МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ МІНІ-МАРКЕТУ ОДЯГУ З ВИКОРИСТАННЯМ REACT NATIVE ТА HEADLESS CMS»**

Виконав: студент 4 курсу, групи 6.1221
спеціальності 122 Комп'ютерні науки
(шифр і назва спеціальності)
освітньої програми Комп'ютерні науки
(назва освітньої програми)
А.П. Желобецький
(ініціали та прізвище)
Керівник доцент кафедри комп'ютерних наук, доцент, к.т.н. Матвіїшина Н.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент завідувач кафедри програмної інженерії доцент, к.ф.-м.н. Лісняк А.О.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра комп'ютерних наук

Рівень вищої освіти бакалавр

Спеціальність 122 Комп'ютерні науки
(шифр і назва)

Освітня програма Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук,
д.т.н., професор

_____ Шило Г.М.
(підпис)

" 24 " 12 2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Желобецькому Арсенію Павловичу
(прізвище, ім'я та по-батькові)

1. Тема роботи Розробка мобільного застосунку для міні-маркету одягу з використанням React Native та headless CMS

керівник роботи Матвіїшина Надія Вікторівна, к.т.н., доцент
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 20 » грудня 2024 року № 2001-с

2. Строк подання студентом роботи 02.06.2025

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Огляд наукових праць і аналіз стану проблеми
2. Вибір та обґрунтування можливих варіантів технічної реалізації
3. Розробка та апробація системи

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
презентація

6. Консультанти розділів роботи

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

7. Дата видачі завдання 23.12.2025

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	25.12.2024	
2.	Збір вихідних даних.	12.01.2025	
3.	Обробка методичних та теоретичних	04.02.2025	
	джерел.		
4.	Розробка першого та другого розділу.	15.02.2025	
5.	Розробка третього розділу.	15.04.2025	
6.	Оформлення та нормоконтроль	02.06.2025	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	18.06.2025	

Студент _____ А.П. Желобецький
(підпис) (ініціали та прізвище)

Керівник роботи _____ Н.В. Матвіїшина
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.Г.Спиця
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка мобільного застосунку для міні-маркету одягу з використанням React Native та headless CMS»: 77 с., 26 рис., 1 табл., 25 джерел, 8 додатків.

ВИБІР ТЕХНОЛОГІЙ, КЕРУВАННЯ ВМІСТОМ, МОБІЛЬНА КОМЕРЦІЯ, ANDROID, AHP-TOPSIS, DIRECTUS, EXPO, HEADLESS CMS, REACT NATIVE

Мета дослідження – теоретичне обґрунтування та практична реалізація Android застосунку, дослідження особливостей інтеграції системи керування вмістом.

Об’єкт дослідження – процес розробки Android застосунку в інтеграції з системою керування вмістом.

Предмет дослідження – React Native застосунок для міні-маркету одягу, інтегрований з headless CMS.

Методи дослідження – аналіз, порівняння, експеримент, формалізація.

Одержані результати – у процесі дослідження створено мобільний застосунок для міні-маркету одягу, який реалізує онлайн-взаємодію клієнтів з магазином. Застосунок інтегрований з системою керування вмістом, яка забезпечує можливість централізованого управління магазином, а також надає перспективу створення кросплатформної інформаційної системи.

Інформація щодо впровадження – система може бути інтегрована в бізнес-процес невеликого інтернет-магазину або адаптована під різні типи подібних проєктів.

SUMMARY

Bachelor's Qualification Thesis «Development of a mobile application for a clothing mini-market using React Native and headless CMS»: 77 pages, 26 figures, 1 table, 25 references, 8 supplements.

TECHNOLOGY SELECTION, CONTENT MANAGEMENT, MOBILE COMMERCE, ANDROID, AHP-TOPSIS, DIRECTUS, EXPO, HEADLESS CMS, REACT NATIVE

Aim of the study – theoretical substantiation and practical implementation of an Android application, study of the features of content management system integration.

Object of the study – the process of developing an Android application in integration with a content management system.

Subject of the study – React Native application for a clothing mini-market integrated with a headless CMS.

Methods of research – analysis, comparison, experiment, formalization.

Results obtained – in the course of the study, a mobile application for a clothing mini-market was created, which implements online interaction between customers and the store. The application is integrated with the content management system, which enables centralized store management and provides the prospect of creating a cross-platform information system.

Information on implementation – the system can be integrated into the business process of a small online store or adapted to different types of similar projects.

ЗМІСТ

Завдання на кваліфікаційну роботу……..…..2
Реферат 4
Summary 5
Скорочення та умовні познаки 8
Вступ..... 9
1 Огляд наукових праць і аналіз стану проблеми 11
1.1 Роль мобільних технологій у сучасному світі 11
1.2 Вплив мобільних застосунків на сферу електронної комерції 11
1.3 Сучасний стан та підходи розробки мобільних застосунків..... 14
1.4 Системи керування вмістом: аналіз традиційного та headless підходів . 15
1.5 Інтеграція систем керування вмістом з мобільними застосунками 18
2 Вибір та обґрунтування можливих варіантів технічної реалізації..... 20
2.1 Визначення функціоналу мобільного застосунку..... 20
2.2 Аналіз та вибір мобільних фреймворків для проєкту..... 21
2.3 Аналіз та вибір headless CMS для проєкту 24
2.3.1 Методи багатокритеріального аналізу 25
2.3.2 Алгоритм застосування методу AHP-TOPSIS..... 26
2.3.3 Аналіз результатів порівняння..... 28
2.4 Рекомендації щодо вибору та впровадження технологій..... 31
3 Розробка та апробація системи 33
3.1 Архітектура проєкту..... 33
3.2 Визначення таблиць в базі даних..... 34
3.3 Розробка запитів до бази даних..... 38
3.4 Особливості взаємодії з Directus..... 39
3.5 Особливості розробки мобільного застосунку..... 41
3.5.1 Структура навігації в застосунку..... 42
3.5.2 Використання App Links..... 44

3.5.3 Використання push-сповіщень 45
3.6 Оптимізація роботи застосунку 46
Висновки 48
Перелік посилань..... 50
Додаток А Реалізація алгоритму AHP-TOPSIS для вибору headless CMS 53
Додаток Б Застосування алгоритму AHP-TOPSIS для вибору headless CMS. 58
Додаток В Файл конфігурації headless CMS та суміжних сервісів..... 59
Додаток Г Приклади API функцій на читання та відправку даних 61
Додаток Д Приклади використання API функцій за допомогою React Query 64
Додаток Е Інтерфейс і модулі Directus 66
Додаток Ж Приклад React Native компоненти..... 70
Додаток И Візуальне представлення інтерфейсу застосунку..... 72

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

ОС	Операційна Система
API	Application Programming Interface
APK	Android Package
AHP	Analytic Hierarchy Process
CMS	Content Management System
MCDM	Multiple-Criteria Decision Making
SDK	Software Development Kit
TOPSIS	Technique for Order of Preference by Similarity to Ideal Solution

ВСТУП

Мобільні пристрої та застосунки стали невіддільною частиною нашого цифрового життя, переосмисливши наші уявлення про продуктивність, розваги та споживання. Ця революція особливо відчутна у сфері електронної комерції, тобто торгівлі товарами та послугами через цифрові платформи. Тут дедалі чіткіше спостерігається зростання обсягів онлайн-покупок саме через мобільні пристрої, що логічно пояснюється швидкістю, доступністю, та прозорістю, яку з легкістю може забезпечити застосунок, тобто аспектами, які є критично важливими для успішного функціонування комерції. Отож, актуальність створення мобільного застосунку обґрунтовується тим, що попит на зручні й ефективні рішення, які надає мобільна розробка, буде лише зростати в найближчому майбутньому.

У цьому контексті також важливо зазначити, що створення подібних інформаційних систем потребує інструменту ефективного керування. Одним із можливих способів забезпечення цього аспекту є впровадження системи керування вмістом (Content Management System, CMS), яка дозволяє одночасно централізовано адмініструвати систему та зменшувати складність щодо її підтримки, що дає змогу більше фокусуватись на розробці клієнтської сторони застосунку. Однак існує велика кількість CMS, усі вони мають свої обмеження, недоліки та особливості роботи, тому вибір відповідної системи для конкретного проєкту може стати нетривіальною задачею. Водночас кількість документованих прикладів інтеграцій CMS із мобільними застосунками є значно меншою у порівнянні з аналогічними прикладами для вебзастосунків. Ба більше, спостерігається нестача наукових публікацій, що детально аналізували б це питання.

Таким чином, необхідність дослідження додатково полягає в вивченні концепції CMS, визначенні критеріїв вибору CMS та демонстрації інтеграції

CMS з мобільним застосунком. Загалом основними завданнями, поставленими на кваліфікаційну роботу є:

– дослідження актуального стану мобільної розробки;

– обґрунтування вибору технології для побудови мобільного застосунку;

– розкриття сутності систем керування вмістом;

– обґрунтування вибору конкретної системи керування вмістом;

– налаштування архітектури інформаційної системи;

– розробка мобільного застосунку;

– формування висновків.

Отже, метою роботи є дослідження процесу розробки мобільного застосунку для операційної системи (ОС) Android в інтеграції із системою керування вмістом для створення гнучкого та масштабованого рішення для малого бізнесу.

1 ОГЛЯД НАУКОВИХ ПРАЦЬ І АНАЛІЗ СТАНУ ПРОБЛЕМИ

1.1 Роль мобільних технологій у сучасному світі

Мобільні пристрої стали одним із найважливіших технологічних досягнень останніх десятиліть. Менше ніж за 25 років вони кардинально трансформували уявлення про комунікацію, доступ до інформації, безпеку та банківські операції.

Сьогодні мобільні пристрої генерують понад половину загального інтернет-трафіку, а глобальна кількість користувачів смартфонів перевищує 7 мільярдів [1, 2]. Об’єм інтернет-трафіку можна пояснити значним розвитком технологій зв’язку, таких як 4G та 5G, які забезпечили доступне високошвидкісне підключення. Однак популярність і багатофункціональність мобільних пристроїв в більшості пов’язана з прикладним програмним забезпеченням. Покладаючись на аналіз Dubel M. [3, с. 8-11], щодо трендів дистрибуції мобільних застосунків можна зрозуміти, що незалежно від ОС офіційні магазини застосунків налічують щонайменше 2 мільйони різноманітних застосунків, сфера застосування яких охоплює буквально кожний аспект повсякденного життя. Поєднання потужної апаратної частини та різноманіття застосунків робить мобільні пристрої незамінними інструментами в сучасному світі.

