.

ЗМІСТ

СПИСОК УМОВНИХ ПОЗНАЧЕНЬ

API – інтерфейс програмування застосунків;

ПЗ – програмне забезпечення;

ОС – операційна система;

ЕОМ – електронно-обчислювальна машина;

MVC – Model-View-Controller;

UML – уніфікована мова моделювання;

XML – розширювана мова розмітки;

JSX – розширення мови JavaScript;

DOM – об’єктна модель документа.

ВСТУП

Всі ми активно купуємо різні товари в Інтернет-магазинах, бо це дуже зручно та може заощадити кошти. Ще більш зручно дивитися товари відразу в декількох магазинах та порівнювати ціни обраних товарів. Порівняння покупок – це звичка, яку виробили клієнти, яка допомагає їм знаходити найкращі пропозиції в Інтернеті. Завдяки великій кількості механізмів порівняння цін та веб-сайтів, що виконують функції порівняння цін, клієнтам все простіше знаходити найкращі пропозиції в Інтернет-магазинах. Незалежно від того, чи хочуть вони просто порівняти ціни на авіаквитки або готелі, чи просто порівняти ціни місцевих магазинів, щоб побачити, де можна отримати найкращі пропозиції. В цьому нам допомагають сервіси порівняння цін, - вони дозволяють вибирати товар та знаходити найнижчу ціну на нього в магазинах.

Більша кількість сервісів порівняння цін в наш час виступають як прайс-агрегатори. Вони об’єднують списки продуктів від різних продавців (магазинів), а гроші заробляють лише на угодах про партнерський маркетинг.

Мобільний додаток для порівняння цін або веб-сайт дозволяє порівнювати ціни на товари, що продаються роздрібними продавцями, щоб показати, де можна придбати товар за доступною ціною, але можуть робити це по різному. Деякі сервіси порівняння цін можуть показувати ціни після сканування штрих-коду, тоді як інші дають інформацію після введення назви товару. Потім сервіс порівняння цін показує список продавців, які продають той самий товар.

Зробивши висновок з вищеописаної інформації, було прийнято рішення розробки мобільного додатку платформи, що дозволить знаходити потрібні користувачеві товари з порівнянням цін з різних магазинів, що пропонують ці товари. Назва цієї платформи – Market Dynamics Analyzer.

Отже завданням даної кваліфікаційної роботи є створення мобільного додатку, що дозволяє порівнювати ціни товарів в різних магазинах.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Загальні відомості з предметної галузі

Споживачі все частіше покладаються на веб-сайти або мобільні додатки порівняння цін для отримання знань про ринок. Ціни, що генеровані цими інструментами можуть виступати в якості контекстних референтних цін, а також впливати на привабливість цін, що виникають пізніше, коли споживачі роблять покупки поза мережею в місцевих магазинах.

Велика сфера веб-сайтів порівняння цін або «веб-агрегаторів» зростає з кожним роком та приносить користь користувачам, посилюючи тиск на конкурентні ціни фірм, ознайомлюючи покупців із більшою кількістю цін [1]. Так як покупці можуть легко порівнювати ціни аналогічних фірм чи брендів за лічені секунди, активне використання сервісів порівняння цін не тільки знижує витрати часу на пошук, але і дає покупцям можливість купувати у фірм, що пропонують кращі пропозиції. Дослідженнями стверджується, що чим вище сприйнята корисність сервісів порівняння цін, тим більше очікується, що покупці будуть використовувати інформацію, доступну на цих сервісах, при прийнятті рішень до покупки, а також після покупки товарів [2].

Різні галузі економічного аналізу (наприклад, зростання та продуктивність) та економічна політика (наприклад, грошово-кредитна та соціальна політика) покладаються на точні показники зміни цін [3].

Для збору інформації для платформи Market Dynamics Analyzer на серверній частині буде використовуватися парсер.

«Синтаксичний аналіз – це друга фаза процесу проектування компілятора, в якій заданий вхідний рядок перевіряється на підтвердження правил та структури формальної граматики [4]». Він аналізує синтаксичну структуру та

перевіряє, чи відповідає вказаний ввід правильному синтаксису мови програмування.

Синтаксичний аналіз у процесі проектування компілятора відбувається після фази лексичного аналізу. Він також відомий як дерево синтаксичного аналізу або дерево синтаксису. Дерево розбору розроблено за допомогою заздалегідь визначеної граматики мови. Аналізатор синтаксису також перевіряє, чи відповідає дана програма правилам, передбаченим контекстно-вільною граматикою. Якщо відповідає, то для цієї вихідної програми парсер створює дерево синтаксичного аналізу. В іншому випадку на екрані відображатимуться повідомлення про помилки.

Парсер (англ. parser; від parse – аналіз, розбір) або синтаксичний аналізатор – це програма, сервіс або скрипт, що збирає дані з вказаних веб-ресурсів, аналізує ці дані, а потім видає у потрібному форматі. Технічно, парсер виконує синтаксичний аналіз даних (наприклад, тексту).

Серед основних сфер використання парсингу виділяються такі:

– пошук товарів та цін в Інтернет-магазинах;

– пошук та наповнення ресурсів текстовим та мультимедійним контентом;

– пошук даних з оголошень, що розміщені на спеціальних ресурсах;

– пошук та збір контактних даних користувачів [5].

Етапи парсингу:

– пошук даних. У програму або скрипт завантажується HTML-код сторінки сайту. С кодом починає працювати скрипт, який розбиває весь текст на лексеми, виділяючи необхідну інформацію;

– вилучення інформації. Пошук даних відбувається завдяки певним наборам знаків, що описують мету пошуку. Цей набір також називається регулярними виразами. Вони дозволяють виділити тільки потрібні фрагменти;

– збереження даних. Після отримання інформація зберігається у вигляді таблиць або зберігається в БД [5].

Для більшого представлення предметної області необхідно виділити такі українські та іноземні сервіси порівняння цін як Hotline, Price.ua, Nadavi (представляє проекти Magazilla та e-Katalog), Google Покупки, BizRate, PriceRunner та інші.

Hotline – це Інтернет-сервіс для порівняння цін на товари.

Щодня користувачі заходять на Hotline понад 450 тис разів. На майданчику представлено майже 4500 магазинів з відгуками. Hotline забезпечує високу конверсію, а також дозволяє націлювати рекламу на певний регіон. Фахівці сервісу кожен день оновлюють каталог та надають партнерам персоналізовану підтримку [6].

З унікальних можливостей Hotline можна виділити можливість переглядання відгуків, які залишені для продавців, що дає можливість обрати певного продавця для своєї покупки.

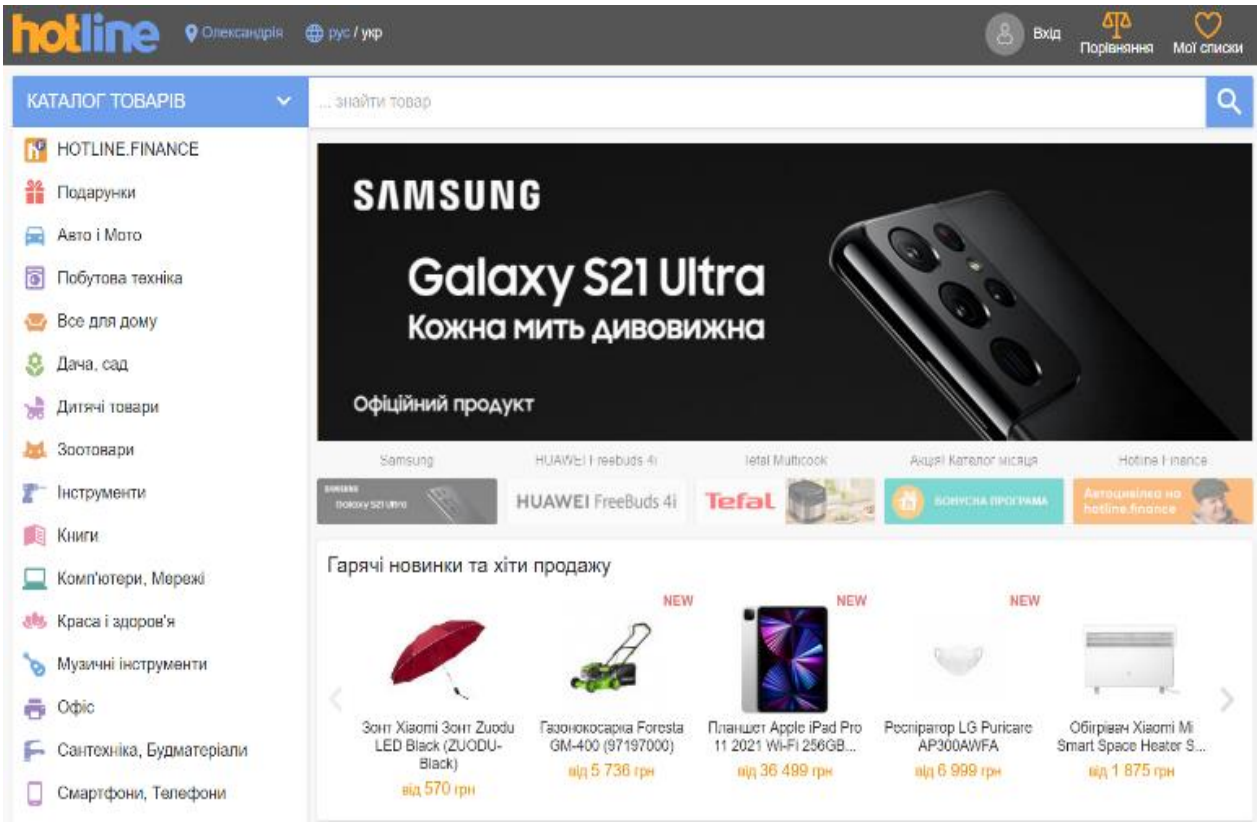

Рис. 1.1. Головна сторінка сервісу «Hotline»

Nadavi – це система, яка забезпечує роботу двох сервісів: e-Katalog (за даними SimilarWeb, 4 млн. користувачів щомісяця) та Magazilla (згідно SimilarWeb, 650 тис. відвідувачів щомісяця). У Magazilla представлені товари більше 1000 інтернет-магазинів. Крім розміщення товарів за категоріями, система Nadavi допомагає з просуванням на рекламних сервісах. Оголошення показуються на сайтах, тематика яких відповідає спрямованості магазину, завдяки чому товари бачить цільова аудиторія [6].

До переваг e-Katalog можна віднести можливість переглянути статті, обзори або корисні поради на певні категорії товарів, що дає деяке представлення користувачеві щодо вибору товару.

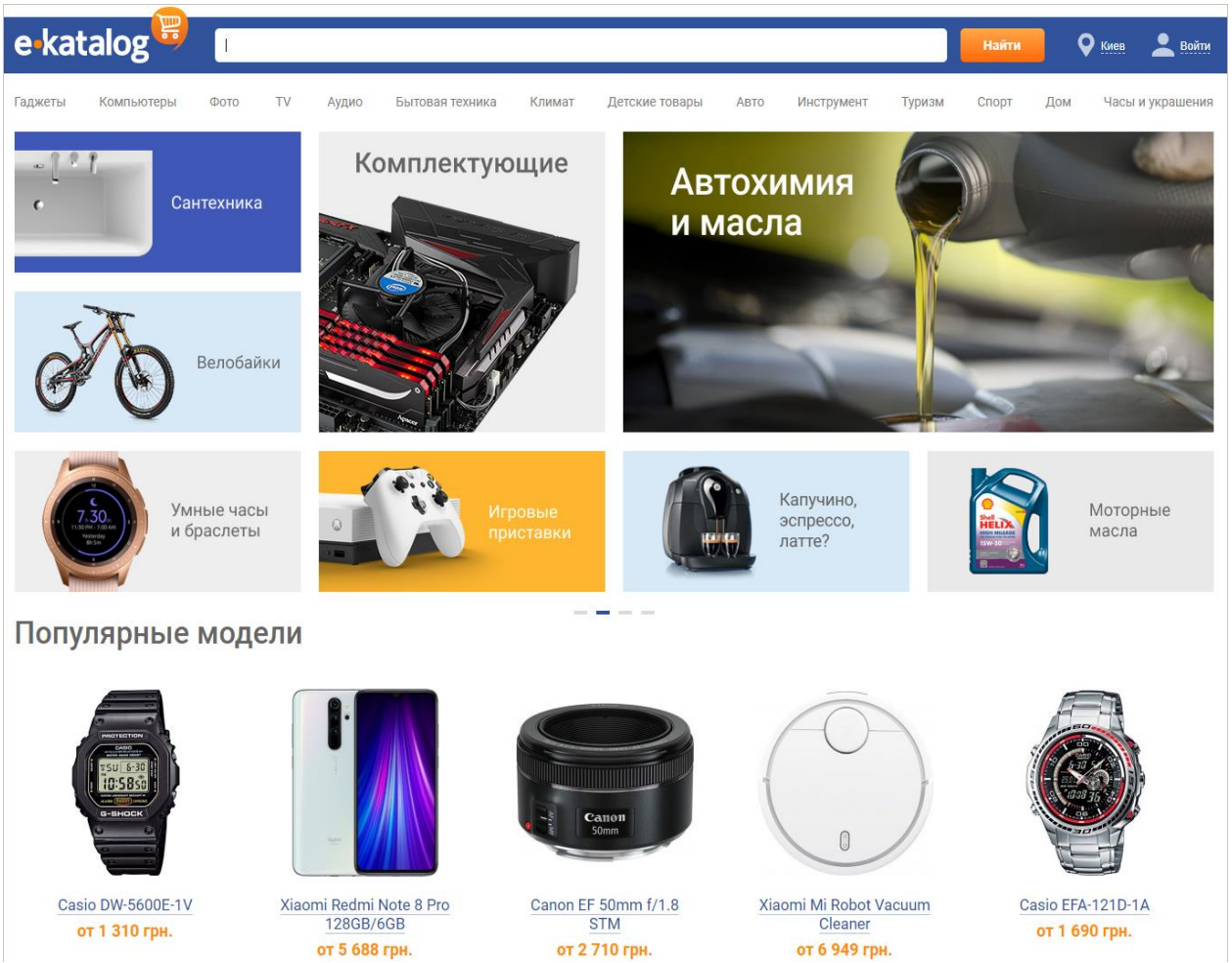

Рис. 1.2. Головна сторінка сервісу «e-Katalog»

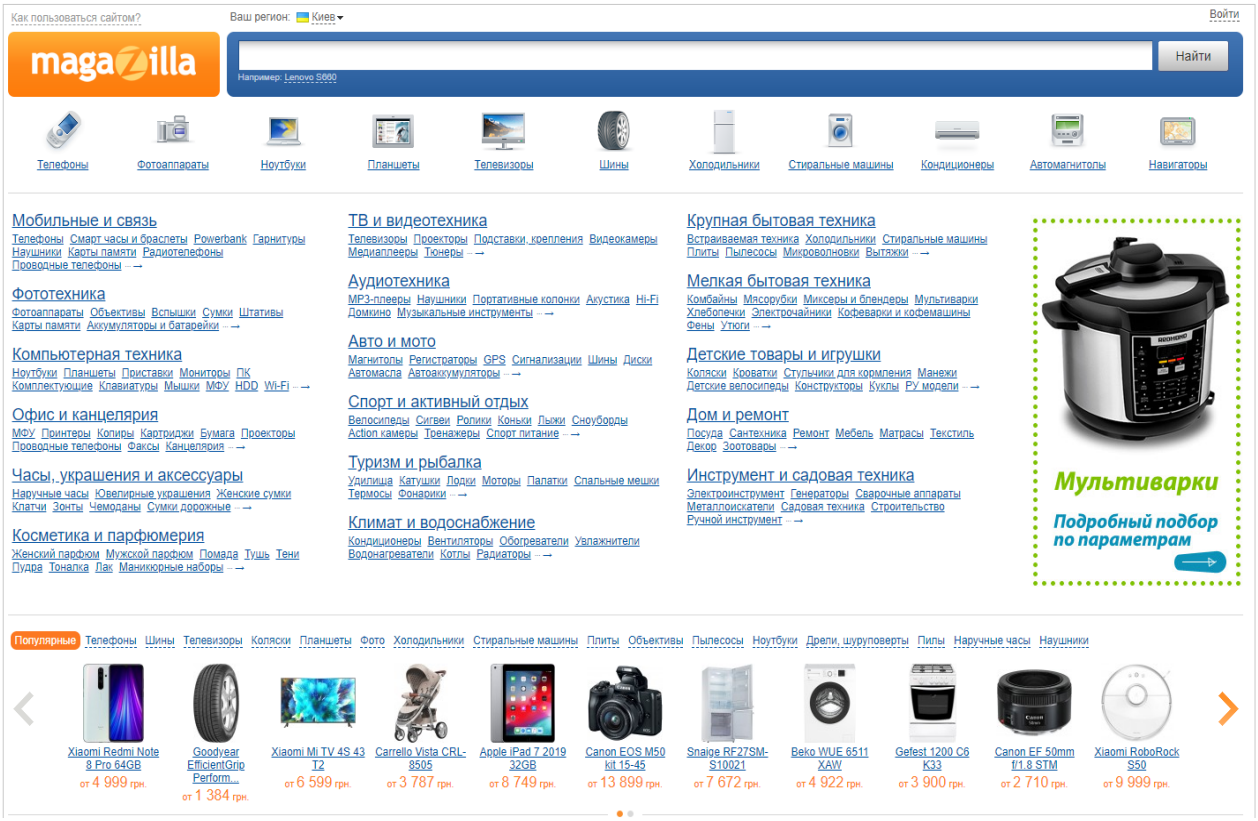

Рис. 1.3. Головна сторінка сервісу «Magazilla»

Google Покупки (раніше відомий як Google Product Ads аудитория, Google PLA) – чудовий ресурс для власників магазинів, які прагнуть знайти конкурентів, що продають подібні товари та порівняти ціни на товари між конкурентами. Власники магазинів можуть навіть додавати свої товари в Google Покупки, щоб залучити більше трафіку назад до своїх магазинів. Користувачі можуть сортувати товари на основі ціни та продавця [7].