1.2 Вплив мобільних застосунків на сферу електронної комерції

Революційний вплив мобільних технологій трансформував не лише повсякденне життя людей, але й істотно змінив ландшафт електронної комерції. Тут дедалі більшою стає роль мобільних застосунків, про що свідчать як численні статті в інтернеті, наукові публікації так й фінансові показники – у 2024

році мобільні застосунки забезпечили прибуток у розмірі 2 трильйонів доларів США, що становить 57% від глобального прибутку електронної комерції. При цьому частка мобільної комерції у загальній структурі електронної торгівлі демонструє стабільну тенденцію до зростання: з 40% у 2017 році до прогнозованих 63% у 2028 році [4]. Разом з цим практика впровадження застосунків сформулювала дві фундаментальні моделі їх імплементації: брендовані застосунки та застосунки, що приносять дохід. Перша модель виконує маркетингові функції, зокрема підвищення впізнаваності бренду, розширення цільової аудиторії або забезпечення омніканальності продажів. Ефективність цієї моделі підтверджується тим, що станом на 2018 рік 90% провідних світових брендів, таких як Nike, Adidas, Coca-Cola мали власні застосунки [5, с. 4]. Друга модель охоплює ті застосунки, що є платними або містять платні функції чи рекламу. Ця модель популяризувала цифрові покупки, і в тому ж 2018 році витрати на застосунки досягли рекордних 101 мільярда доларів [5, с. 4].

Однак важливо розуміти, що мобільні застосунки у цій сфері набули такого поширення завдяки своїм суттєвим перевагам над іншими каналами електронної комерції. Ґрунтовний аналіз цих переваг представлений у дослідженні Khrais L. T. [6], яке фокусується на впливі мобільних застосунків та соціальних мереж на користувацький досвід онлайн-покупок. Методологія дослідження базується на якісному та кількісному аналізі даних, отриманих шляхом проведення інтерв'ю респондентів із Саудівської Аравії. Результати дослідження дозволяють ідентифікувати критерії, які мобільні технології забезпечують ефективніше, а також ті, що стикаються з потенційними обмеженнями, на які слід звернути увагу під час власної розробки. Для наочності доцільно виділити та візуалізувати ці критерії (див. рис. 1.1).

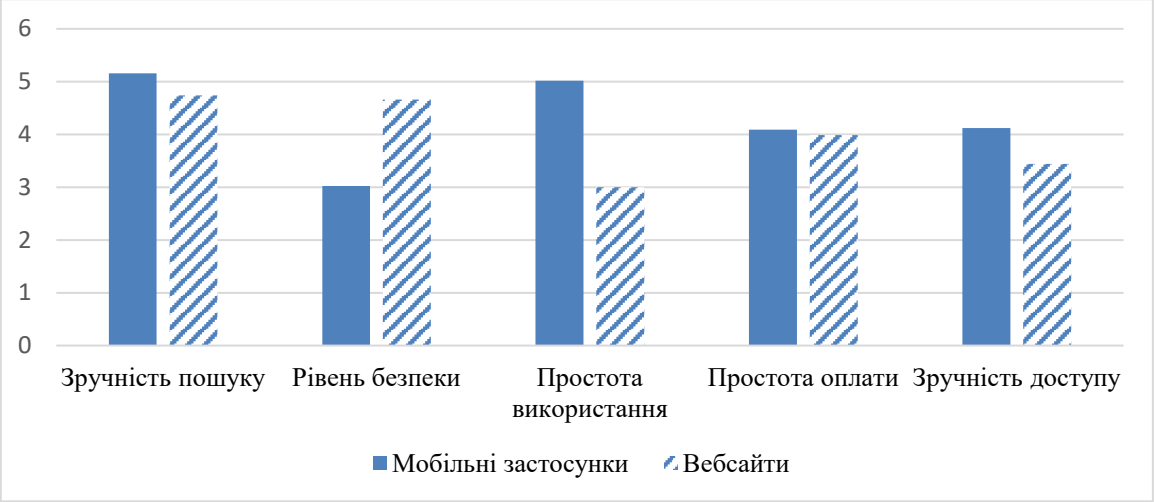


Рисунок 1.1 – Порівняння досвіду онлайн-покупок між двома платформами

На основі рисунка 1.1 можна зробити висновок, що лише рівень безпеки серед усіх критеріїв мобільних застосунків оцінюється нижче, ніж у вебсайтів. Це пов’язано з ризиками, що ідентифіковані респондентами, такими як можливість викрадення пристроїв, підключення до незахищених публічних мереж та ймовірність зловживання конфіденційними даними самими застосунками. Хоча подібні загрози можуть бути характерні й для вебсайтів, такий результат вказує на необхідність забезпечення відповідної уваги безпеці при роботі з мобільними платформами, а також наводить на думку, що хоч мобільні застосунки на сьогодні є популярним способом взаємодії з комерційними сервісами серед 70% користувачів [7], бізнесу варто уникати одностороннього фокусування лише на мобільних платформах. Раціональнішим підходом є їх інтеграція в широку бізнес-стратегію, яка враховує мотивацію, поведінку та потреби цільової аудиторії. Наприклад, у категорій високовартісних товарів, таких як нерухомість, меблі або ювелірні вироби, ефективність мобільних платформ може бути обмеженою через відповідну вартість продуктів та тривалий цикл прийняття рішень. Водночас для роздрібних товарів, таких як одяг, рівень конверсії може бути значно вищим, що свідчить про доцільність використання мобільної платформи [7].

Зважаючи на стабільне зростання ролі як мобільної комерції, так й мобільних технологій, можна стверджувати про актуальність дослідження особливостей мобільної розробки.

1.3 Сучасний стан та підходи розробки мобільних застосунків

Попри динамічний розвиток, мобільна розробка, як й будь-яка інша галузь програмування, зіткнулась з низкою викликів. Серед них – необхідність забезпечення швидкого циклу розробки для вчасного випуску продуктів, стандартизація процесів розробки та методологій [8]. У відповідь на ці виклики сформувалося кілька підходів до створення мобільних застосунків: нативна, гібридна, кросплатформна [9, с. 23].

Нативна модель передбачає створення застосунків для конкретної ОС (наприклад, iOS або Android) з використанням характерних мов програмування: Swift та Objective-C для iOS, Kotlin та Java для Android. Ця модель забезпечує максимальну продуктивність, повний доступ до функцій пристрою, проте потребує детального вивчення особливостей написання коду під певну платформу, що збільшує витрати часу та ресурсів.

Гібридна модель розробки використовує вебтехнології (HTML, CSS, JavaScript) для створення інтерфейсу користувача та основного функціоналу застосунку, який запускається у спеціальній обгортці для доступу до функцій пристрою. Хоча цей підхід дозволяє швидко створювати кросплатформні застосунки, він має значні обмеження у продуктивності та доступі до апаратного забезпечення через те, що фактично працює у вебоболонці. Це додатково негативно впливає на якість користувацького досвіду, а також вимагає більш ретельного тестування. Прикладами фреймворків для гібридної розробки, тобто програмних платформ, що надають набір готових інструментів, бібліотек і структур для спрощення розробки, є Apache Cordova та Ionic.

Кросплатформна модель дозволяє створювати застосунки з єдиним базовим кодом для кількох платформ. Ключовою відмінністю від гібридної моделі є компіляція коду у нативний код платформи, що забезпечує високу продуктивність. Водночас модель також має свої обмеження. Зокрема, доступ до нативних функцій здійснюються за допомогою бібліотек, які підтримуються фреймворком (наприклад, Flutter, React Native). Це може ускладнити реалізацію окремих функцій або обмежити їх доступність. Крім того, хоча продуктивність таких застосунків зазвичай наближається до нативної, у деяких випадках швидкодія та якість користувацького досвіду може поступатися нативним рішенням, що може вимагати додаткового доопрацювання коду.

Таким чином, кожна з моделей мобільної розробки має свої переваги та обмеження, а вибір оптимальної (див. підрозділ 2.2) залежить від цілей, ресурсів та вимог проєкту.

1.4 Системи керування вмістом: аналіз традиційного та headless підходів

Інформаційні системи, від електронної комерції, до освітніх платформ характеризуються необхідністю ефективного управління значними обсягами динамічної інформації та станом системи. Ключовими викликами є забезпечення адміністрування системи, оптимального процесу створення та оновлення інформації, а також її адаптації під різні формати подання та канали комунікації. Вирішенням цих завдань може стати використання спеціалізованих систем керування вмістом.

На сьогодні у сфері CMS сформувалися два основні підходи: традиційні CMS та headless CMS. Історично першими є традиційні або монолітні CMS, які виникли як відповідь на проблему масштабування вебресурсів та оптимізації процесу управління інформацією, що з розвитком вебтехнологій ставала дедалі складнішою та вимагала більших ресурсів. Ці системи запропонували

інтегрований підхід, об’єднуючи функції створення, редагування та публікації інформації з інструментами її відображення. За період еволюції функціональність цих систем значно розширилась від базового управління інформацією до комплексних рішень, що включають управління правами доступу, аналітику, інтеграцію з електронною комерцією та інші функції. [10].

Headless CMS системи виникли як послідовник ідей традиційних систем. Їх ключова відмінність полягає у забезпеченні можливості агностичного доступу до вмісту інформаційної системи, що відкриває можливість створювати та обслуговувати різні клієнтські застосунки (вебсайти, мобільні застосунки та інші платформи).

Для систематизації відмінностей між двома підходами розроблено схематичну структуру систем (див. рис. 1.2) та проведено їх порівняльний аналіз (див. табл. 1.1).

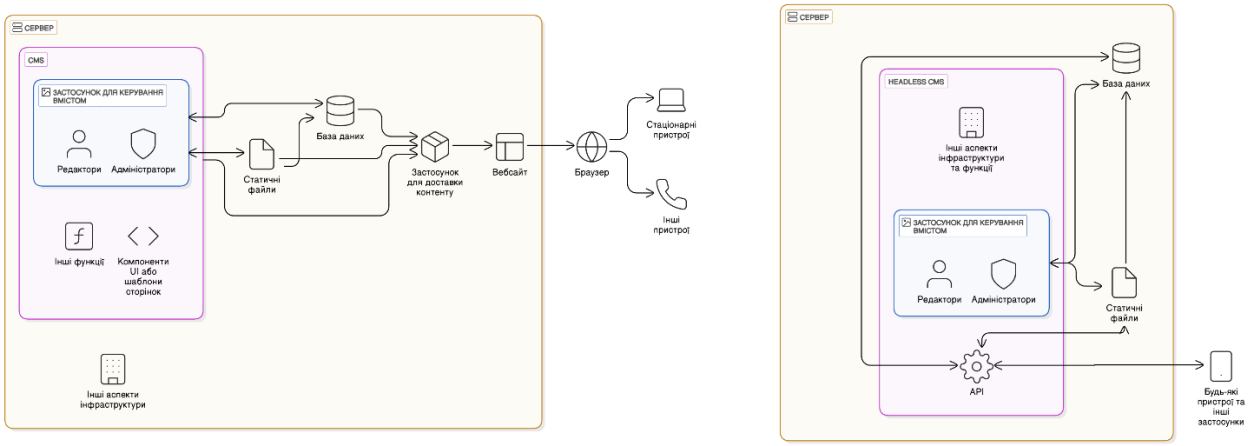


Рисунок 1.2 – Загальна схема структури традиційної та headless CMS

Як показано на рисунку 1.2 архітектурна відмінність між двома підходами полягає в способі взаємодії з клієнтською частиною. У традиційних CMS логіка відображення інформації безпосередньо пов'язана з системою керування, створюючи монолітну структуру. Натомість headless CMS забезпечує гнучкість завдяки архітектурі, орієнтованій на програмний інтерфейс застосунків (Application Programming Interface, API), що й дозволяє створювати різноманітні

клієнтські інтерфейси незалежно від базової системи керування вмістом. Подальше порівняння двох систем представлено таблицею (див. табл. 1.1).

Таблиця 1.1 – Порівняння двох видів CMS систем

Характеристика	Традиційна CMS	Headless CMS
Сфера застосування	Оптимально підходить для простих сайтів, блогів, новинних порталів та комерції та систем, які потребують постійного оновлення інформації.	Ширша, адже поєднує можливості традиційних CMS та дозволяє адаптовувати клієнтські застосунки під різні вимоги та потреби.
Необхідність технічних навичок	Необов’язкова, користувач може вносити інформацію та управляти платформою без глибоких технічних знань.	Обов’язкова, оскільки розробка клієнтської сторони системи та використання API вимагає знань програмування.
Час розробки	Зазвичай менший завдяки готовим шаблонам та компонентам.	Більший, особливо у випадку інтеграції власної бізнес-логіки.
Технічні аспекти (хостинг, безпека, обслуговування тощо)	Забезпечуються платформою	Можуть бути забезпечені платформою. У випадку розгортання у власному середовищі, може знадобитись ручне налаштування.

Проведений аналіз демонструє, що вибір між традиційною та headless архітектурою має базуватись на комплексній оцінці вимог проєкту. Традиційні CMS можуть демонструвати ефективність у проєктах з усталеними та скромними вимогами до відображення інформації й персоналізації та обмеженою кількістю каналів комунікації. Водночас headless CMS є оптимальним вибором для проєктів, що потребують мультиканальної дистрибуції, високого рівня персоналізації та гнучкості щодо майбутніх технологічних змін.

1.5 Інтеграція систем керування вмістом з мобільними застосунками

Зі зростанням значення цифрових технологій роль систем керування стає дедалі помітнішою. Це легко підтверджується аналізом науково-технічної літератури та галузевих джерел, які демонструють значний інтерес до впровадження CMS у різні інформаційні системи, що корелюється з інформацією у підрозділі 1.4.

Проведений пошук та аналіз наукових публікацій на платформах Semantic Scholar [11], Academia.edu [12], ResearchGate [13] та Google Scholar [14], лише за 2021-2024 роки виявив суттєвий масив досліджень у сфері застосування CMS для електронної комерції. Зокрема, окремі наукові праці розглядають практичні аспекти інтеграції headless CMS з комерційними застосунками, інші присвячені інтеграції традиційних CMS, а також існує ряд робіт, які зосереджені на теоретико-методологічних засадах даної технології. Важливо зазначити, що в контексті мобільних застосунків доцільно розглядати саме headless CMS, оскільки традиційні CMS здебільшого орієнтовані на використання у межах однієї платформи – вебсайтів. Хоча теоретично можлива інтеграція мобільного застосунку з традиційною CMS, такий підхід є малопрактичним, через обмежену масштабованість, продуктивність та гнучкість. У будь-якому випадку в проаналізованому масиві публікацій жодних прикладів інтеграції headless CMS з мобільним застосунком або відмінною від вебсередовища платформою не було виявлено.

Примітним є також те, що, попри актуальність та популярність, кількість документованих прикладів інтеграції headless CMS саме з мобільними застосунками на офіційних сторінках цих систем значно поступається аналогічним показникам для вебрішень. Точно встановити передумову такої тенденції складно, множина причин може пояснювати цю закономірність, зокрема історична орієнтація CMS інструментів на вебсередовище або більша доступність веброзробки для малих бізнесів, наявність миттєвого попереднього перегляду для вебверсії проєкту, що може бути важливим чинником для

інвесторів або просто обмежена увага офіційної документації до мобільних інтеграцій. В будь-якому випадку, наявні приклади успішної імплементації від провідних платформ, зокрема Strapi, Sanity, Contentful, Payload CMS, демонструють практичну можливість та потенційну ефективність такої інтеграції, особливо у сфері електронної комерції, що дозволяє відібрати їх для порівняння (див. підрозділ 2.3).

Комплексний аналіз виявлених тенденцій та наявних прогалин у дослідженнях свідчить про актуальність та необхідність вивчення особливостей інтеграції headless CMS із мобільними застосунками, зокрема у контексті розробки системи електронної комерції.

2 ВИБІР ТА ОБҐРУНТУВАННЯ МОЖЛИВИХ ВАРІАНТІВ ТЕХНІЧНОЇ РЕАЛІЗАЦІЇ

2.1 Визначення функціоналу мобільного застосунку

З метою формування оптимального набору функціональних вимог до розроблюваної системи проведено порівняльний аналіз функціональних можливостей наявних мобільних застосунків електронної комерції у сфері продажу одягу. Емпіричну базу дослідження склали 16 провідних застосунків, представлених на платформі Google Play Market [15], зокрема в порівняння увійшли такі відомі рішення як Asos, Puma, Bershka, Shein, Zara та інші.

За результатами проведеного аналізу ідентифіковано ключові функціональні компоненти, що є спільними для більшості досліджених застосунків, та, відповідно, можуть вважатися базовими вимогами для розроблюваної системи. Для структурованого представлення виявлених вимог обрано методологію історій користувача (user stories) (див. рис. 2.1), що дозволяє чітко визначити як потреби користувачів, так й необхідні сторінки в застосунку. При цьому залишається можливість представити адміністративний функціонал, який можна використовувати при виборі відповідної headless CMS.

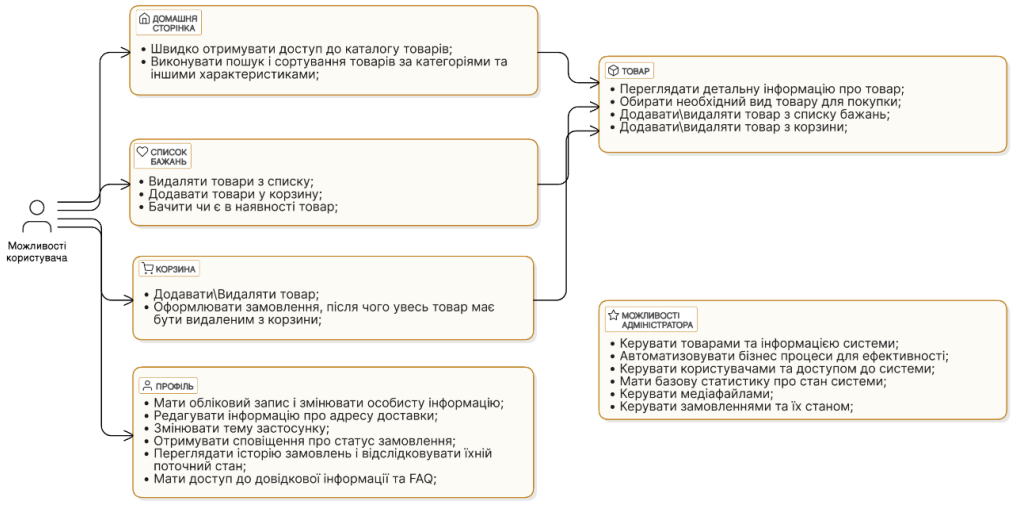


Рисунок 2.1 – Основний функціонал застосунку та CMS

Враховуючи визначені критерії з підрозділу 1.2, прийнято рішення про делегування функціоналу обробки платежів на адміністративний рівень системи. Це дозволяє мінімізувати потенційні ризики безпеки та спростити архітектуру, водночас зберігаючи повну функціональність системи, адже опрацювання платежів може відбуватись під час взаємодії з адміністратором системи, що для малого бізнесу є цілком прийнятим. За винятком цього специфічного аспекту, розроблена система забезпечує задовільний рівень безпеки, простоту використання, а також зручність пошуку та доступу, що відповідає визначеним критеріям підрозділу 1.2.

2.2 Аналіз та вибір мобільних фреймворків для проєкту

У підрозділі 1.3 проаналізовано фундаментальні моделі розробки мобільних застосунків, які є основою для функціонування різних фреймворків та технологій. Для обґрунтованого вибору інструментарію розробки необхідно розглянути як нативну технологію Android Native, так і кросплатформні (Flutter, React Native) та гібридні (Apache Cordova, Ionic) фреймворки. Відповідно до комплексного дослідження проведеного Banwarie A. [16], присвяченого проблематиці вибору оптимального фреймворку для Android-розробки можна виокремити ключові фактори вибору фреймворку: продуктивність, складність вивчення та розробки, час на розробку, цільові платформи, технічна підтримуваність та наявність необхідних бібліотек. Додатково можна підкреслити важливу закономірність – загальну непопулярність гібридних фреймворків – які посідають останні місця в результатах як якісного, так і кількісного аналізу [16].

Зазначена тенденція корелюється з даними власного дослідження. Зокрема на рисунку 2.2 зображена статистика, отримана з ресурсу Google Trends [17], на якій демонструється глобальна зацікавленість фреймворками, де значення 100 відповідає максимальному рівню зацікавленості в певний період часу.