Google збирає дані з Інтернету та представляє їх користувачеві. Функція порівняння цін у Google Покупки – це лише частина самої пошукової системи. Все, що користувачеві потрібно зробити, це шукати товар у розділі «Покупки». Користувачеві буде представлено кілька фотографій товару, опис, відгуки та інформація про ціни від різних роздрібних продавців. Товари автоматично перераховуються на основі найнижчої ціни, яка вказана як перший варіант, але є можливість розширити параметри та переглянути повне порівняння.

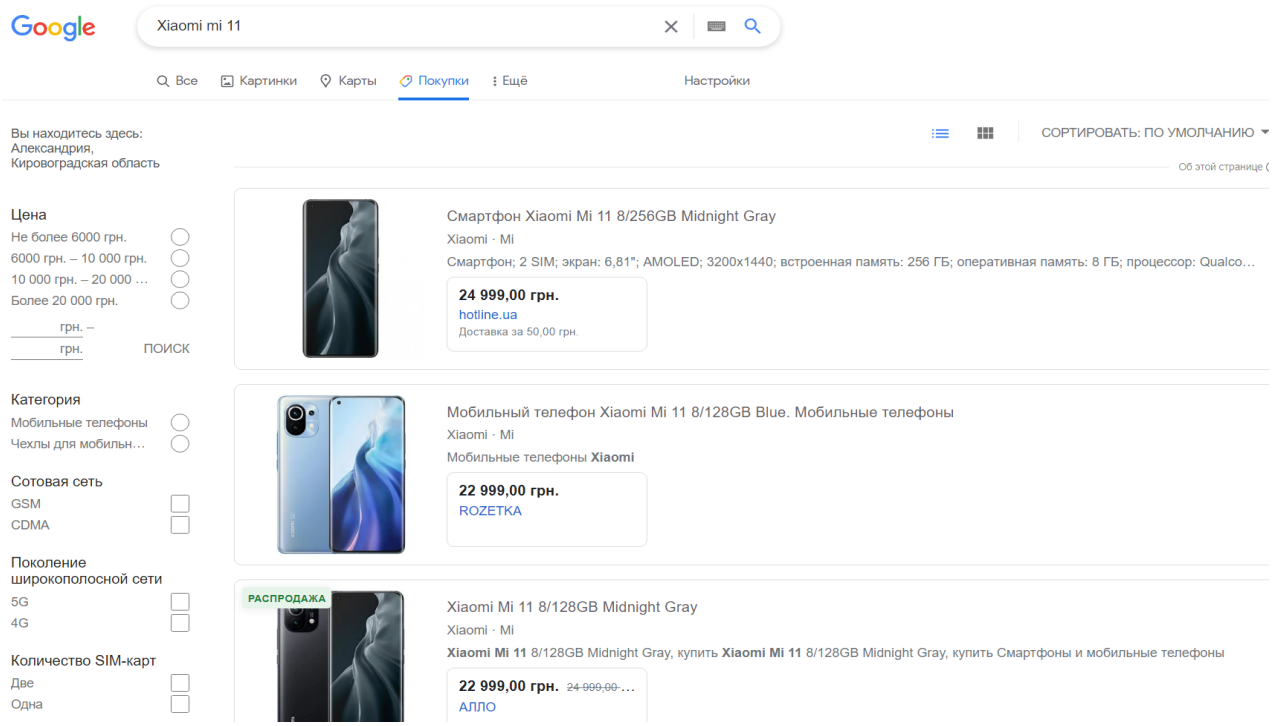

Рис. 1.4. Головна сторінка сервісу «Google Покупки»

BizRate.com працює на відкритому Інтернет-ринку, що регулюється рейтингами клієнтів.Компанія поєднує інформацію, яку генерують клієнти, щоб допомоги покупцям в Інтернеті визначити кращу пропозицію.

BizRate дозволяє клієнтам знаходити найкращі ціни, встановлювати попередження про ціни та здійснювати пошук за незліченними пропозиціями на їх пошуковій системі порівняння цін. Незалежно від того, шукають клієнти конкретний продукт чи дивляться колекції, платформа BizRate пропонує велику різноманітність товарних пропозицій [7]. Коли мова заходить про найкращі механізми порівняння цін, BizRate є високо в списку, що пов’язано з широким спектром результатів. Кілька функцій, які виділяють BizRate, включають можливість завантаження посилань на посібники користувача для PDF для сотень пристроїв та гаджетів. Він також має попередження про ціну, яка проста у використанні. Все що потрібно зробити, це ввести адресу електронної пошти та ціновий поріг, а BizRate повідомить, коли ціна потрапить у діапазон попереджень.

PriceRunner – це сервіс порівняння цін, який запущено у Швеції в 1999 році.

PriceRunner порівнює веб-сайти провідних роздрібних продавців, таких як Amazon, ASOS, House of Fraser та інших провідних брендів Великобританії [7]. Власники магазинів можуть порівняти ціни з авторитетними брендами, щоб визначити, як найкраще оцінити товари на своєму веб-сайті. Цей сервіс порівняння цін має свіжий та простий у використанні сучасний інтерфейс. Користувач має доступ до історії цін, попереджень про ціни, інформації про ціни та відгуки. Якщо користувач віддає перевагу купівлі товару на місцевому рівні, то це може направити його, куди йти.

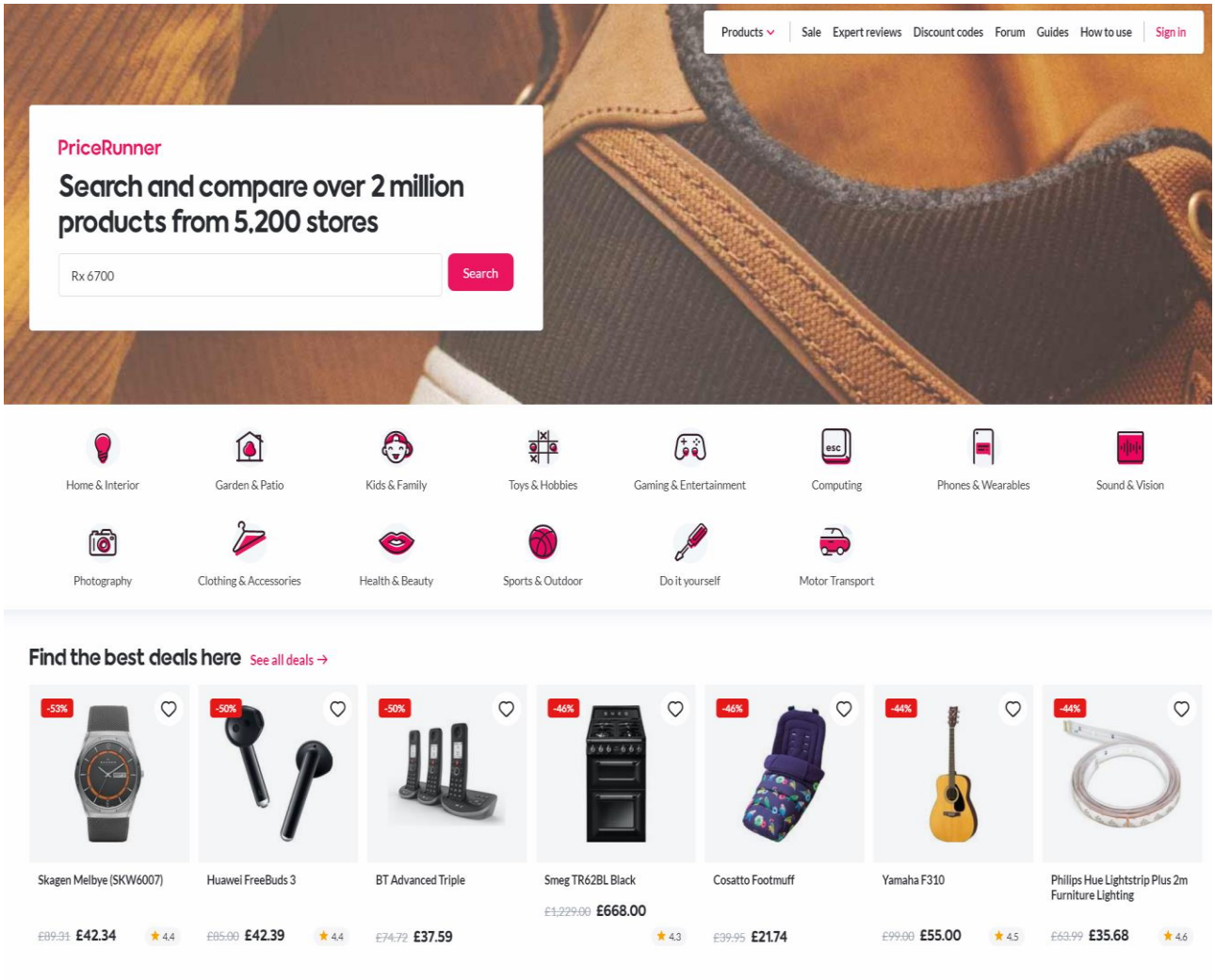

Рис. 1.5. Головна сторінка сервісу «PriceRunner»

1.2. Призначення розробки та галузь застосування

Інформаційна система, що розроблена для кваліфікаційної роботи, має назву «Market Dynamics Analyzer», а сама тема кваліфікаційної роботи має таку назву – «Розробка мобільного додатку під керуванням ОС Android для платформи Market Dynamics Analyzer для моніторингу динаміки цін».

Призначення розробки – надання користувачеві зручно та швидко знайти потрібний товар та порівняти ціни у різних продавців для допомоги зробити правильний вибір та заощадити власні кошти.

1.3. Підстава для розробки

В сучасному світі сервіси порівняння цін користуються досить великою популярністю, але в Україні до сих пір досить мало аналогів таких сервісів, але у вигляді мобільних додатків.

Відповідно до освітньої програми, згідно навчального плану та графіків навчального процесу, в кінці навчання студент виконує кваліфікаційну роботу.

Тема роботи узгоджується з керівником проекту, випускаючою кафедрою, та затверджується наказом ректора.

Отже, підставами для розробки (виконання кваліфікаційної роботи) є:

– досить мала кількість мобільних додатків у галузі порівняння цін;

– освітня програма 121 «Інженерія програмного забезпечення»;

– навчальний план та графік навчального процесу;

– наказ ректора Національного технічного університету «Дніпровська політехніка» № 317-с від 07.06.2021 р;

– завдання на кваліфікаційну роботу на тему «Розробка мобільного додатку під керуванням ОС Android для платформи Market Dynamics Analyzer для моніторингу динаміки цін».

1.4. Постановка завдання

Метою проекту є розробка інформаційної системи для надання інформації про різні товари та їх ціни у різних продавців, яка дозволить користувачеві досить легко робити вибір на користь продавця, що дозволить заощадити власні кошти.

Для досягнення мети необхідно спроектувати та розробити інформаційну систему для надання інформації про товари та їх продавців.

Інтерфейс мобільного додатку повинен бути зручним та зрозумілим для користувача.

Кінцевий розроблений проект має представляти собою мобільний додаток, що буде працювати на операційній системі Android та добре працювати як на телефонах, так і на планшетах.

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

Мобільний додаток повинен мати можливість працювати як з гостями (неавторизованими користувачами) та з авторизованими користувачами.

Гості будуть мати можливість користуватися такими функціями:

- перегляд товарів та категорій товарів;
- пошук товарів за найменуванням;
- перегляд історії пошуку;
- фільтрація та сортування товарів по різним параметрам;
- перегляд цінових пропозицій на товари;
- перегляд графіку цін на товари, а також їх середньої ціни.

Авторизовані користувачі будуть мати всі можливості гостей, а також такі функції:

– додання в “обране” товарів та цікавих для користувача категорій;

– змінювати інформацію свого профіля.

1.5.2. Вимоги до інформаційної безпеки

Для забезпечення надійного функціонування системи необхідно реалізувати основні наступні вимоги:

– перевірка введених даних на валідність: дані при реєстрації, авторизації, пошуку та редагуванні профілю;

– обробка виняткових ситуацій;

– вивід повідомлень про помилки, якщо вони виникають;

– захист від SQL-ін’єкцій;

– захист від несанкціонованого доступу до функціоналу авторизованого користувача.

1.5.3. Вимоги до складу та параметрів технічних засобів

Так як мобільний додаток буде створюватися за допомогою фреймворку React Native, то слід мати такі мінімальні параметри пристрою:

– версія Android: 5+ (API level 21 та вище);

– оперативна пам’ять (RAM): 512 МБ;

– не менш, ніж 100+ МБ вільного місця на накопичувачі;

– також повинен бути встановлений Android WebView для коректного відображення графіків.

1.5.4. Вимоги до інформаційної та програмної сумісності

Для розробки мобільного додатку використовується фреймворк React Native. Вибір на користь даного фреймворку був зробленим тому, що принцип роботи при створенні додатку схожий з розробкою на бібліотеці React. Даний фреймворк добре показує себе при створенні малих та середніх додатків (з досить великими додатками досвіду не було), що підтверджено робочим досвідом.

Для розробки мобільного додатку необхідна наявність наступних програм та систем:

- Node.js (версії 12 та вище);
- JDK (Java Development Kit, версії 8 та вище);
- Android Studio та Android SDK;
- Metro (компілятор для React Native);
- Gradle (система збирання проекту під операційну систему Android).

РОЗДІЛ 2
ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

2.1. Функціональне призначення програми

Результатом даної кваліфікаційної роботи має бути мобільний додаток, який повинен працювати на телефонах та планшетах на ОС Android. Основне призначення додатку – відстеження цін на різні товари.

Мобільний додаток має дотримуватися наступних функціональних характеристик:

- перегляд актуальних цін на товари;
- перегляд графіків середньої ціни та пропозицій;
- перегляд товарів за категоріями;
- фільтрація та сортування товарів;
- пошук товарів за найменуванням та перегляд історії пошуку;
- реєстрація та авторизація користувача;
- можливість додання категорій або товарів в “обране”;
- зміна інформації у профілі.

2.2. Опис застосованих математичних методів

За час проектування та розробки даного мобільного додатку використовувалися лише прості арифметичні дії, наприклад, для отримання адаптивної сітки товарів на головній сторінці та сторінках товарів окремої категорії, а також у профілі користувача. Інші математичні методи не використовувалися.

2.3. Опис використаної архітектури та шаблонів проектування

При проектуванні даного програмного продукту у якості основного архітектурного підходу було обрано архітектуру клієнт-сервер для більш швидкої роботи.

Клієнт-сервер – це обчислювана або мережева архітектура, що розділяє різні завдання та робочі навантаження між провайдерами ресурсу чи послуг, які називаються серверами, та між запитувачами послуг, які називаються клієнтами.

Початкова ідея, яка полягає у вивантаженні функцій управління базами даних на спеціальний сервер, бере свій початок у 1970 році [8]. Тоді комп'ютер, на якому працювала система баз даних, називався комп'ютером баз даних або комп'ютером-сервером, а комп'ютер, на якому працювали програми, називався хост-комп'ютером. Більш пізні терміни для них – це сервер баз даних та сервер додатків, відповідно.

Архітектура клієнт-сервер, як це виглядає сьогодні, стала популярною архітектурою приблизно у 1990 році [8]. До цього розподіл функціональних можливостей бази даних передбачав, що між клієнтськими машинами та серверами не існує ніякої функціональної різниці (тобто, більш рання форма сьогоднішньої однорангової архітектури).

Клієнт-серверні обчислення відіграють життєво важливу роль у формі доступу до даних або інформації, що зберігається віддалено, серед більшості. Система клієнт-сервер відіграє значну роль в розвитку інформаційних технологій. Компоненти, що беруть участь у системі клієнт-сервер, розділені на два великі розділи фізичні та логічні компоненти. Фізичні компоненти - це сервери, клієнтські пристрої, пристрої введення/виведення, мережі та джерела живлення. Логічні компоненти – це веб-сторінки, дані, сценарії програмування, протоколи, наприклад, HTTP, HTTPS, telnet, IP та API (наприклад, JDBC) [9].

Розробка мобільного додатку на React Native здійснюється за допомогою модульної архітектури.

Модульність є одним з найкорисніших інструментів, які використовуються при розробці продуктів. Щодо функціональності, використання модулів є загальним для створення гнучких платформ для виробництва продуктів та сімейств продуктів, які потребують функціональних змін.

Завдяки модульності можна досить легко розробляти компоненти (модулі), а самі компоненти не будуть досить великими, що дає змогу більш легко вносити до них зміни та тестувати.

При розробці ПЗ застосовано ітеративну модель життєвого циклу.

Життєвий цикл ПЗ (Модель життєвого циклу ПЗ) представляє собою процес, що складається з детального плану, що описує, як розробляти, підтримувати, змінювати або вдосконалювати конкретне ПЗ. Цей цикл являє собою процес розробки та розвитку ПЗ.

У ітеративній моделі ітеративний процес починається з простої реалізації невеликого набору вимог до ПЗ та ітеративно вдосконалює нові версії, поки повна система не буде впроваджена та готова до розгортання. Ітеративна модель поєднує в собі послідовні функції моделі водоспаду із швидкими ітеративними функціями моделі-прототипу [10].

Ітеративна модель не потребує повної специфікації вимог для початку розробки. Спочатку розробляється частина функціональних можливостей, яка стає основою для визначення подальших вимог. Цей процес повторюється.

2.4. Опис використаних технологій та мов програмування

При розробці даного мобільного додатку було використано платформу React Native, яка базується на мові програмування JavaScript та бібліотеці React.

Нижче наведено список основних технологій, які використовувалися при розробці:

– JavaScript;
– React Native (включаючи React);
– React Navigation 5;
– Async Storage.

JavaScript (часто скорочується до JS) – це легка, інтерпретована, об’єктно-орієнтована мова з функціями першого порядку і найвідоміша як мова сценаріїв веб-сторінок. Також JavaScript може використовуватися в багатьох середовищах, які не належать до браузера, наприклад, для створення мобільних додатків або стаціонарних застосунків. Це мова сценаріїв на основі прототипів, що є динамічною та має підтримку об’єктно-орієнтованого, імперативного та функціонального стилів програмування.

JavaScript властиві такі особливості, які присутні у функціональних мовах програмування – функції як об’єкти першого класу, об’єкти як списки, каррінг, анонімні та стрілочні функції, замикання, а автоматична збірка сміття та динамічна типізація дає додаткову гнучкість при розробці.

JavaScript працює на стороні клієнта, що може використовуватися для програмування поведінки веб-сторінок при настанні яких-небудь подій.