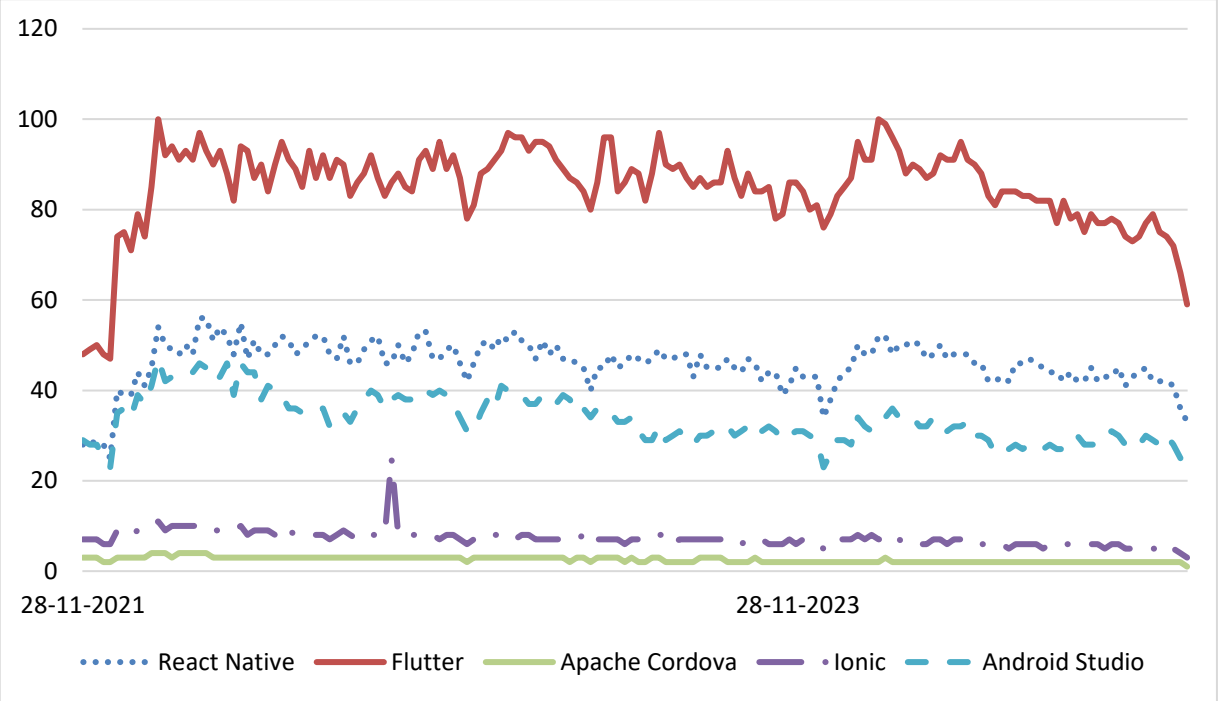


Рисунок 2.2 – Глобальна зацікавленість фреймворками для мобільної розробки за даними Google Trends

Відповідно до рисунка 2.2 гібридні фреймворки, як Apache Cordova та Ionic, стабільно займають найменші позиції порівняно з кросплатформними фреймворками, такими як Flutter та React Native. Схожу закономірність відображає активність у репозиторіях GitHub [18] відповідних фреймворків та статистичні дані зі Stack Overflow [19], за винятком зміщення на користь Android-розробки. Розбіжність в даних можна пояснити спеціалізованістю цих платформ на технічних аспектах розробки та професійній аудиторії розробників, в той час, як Google Trends охоплює лиш загальний інтерес та не надає можливості розглянути питання саме нативної розробки. Отже, отримана інформація щодо гібридних фреймворків може свідчити про слабку підтримку спільноти, що своєю чергою може ускладнити розв'язання технічних проблем і призвести до труднощів у реалізації функціоналу, що взаємодіє з апаратними функціями пристрою, через застарілі або обмежені бібліотеки. Додатково, враховуючи, що гібридні фреймворки поступаються в продуктивності та гнучкості іншим альтернативам, можна зробити висновок, що їх використання є доцільним для розробки простих та невимогливих застосунків, які не

передбачають тривалої підтримки чи масштабування. Це обмежує їх перспективність у контексті розробки застосунку для міні-маркету одягу.

Для більш ґрунтовного вибору між Android Native, Flutter та React Native для реалізації проєкту, застосунки, обрані у підрозділі 2.1, використано для визначення фреймворку, який було застосовано для їх створення (див. рис. 2.3). Для цього здійснено наступні кроки:

– отримано Android Package (APK) версію застосунку за допомогою ресурсу ApkMirror [20];

– проведено зворотний інжиніринг для встановлення фреймворку, який використовувався для розробки, шляхом аналізу за допомогою Android Studio (офіційного середовища розробки для Android-застосунків) вихідних файлів. Для ідентифікації Flutter достатньо було знайти файл libflutter.co, для React Native – libreactnativejni.so або assets/index.android.bundle. У разі гібридних фреймворків серед файлів містилася б велика кількість HTML та JavaScript, зокрема index.html. Якщо ж такі файли відсутні, застосунок створено за допомогою нативних технологій (Kotlin\Java).

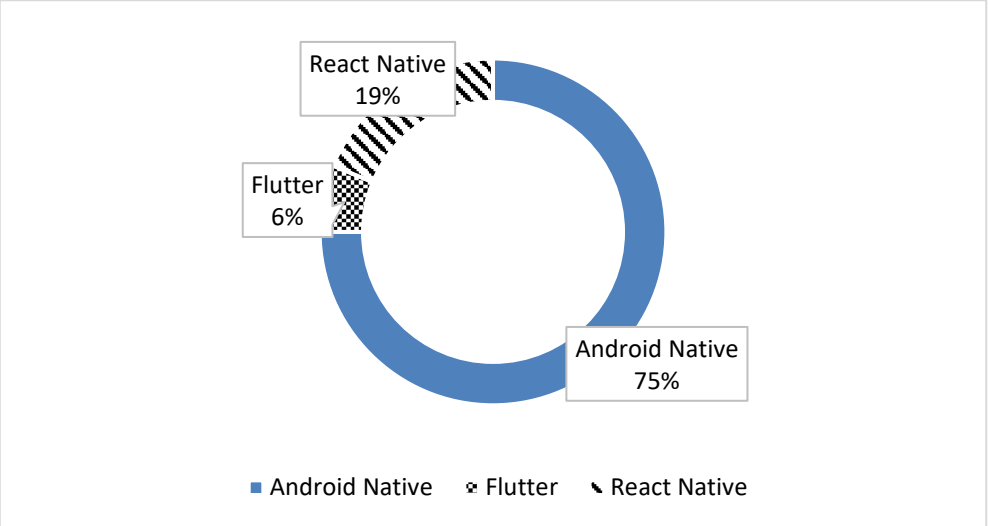


Рисунок 2.3 – Аналіз використання фреймворків у популярних застосунках для електронної комерції одягу

Результати, представлені на рисунку 2.3, показують, що, попри суттєво більшу популярність Flutter за останні 4 роки, більшу кількість застосунків демонструє саме React Native. Це пояснюється більшою спільнотою та зрілістю React Native, створеного у 2015 році, порівняно з Flutter, який з'явився лише у 2018. Водночас попри очевидну перевагу Android Native, необхідно оцінити цю технологію згідно з визначеними вище факторами. Безперечні переваги Android Native – висока продуктивність та підтримуваність – можуть бути нівельовані значними витратами часу та складністю розробки. Крім того, при використанні цієї технології обмежується перспектива подальшого розширення проєкту через неможливість легкого додавання підтримки іншої ОС чи вебверсії. Натомість Flutter та React Native пропонують оптимальний баланс між продуктивністю та легкістю розробки й можливістю подальшого розширення. Вибір на користь React Native обґрунтовується двома ключовими факторами: наявністю значного практичного досвіду з бібліотекою React та простотою її екосистеми, тоді як розробка на Flutter потребує володіння мовою програмування Dart, застосування якої обмежується переважно цим фреймворком; а також наявністю Software Development Kit (SDK) для обраної в пункті 2.3.3 headless CMS лише на JavaScript/TypeScript, використання якого забезпечить безперешкодну інтеграцію з CMS.

2.3 Аналіз та вибір headless CMS для проєкту

З огляду на різноманітність доступних на ринку headless CMS – кількість яких перевищує 100 рішень – процес вибору найбільш відповідної системи може бути досить складним. Кожен варіант має свої специфічні характеристики та обмеження, що можуть істотно вплинути на успішність розробки та масштабування проєкту. Тому для прийняття обґрунтованого рішення є необхідним проведення аналізу доступних систем.

Для порівняння, враховуючи інформацію з підрозділу 2.1, було визначено низку ключових критеріїв, які є важливими для успішної реалізації проєкту:

– зрілість системи – включає стабільність роботи, наявність активної спільноти та відповідей на поширені запитання, тривалість існування;

– гнучкість управління зв’язками – можливість налаштування взаємозв’язків між даними та визначення поведінки при оновленні або видаленні даних;

– гнучкість використання – включає можливість розгортання у власному середовищі, підтримку бізнес-логіки, наявність інструментів для електронної комерції, а також інструментів для управління користувачами та доступом до API;

– зручність використання – повноцінність документації, простота вивчення та використання, а також загальна швидкодія;

– цінова політика – вартість використання системи та наявність обмежень, таких як ліміти на кількість запитів до API або функціональні обмеження у безплатних версіях.

Для порівняння обрано 7 найбільш популярних headless CMS, які згідно з інформацією на офіційних вебсайтах або в блозі розробників, мають можливість застосовуватися в контексті електронної комерції та мають документовані приклади такої інтеграції. Ці системи: Payload CMS, Directus, Strapi, Contentful, Sanity, MedusaJS та Vendure. Оскільки порівняння таких рішень є складним і багатогранним, для формалізації процесу аналізу обрано метод багатокритеріального аналізу (Multiple-Criteria Decision Making, MCDM), який забезпечує структурований та оптимізований підхід до оцінки та дозволяє зробити правильний вибір.

2.3.1 Методи багатокритеріального аналізу

Багатокритеріальний аналіз є важливим етапом в процесі прийняття рішення, особливо у випадках, коли необхідно враховувати різноманітні, часто

суперечливі критерії. Методи MCDM дозволяють структурувати процес вибору, оцінювати альтернативи за допомогою кількісних та якісних показників, а також приймати зважені рішення в умовах невизначеності [21]. Серед численних методів MCDM можна виділити класичні методи Analytic Hierarchy Process (AHP) та Technique for Order of Preference by Similarity to Ideal Solution (TOPSIS), які вважаються найпоширенішими та найбільш перевіреними у багатьох галузях застосування [21, с. 188].

Для вибору відповідного методу в контексті вибору headless CMS доцільно звернутись до головних недоліків, які мають ці два методи. Для AHP – недоцільність використання у випадку великої кількості критеріїв або альтернатив (понад 7), що приводить до суттєвого збільшення обчислювальної складності та зменшення точності методу [22, с. 5643]. У випадку TOPSIS – невизначеність щодо правильного способу визначення ваг для критеріїв [22, с. 5644]. Однак в літературі, часто пропонується комбінувати різні методи для обходу їх недоліків. Одним з таких підходів є комбінація AHP та TOPSIS, яка дозволить розв’язати визначені проблеми [21, с. 189].

2.3.2 Алгоритм застосування методу AHP-TOPSIS

Покроково алгоритм реалізується у два етапи – AHP та TOPSIS [23]. **Етап AHP складається з наступних кроків:**

– постановка задачі – визначення цілі порівняння, критеріїв та альтернатив;

– побудова матриці парних порівнянь для критеріїв (C) – порівняння кожного критерію з іншими за 9-бальною шкалою T. Saaty [24, с. 6]. Принцип порівняння полягає в тому, що той критерій, який порівнюється з іншим, отримує оцінку за шкалою, а той, з яким порівнювався, отримує обернену оцінку (1/оцінка). Елементи на головній діагоналі отримують завжди оцінку 1.

– нормалізація матриці з критеріями (N) – кожен елемент C ділиться на суму свого стовпця;

– розрахунок вектора ваг (W) – обчислення середнього значення кожного рядка N;

– перевірка узгодженості – здійснення обчислення за формулою (2.1). Якщо отримане значення є меншим від 0.1, то це значить, що узгодженість є прийнятною. У іншому випадку необхідно переглянути парні порівняння, або бути обережним при прийнятті рішення.

$$CR = \frac{CI}{RI}, \tag{2.1}$$

де $CI = \frac{\lambda_{max} - n}{n-1}, \lambda_{max}$ – найбільше власне значення матриці;

RI – середнє значення індексу узгодженості. Визначається за таблицею випадкового індексу T. Saaty для різної кількості критеріїв [24, с. 9].

Етап TOPSIS налічує наступні кроки:

– оцінка альтернатив – кожна альтернатива отримує оцінку за визначеними критеріями. В результаті формується матриця X ($[x_{ij}]_{m \times n}$);

– побудова нормалізованої матриці оцінок (R) – оцінки перетворюються у безрозмірні величини, щоб забезпечити порівнянність між критеріями за наступною формулою:

$$r_{ij} = \frac{x_{ij}}{\sqrt{\sum_i x_{ij}^2}}, i = i, \dots, m, j = 1, \dots, n, \tag{2.2}$$

– побудова зваженої нормалізованої матриці (V) – кожен стовпець R множиться на вагу критерія (w_{ij}), отриману в результаті етапу AHP;

– розрахунок позитивного (I^+) та негативного (I^-) ідеального рішення – для кожного стовбця з V вибирається максимальне значення (позитивне рішення) в результаті чого формується вектор $I^+ = \{v_1^*, \dots, v_n^*\}$. Так само вибирається мінімальне значення (негативне рішення) в результаті чого формується вектор $I^- = \{v_1', \dots, v_n'\}$;

– розрахунок відстаней до ідеальних рішень (S_i^+, S_i^-) – для кожного рядка в V застосовується формула:

$$S_i^+ = \left(\sum_j \left(v_{ij} - v_j^* \right)^2 \right)^{\frac{1}{2}}, j = 1, \dots, m \tag{2.3}$$

$$S_i^- = \left(\sum_j \left(v_{ij} - v_j' \right)^2 \right)^{\frac{1}{2}}, j = 1, \dots, m \tag{2.4}$$

– розрахунок відносної близькості до ідеального рішення (C^*) – відбувається за формулою (2.5). Після чого обирається варіант найближчий до 1.

$$C_i^* = \frac{S_i^-}{S_i^- + S_i^+}, 0 < C_i^* < 1 \tag{2.5}$$

2.3.3 Аналіз результатів порівняння

Для реалізації методу AHP-TOPSIS створено власну імплементацію з використанням мови програмування Python (див. додаток А). Правильність роботи методу було перевірено на декількох простих прикладах, після чого застосовано для оцінювання headless CMS. Результати представлено у вигляді графіку (див. рис 2.4).

За результатами проведеного аналізу (див. рис. 2.4) встановлено, що Contentfull демонструє найнижчі показники серед досліджуваних платформ. Основними факторами, що зумовлюють такий результат є непропорційно висока вартість послуг та істотні функціональні обмеження безплатної версії, що створює економічний тиск щодо переходу на платну підписку. Додатково, попри те, що Contentfull позиціонує себе як інструмент, який може бути використаний для розробки рішень в електронній комерції, емпіричний аналіз виявив ключову

деталь, яка формує певне функціональне обмеження у цьому напрямку. Ця деталь полягає в делегації усіх комерційних функцій стороннім сервісам (Shopify, BigCommerce, Commercetools тощо).

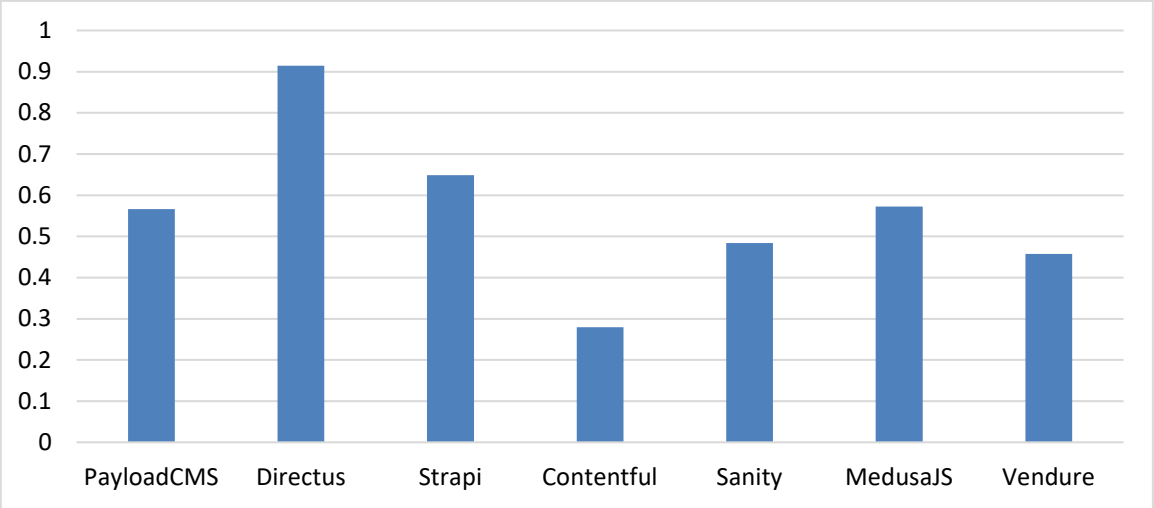


Рисунок 2.4 – Результати порівняння headless CMS за визначеними критеріями

Аналогічна тенденція спостерігається у таких системах, як Strapi, Sanity та ряду інших подібних платформ. Хоча такий підхід може бути виправданим для швидкого розгортання системи без значних інвестицій в розробку та час, він породжує щонайменше два фундаментальних обмеження:

– втрата контролю над бізнес-логікою, що може ускладнити персоналізацію та управління;

– ефект vendor lock-in, який збільшує ланцюг постачальників і водночас підвищує залежність від сторонніх сервісів. Це може створювати значні труднощі при масштабуванні або міграції на інші технології, оскільки подібні зміни можуть супроводжуватися високими фінансовими витратами та додатковими технічними труднощами.

Додатково, впровадження таких рішень може призвести до зростання операційних витрат через необхідність підтримки декількох інтеграцій, регулярних платежів за сторонні сервіси та додаткових витрат на технічну підтримку. В поєднанні зі слабкою документацією Contentfull, відсутністю можливостей для розгортання у власному середовищі, гнучкого управління

користувачами, доступом до API та зв’язками між даними можна зробити висновок, що вибір цієї платформи є доцільним для підтримки блогів, новинних порталів та сайтів, де основним завданням є просте та швидке керування текстовим вмістом. Схожий висновок можна зробити й щодо Sanity, не дивлячись на кращу реалізацію визначених аспектів.

У цьому контексті більш привабливими виглядають платформи, які інтегрують більшість функцій безпосередньо в себе або дозволяють інтеграцію за допомогою відкритої архітектури із можливістю глибокої персоналізації. Прикладами таких систем є MedusaJS, Vendure, Payload CMS, Directus та в певному ступені Strapi.

MedusaJS та Vendure забезпечує виняткову функціональність для електронної комерції, забезпечуючи повний контроль над бізнес-логікою та надаючи гнучкі модулі для управління товарами, замовленнями, валютою, користувачами, користувацькими кошиками тощо. Однак детальний аналіз виявив суттєві недоліки в документації обох платформ, зокрема брак практичних прикладів, послідовності та чіткості. Додатковими факторами ризику є обмежена активність спільноти та підвищена складність конфігурації, що може вимагати прямої взаємодії з розробниками платформ для розробки проєкту. Ці проблеми значно ускладнюють використання цих інструментів, що стало причиною відмови від їх використання для цього проєкту.

Payload CMS пропонує збалансоване рішення з вбудованими шаблонами для електронної комерції та високим ступенем гнучкості в налаштуванні всіх компонентів системи. Однак, як і MedusaJS, система демонструє нестабільність у роботі та проблеми зі швидкодією, що пояснюється активною фазою розробки цих платформ. Через ці обмеження вибір Payload CMS також був відкинутий для реалізації даного проєкту.

Вибір між Strapi та Directus обґрунтовується вищими показниками стабільності та швидкодії Directus, а також розширеними можливостями конфігурації бізнес-логіки, управління зв’язками між даними, користувачами та

API-інтерфейсом, що є критичними обмеженнями Strapi. Повні результати аналізу можна переглянути в додатку Б.

2.4 Рекомендації щодо вибору та впровадження технологій

На основі проведеного підбору технологій для проєкту, сформульовано наступні рекомендації щодо проєктування та впровадження подібних систем.

Першим етапом має стати комплексний передпроєктний аналіз. Він включає детальне дослідження цільової аудиторії та порівняльний аналіз наявних ринкових рішень. На основі якого формується висновок щодо актуальності розробки та функціональних вимог до мобільного застосунку.

При виборі підходу до розробки мобільного застосунку необхідно зважати на наявний час та ресурси на розробку, а також вимоги до продуктивності. За наявності достатніх ресурсів рекомендується віддавати перевагу нативній розробці, в іншому випадку – кросплатформній.

Вибір CMS потребує розробки чіткої системи критеріїв оцінювання на основі функціональних вимог проєкту. Для об’єктивізації процесу вибору доцільно застосувати формалізовані методи оцінювання, такі як метод експертних оцінок (метод Дельфі, безпосереднього ранжування тощо), метод багатокритеріального аналізу тощо.

Щодо конкретних CMS-рішень, у ході дослідження встановлено, що поза межами вебсередовища актуально розглядати headless підхід. У цьому сегменті Directus демонструє оптимальний баланс у ціновій політиці, зрілості, гнучкості та стабільності. Для проєктів із переважно текстовим наповненням доцільно розглянути Sanity або Strapi.

MedusaJS є перспективним рішенням для комерційних застосунків, так само як і Payload CMS, який може вважатись універсальним варіантом. Проте важливо відзначити, що повноцінне впровадження цих систем у виробниче

середовище доцільне лише після суттєвого вдосконалення їхньої документації та підвищення загальної стабільності роботи.

Критичним фактором успіху впровадження є попереднє тестування обраної CMS на забезпечення усіх поставлених цілей, адже неправильний вибір може призвести до небажаних витрат, неможливості масштабування або впровадження необхідного функціоналу.

3 РОЗРОБКА ТА АПРОБАЦІЯ СИСТЕМИ

3.1 Архітектура проєкту

Першим етапом розробки застосунку є визначення загальної структури проєкту, що створює основу для ефективного управління його компонентами та забезпечує узгодженість. Для кращого розуміння структура проєкту представлена у вигляді схеми (див. рис. 3.1).