Також JavaScript може застосовуватися як скриптова мова на стороні сервера. JavaScript сервер надає об’єкти середовища, які мають об’єкти HTTP запитів та відповідей, що можуть бути використані програмою на JavaScript для динамічної генерації веб-сторінок.

JavaScript може використовуватися для:

– створення сценаріїв веб-сторінок, щоб надати їм інтерактивності;
– створення односторінкових (SPA) та прогресивних веб-застосунків (з застосуванням React, Vue.js або AngularJS);
– створення серверів (Node.js та Express для Node.js);

– створення стаціонарних застосунків (з застосуванням Electron, NW.js);
– створення мобільних додатків (з застосуванням React Native, Cordova);
– створення сценаріїв в прикладних програмах (наприклад, в Apache JMeter) [11].

Mozilla в даний момент надає дві реалізації JavaScript. Найперша реалізація JavaScript була розроблена Бренданом Ейхом (Brendan Eich) у компанії Netscape у 1995 році [12]. Ця реалізація оновлюється з тих пір для того, щоб відповідати ECMA-262 Edition 5 і пізніших версій. Цей движок називається SpiderMonkey та реалізований на мові C/C++. Движок Rhino створений Норрісом Бойдом (Norris Boyd) та реалізований на мові Java. Як і SpiderMonkey, Rhino відповідає ECMA-262 Edition 5 [13].

Крім вище приведених існують також інші реалізації, наприклад, V8 від Google, який використовується в браузерах Google Chrome, Opera та інших, які використовують Chromium.

Актуальною на даний момент є версія ES2020 (ES11), що вийшла у червні 2020 року. Вона додала до мови новий тип даних BigInt, оператор ?? для перевірки на null та undefined, оператор опціональної послідовності в об'єкті, динамічні виклики імпортів, об'єкт globalThis, нові методи String.prototype.matchAll для пошуках в рядках з використанням регулярних виразів та Promise.allSettled, що повертає проміс (promise), який виконується тоді, коли всі отримані проміси завершені, який містить масив результатів виконання отриманих промісів [14].

Також JavaScript набуде нових можливостей у червні 2021 року після виходу нової версії ECMAScript 2021 (ES12).

React – це бібліотека JavaScript, що застосовується для створення швидких та інтерактивних користувацьких інтерфейсів для веб-застосунків та мобільних додатків. Це компонентна та інтерфейсна бібліотека з відкритим кодом, яка відповідає тільки за рівень представлення додатку. В архітектурі Model View

Controller (MVC) рівень представлення відповідає за те, як додаток виглядає та відчувається. React був розроблений Facebook для розробки інтерфейсу користувача в 2013 році. Також React часто використовується для створення односторінкових застосунків (SPA) [15].

Вся структура веб-сторінок може бути представлена за допомогою Document Object Model (DOM). Для взаємодії з DOM застосовується мова JavaScript. Однак, коли здійснюється маніпулювання з HTML-елементами за допомогою JavaScript, можна зіткнутися зі зниженням продуктивності, особливо при маніпулюванні з великою кількістю елементів. Операції над елементами можуть зайняти якийсь час, що негативно позначиться на користувацькому досвіді. Для вирішення проблем продуктивності з’явилася концепція віртуального DOM (Virtual DOM).

React використовує віртуальний DOM для оптимізації оновлення змін на сторінці. «Віртуальний DOM – це легковажна копія DOM, що має всі його властивості. Коли відбувається оновлення, весь віртуальний DOM оновлюється для відслідковування майбутніх змін [16]».

При розробці на React використовується модульна архітектура – можна створити окремі компоненти, а потім легко переносити їх з проекту в проект. Компоненти можуть створюватися за допомогою JSX. JSX представляє комбінацію коду JavaScript та XML і надає простий та інтуїтивно зрозумілий спосіб для визначення коду візуального інтерфейсу.

React Native – це кросплатформний фреймворк (платформа) з відкритим кодом для створення додатків для Android та iOS за допомогою React та власних можливостей платформи. За допомогою React Native використовується мова JavaScript для доступу до API платформи Android чи iOS.

Основні принципи розробки на React Native схожі з принципами розробки React, за виключенням того, що React Native керує не браузерним DOM, а платформними інтерфейсними компонентами.

Подібно до React розробки для веб-застосунків, додатки на React Native пишуться з використанням JavaScript та XML, відомих як JSX. Додаток React Native складається з двох сторін, сторони JavaScript та нативної сторони. Нативною стороною може бути Swift для iOS, або Java/Kotlin для Android. React Native Bridge дозволяє нативному коду та коду на JavaScript спілкуватися між собою. Без моста для нативного коду неможливо надіслати будь-яку інформацію до коду JavaScript та навпаки [17]. Таким чином програма, що розроблена на React Native, відображатиметься за допомогою реальних компонентів мобільного інтерфейсу, а не веб-переглядів (webview), і виглядатиме та відчуватиметься як будь-яка інша мобільна платформа.

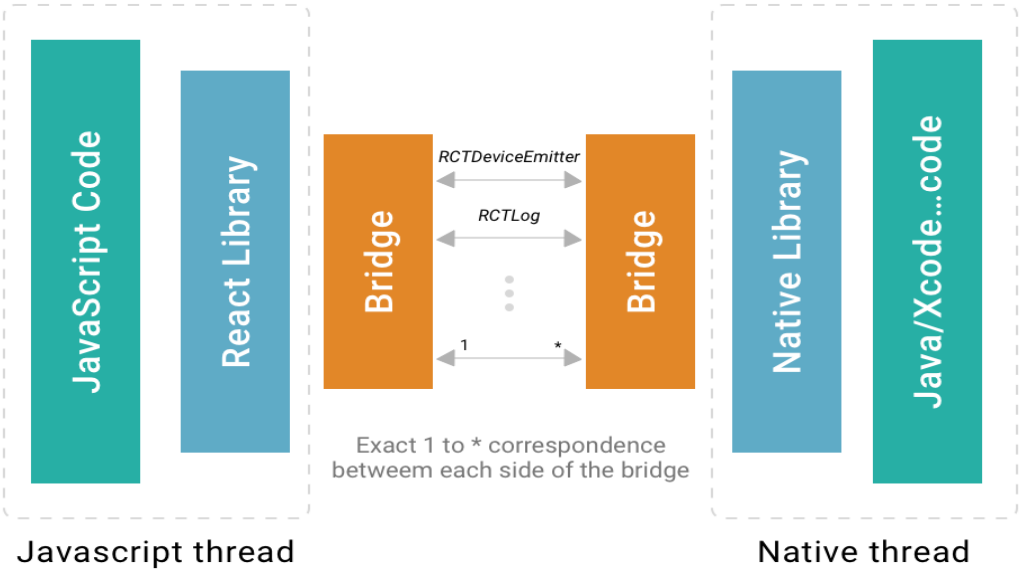

Рис. 2.1. Структура спілкування між JavaScript та нативною стороною

Можна виділити такі переваги:

– однією з найважливіших переваг React Native для спільноти розробників є багаторазове використання коду. Все, що потрібно зробити розробникам, - це просто повторно використовувати 90% коду React Native для iOS та Android, замість того, щоб створювати окремі програми для iOS та Android;

– React Native дозволяє всій спільноті розробників безкоштовно вивчити всю документацію, що стосується цієї технології, а також дозволяє їм брати участь у ній, коли вони цього хочуть;

– ефективність платформи полягає у підвищенні продуктивності за допомогою власних модулів та елементів управління. React Native працює, взаємодіючи з власними компонентами Android і iOS;

– є можливість Hot Reloading, що відображає зміни в коді без перезапуску додатку;

– основній структурі React Native не вистачає певних компонентів. Щоб заповнити цей недолік, він забезпечує розробникам доступ до сторонніх плагінів, таких як модулі JavaScript та власні модулі.

– навички, отримані в React Native, можуть бути застосовані до розробки на React та навпаки [18].

Також виділяють такі недоліки:

– молодий фреймворк. В даний час це не версія 1.0.0, тому деякі компоненти все ще відсутні і сама технологія досить часто оновлюється. Для розробника «незрілість» React Native означає, що йому потрібно стежити за версією як самого React Native, так і бібліотек залежать від нього.

– хоча React Native підтримує величезну кількість API-інтерфейсів, все ще існує потреба у використанні інших API через вбудовані модулі. Ці модулі в основному є частиною коду, написаного на нативній мові, після чого інтегрованого в існуючий код [18].

React Navigation – це популярна бібліотека для організації маршрутизації та навігації в додатках React Native. Ця бібліотека дозволяє вирішити проблему навігації між різними екранами та використанням спільних даних між різними екранами.

React Navigation написана на JavaScript та не використовує напряму власні навігаційні API на Android та iOS. Це дозволяє інтегрувати сторонні бібліотеки JavaScript без необхідності вивчати Swift, Java або Kotlin.

На даний момент стабільною версією є React Navigation 5, яка випущена в лютому 2020 року. «За даними блогу React Navigation, версія 5 була направлена на те, щоб зробити базову бібліотеку React Navigation та API більш динамічними [19]». Нижче представлені основні зміни у версії 5:

– динамічна конфігурація на основі компонентів;

– нові хуки для загальних випадків використання, включаючи useNavigation, useRoute, useNavigationState;

– новий метод setOptions, що робить налаштування параметрів навігації по екранам більш інтуїтивним;

– інтеграція Redux DevTools;

– власний стек-навігатор, який використовує власні примітивні навігації для навігації з використанням рідних екранів.

Є також інші бібліотеки для вирішення проблем навігації, наприклад: React Native Navigation, React Native Router та React Router. Останній використовується не тільки для React Native, а також і для React.

Обрано саме React Navigation тому, що він повністю написаний на JavaScript, що дозволяє уникнути деяких проблем при розробці додатку, а також у React Navigation зручна документація.

Async Storage – це простий, асинхронний, незашифрований за замовчуванням модуль, який дозволяє зберігати дані в автономному режимі у додатках React Native. Збереження даних здійснюється в системі зберігання «ключ – значення».

Існує безліч сценаріїв, коли цей модуль може бути корисним. Збереження даних у мобільному додатку має такі переваги, наприклад у такому випадку, коли користувач перезапустив мобільний додаток, а дані або змінні

налаштування доступні користувачеві в стані, в якому він їх залишив перед закриттям додатку.

При розробці даного проекту Async Storage використовується для збереження токену користувача, що авторизувався у системі, а також для збереження історії пошуків.

2.5. Опис структури програми та алгоритмів її функціонування

Мобільний додаток базується на посиланні запитів на серверну частину. У якості формату даних для запитів та відповідей на запити використовується формат збереження даних JSON.

JSON (JavaScript Object Notation) – це формат для збереження даних та обміну інформацією, яка зрозуміла для людини. Він являє собою текст, що містить числа, рядки, колекції пар «ім’я – значення», масиви даних. Не дивлячись на той факт, що JSON походить від JavaScript («а якщо уточнювати, то від підмножини мови стандарту 1999 року – ECMA-262, 3 видання [20]»), він вважається незалежним від JavaScript та може бути використаний у всіх мовах програмування. Також у більшості мов програмування є вбудовані функції та методи для роботи з форматом JSON.

При проектуванні було розроблено декілька діаграм мови UML (Unified Modeling Language) для кращого розуміння розробки мобільного додатку, а саме Use Case та Sequence діаграми.

«Use Case (варіантів використання або прецедентів) діаграми використовуються для аналізу загальних вимог до системи. Ці вимоги виражаються в різних випадках використання. При цьому нічого не говориться про реалізацію цих вимог [21]».

Система, що проектується, представляється на діаграмі у вигляді акторів, прецедентів (варіантів використання) та відношень між ними. Актором може

бути людина, організація або внутрішній/зовнішній додаток. Усередині кругових контейнерів виражаються дії, які актори виконують.

На діаграмі, що зображена на рис. 2.2 видно два види користувачів.

Рис. 2.2. Use Case діаграма мобільного додатку

Тепер більш докладніше про них:

– гість – користувач, який не здійснив вхід до системи, він має обмежений функціонал мобільного додатку;

– авторизований користувач – користувач, який здійснив вхід до системи та має повний доступ до всього функціонала мобільного додатку.

Гість не буде мати доступу до свого профілю у системі, тому не зможе редагувати свої списки товарів та категорій, що його цікавлять, а також редагувати інформацію у профілі (аватар, ім’я та прізвище).

Щодо функціональних можливостей мобільного додатку, що зображено на рис. 2.2, то їх можна пояснити так:

– перегляд товарів – користувач має можливість переглядати сітку товарів, інформацію про певний товар, що його цікавить (перегляд графіків цін у продавців та середньої ціни);

– перегляд категорій товарів – користувач має можливість обрати категорію товарів, яка його цікавить, щоб показати товари лише цієї категорії з можливістю сортування та фільтрації;

– перегляд профілю – авторизований користувач має можливість переглядати власний профіль у системі;

– пошук товарів – користувач має можливість користуватися пошуком товарів у системі;

– перегляд історії пошуків – користувач має можливість обрати декілька з останніх варіантів пошуку;

– фільтрація товарів – користувач має можливість користуватися фільтром з різними налаштуваннями;

– сортування товарів – користувач має можливість використати сортування по ціні або за замовчуванням;

– додання товару в “обране” – авторизований користувач має можливість додати товар в “обране”, щоб зручніше слідкувати за ціною;

– додання категорії в “обране” – авторизований користувач має можливість додати категорію в “обране”, тоді на головній сторінці одними з перших товарів будуть йти саме товари з обраних категорій.

Окрім Use Case діаграми також було розроблено дві Sequence діаграми, які будуть нижче.

Sequence (послідовності) діаграма – це діаграми взаємодії, що детально описують спосіб виконання операцій. Вони фіксують взаємодію між об’єктами в контексті співпраці. Діаграми послідовності орієнтовані на час, вони візуально

показують порядок взаємодії, щоб представити час, які повідомлення надсилаються та коли.

Sequence діаграми складаються з таких основних компонентів:

– актори – представляє тип ролі, коли здійснюється взаємодія із системою та її компонентами;

– об’єкти – показують взаємодію в рамках певного сценарію;

– повідомлення (виклики методів) – лінії зі стрілками;

– повідомлення з результатом – пунктирні лінії зі стрілками [22].

На діаграмі послідовності, що зображена на рис. 2.3, видно алгоритм додання певного товару до “обраного”.

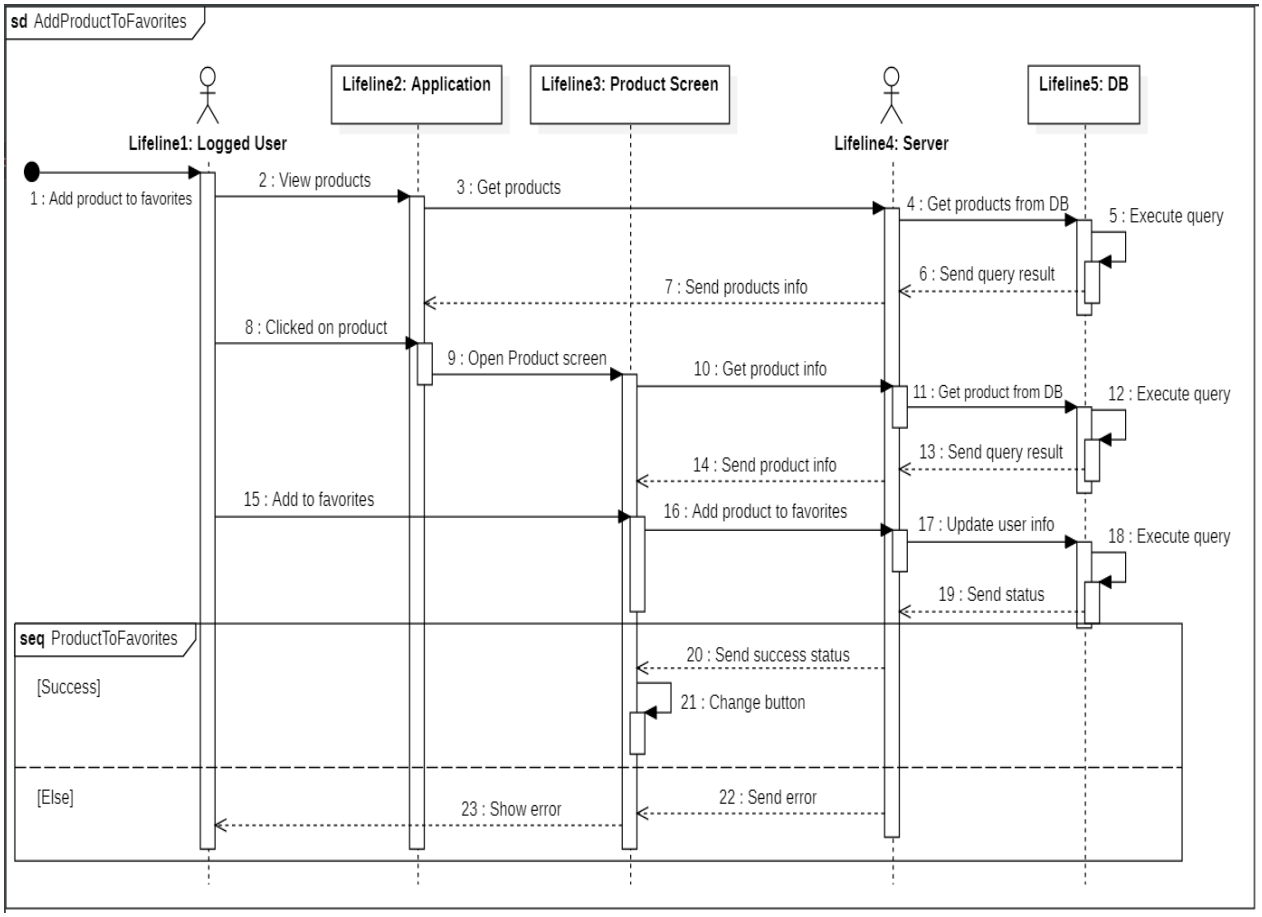

Рис. 2.3. Sequence діаграма мобільного додатку (додання товарів в “обране”)

Додати товар до “обраного” можливо, якщо користувач увійшов в систему та перейшов до сторінки певного товару, там буде кнопка додання до “обраного”.

Нижче, на рис. 2.4 показано ще одну діаграма послідовностей, де представлений алгоритм проходження реєстрації та авторизації в системі.

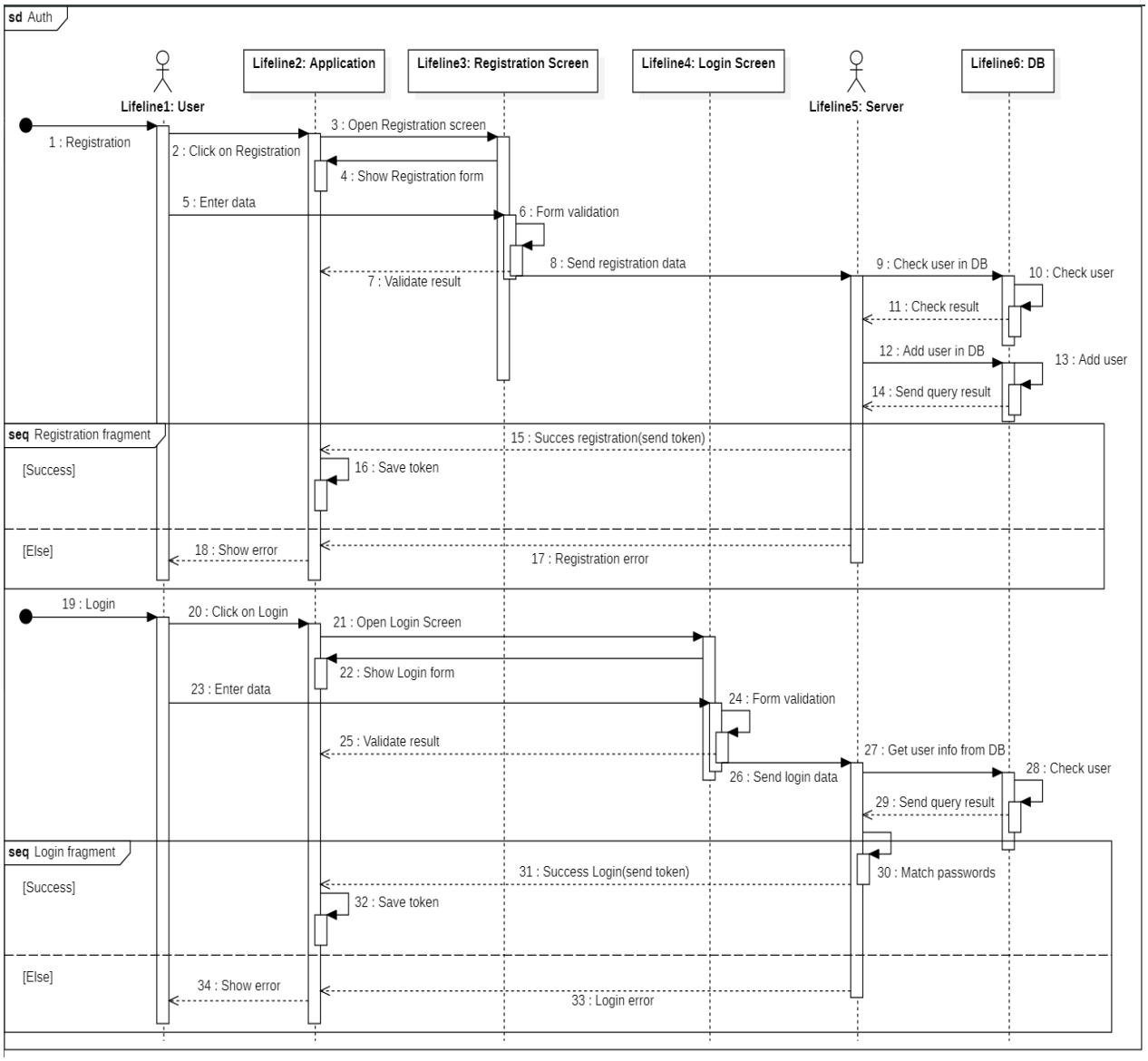

Рис. 2.4. Sequence діаграма мобільного додатку (реєстрація та авторизація)

На рис. 2.5 показано список екранів, з якими користувач має можливість працювати.

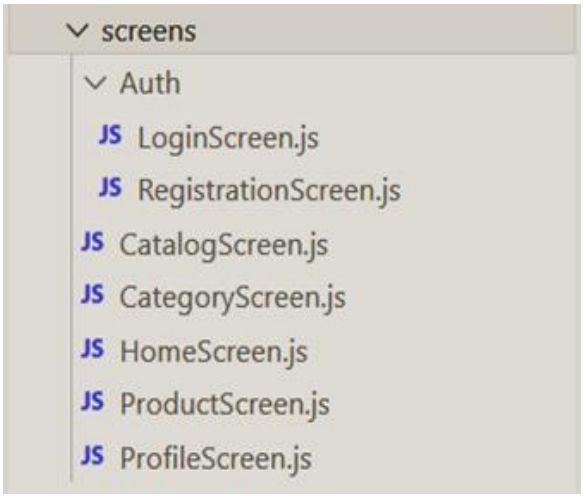

Рис. 2.5. Список екранів, що розроблені для мобільного додатку

Зараз інформація про кожен екран окремо:

– HomeScreen.js – являє собою головну сторінку мобільного додатку, показує всі товари з можливістю пошуку, також на зміст товарів впливають обрані категорії авторизованим користувачем;

– ProductScreen.js – відображає інформацію про обраний товар (графіки цін, опис, фотографії товару);

– CatalogScreen.js – відображає список усіх доступних категорій;

– CategoryScreen.js – відображає список товарів обраної категорії з можливістю сортування та фільтрації;

– ProfileScreen.js – доступний для авторизованого користувача екран, де користувач може редагувати свою інформацію, додавати та видаляти категорії/товари з “обраного”;

– RegistrationScreen.js – являє собою екран з формою реєстрації;

– LoginScreen.js – являє собою екран з формою авторизації в систему.

Також при розробці мобільного додатку було створено декілька власних компонентів для введення даних, таких як: текстове поле для введення даних (text input), прапорець (checkbox), радіокнопка (radio button).

2.6. Обґрунтування та організація вхідних та вихідних даних програми

При проектуванні даної платформи було визначено вхідні та вихідні дані, що стосуються товарів та користувачів, що в системі реєструються.

Вхідними даними системи є:

– дані, що вводяться користувачем при реєстрації (ім’я, прізвище, електронна пошта, пароль та повторений пароль);

– дані, що вводяться користувачем при авторизації (електронна пошта та пароль);

– дані налаштування відображення товарів (обрані користувачем опції фільтрації та сортування), а також дані при пошуку товарів (найменування товарів);

– дані, що стосуються додання категорій/товарів до “обраного” (ідентифікатор категорії або товару);

– дані, що вводяться користувачем при зміні інформації профілю (фотографія, ім’я та прізвище).

Вихідними даними є:

– вся інформація про товари (ідентифікатор, назва, опис, фотографії, ціни, кількість товарів певної категорії);

– вся інформація про категорії (ідентифікатор, назва);

– вся інформація, що стосується користувача (список “обраного”, інформація користувача, токен авторизації);

– дані, що стосуються налаштування відображення товарів (опції фільтрації та сортування, історія пошуку користувача).

Усі вхідні дані передаються на сервер у форматі JSON. Відповіді з серверної частини також приходять до клієнта у форматі JSON.

2.7. Опис роботи розробленого програмного продукту

2.7.1. Використані технічні засоби

Так як мобільний додаток буде створюватися за допомогою фреймворку React Native, то слід мати такі мінімальні параметри пристрою:

– версія Android: 5+ (API level 21 та вище);

– оперативна пам’ять (RAM): 512 МБ;

– не менш, ніж 100+ МБ вільного місця на накопичувачі;

– також повинен бути встановлений Android WebView для коректного відображення графіків.

2.7.2. Використані програмні засоби

Під час розробки даного застосунку були використані такі програмні засоби:

– Visual Studio Code;

– Android Studio;

– Gradle;

– Metro;

– Git та GitHub.

Visual Studio Code (VS Code) – це легкий та потужний редактор вихідного коду, що доступний на операційних системах Windows, macOS та Linux [23]. Також VS Code має багато розширень для інших мов та середовищ виконання (.NET та Unity).

«VS Code створений з використанням Electron, Node.js, TypeScript та протоколу Language Server [24]». Багатство підтримки варіюється в залежності від різних мов програмування (JavaScript, C++, C#, PHP, Python та інші) та їх

розширень, починаючи від простої підсвітки синтаксису та відповідності дужок до налагодження та рефакторингу. Також VS Code підтримує інтеграцію з Git.

VS Code є проектом з відкритим кодом під ліцензією MIT. Сам VS Code постачається за стандартною ліцензією продукту Microsoft, оскільки має невеликий відсоток налаштувань, специфічних для Microsoft.

Android Studio – це офіційне інтегроване середовище розробки (IDE) для розробки додатків на Android, засноване на IntelliJ IDEA [25]. На додаток до потужного редактора коду та інструментів розробника IntelliJ, Android Studio пропонує ще більше функцій, які підвищать продуктивність при створенні програм для Android, таких як:

– інструмент автоматичної збірки – Gradle;
– емулятор Android (Android Virtual Device);
– широкі інструменти для тестування;
– інтеграція з GitHub.

Використовувався при розробці даного проекту у якості інструменту для тестування на мобільних та планшетних пристроях, що можливо завдяки емулятору Android.

Gradle – це інструмент автоматизації збірки з відкритим кодом, що розроблений для того, щоб бути досить гнучким для створення майже будь-якого типу програмного забезпечення [26]. Gradle поширюється як програмне забезпечення з відкритим кодом під ліцензією Apache 2.0 та вперше вийшов у 2007 році.

Gradle був розроблений для багатопроектних збірок, які можуть зростати до великих розмірів. Він працює на основі ряду завдань збірки, які можуть виконуватися послідовно або паралельно. Покрокові збірки підтримуються шляхом визначення частин дерева збірки, які вже оновлені. Gradle також підтримує кешування компонентів збірки за допомогою Gradle Build Cache.

Використовується при розробці проекту у якості інструменту для збірки мобільного додатку в файл .APK, щоб його можна було встановити на будь-якому пристрої, що відповідає мінімальним технічним параметрам.

Metro – це збірник JavaScript, який приймає опції, вхідний файл та повертає файл JavaScript, що включає всі файли JavaScript [27]. Кожен раз, коли запускається проект React Native, відбувається компіляція багатьох файлів в один файл.

Metro запускається разом із сервером вузлів, який запускається за допомогою запуску «npm».

Використовується при розробці проекту у якості інструменту для відладки на фізичних пристроях та емуляторі в Android Studio.

Git – це безкоштовна система управління розподіленими версіями з відкритим кодом [28]. Git зазвичай використовується для зручної розробки програмного забезпечення.

Git був створений Лінусом Торвальдсом у 2005 році для розробки ядра Linux, а інші розробники ядра сприяли його початковій розробці [29].

Git має віддалене сховище, яке зберігається на сервері, та локальне сховище, яке зберігається на комп’ютері кожного розробника [30]. Це означає, що код не просто зберігається на центральному сервері, а ще й присутній у кожного розробника.

Git дозволяє отримати повний доступ до будь-якого файлу, гілки та ітерації проекту, що збережений на сервері. Це дозволяє користувачеві отримати доступ до повної історії всіх змін проекту.

GitHub – це один з найбільших веб-сервісів для спільної розробки програмного забезпечення. Він дозволяє спільно працювати над проектами з будь-якої точки світу та зберігати версії проекту на віддаленому сервері.

GitHub був придбаний Microsoft 4 червня 2018 року за 7.5 млрд. доларів [31]. Перехід до Microsoft дав свої переваги, наприклад, у 2019 році можливість створювати приватні репозиторії стала безкоштовною.

2.7.3. Виклик та завантаження програми

Створений Gradle файл .APK потрібно відкрити на пристрої, що відповідає технічним параметрам мобільного додатку, після чого встановити додаток. Після встановлення достатньо просто запустити додаток, клацнувши по іконці додатку (рис. 2.6).

Рис. 2.6. Іконка та назва встановленого додатку

2.7.4. Опис інтерфейсу користувача

На початку роботи з мобільним додатком користувачу відображається завантажувальний екран (рис. 2.7). Далі відображається головна сторінка (рис. 2.8 та рис. 2.9), на цій сторінці будуть відображатися товари різних категорій, саме відображення товарів залежить від взаємодії користувача зі своїм профілем, а також поле для пошуку. Екрани для реєстрації та авторизації зображені на рис. 2.10 та рис. 2.11. Користувач також має можливість переглядати категорії (рис. 2.12 та рис. 2.13), сортувати та фільтрувати товари на екрані обраної категорії (рис. 2.134 та рис. 2.15), переглядати свій профіль (рис. 2.16 та рис. 2.17), а також переглядати інформацію про товари (рис. 2.18).

Інтерфейс мобільного додатку розроблявся з властивістю адаптуватися до різних розмірів екранів смартфонів та планшетів.

Рис. 2.7. Завантажувальний екран додатку

Рис. 2.8. Головний екран додатку (вигляд на смартфоні)

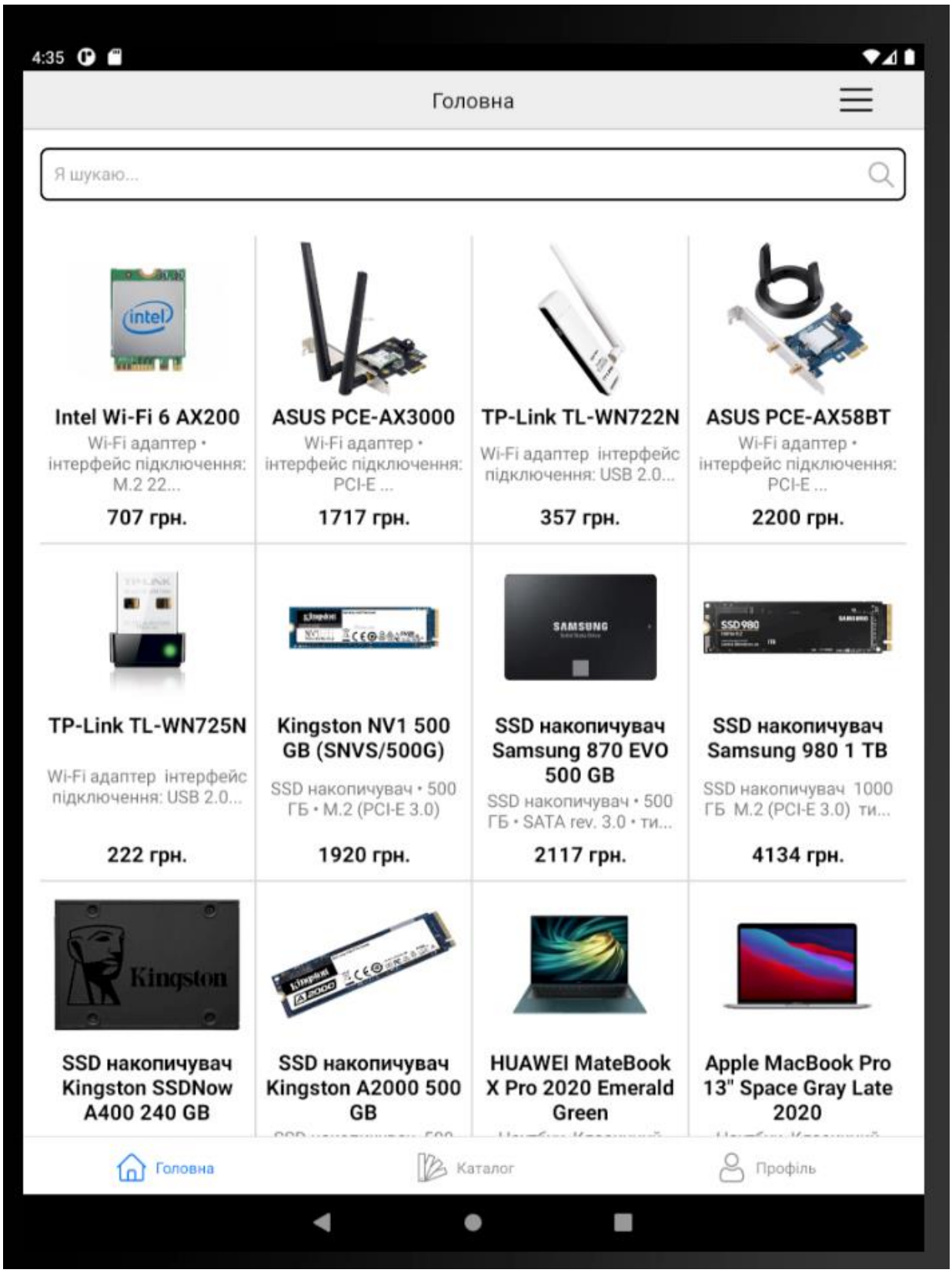

Рис. 2.9. Головний екран додатку (вигляд на планшеті)