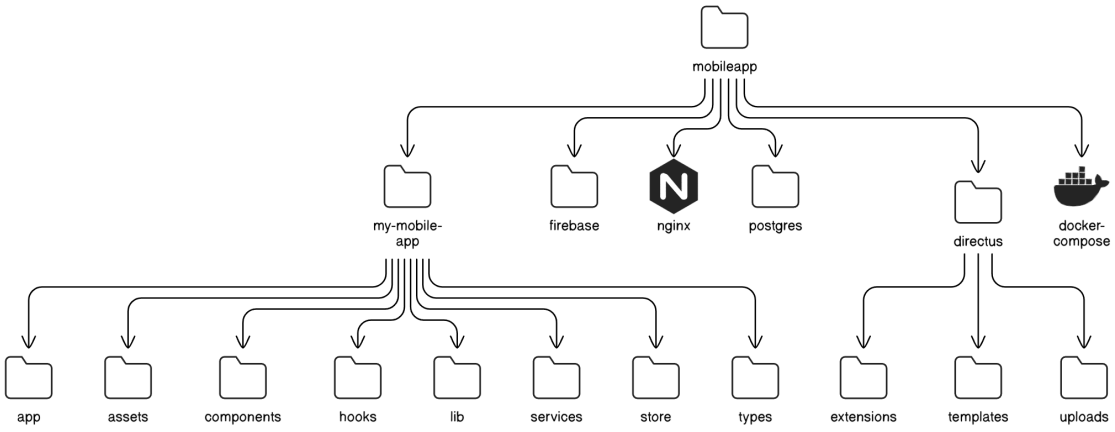


Рисунок 3.1 – Структура проєкту

Таким чином, інформаційна система будується як монолітна директорія, яка містить як мобільний застосунок (my-mobile-app), так і CMS (directus). Деталізація функціонального призначення основних директорій подана нижче:

– firebase – директорія, що містить конфігураційні файли для налаштування доменного імені. Це необхідно для реалізації універсальних посилань (Android App Links) (див. пункт 3.5.2), що забезпечують функціонал підтвердження реєстрації, скидання пароля тощо;

– postgres – директорія з файлами бази даних, що забезпечує збереження інформації навіть у випадку знищення контейнерів, пов’язаних із проєктом;

– nginx – містить приклад налаштування проксі-сервера з урахуванням використання SSL (Secure Sockets Layer) сертифікату, що дозволить розгорнути систему у реальному середовищі;

– templates – директорія, що дозволяє визначати власні шаблони електронних листів за допомогою Liquid – шаблонної мови, використовуваної для генерації листів із динамічною інформацією;

– uploads – директорія, що зберігає усі статичні файли завантажені в базу даних;

– extensions – директорія для реалізації додаткової бізнес-логіки, зокрема додання власних кінцевих точок API, розширень, які використовують сторонні бібліотеки тощо.

Рекомендована конфігурація системи Directus базується на застосуванні технології контейнеризації (Docker). Приклад налаштування системи подано в додатку В.

3.2 Визначення таблиць в базі даних

Визначивши загальну архітектуру проєкту, наступним етапом є проєктування бази даних, яка є фундаментальним компонентом системи. Обрана headless CMS Directus демонструє гнучкість у виборі системи керування базами даних (СКБД), підтримуючи широкий спектр реляційних СКБД. Для реалізації даного проєкту обрано PostgeSQL, оскільки вона є рекомендованим варіантом у загальному випадку згідно із документацією Directus.

Для роботи зі структурою бази даних, Directus надає інтуїтивно зрозумілий графічний інтерфейс. Увесь процес моделювання даних здійснюється на сторінці Data Model, де надається повний функціонал для створення та редагування сутностей (див. рис. 3.2).

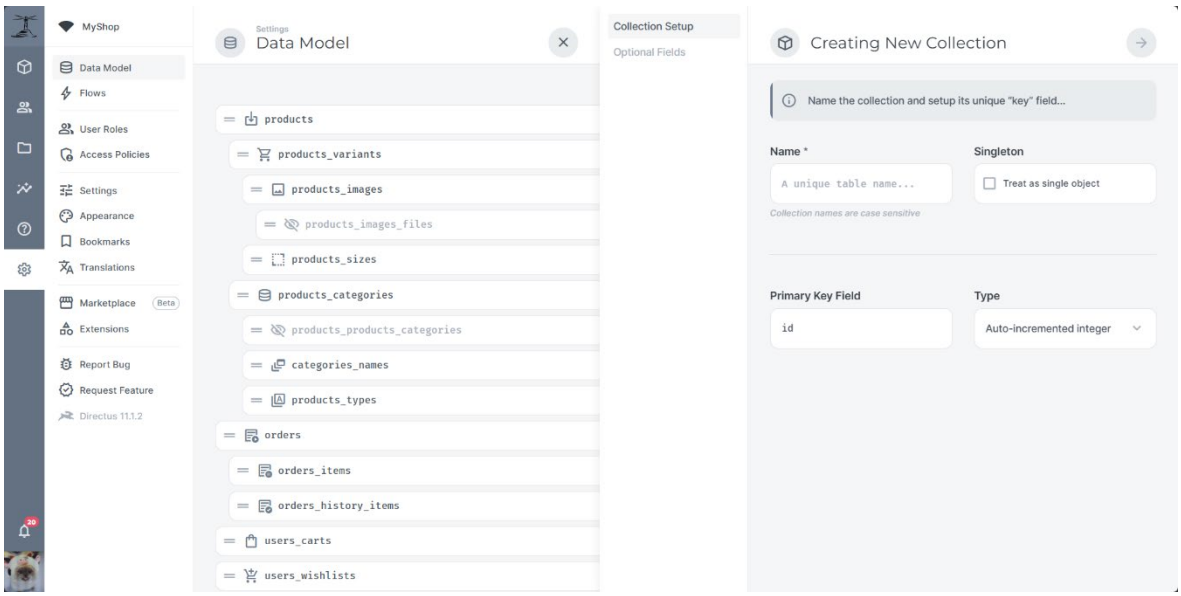


Рисунок 3.2 – Інтерфейс Directus для створення та редагування сутностей

Особливістю проєктування в Directus є те, що замість вибору стандартних типів даних для визначення атрибутів, вибір здійснюється з ряду компонентів user interface (UI), таких як текстові поля, медіафайли, перемикачі, випадні списки, багаторядкові текстові зони тощо (див. рис. 3.3).

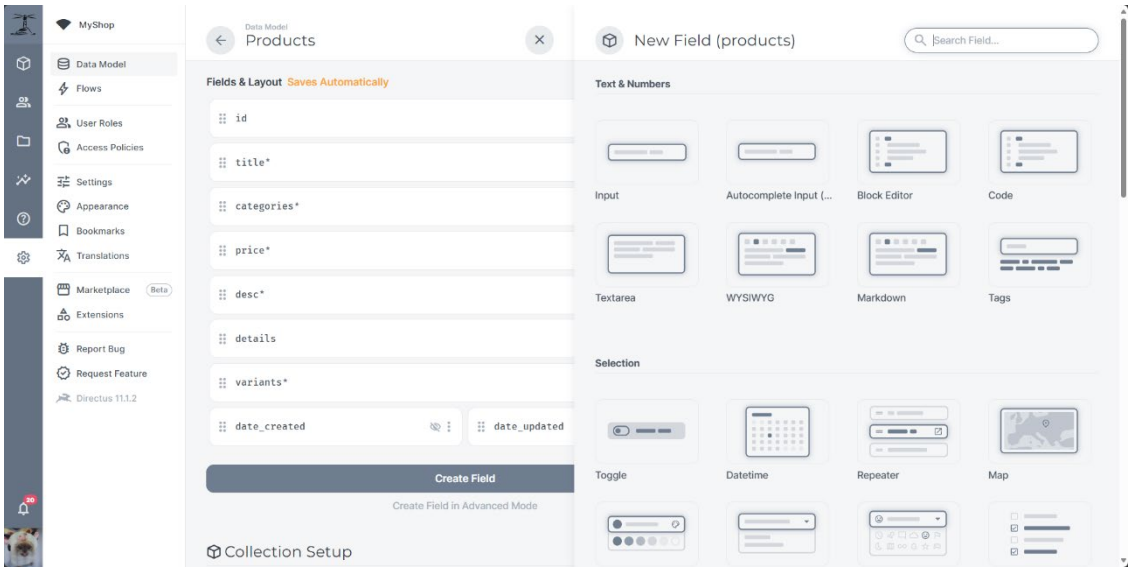


Рисунок 3.3 – Інтерфейс Directus для редагування атрибутів сутності

При цьому Directus зберігає можливість подальшого редагування типів даних у разі необхідності. Такий підхід робить розширення системи доступним

не тільки для розробників, а й для користувачів, які можуть не мати технічних знань.

Крім визначення атрибутів Directus також надає можливість налаштування зв’язків між сутностями, які теж реалізуються через компоненти, які дозволяють визначати тип зв’язку (один-до-багатьох, багато-до-одного, багато-до-багатьох тощо). Разом з цим зберігається й можливість встановлення каскадних операцій між зв’язками, таких як каскадне оновлення або видалення. Це дозволяє автоматично синхронізувати зміни між пов’язаними сутностями, що є критично важливим для підтримки цілісності даних, особливо у рамках визначеного проєкту.

Отже, для визначення сутностей для міні-маркету одягу було використано інтерфейс, який надає Directus. При проєктуванні враховувався необхідний спектр функціональності застосунку, який був визначений у підрозділі 2.1, та застосовувались принципи нормалізації баз даних, що сприяло мінімізації надлишковості, підвищенню ефективності зберігання інформації та забезпеченню її консистентності.

Структура бази даних проєкту побудована навколо центральних сутностей: товарів (products) та їх варіацій (products_variants). Сутність products містить загальну інформацію про товар (назва, ціна, опис), тоді як products_variants деталізує варіанти товару за різними параметрами (колір, розмір тощо).

Для класифікації товарів використовується ієрархічна структура категорій, реалізована через сутності products_categories, products_types та categories_names. Такий підхід дозволяє адміністратору гнучко групувати товари.

Для відстеження замовлень та забезпечення їх цілісності застосовується не менш важливий комплекс сутностей: orders, oders_items та orders_history_items, де сутність orders_history_items призначена для архівування інформації про замовлення у випадку видалення товару з каталогу.

Детальна схема цих та деяких інших сутностей й зв’язків між ними, що були покладені в основу розробленого застосунку, наведена на рисунку 3.4.