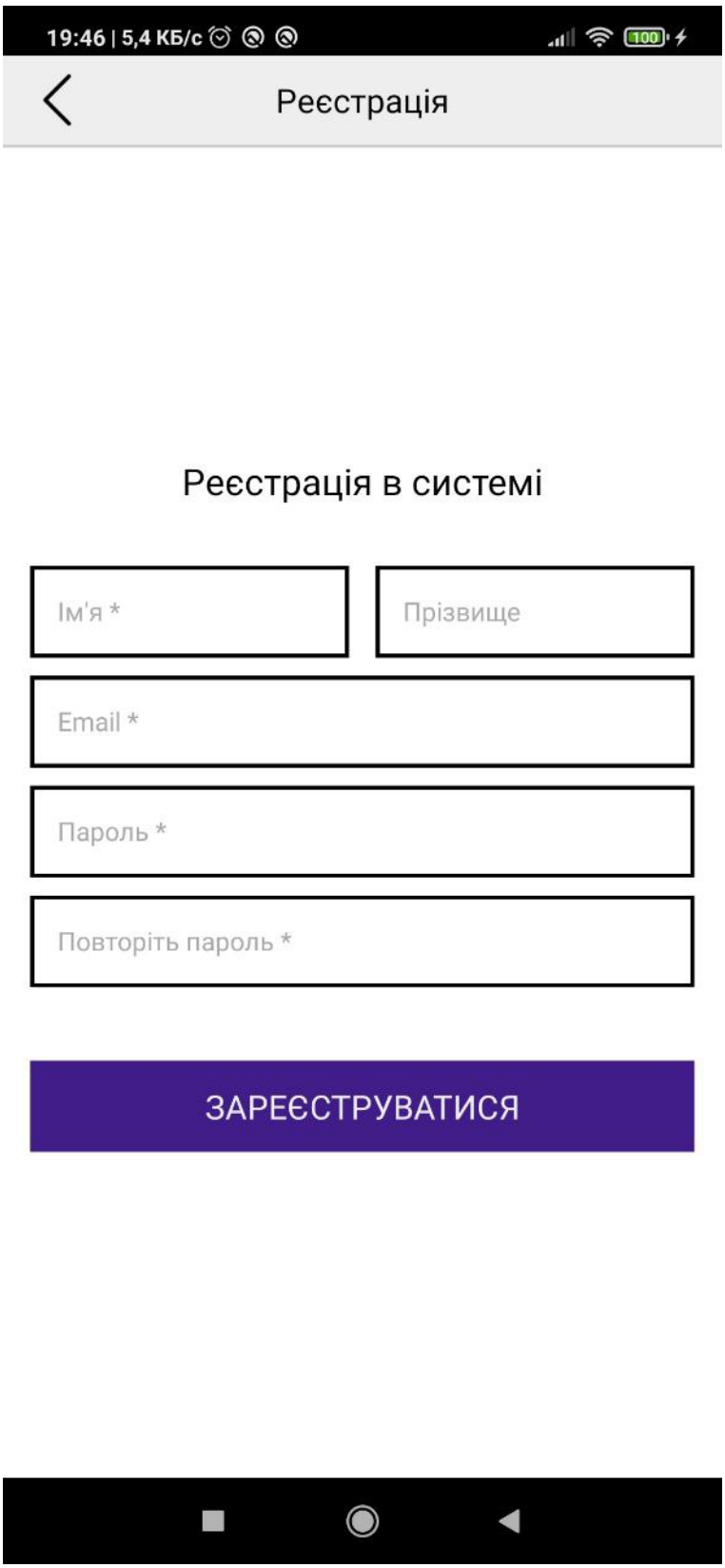

Рис. 2.10. Екран реєстрації користувача

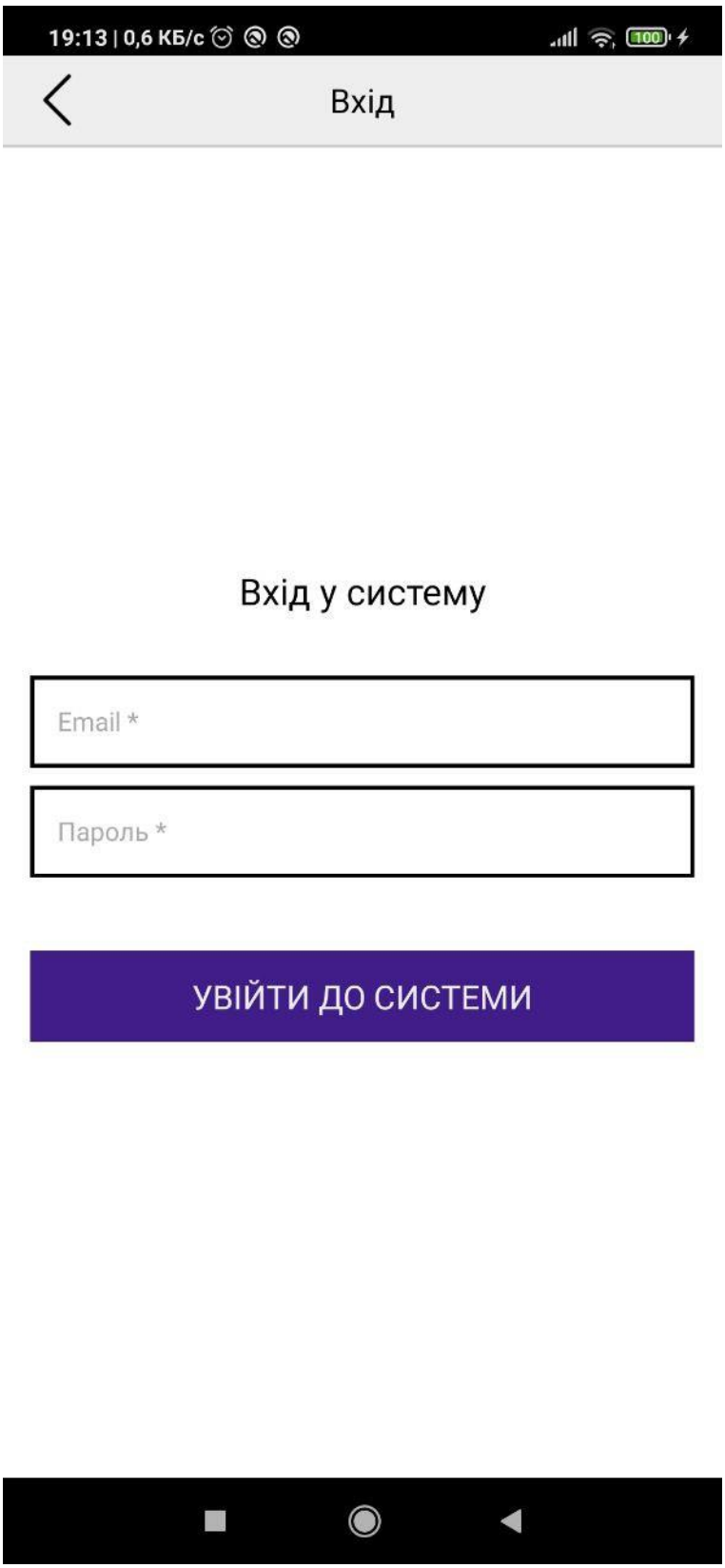

Рис. 2.11. Екран авторизації користувача

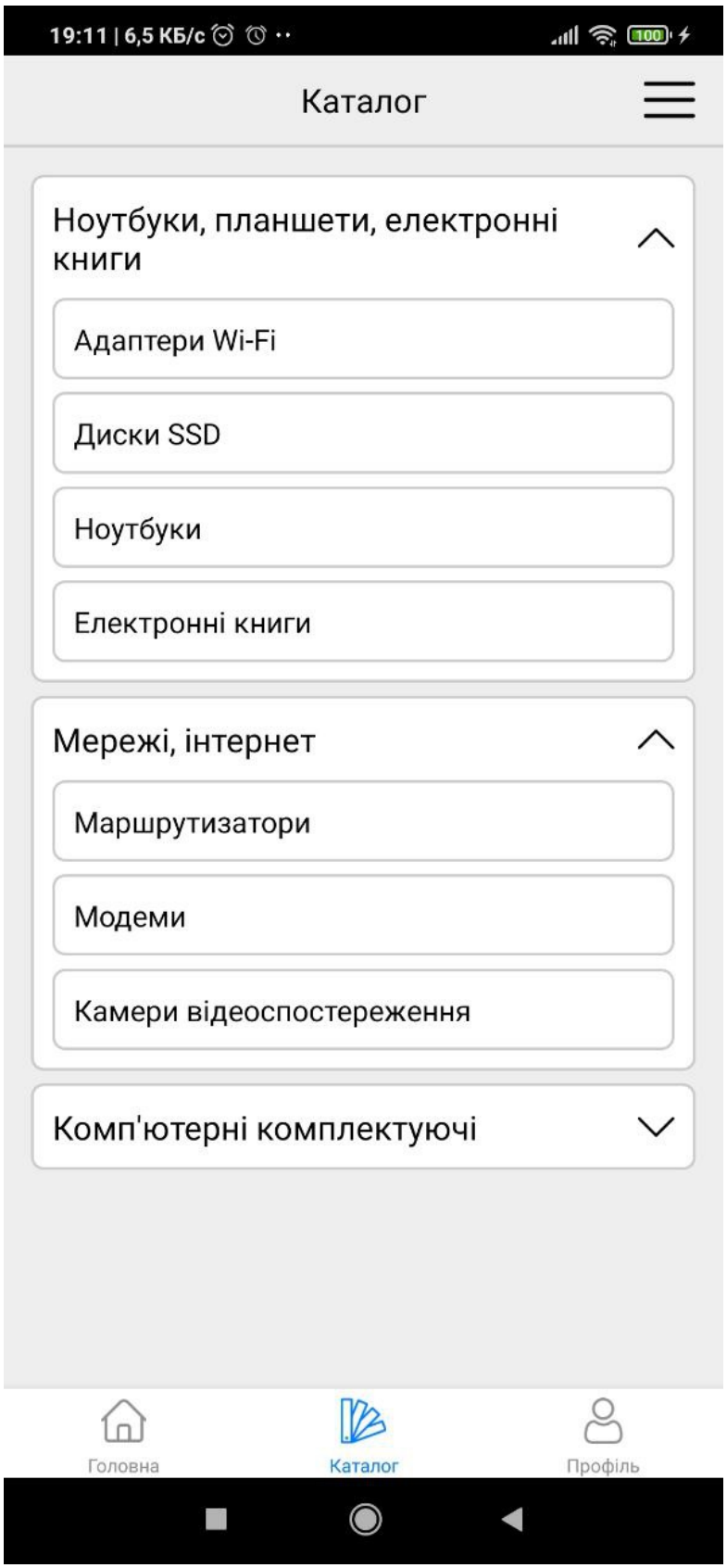

Рис. 2.12. Екран каталогу (вигляд на смартфоні)

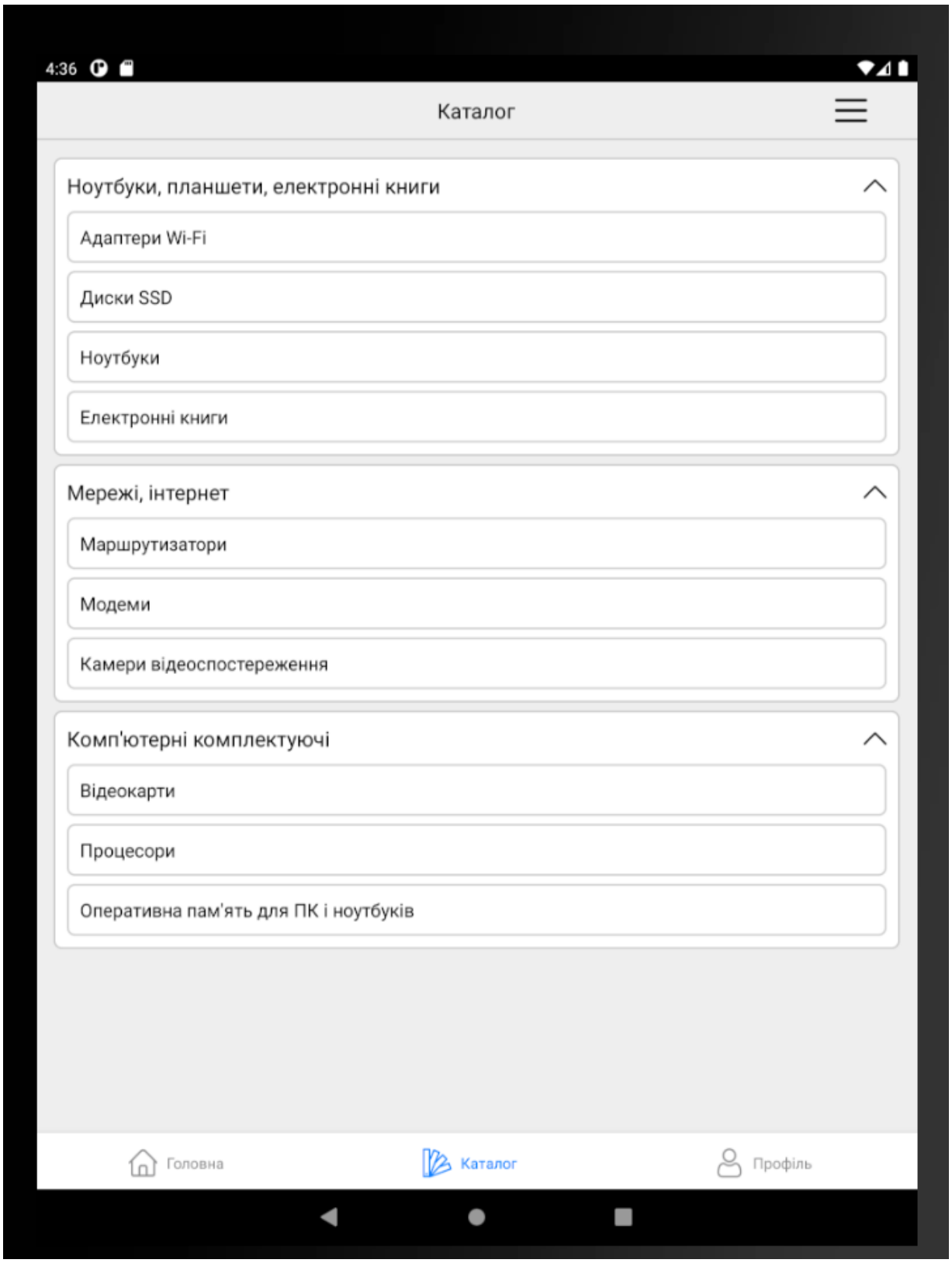

Рис. 2.13. Екран каталогу (вигляд на планшеті)

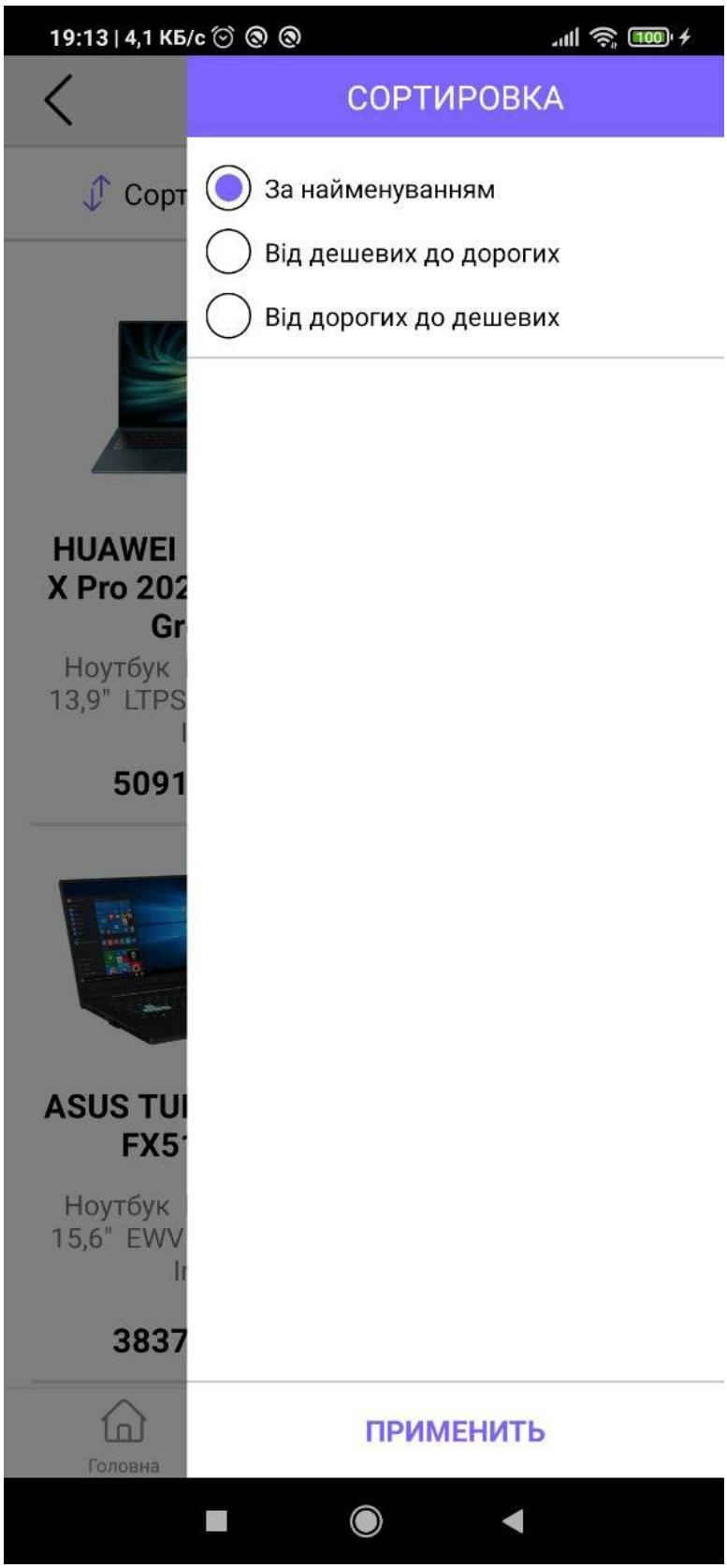

Рис. 2.14. Меню з опціями сортування

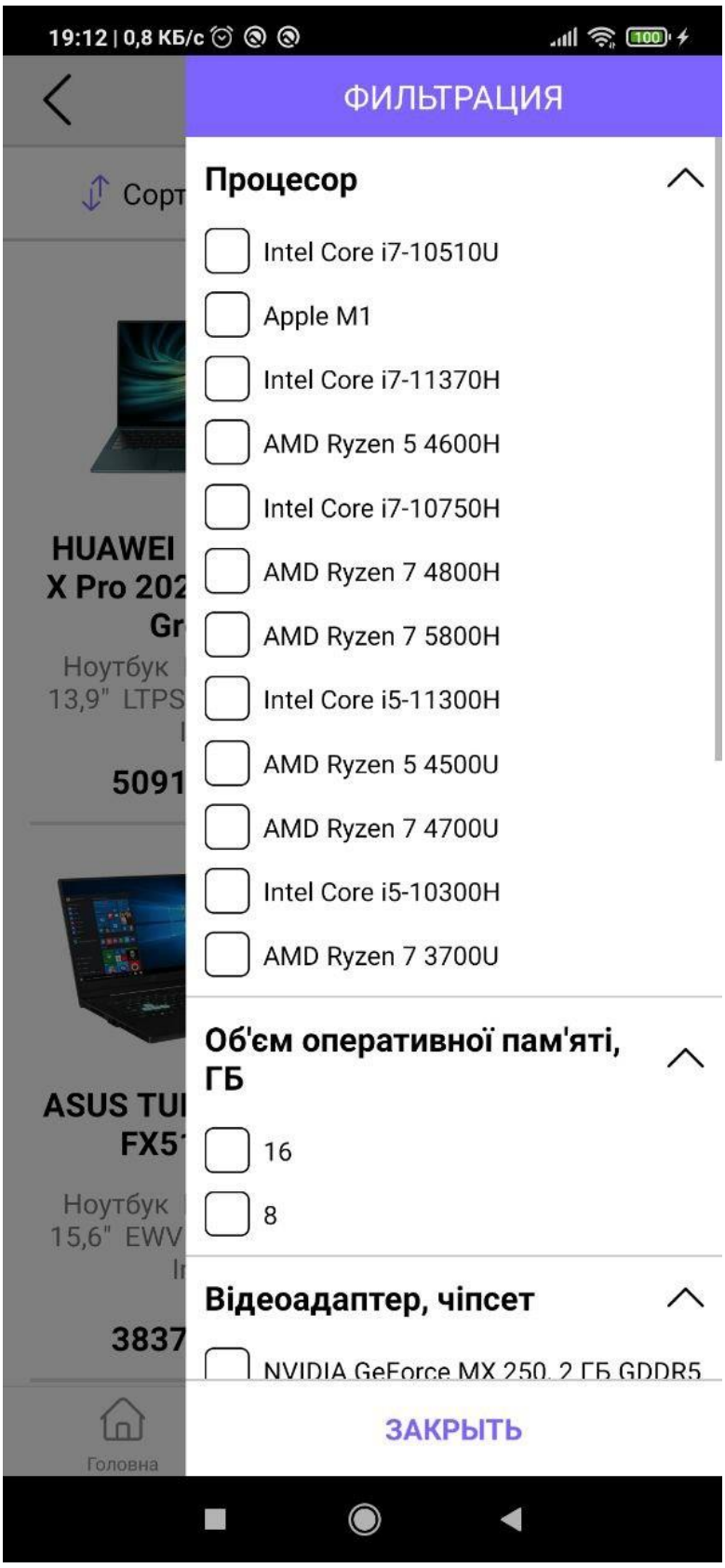

Рис. 2.15. Меню з опціями фільтрації

Рис. 2.16. Екран профілю користувача (вигляд на смартфоні)

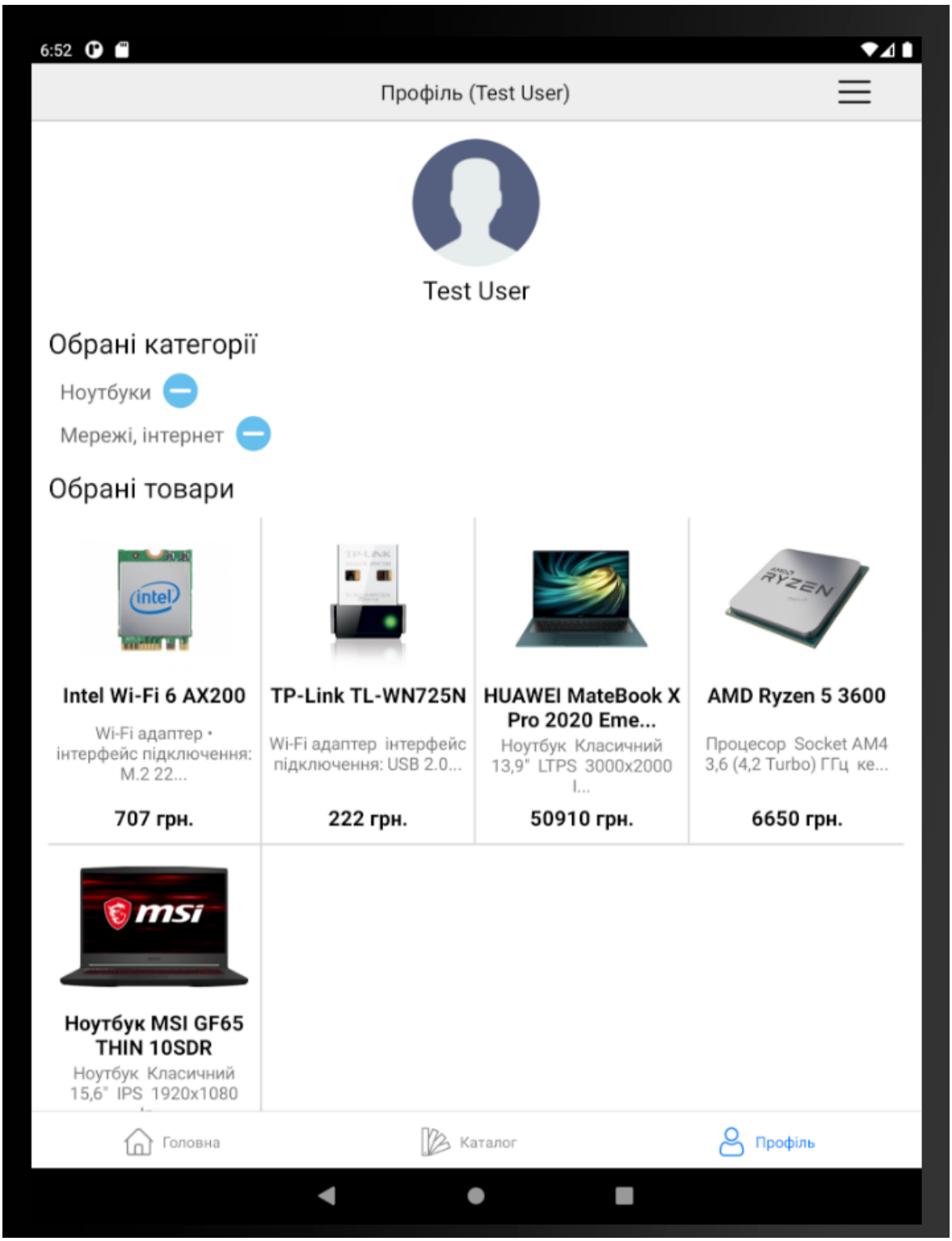

Рис. 2.17. Екран профілю користувача (вигляд на планшеті)

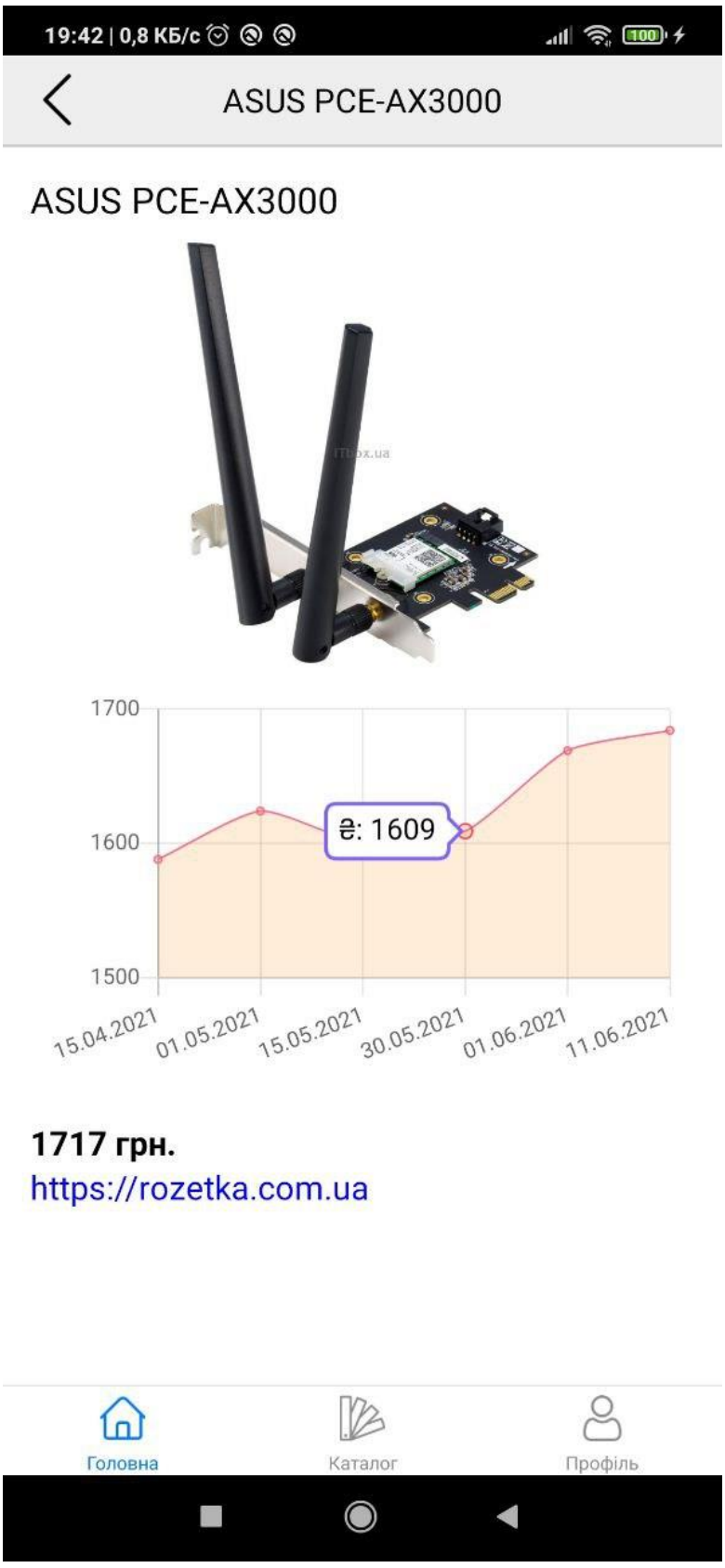

Рис. 2.18. Екран товару

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Розрахунок трудомісткості та вартості розробки програмного продукту

Початкові дані:

1. передбачуване число операторів програми – 1300;

2. коефіцієнт складності програми – 1,4;

3. коефіцієнт корекції програми в ході її розробки – 0,06;

4. годинна заробітна плата програміста – 140 грн/год;

5. коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі – 1,2;

6. коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 1,1;

7. вартість машино-години ЕОМ – 15 грн/год.

Нормування праці в процесі створення ПЗ істотно ускладнено в силу творчого характеру праці програміста. Тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_o + t_u + t_a + t_n + t_{отл} + t_д, \text{ людино-годин}, \qquad (3.1)$$

де t_o – витрати праці на підготовку й опис поставленої задачі (приймається 50 людино-годин);

t_u – витрати праці на дослідження алгоритму рішення задачі;

t_a – витрати праці на розробку блок-схеми алгоритму;

t_n – витрати праці на програмування по готовій блок-схемі;

$t_{отл}$ – витрати праці на налагодження програми на ЕОМ;

$t_д$ – витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у ПЗ, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q \cdot C \cdot (1 + p), \text{ де} \qquad (3.2)$$

q – передбачуване число операторів;

C – коефіцієнт складності програми;

p – коефіцієнт кореляції програми в ході її розробки.

$$Q = 1300 \cdot 1{,}4 \cdot (1 + 0{,}06) = 1929{,}2;$$

Витрати праці на вивчення опису задачі t_u визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q \cdot B}{(75 \ldots 85) \cdot K}, \text{ людино-годин,} \qquad (3.3)$$

де B – коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі;

K – коефіцієнт кваліфікації програміста, обумовлений стажем роботи з даної спеціальності;

$$t_u = \frac{1929.2 \cdot 1{,}2}{85 \cdot 1{,}1} = 24{,}76, \text{ людино-годин.}$$

Витрати праці на розробку алгоритму рішення задачі:

$$t_a = \frac{Q}{(20...25) \cdot K}; \quad (3.4)$$

$$t_a = \frac{1929,2}{20 \cdot 1,1} = 87,69, \text{ людино-годин.}$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20...25) \cdot K}; \quad (3.5)$$

$$t_n = \frac{1929,2}{25 \cdot 1,1} = 70,15, \text{ людино-годин.}$$

Витрати праці на налагодження програми на ЕОМ:

– за умови автономного налагодження одного завдання:

$$t_{отл} = \frac{Q}{(4...5) \cdot K}; \quad (3.6)$$

$$t_{отл} = \frac{1929,2}{5 \cdot 1,1} = 350,76, \text{ людино-годин,}$$

– за умови комплексного налагодження завдання:

$$t_{отл}^{к} = 1,2 \cdot t_{отл}; \quad (3.7)$$

$$t_{отл}^{к} = 1,2 \cdot 350,76 = 420,91, \text{ людино-годин}$$

Витрати праці на підготовку документації:

$$t_д = t_{дp} + t_{до}; \quad (3.8)$$

де $t_{др}$ – трудомісткість підготовки матеріалів і рукопису

$$t_{др} = \frac{Q}{(15...20)\cdot K}; \qquad (3.9)$$

$$t_{др} = \frac{1929,2}{20 \cdot 1,1} = 87,69, \text{ людино-годин.}$$

$t_{до}$– трудомісткість редагування, печатки й оформлення документації

$$t_{до} = 0,75 \cdot t_{др}; \qquad (3.10)$$

$$t_{до} = 0,75 \cdot 87,69 = 65,77, \text{ людино-годин.}$$

$$t_{д} = 87,69 + 65,77 = 153,46, \text{ людино-годин.}$$

Отримаємо трудомісткість розробки програмного забезпечення:

$$t = 50 + 24,76 + 87,69 + 70,15 + 350,76 + 153,46 = 736,82, \text{ людино-годин.}$$

У результаті було розраховано, що в загальній складності необхідно 736,82 людино-годин для розробки даного програмного забезпечення.

3.2. Розрахунок витрат на створення програми

Витрати на створення ПЗ Кпо включають витрати на заробітну плату виконавця програми Зз/п і витрат машинного часу, необхідного на налагодження програми на ЕОМ.

$$К_{ПО} = З_{ЗП} + З_{МВ}, \text{ грн,} \qquad (3.11)$$

де $З_{ЗП}$ – заробітна плата виконавців, яка визначається за формулою:

$$З_{ЗП} = t \cdot С_{ПР}, \text{ грн,} \qquad (3.12)$$

де t – загальна трудомісткість, людино-годин;

$С_{ПР}$ – середня годинна заробітна плата програміста, грн/година

$$З_{ЗП} = 736{,}82 \cdot 140 = 103154{,}8, \text{ грн.}$$

$З_{МВ}$ – Вартість машинного часу, необхідного для налагодження програми на ЕОМ:

$$З_{МВ} = t_{отл} \cdot С_{М}, \text{ грн,} \qquad (3.13)$$

де $t_{отл}$ – трудомісткість налагодження програми на ЕОМ, год.

$С_{МЧ}$ – вартість машино-години ЕОМ, грн/год.

$$З_{МВ} = 350{,}76 \cdot 15 = 5261{,}4 \text{ грн.}$$

$$К_{ПО} = 103154{,}8 + 5261{,}4 = 108416{,}2 \text{ грн.}$$

Очікуваний період створення ПЗ:

$$T = \frac{t}{B_k \cdot F_p}, \text{ мес.} \qquad (3.14)$$

де B_k- число виконавців;

F_p – місячний фонд робочого часу (при 40 годинному робочому тижні Fp=176 годин).

$$T = \frac{736,82}{1 \cdot 176} = 4,19 \text{ міс.}$$

Висновки. На розробку даного програмного забезпечення піде 736,82 людино-годин. Тобто, ймовірна очікувана тривалість розробки складатиме 4,19 місяці при стандартному 40-годинному робочому тижні та 176-годинному робочому місяці. Очікувані витрати на створення програмного забезпечення складатимуть 108416,2 грн.

ВИСНОВКИ

В даній кваліфікаційній роботі було спроектовано та розроблено мобільний додаток платформи Market Dynamics Analyzer.

Мобільний додаток надає два рівні доступу: гість та авторизований користувач, від чого залежить доступ до різного функціоналу платформи Market Dynamics Analyzer.

Цей мобільний додаток призначений для надання інформації про різні товари та їх ціни у різних продавців. Це дозволить користувачам легко робити свій вибір на користь більш вигідної пропозиції, що заощадить власні кошти.

Під час виконання даної кваліфікаційної роботи було виконано такі етапи:

– проаналізовано предметну галузь;

– порівняно можливості веб-сайтів та мобільних додатків з даної предметної галузі;

– обрано зручну архітектуру для розробки мобільного додатку;

– написано програмний код мобільного додатку.

Мобільний додаток працює на ОС Android, яка широко використовується в наш час. Розробка велась за допомогою фреймворку React Native на мові програмування JavaScript.

Також у кваліфікаційній роботі було визначено трудомісткість розробленого програмного продукту – 736,82 люд-год, проведений підрахунок вартості роботи по створенню програми – 108416,2 грн та розраховано час на його створення – 4,19 міс.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. The Influence of Price Comparison Websites on Online Switching Behavior: A Consumer Empowerment Perspective. DOI: https://doi.org/10.1007/978-3-030-44999-5_18.

2. Consistent aggregation with superlative and other price indexes. DOI: https://doi.org/10.1111/rssa.12633.

3. Online Price Search: Impact of Price Comparison Sites on Offline Price Evaluations. Journal of Retailing. DOI: https://doi.org/10.1016/j.jretai.2014.09.003.

4. Syntax Analysis: Compiler Top Down & Bottom Up Parsing Types. URL: https://www.guru99.com/syntax-analysis-parsing-types.html (дата звернення: 24.04.2021).

5. Что такое парсинг и как правильно парсить. URL: https://blog.calltouch.ru/chto-takoe-parsing/ (дата звернення: 24.04.2021).

6. Прайсагрегаторы Украины – эффективный способ увеличения продаж. URL: https://horoshop.ua/blog/praysagregatory-ukrainy/ (дата звернення: 25.04.2021).

7. 25+ Best price comparison websites and apps you need to try. URL: https://www.oberlo.com/blog/25-best-price-comparison-websites (дата звернення: 25.04.2021).

8. Özsu M.T. Client-Server Architecture. In: Liu L., Özsu M. Encyclopedia of Database Systems, 2016.