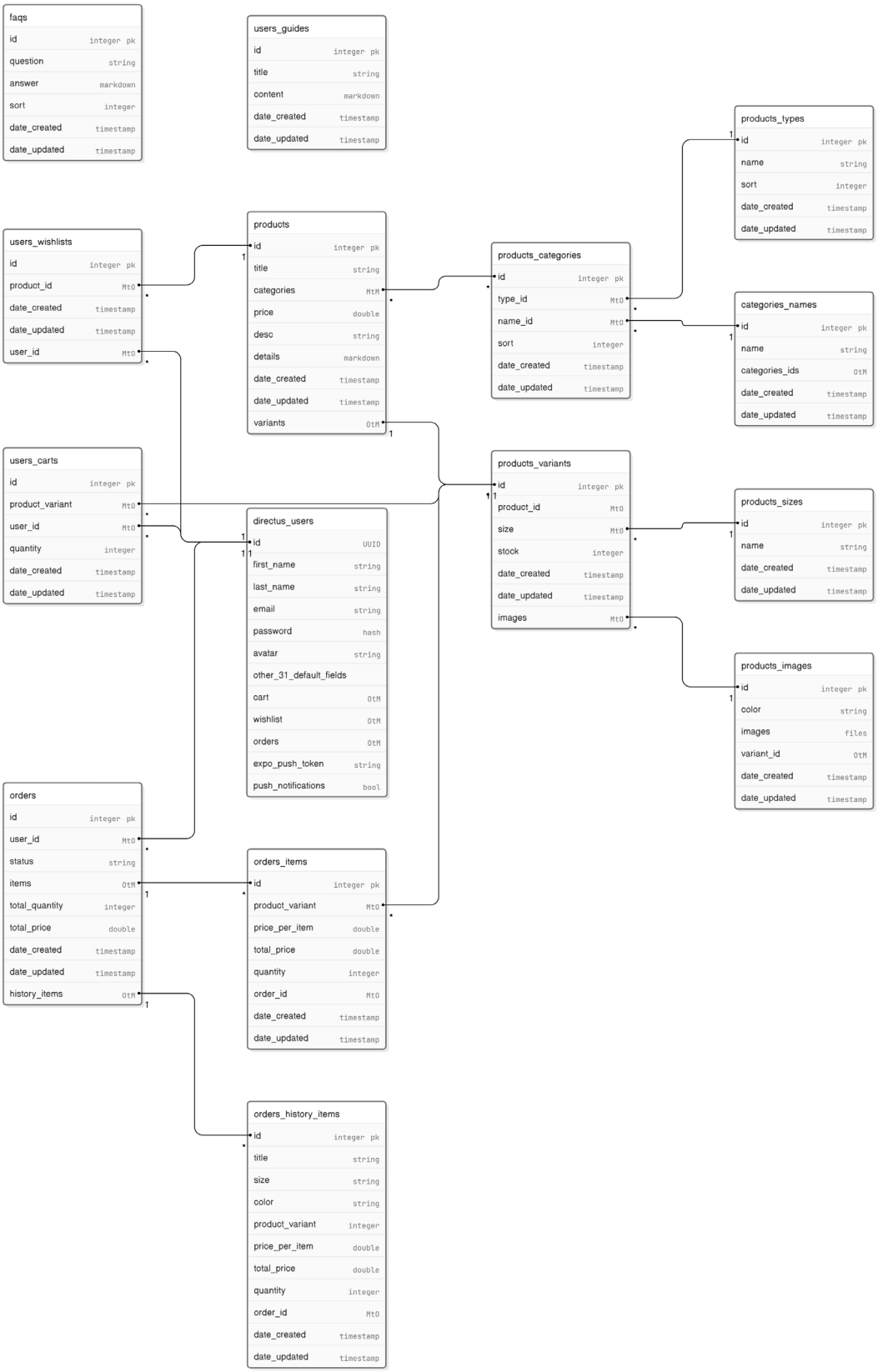


Рисунок 3.4 – Схема бази даних для міні-маркету одягу

3.3 Розробка запитів до бази даних

У процесі розробки мобільного застосунку для електронної комерції, де ключовий функціонал побудовано на основі взаємодії з базою даних, постала потреба у впровадженні ефективного та масштабованого механізму цієї взаємодії. Для досягнення цієї мети використано Directus SDK, який надав зручний інструментарій у вигляді готових методів та функцій для взаємодії з API, яке автоматично генерує Directus.

Паралельно з цим важливу роль у забезпечені продуктивної роботи з даними відіграла бібліотека React Query. Її використання дозволило вирішити такі ключові завдання: управління станом запитів, кешування отриманих даних, структуризацію процесу звернення до API функцій, нескінченних запитів, взаємодії з асинхронними модулями пристрою (камери, сповіщень, геолокації, локальних сховищ тощо).

Для оптимізації такої архітектури створено окремі директорії:

– services – містить функції для взаємодії з API;

– hooks – в більшості складається з конфігурації окремих запитів із використанням React Query, готових до впровадження у компоненти застосунку;

Відповідно на рисунку 3.5 наведено схему, яка ілюструє процес взаємодії застосунку з базою даних на прикладі отримання товарів для відображення в каталозі через API Directus з використанням механізмів, які надає React Query.

Власне приклади функцій на читання та відправлення даних, а також їхнє використання з React Query наведено у додатках Г та Д відповідно. За аналогічним алгоритмом було реалізовано інші Create, Read, Update, Delete (CRUD) запити, які дозволили у структурованому вигляді впровадити велику кількість функціоналу, починаючи з керування кошиком та списком бажань, фільтрації товарів, закінчуючи отриманням доступу до довідної інформації, замовлень тощо.

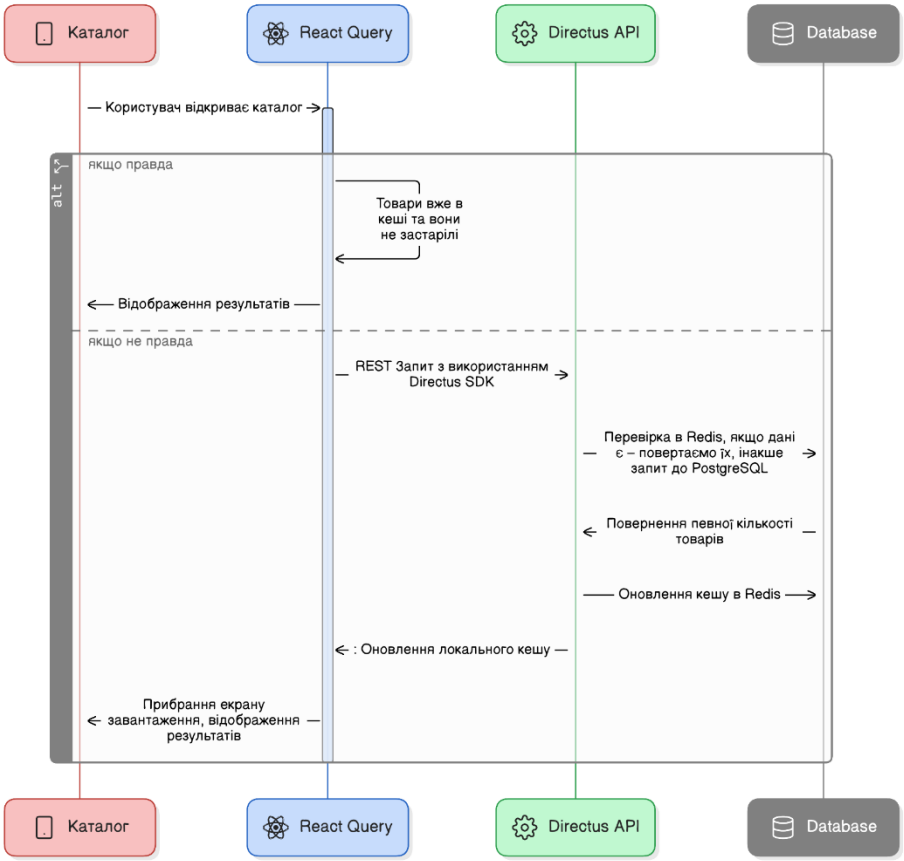


Рисунок 3.5 – Взаємодія застосунку з Directus: отримання товарів для відображення в каталозі

3.4 Особливості взаємодії з Directus

Після визначення сутностей та відношень між ними повністю розкривається функціональний потенціал Directus, зокрема його можливості для ефективного управління даними, медіафайлами та користувачами. Для виконання цих та деяких інших завдань, платформа забезпечує інтуїтивний інтерфейс розбитий за відповідними модулями:

– Content є центром керування інформацією та, у випадку проєкту, товарами й відповідними до них даними. Цей модуль є основним способом взаємодії з визначеними у Data Model сутностями через відповідні компоненти. Сильною стороною цього модулю є можливість гнучкого налаштування відображення та валідації даних;

– Files library репрезентує агреговане сховище усіх медіафайлів системи, які в Content можна зручно використовувати для моделювання даних. Directus підтримує інтеграцію різних хмарних сховищ, таких як Amazon S3, Azure Storage, Google Storage, що робить систему масштабованою та економною, адже база даних зберігає в собі лише посилання на відповідні файли;

– User’s directory забезпечує можливість управління користувачами системи. Модуль реалізує функціонал, схожий до Content, дозволяючи додавати користувачів та редагувати їх дані. В поєднанні з можливістю налаштування ролей та їх відповідних прав доступу, Directus реалізує можливість створення складних сценаріїв управління доступу до системи, адаптованих до будь-якої потреби. У випадку мобільного застосунку реалізовано дві ролі – користувач та адміністратор.

Однак однією з найбільш унікальних і потужних можливостей Directus стало використання так званих Flows, які дозволяють автоматизувати бізнес-процеси системи. Принцип моделювання Flows нагадує побудову блок-схеми, де шляхом розташування та поєднання між собою різноманітних компонентів реалізується ланцюг потрібної бізнес-логіки. Отже, ключовими двома компонентами Flow є тригер – подія або дія, яка ініціює виконання Flow, та відповідно операція, яка може взаємодіяти з базою даних, зі сторонніми сервісами або з даними, отриманими з інших операцій. У рамках міні-маркету одягу реалізовано Flow, який автоматизує зменшення кількості доступного товару після замовлення, далі він архівує успішне замовлення та надсилає повідомлення про його статус користувачу.

Візуалізація відповідних модулів наведена у додатку Е. Загалом поєднання можливостей керування вмістом, автоматизації, керування користувачами та аналітичних можливостей модуля Insights, який дозволяє створювати інформаційні панелі для моніторингу нових замовлень, залишків на складі, прибутку тощо, зробило Directus потужним інструментом для міні-маркету одягу. Це рішення значно оптимізувало процеси управління товарами, замовленнями та системою загалом.

3.5 Особливості розробки мобільного застосунку

React Native є прикладом кросплатформного фреймворку для розробки застосунків. Його головною концепцією є реалізація React парадигми програмування, яка лежить в основі однойменної бібліотеки, що використовується для створення вебзастосунків. У рамках цього підходу розробники створюють компоненти UI, які відповідають різним станам застосунку. React автоматично оновлює ці компоненти відповідно до змін у даних, що з ними пов’язані.

React Native розширює цю архітектуру, пропонуючи уніфіковані будівельні блоки – компоненти, які після компіляції, трансформуються у нативні елементи цільової платформи (наприклад, Android або iOS). Це дозволяє досягти високої продуктивності застосунків, а також забезпечує консистентність інтерфейсу на різних платформах. Приклад компоненти наведено у додатку Ж.