9. Santosh Kumar, A Review on Client-Server Based Applications and Research Opportunity, 2019.

10. Research Process on Software Development Model. URL: https://iopscience.iop.org/article/10.1088/1757-899X/394/3/032045/pdf (дата звернення: 08.05.2021).

11. Programming languages in chemistry: a review of HTML5/JavaScript. URL: https://jcheminf.biomedcentral.com/articles/10.1186/s13321-019-0331-1 (дата звернення: 08.05.2021).

12. About JavaScript. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript (дата звернення: 09.05.2021).

13. Крокфорд. Д. Как устроен JavaScript. Питер, 2018. 304 с.

14. Хавербеке М. Выразительный JavaScript. Современное веб-программирование. 3-е издание. Питер, 2019. 480 с.

15. Robin Wieruch. The Road to Learn React: Your Journey to Master Plain Yet Pragmatic React. Js. CreateSpace Independent Publishing Platform, 2018. 208 p.

16. Banks A., Porcello E. Learning React. Modern Patterns for Developing React Apps. O'Reilly Media, 2020. 310 p.

17. Ward D. React Native Cookbook: Recipes for solving common React Native development problems, 2nd Edition. Packt Publishing, 2019.

18. React Native Pros and Cons - Facebook's Framework in 2021. URL: https://www.netguru.com/blog/react-native-pros-and-cons (дата звернення: 09.05.2021).

19. React Navigation 5.0 – A new way to navigate. URL: https://reactnavigation.org/blog/2020/02/06/react-navigation-5.0/ (дата звернення: 09.05.2021).

20. A brief history of JSON. URL: https://blog.sqlizer.io/posts/json-history/ (дата звернення: 11.05.2021).

21. What is Use Case Diagram? URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/ (дата звернення: 11.05.2021).

22. Unified Modeling Language (UML) | Sequence Diagrams. URL: https://www.geeksforgeeks.org/unified-modeling-language-uml-sequence-diagrams/ (дата звернення: 11.05.2021).

23. Visual Studio Code docs. URL: https://code.visualstudio.com/docs (дата звернення: 12.05.2021).

24. Visual Studio Code vs. Visual Studio: How to choose. URL: https://dineroclub.net/visual-studio-code-vs-visual-studio-how-to-choose/ (дата звернення: 12.05.2021).

25. Meet Android Studio. URL: https://developer.android.com/studio/intro (дата звернення: 12.05.2021).

26. What is Gradle? URL: https://docs.gradle.org/current/userguide/-what_is_gradle.html (дата звернення: 12.05.2021).

27. Metro bundler in React Native. URL: https://www.codementor.io/-@rishabhsharma984/metro-bundler-in-react-native-wub92qcp5 (дата звернення: 12.05.2021).

28. All about Git and GitHub. URL: https://shiivangii.medium.com/all-about-git-and-github-c4b987df1b16 (дата звернення: 12.05.2021).

29. Getting Started – A Short History of Git. URL: https://git-scm.com/book/en/v2/Getting-Started-A-Short-History-of-Git (дата звернення: 12.05.2021).

30. Tsitoara M. Beginning Git and GitHub: A Comprehensive Guide to Version Control, Project Management, and Teamwork for the New Developer. Apress, 2019. 308 p.

31. A bright future for GitHub. URL: https://github.blog/2018-06-04-github-microsoft/ (дата звернення: 12.05.2021).

ДОДАТОК А

КОД ПРОГРАМИ

App.js

```
import React from 'react';
import { StatusBar } from 'react-native';
import { enableScreens } from 'react-native-screens';
import { NavigationContainer } from '@react-navigation/native';
import RNBootSplash from 'react-native-bootsplash';

import AppNavigator from './client/navigators/AppNavigator';

export default function App() {
   enableScreens(); // Screen memory optimization in React Navigation
   window.navigator.userAgent = 'ReactNative';

   return (
      <NavigationContainer onReady={() => RNBootSplash.hide()}>
         <StatusBar barStyle='light-content' backgroundColor='#000'/>
         <AppNavigator/>
      </NavigationContainer>
   );
};
```

Навігатори

SidebarNavigator.js

```
import React from 'react';
import { createDrawerNavigator } from '@react-navigation/drawer';

import TabNavigator from './TabNavigator';
import AuthStack from './AuthStack';

const SidebarDrawer = createDrawerNavigator();

export default function SidebarNavigator() {
   return (
      <SidebarDrawer.Navigator>
         <SidebarDrawer.Screen
            name="Home"
            component={TabNavigator}
            options={{title: 'Главная'}}
         />
         <SidebarDrawer.Screen
            name="Login"
            component={AuthStack}
```

```
            initialParams={{screen: 'Login'}}
            options={{title: 'Вход'}}
          />
          <SidebarDrawer.Screen
            name="Registration"
            component={AuthStack}
            initialParams={{screen: 'Registration'}}
            options={{title: 'Регистрация'}}
          />
        </SidebarDrawer.Navigator>
    )
}
```

AppNavigator.js

```
import React from 'react';
import SidebarNavigator from './SidebarNavigator';

export default function AppNavigator() {
    return (
        <SidebarNavigator/>
    );
}
```

TabNavigator.js

```
import React from 'react';
import { Image } from 'react-native';
import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';

import HomeStack from './HomeStack';
import CatalogStack from './CatalogStack';
import ProfileStack from './ProfileStack';

const Tab = createBottomTabNavigator();

export default function TabNavigator() {
    return (
        <Tab.Navigator>
            <Tab.Screen
                name='Home'
                component={HomeStack}
                options={{
                    tabBarLabel: 'Головна',
                    tabBarIcon: ({ color, size }) => (
                        <Image
                            style={{width:size, height:size, tintColor: color}}
                            source={require('../img/home-icon.png')}
                        />
                    )
                }}
            />
            <Tab.Screen
```

```
            name='Catalog'
            component={CatalogStack}
            options={{
              tabBarLabel: 'Каталог',
              tabBarIcon: ({ color, size }) => (
                <Image
                  style={{width:size, height:size, tintColor: color}}
                  source={require('../img/catalog-icon.png')}
                />
              )
            }}
          />
          <Tab.Screen
            name='Profile'
            component={ProfileStack}
            options={{
              tabBarLabel: 'Профіль',
              tabBarIcon: ({ color, size }) => (
                <Image
                  style={{width:size, height:size, tintColor: color}}
                  source={require('../img/user-icon.png')}
                />
              )
            }}
          />
        </Tab.Navigator>
    )
}
```

HomeStack.js

```
import React from 'react';
import { createStackNavigator, CardStyleInterpolators } from '@react-navigation/stack';

import HomeScreen from '../screens/HomeScreen';
import ProductScreen from '../screens/ProductScreen';

const StackHome = createStackNavigator();
const navOptions = () => ({
              headerShown: false,
              cardStyleInterpolator: CardStyleInterpolators.forHorizontalIOS
});

export default function HomeStack() {
   return (
      <StackHome.Navigator>
        <StackHome.Screen name="Home" component={HomeScreen} options={navOptions}/>
        <StackHome.Screen name="Product" component={ProductScreen} options={navOptions}/>
      </StackHome.Navigator>
   )
}
```

Екрани

HomeScreen.js

```
import React, { useState, useEffect } from 'react';
import { View, ScrollView } from 'react-native';
import { Wrapper, Preloader, CustomHeader } from '../components/Custom';
import { Header, Content } from '../components/Home';
import { URL, ITEMS_PER_PAGE } from '../constants';

export default function HomeScreen({ navigation }) {
  const [products, setProducts] = useState([]);
  const [totalItems, setTotalItems] = useState(null);
  const [page, setPage] = useState(1);
  const [searchValue, setSearchValue] = useState('');
  const [loading, setLoading] = useState(true);

  const getProducts = async () => {
    setLoading(true);

    const res = await (fetch(`${URL}/products?limit=${ITEMS_PER_PAGE}&page=${page}
        ${searchValue ? `&search=${searchValue}` : '}`)
      .then(data => data.json())
    );

    setProducts([...products, ...res['products']]);
    setTotalItems(res['totalItems']);
    setLoading(false);
  }

  const onSearchHandler = (search, callback) => {
    if (search.trim().length && searchValue !== search) {
      setPage(1);
      setSearchValue(search);
      callback(searchValue);
    }
  }

  useEffect(() => {
    getProducts();
  }, [page, searchValue]);

  return (
    <View style={{ flex: 1 }}>
      <CustomHeader
        title={'Головна'}
        isMain={true}
        navigation={navigation}
      />
```

```
        <ScrollView keyboardShouldPersistTaps='handled'>
          <Wrapper>
            <Header
              onSearch={onSearchHandler}
              navigation={navigation}
            />

            <Content
              products={products}
              totalItems={totalItems}
              page={page}
              setPage={setPage}
              loading={loading}
              navigation={navigation}
            />
          </Wrapper>
        </ScrollView>
      </View>
    );
}
```

CatalogScreen.js

```
import React, { useState, useEffect } from 'react';
import { View, ScrollView } from 'react-native';
import { Wrapper, Preloader, CustomHeader } from '../components/Custom';
import { Content } from '../components/Catalog/';
import { URL } from '../constants';

export default function CatalogScreen({ navigation }) {
  const [categories, setCategories] = useState([]);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    async function fetchData() {
      setLoading(true);

      const categories = await (fetch(`${URL}/categories`)
        .then(data => data.json())
      );

      setCategories(categories);
      setLoading(false);
    }

    fetchData();
  }, []);
```

```
    return (
        <View style={{ flex: 1 }}>
            <CustomHeader
                title={'Каталог'}
                isMain={true}
                navigation={navigation}
            />

            <ScrollView
                style={{ backgroundColor: '#eee' }}
                keyboardShouldPersistTaps='handled'
            >
                <Wrapper backgroundColor={'#eee'}>
                    { !loading ? <Content
                        categories={categories}
                        navigation={navigation}
                    /> : <Preloader/> }
                </Wrapper>
            </ScrollView>
        </View>
    );
}
```

CategoryScreen.js

```
import React, { useState, useEffect } from 'react';
import { View, ScrollView } from 'react-native';
import { Wrapper, Preloader, CustomHeader } from '../components/Custom';
import { Header, Content, CategorySettingsSidebar } from '../components/Category';
import { getCategory } from '../logic/categories';
import categoryOptions from '../logic/categoryOptions';
import { URL, ITEMS_PER_PAGE } from '../constants';

export default function CategoryScreen({ route, navigation }) {
    const { id } = route.params;
    const [categoryInfo, setCategoryInfo] = useState(null);
    const [products, setProducts] = useState([]);
    const [options, setOptions] = useState(null);
    const [filtersActive, setFiltersActive] = useState(false);
    const [page, setPage] = useState(1);
    const [loading, setLoading] = useState(true);
    const [toggleModal, setToggleModal] = useState(null); // null || 'sort' || 'filter'

    const onChangeOption = (type, code, value) => {
        const newOptions = {...options};
        const option = categoryOptions.getOption(newOptions[type], code, value);

        if (type == 'filters') {
            option['active'] = !option['active'];
```

```
        setFiltersActive(categoryOptions.isOptionsActive(newOptions[type]));
    } else {
        categoryOptions.resetOptions(newOptions[type]);
        option['active'] = true;
    }

    setOptions(newOptions);
}

const onResetFilters = () => {
    const newOptions = {...options};
    categoryOptions.resetOptions(newOptions['filters']);

    setOptions(newOptions);
    setFiltersActive(false);
}

const onApplyOptions = () => {
    setToggleModal(null);
    console.log('apply');
}

useEffect(() => {
    async function fetchData() {
        setLoading(true);

        const categories = await (fetch(`${URL}/categories/${id}?limit=${ITEMS_PER_PAGE}&page=${page}`)
            .then(data => data.json())
        );

        setCategoryInfo({
            'name': categories['name'],
            'totalItems': categories['totalItems']
        });
        setProducts([...products, ...categories['products']]);
        setOptions({
            'filters': categories['filterOptions'],
            'sorts': categories['sortOptions']
        });

        setLoading(false);
    }

    fetchData();
}, [id, page]);

return (
    <View style={{ flex: 1 }}>
        <CustomHeader
            title={ categoryInfo?.['name'] || 'Loading...'}
            navigation={navigation}
        />
```

```
        <Header
          options={options}
          setToggleModal={setToggleModal}
          navigation={navigation}
        />

        <ScrollView keyboardShouldPersistTaps='handled'>
          <Wrapper>
            { categoryInfo ? <Content
              products={products}
              totalItems={categoryInfo['totalItems']}
              page={page}
              setPage={setPage}
              loading={loading}
              navigation={navigation}
            /> : <Preloader/> }
          </Wrapper>
        </ScrollView>

        { toggleModal && <CategorySettingsSidebar
          type={toggleModal}
          data={ toggleModal == 'filter'
            ? options['filters']
            : options['sorts']
          }
          filtersActive={filtersActive}
          onApplyOptions={onApplyOptions}
          onChangeOption={onChangeOption}
          onResetFilters={onResetFilters}
          setToggleModal={setToggleModal}
        /> }
      </View>
    );
}
```

ProductScreen.js

```
import React, { useState, useEffect } from 'react';
import { View, ScrollView, StyleSheet } from 'react-native';
import { Header, Content } from '../components/Product';
import { Wrapper, CustomHeader } from '../components/Custom';
import { URL } from '../constants';

export default function ProductScreen({ route, navigation }) {
  const { id } = route.params;
  const [product, setProduct] = useState(null);
  const [loading, setLoading] = useState(true);
```

```
  useEffect(() => {
    async function fetchData() {
      setLoading(true);

      const res = await fetch(`${URL}/products/${id}`)
        .then(data => data.json()
      );

      setProduct(res);
      setLoading(false);
    }

    fetchData();
  }, [id]);

  return (
    <View style={styles.container}>
      <CustomHeader
        title={!loading ? product['name'] : 'Завантаження...'}
        navigation={navigation}
      />

      <ScrollView keyboardShouldPersistTaps='handled'>
        <Wrapper>
          { !loading && (
            <View>
              <Header item={product}/>
              <Content item={product}/>
            </View>
          ) }
        </Wrapper>
      </ScrollView>
    </View>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#fff'
  }
});
```

ProfileScreen.js

```
import React, { useState, useEffect } from 'react';
import { View, ScrollView } from 'react-native';
import { Header, Content } from '../components/Profile';
import { Wrapper, CustomHeader } from '../components/Custom';
```

```
import { URL } from '../constants';

export default function ProfileScreen({ navigation }) {
  const [user, setUser] = useState(null);
  const [loading, setLoading] = useState(true);

  const removeCategory = (id) => {
    setCategories(categories.filter(item => item['id'] !== id));
  };

  useEffect(() => {
    async function fetchData() {
      setLoading(true);
      const userInfo = await fetch(`${URL}/users/1`).then(data => data.json());
      setUser(userInfo);
      setLoading(false);
    }

    fetchData();
  }, []);

  return (
    <View style={{ flex: 1, backgroundColor: '#fff' }}>
      <CustomHeader
        title={ !loading ? `Профіль (${user['firstName']} ${user['lastName']})` : 'Завантаження...'}
        isMain={true}
        navigation={navigation}
      />

      <ScrollView keyboardShouldPersistTaps='handled'>
        <Wrapper>
          { !loading && (
            <View>
              <Header
                user={user}
                navigation={navigation}
              />

              <Content
                user={user}
                removeCategory={removeCategory}
                navigation={navigation}
              />
            </View>
            )
          }
        </Wrapper>
      </ScrollView>
    </View>
  );
}
```

RegistrationScreen.js

```
import React, { useState } from 'react';
import { View, Text, StyleSheet } from 'react-native';
import { Wrapper, CustomHeader, CustomInput, CustomButton } from '../../components/Custom';
import { AuthContext } from '../../context/AuthContext';
import { validateRegistrationForm } from '../../logic/validate';
import { URL } from '../../constants';

export default function RegistrationScreen({ navigation }) {
    const [firstName, setFirstName] = useState('');
    const [lastName, setLastName] = useState('');
    const [email, setEmail] = useState('');
    const [password, setPassword] = useState('');
    const [repeatPassword, setRepeatPassword] = useState('');
    const [loading, setLoading] = useState(false);
    const { login } = useContext(AuthContext);

    const onRegisterHandler = () => {
        if (!loading) {
            setLoading(true);

            if (validateRegistrationForm({ firstName, lastName, email, password, repeatPassword })) {
                const data = {
                    'firstName': firstName.trim(),
                    'lastName': lastName.trim(),
                    'email': email,
                    'password': password,
                };

                const res = await fetch(`${URL}/register`, {
                    method: 'POST',
                    body: JSON.stringify(data),
                    headers: {
                        'Content-Type': 'application/json',
                    }
                }),
                result = await res.json();

                if (res.status === 200) {
                    showAlertOneBtn('Внимание!', result['message']);
                } else if (res.status === 201) {
                    login(result['token'], result['memberId']);
                }
            }
        }
    };

    return (
        <View style={{ flex: 1 }}>
```

```
<CustomHeader
   title={'Реєстрація'}
   navigation={navigation}
/>

<Wrapper isAuth={true}>
   <View style={styles.rowContainer}>
      <Text style={styles.titleText}>Реєстрація в системі</Text>
   </View>

   <View style={styles.fullnameContainer}>
      <View style={[styles.rowContainer, {flex: .48}]}>
         <CustomInput
            value={firstName}
            setValue={setFirstName}
            borderWidth={2}
            borderColor={'#000'}
            activeBorderColor={'#7e64ff'}
            placeholder={'Ім\'я *'}
         />
      </View>

      <View style={[styles.rowContainer, {flex: .48}]}>
         <CustomInput
            value={lastName}
            setValue={setLastName}
            borderWidth={2}
            borderColor={'#000'}
            activeBorderColor={'#7e64ff'}
            placeholder={'Прізвище'}
         />
      </View>
   </View>

   <View style={styles.rowContainer}>
      <CustomInput
         value={email}
         setValue={setEmail}
         borderWidth={2}
         borderColor={'#000'}
         activeBorderColor={'#7e64ff'}
         placeholder={'Email *'}
         keyboardType={'email-address'}
      />
   </View>

   <View style={styles.rowContainer}>
      <CustomInput
         value={password}
         setValue={setPassword}
         secureTextEntry={true}
         borderWidth={2}
```

```
                borderColor={'#000'}
                activeBorderColor={'#7e64ff'}
                placeholder={'Пароль *'}
              />
            </View>

            <View style={styles.rowContainer}>
              <CustomInput
                value={repeatPassword}
                setValue={setRepeatPassword}
                secureTextEntry={true}
                borderWidth={2}
                borderColor={'#000'}
                activeBorderColor={'#7e64ff'}
                placeholder={'Повторіть пароль *'}
              />
            </View>

            <View style={[styles.rowContainer, styles.buttonContainer]}>
              <CustomButton
                onPress={onRegisterHandler}
                backgroundColor={'#401d88'}
                textColor={'#eee'}
              >{!loading ? 'Зареєструватися' : 'Завантаження...'}</CustomButton>
            </View>
          </Wrapper>
        </View>
    );
}

const styles = StyleSheet.create({
    titleText: {
        fontSize: 20,
        textAlign: 'center'
    },
    fullnameContainer: {
        flexDirection: 'row',
        justifyContent: 'space-between'
    },
    rowContainer: {
        height: 50,
        marginBottom: 10
    },
    buttonContainer: {
        marginTop: 30
    }
});
```

Компоненти

CheckBoxGroup.js

```
import React, { useState } from 'react';
import { View, Text, Image, TouchableOpacity, StyleSheet } from 'react-native';
import CheckBox from './CheckBox';

export default function CheckBoxGroup({ values, onChangeOption }) {
   const [open, setOpen] = useState(true);

   const onChangeOptionHandler = (optionCode) => {
      onChangeOption('filters', values['code'], optionCode);
   }

   return (
      <View>
         { values['title'] && <TouchableOpacity
            style={[styles.groupHeader, open && styles.groupHeaderOpen]}
            activeOpacity={.7}
            onPress={() => setOpen(prev => !prev)}
         >
            <Text style={styles.groupTitleText}>{values['title']}</Text>
            <View style={styles.toggleBtn}>
               <Image
                  style={open ? [styles.toggleBtnImage, styles.toggleBtnRotateImage] : styles.toggleBtnImage}
                  source={require('../../../img/back-icon.png')}
               />
            </View>
         </TouchableOpacity> }

         { open && <View style={styles.groupContent}>
            { values['values'].length > 0 && values['values'].map((item, idx) => (
               <CheckBox
                  key={`checkBox-${item['code']}`}
                  item={item}
                  onChangeOption={onChangeOptionHandler}
                  isLast={idx == values['values'].length - 1}
               />
            )) }
         </View> }
      </View>
   );
}

const styles = StyleSheet.create({
   groupHeader: {
      padding: 10,
      flexDirection: 'row',
```

```
    justifyContent: 'space-between',
    alignItems: 'center',
    borderBottomWidth: 1,
    borderColor: '#ccc'
  },
  groupHeaderOpen: {
    paddingBottom: 0,
    borderBottomWidth: 0
  },
  groupTitleText: {
    flex: 1,
    fontSize: 18,
    fontWeight: '700'
  },
  groupContent: {
    padding: 10,
    borderBottomWidth: 1,
    borderColor: '#ccc'
  },
  toggleBtn: {
    width: 20,
    transform: [{ rotate: '-90deg' }]
  },
    toggleBtnImage: {
    width: '100%',
    height: 20,
  },
  toggleBtnRotateImage: {
    transform: [{ rotate: '180deg'}]
  },
});
```

CheckBox.js

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

export default function CheckBox({ item, onChangeOption, isLast }) {
  return (
    <TouchableOpacity
      style={[styles.container, isLast && { paddingBottom: 0 }]}
      onPress={() => onChangeOption(item['code'])}
      activeOpacity={.7}
    >
      <View style={styles.box}>
        <View style={item['active'] && styles.innerBox}></View>
      </View>

      <Text style={styles.titleText}>{item['name']}</Text>
```

```
      </TouchableOpacity>
    )
}

const styles = StyleSheet.create({
    container: {
      flexDirection: 'row',
      alignItems: 'center',
      paddingVertical: 5,
    },
    box: {
      width: 25,
      height: 25,
      borderWidth: 1,
      borderRadius: 5,
      marginRight: 7,
      justifyContent: 'center',
      alignItems: 'center'
    },
    innerBox: {
      width: '65%',
      height: '65%',
      backgroundColor: '#7e64ff',
      borderRadius: 3
    },
    titleText: {
      flex: 1,
      fontSize: 14
    }
});
```

RadioButton.js

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

export default function RadioButton({ item, onChangeOption, isLast }) {
    return (
      <TouchableOpacity
        style={[styles.container, isLast && { paddingBottom: 0 }]}
        onPress={() => onChangeOption(item['code'])}
        activeOpacity={.7}
      >
        <View style={styles.circle}>
          <View style={item['active'] && styles.innerCircle}></View>
        </View>

        <Text style={styles.titleText}>{item['name']}</Text>
      </TouchableOpacity>
```

```
    )
}

const styles = StyleSheet.create({
    container: {
        flexDirection: 'row',
        alignItems: 'center',
        paddingVertical: 5
    },
    circle: {
        width: 25,
        height: 25,
        borderWidth: 1,
        borderRadius: 50,
        marginRight: 7,
        justifyContent: 'center',
        alignItems: 'center'
    },
    innerCircle: {
        width: '65%',
        height: '65%',
        backgroundColor: '#7e64ff',
        borderRadius: 50
    },
    titleText: {
        flex: 1,
        fontSize: 14
    }
});
```

CustomHeader.js

```
import React from 'react';
import { View, Text, TouchableOpacity, Image, ToastAndroid, StyleSheet } from 'react-native';

export default function CustomHeader({ title, isMain = false, navigation }) {
    const onBackHandler = () => {
        if (navigation.goBack) {
            navigation.goBack();
        } else if (navigation.navigate) {
            navigation.navigate('Home');
        } else {
            ToastAndroid.show('Произошла ошибка при возврате назад', ToastAndroid.LONG);
        }
    }

    const onOpenDrawer = () => {
        if (navigation.openDrawer) {
            navigation.openDrawer();
```

```
      } else {
        ToastAndroid.show('Произошла ошибка при открытии меню', ToastAndroid.LONG);
      }
    }

    return (
      <View style={styles.container}>
        <View style={[styles.block, styles.leftBlock]}>
          { !isMain && <TouchableOpacity
            onPress={onBackHandler}
          >
            <Image
              style={styles.backImg}
              source={require('../../img/back-icon.png') }
            />
          </TouchableOpacity> }
        </View>

        <View style={[styles.block, styles.centerBlock]}>
          <Text style={styles.titleText}>{title}</Text>
        </View>

        <View style={[styles.block, styles.rightBlock]}>
          { isMain && <TouchableOpacity
            onPress={onOpenDrawer}
          >
            <Image
              style={styles.barImg}
              source={require('../../img/menu_bar-icon.png')}
            />
          </TouchableOpacity> }
        </View>
      </View>
    );
}

const styles = StyleSheet.create({
    container: {
      minHeight: 50,
      backgroundColor: '#eee',
      flexDirection: 'row',
      justifyContent: 'space-between',
      borderBottomWidth: 1.5,
      borderColor: '#ccc'
    },
    block: {
      justifyContent: 'center',
      alignItems: 'center',
    },
    leftBlock: {
      width: '15%',
    },
```

```
    centerBlock: {
      flex: 1,
    },
    rightBlock: {
      width: '15%',
    },
    titleText: {
      fontSize: 18,
      textAlign: 'center'
    },
    backImg: {
      width: 28,
      height: 28
    },
    barImg: {
      width: 28,
      height: 28
    }
});
```

CustomButton.js

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

export default function CustomButton({ ...props }) {
    const [isFocus, setIsFocus] = useState(false); // TouchableHighlight focus state

    const { children, onPress, backgroundColor, textColor, fontWeight, borderWidth,
      borderColor, borderRadius, highlight = false, highlightColor, highlightTextColor } = props;

    const btnStyles = {
      ...styles.btn,
      backgroundColor: backgroundColor,
      borderWidth: borderWidth || 0,
      borderColor: borderColor || '',
      borderRadius: borderRadius || 0
    };
    const btnTextStyles = {
      ...styles.btnText,
      color: textColor,
      fontWeight: fontWeight || '300'
    };

    return (
      !highlight ?
      <TouchableOpacity
        style={btnStyles}
        onPress={onPress}
```

```
        activeOpacity={.7}
      >
        <Text style={btnTextStyles}>{children}</Text>
      </TouchableOpacity>
      : <TouchableHighlight
        style={btnStyles}
        onPress={onPress}
        activeOpacity={1}
        underlayColor={highlightColor}
        onShowUnderlay={() => setIsFocus(true)}
        onHideUnderlay={() => setIsFocus(false)}
      >
        <Text style={[btnTextStyles, isFocus && {color: highlightTextColor}]}>{children}</Text>
      </TouchableHighlight>
    );
}

const styles = StyleSheet.create({
  btn: {
    flex: 1,
    borderRadius: 7,
    justifyContent: 'center'
  },
  btnText: {
    textAlign: 'center',
    fontSize: 18,
    textTransform: 'uppercase',
  }
})
```

CustomInput.js

```
import React, { useState, useRef } from 'react';
import { View, TextInput, TouchableOpacity, Image, StyleSheet, Animated } from 'react-native';

export default function CustomInput({ ...props }) {
  const [active, setActive] = useState(false);
  const input = useRef(null);
  const positionValue = useRef(new Animated.ValueXY({ x: 15, y: 15 })).current;

  const {
    value,
    setValue,
    borderWidth,
    borderColor,
    borderRadius,
    activeBorderColor,
    placeholder,
    secureTextEntry = false,
```

```
    keyboardType = 'default',
    searching = false,
    onSubmit = () => {},
    setInputFocus = () => {}
} = props;

const inputStyles = {
    ...styles.input,
    borderWidth: borderWidth || 0,
    borderRadius: borderRadius || 0,
    borderColor: borderColor || '#000',
    paddingRight: searching ? 55 : styles.input.paddingHorizontal
};
const activeInputStyles = {
    borderColor: activeBorderColor || '#000',
    borderWidth: 3
}

const activePlaceholder = () => {
    Animated.timing(positionValue, {
        toValue: { x: 15, y: -9 },
        duration: 130,
        useNativeDriver: false
    }).start();

    Animated.timing(positionValue, {
        toValue: { x: 10, y: -9 },
        duration: 130,
        useNativeDriver: false
    }).start();
}

const unactivePlaceholder = () => {
    Animated.timing(positionValue, {
        toValue: { x: 15, y: 15 },
        duration: 130,
        useNativeDriver: false
    }).start();
}

const onFocusHandler = () => {
    setActive(true);
    setInputFocus(true);
    input.current.focus();
    activePlaceholder();
}

const onBlurHandler = () => {
    setActive(false);
    setInputFocus(false);

    if (!value) {
```

```
            unactivePlaceholder();
        }
    }

    return (
        <View style={styles.container}>
            <TextInput
                ref={input}
                style={[inputStyles, active && activeInputStyles]}
                value={value}
                onChangeText={text => setValue(text)}
                placeholder={searching && placeholder || ''}
                secureTextEntry={secureTextEntry}
                keyboardType={keyboardType}
                onFocus={onFocusHandler}
                onBlur={onBlurHandler}
                returnKeyType={searching ? 'search' : 'default'}
                onSubmitEditing={onSubmit}
            />

            { searching &&
            <TouchableOpacity
                style={styles.iconContainer}
                activeOpacity={.7}
                onPress={onSubmit}
            >
                <Image
                    source={require('../../img/search-icon.png')}
                    style={styles.icon}
                />
            </TouchableOpacity> }
            { placeholder && !searching && (
                <Animated.Text
                    style={[
                        {...styles.inputPlaceholder, left: positionValue.x, top: positionValue.y},
                        (active || value) && {...styles.activeInputPlaceholder, left: positionValue.x, top: positionValue.y},
                        active && { color: activeBorderColor }
                    ]}
                    onPress={onFocusHandler}
                >
                    {placeholder}
                </Animated.Text>
            ) }
        </View>
    );
}

const styles = StyleSheet.create({
    container: {
        position: 'relative',
        flex: 1
    },
```

```
  input: {
    flex: 1,
    paddingHorizontal: 12,
    fontSize: 15
  },
  iconContainer: {
    position: 'absolute',
    right: 10,
    top: 12,
  },
  icon: {
    width: 22,
    height: 22,
    tintColor: '#aaa'
  },
  inputPlaceholder: {
    position: 'absolute',
    color: '#aaa'
  },
  activeInputPlaceholder: {
    backgroundColor: '#fff',
    paddingHorizontal: 3,
    fontSize: 14
  }
});
```

SearchForm.js

```
import React, { useState } from 'react';
import { View, Text, Image, TouchableOpacity, Keyboard, StyleSheet } from 'react-native';
import CustomInput from './CustomInput';
import useSearchHistory from '../../hooks/useSearchHistory';

export default function SearchForm({ onSearch }) {
  const [searchValue, setSearchValue] = useState('');
  const [inputFocus, setInputFocus] = useState(false);
  const { history, addToHistory, removeHistory } = useSearchHistory();

  const onCloseHandler = () => {
    Keyboard.dismiss(); // Disable input focus
  }
  const onClickHistory = (searchStr) => {
    setSearchValue(searchStr);
  }

  return (
    <View style={styles.container}>
      <View style={styles.searchContainer}>
        <CustomInput
```

```
      value={searchValue}
      setValue={setSearchValue}
      borderWidth={2}
      borderRadius={7}
      borderColor={'#000'}
      activeBorderColor={'#7e64ff'}
      placeholder={'Я шукаю...'}
      searching={true}
      onSubmit={() => onSearch(searchValue, addToHistory)}
      setInputFocus={setInputFocus}
    />
  </View>

  { (inputFocus && history.length > 0) &&
  <View style={styles.searchHistoryContainer}>
    <View style={styles.searchHistoryHeader}>
      <Text style={styles.searchHistoryTitle}>История поиска</Text>
      <TouchableOpacity
        activeOpacity={.5}
        onPress={removeHistory}
      >
        <Text style={styles.searchBtnText}>Очистить</Text>
      </TouchableOpacity>
    </View>

    <View>
      { history.map((item, idx) => (
        <TouchableOpacity
          key={`searchHistory-${idx}`}
          style={styles.searchHistoryItem}
          activeOpacity={.5}
          onPress={() => onClickHistory(item)}
        >
          <Image
            style={styles.searchHistoryIcon}
            source={require('../../img/search-icon.png')}
          />
          <Text style={styles.searchHistoryItemText}>
            { item.length > 25 ? item.slice(0, 25) + '...' : item }
          </Text>
        </TouchableOpacity>
      )) }
    </View>

    <TouchableOpacity
      style={styles.searchCloseContainer}
      activeOpacity={.5}
      onPress={onCloseHandler}
    >
      <Text style={styles.searchBtnText}>Закрыть</Text>
    </TouchableOpacity>
  </View> }
```

```
        </View>
    );
}

const styles = StyleSheet.create({
    container: {
        flex: 1
    },
    searchContainer: {
        height: 45,
        zIndex: 500
    },
    searchHistoryContainer: {
        width: '100%',
        position: 'absolute',
        justifyContent: 'space-between',
        top: 40,
        paddingHorizontal: 10,
        borderWidth: 3,
        borderTopWidth: 0,
        borderBottomStartRadius: 5,
        borderBottomEndRadius: 5,
        borderColor: '#7e64ff',
    },
    searchHistoryHeader: {
        flexDirection: 'row',
        justifyContent: 'space-between',
        alignItems: 'center',
    },
    searchHistoryTitle: {
        fontSize: 16
    },
    searchHistoryItem: {
        flexDirection: 'row',
        alignItems: 'center',
        paddingBottom: 10
    },
    searchHistoryIcon: {
        width: 20,
        height: 20,
        marginRight: 7
    },
    searchHistoryItemText: {
        fontSize: 18
    },
    searchCloseContainer: {
        paddingHorizontal: 10,
        paddingBottom: 5
    },
    searchBtnText: {
        fontSize: 16,
        color: '#555',
```

```
      textTransform: 'uppercase',
      fontWeight: '700'
    }
});
```

package.json

```
{
  "name": "MDA_mobile_app",
  "version": "1.0.0",
  "private": true,
  "scripts": {
    "android": "react-native run-android",
    "ios": "react-native run-ios",
    "start": "react-native start",
    "test": "jest",
    "lint": "eslint ."
  },
  "dependencies": {
    "@react-native-async-storage/async-storage": "^1.14.1",
    "@react-native-community/masked-view": "^0.1.10",
    "@react-navigation/bottom-tabs": "^5.11.7",
    "@react-navigation/drawer": "^5.12.3",
    "@react-navigation/native": "^5.9.2",
    "@react-navigation/stack": "^5.14.2",
    "react": "16.13.1",
    "react-native": "0.63.4",
    "react-native-bootsplash": "^3.2.3",
    "react-native-gesture-handler": "^1.10.1",
    "react-native-reanimated": "^1.13.2",
    "react-native-safe-area-context": "^3.1.9",
    "react-native-screens": "^2.17.1",
    "react-native-webview": "^11.2.4"
  },
  "devDependencies": {
    "@babel/core": "^7.8.4",
    "@babel/runtime": "^7.8.4",
    "@react-native-community/eslint-config": "^1.1.0",
    "babel-jest": "^25.1.0",
    "eslint": "^6.5.1",
    "jest": "^25.1.0",
    "metro-react-native-babel-preset": "^0.59.0",
    "react-test-renderer": "16.13.1"
  },
  "jest": {
    "preset": "react-native"
  }
}
```

ДОДАТОК Б

ВІДГУК
Керівника економічного розділу
на кваліфікаційну роботу бакалавра
на тему:
"Розробка мобільного додатку під керуванням ОС Android для платформи Market Dynamics Analyzer для моніторингу динаміки цін."
студента групи 121-18ск-1 Надтоки Богдана Віталійовича

Керівник економічного розділу
Зав. каф. ПЕП та ПУ, д.е.н. **О.Г. Вагонова**

ДОДАТОК В

ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ

Ім’я файлу	Опис
Пояснювальні документи	
Кваліфікаційна робота Надтока.docx	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Кваліфікаційна робота Надтока.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF
Програма	
Надтока.rar	Архів. Містить коди мобільного додатку і .APK файл додатку
Презентація	
Надтока.pptx	Презентація кваліфікаційної роботи