З огляду на ці особливості, стає очевидним, що реалізація застосунку для міні-маркету одягу виключно за допомогою React Native є неможливою. Це пояснюється тим, що невирішеними залишається ряд аспектів, зокрема: навігація, доступ до функціоналу пристрою, забезпечення зручного процесу розробки та тестування. Для розв’язання цих проблем, згідно із документацією React Native, рекомендується використовувати Expo – платформу та набір інструментів для полегшення розробки, тестування та дистрибуції.

Однією зі значних переваг Expo для проєкту став SDK, який надає різноманітні бібліотеки, утиліти та компоненти, що значно спрощує процес розробки та інтеграції функціоналу. Зокрема, для реалізації функцій, що залежали від апаратних можливостей використано щонайменше 12 різноманітних пакетів, наприклад:

– Expo-image-picker – забезпечив інтеграцію з камерою та галереєю пристрою, що дозволило реалізувати функцію завантаження фотографій профілю користувачем;

– Expo-location – дозволив отримувати геолокацію користувача з метою автоматичного заповнення полів адреси;

– Expo-secure-store – використаний для зберігання чутливої інформації;

– Expo-splash-screen – забезпечив налаштування та управління екраном завантаження.

Інтеграція модулів у проєкт відбувалася швидко та ефективно, що стало можливим завдяки добре структурованій документації та інтуїтивно зрозумілому інтерфейсу. Зважаючи на загальну зручність роботи з SDK, доцільно детально зупинитися лише на кількох ключових модулях, які мають принципове значення для реалізації основного функціоналу (див. пункти 3.5.1-3.5.3).

Окрім цього, Expo забезпечив:

– Уніфіковане керування налаштуваннями та конфігурацією модулів застосунку через файл app.json, що дозволило декларативно, без необхідності в зміні нативних файлів редагувати відповідні параметри;

– хмарний сервіс Expo Application Services (EAS), використаний для створення APK-версії застосунку для тестування та дистрибуції, а також Expo Development Build – інструмента, що дозволив розробляти застосунок на фізичному пристрої.

Водночас для розробки використовувалися не лише Expo модулі, а й низка інших бібліотек. Наприклад, бібліотеки для роботи з анімаціями, жестами, SVG-зображеннями, асинхронним сховищем, для відображення markdown інформації та деякі інші. Застосування цих інструментів дозволило ефективно впровадити комплексний функціонал, одночасно зберігаючи високу продуктивність та простоту підтримки застосунку.

3.5.1 Структура навігації в застосунку

Для реалізації навігації в застосунку застосовано Expo Router, в основі якого лежить маршрутизація на основі структури директорій (див. рис. 3.6).

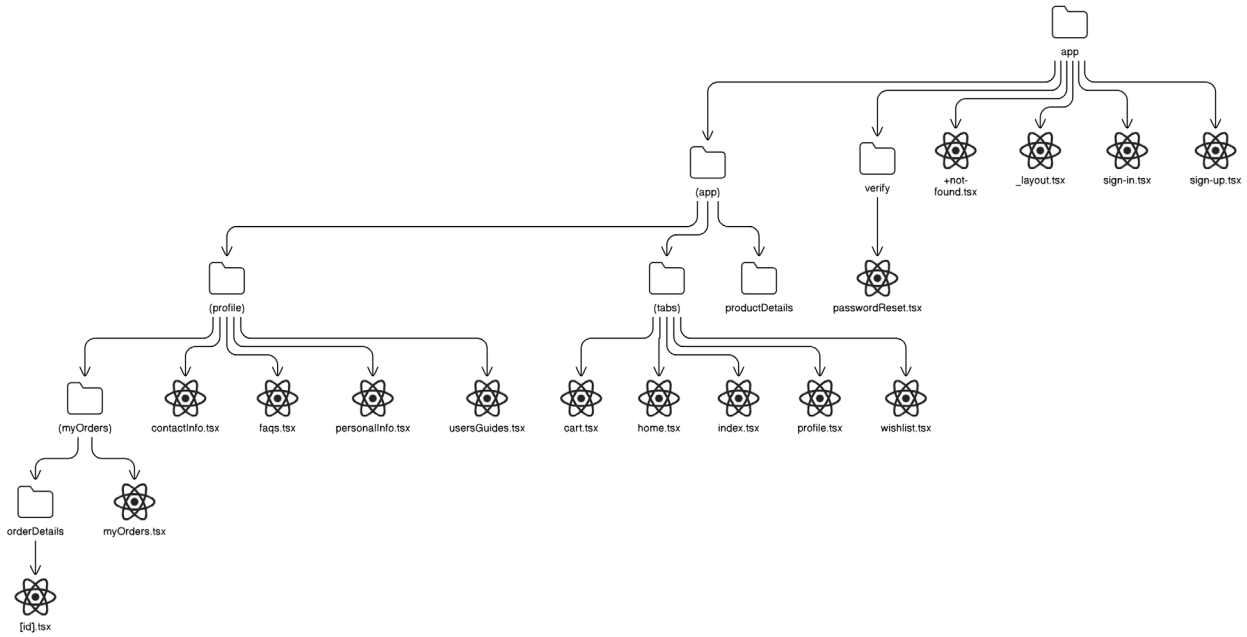


Рисунок 3.6 – Структура навігації (директорій) у застосунку

Отже, окремі tsx файли формують сторінки в застосунку, окрім _layout.tsx, необхідного в кожній директорії (на схемі не відображено) для визначення спільних елементів інтерфейсу (навігаційні панелі, меню тощо) та інших загальних налаштувань сторінок. Директорії відображають вкладені маршрути, окрім тих, що взяті у круглі дужки, що зумовлює відсутність відповідного Uniform Resource Locator (URL) сегменту й необхідно для логічного групування схожих по суті сторінок. Сторінки, взяті у квадратні дужки відображають динамічні маршрути. У рамках цього проєкту, структура маршрутизації поділена на дві основні частини: маршрути для авторизованих користувачів, що знаходяться в директорії app (яка взята у круглі дужки), та маршрути для неавторизованих користувачів, які розміщені в app, що дозволяє керувати доступом користувачів до функцій застосунку.

Загальною особливістю такої системи маршрутизації є її інтуїтивність і зручність у використанні, оскільки ієрархія файлів і директорій чітко відображає структуру навігації в застосунку.

3.5.2 Використання App Links

Важливою частиною функціонування застосунку є наявність облікового запису користувача, оскільки саме з ним пов’язуються основні функціональні можливості, такі як кошик покупок, історія замовлень та список бажань. Реалізація механізмів авторизації та реєстрації виявилась прямолінійним процесом, особливо з використанням Directus SDK, яке значно полегшило інтеграцію із серверною частиною, зокрема в аспекті оновлення токена доступу. Проте, попри відносну простоту впровадження загальних функцій автентифікації, деякі аспекти, зокрема підтвердження реєстрації через електронну пошту, виявились менш очевидними та вимагали більш детального підходу.

Зокрема, постало питання реалізації переадресації користувача до застосунку після відкриття посилання для підтвердження реєстрації. Першим можливим рішенням стало використання механізму DeepLink, що дозволяє безпосередньо відкривати застосунок на певній сторінці. Однак, це рішення мало суттєву проблему: посилання на відкриття застосунку мають специфічну схему, визначену в app.json (наприклад, myshop://), що робить їх непідтримуваними у більшості поштових сервісів, які очікують посилання з протоколом Hypertext Transfer Protocol (HTTP) або Hypertext Transfer Protocol Secure (HTTPS).

Щоб подолати це обмеження, було використано Android App Links, що дозволяють робити необхідну переадресацію й разом з тим підтримують необхідні протоколи. Однак для реалізації цього функціоналу необхідно мати робочий домен з розміщеним файлом assetlinks.json, що містить відповідні налаштування для ідентифікації та переадресації запитів на конкретний застосунок. Враховуючи ці вимоги, проблему вдалося вирішити шляхом використання сервісу Firebase, який надає безплатний домен з підтримкою SSL сертифікатів, а також налаштуванням відповідного файлу, дані для якого отримано з Expo Command Line interface (CLI). Після чого, додавши в app.json

конфігурацію для роботи зі створеним доменом, вдалося реалізувати необхідну поведінку для підтвердження реєстрації та скидання пароля, що зробило процес автентифікації повноцінним, а також відкрило шлях до подальшого розширення застосунку на вебверсію.

3.5.3 Використання push-сповіщень

Сповіщення є ще одним невіддільним компонентом сучасної електронної комерції, адже вони забезпечують оперативну взаємодію з клієнтами та можуть підвищувати рівень довіри та лояльності. У рамках проєкту функцію сповіщень реалізовано у вигляді електронних листів, які надсилаються користувачам у разі зміни стану їхніх замовлень. Цей механізм можна суттєво розширити шляхом інтеграції push-сповіщень, що сприятиме покращенню користувацького досвіду.

Для реалізації push-сповіщень в Expo використовується спеціалізований сервіс – Expo Push Service. Налаштування цього сервісу потребує інтеграції з Firebase Cloud Messaging, який є платформою для передачі повідомлень. Отже, спочатку необхідно отримати секретний ключ API в консолі Firebase. Цей ключ завантажується на сторінці налаштувань проєкту в Expo. Далі потрібно завантажити файл конфігурації google-services.json з Firebase і розмістити його в кореневій директорії застосунку. У файлі app.json додається шлях до цього конфігураційного файлу, що забезпечує правильну роботу сервісу.

Після виконання цих налаштувань застосунок набуває можливості взаємодіяти з Expo Push Service шляхом отримання унікального токена, що ідентифікує пристрій. Після генерації токен доцільно зберігати у базі даних для подальшого використання. Схематично принцип роботи push-сповіщень зображено на рисунку 3.7.

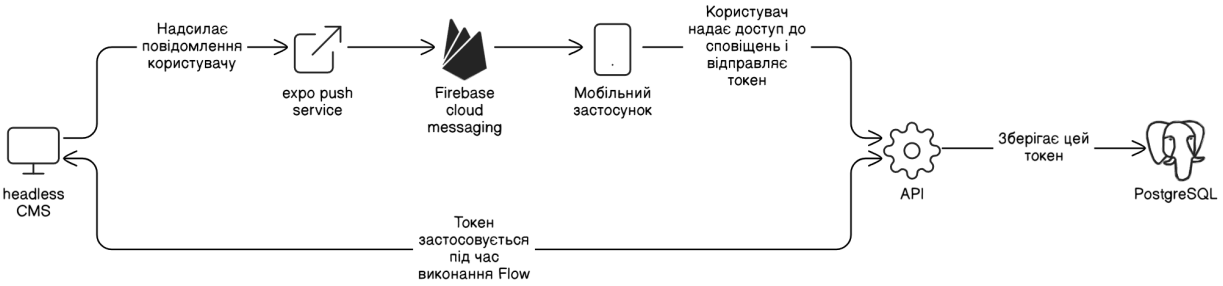


Рисунок 3.7 – Схема роботи push-сповіщень у застосунку

Отже, у рамках проєкту, токен інтегрується у Flow, який через використання вебхуків, може звертатись до Expo Push Service, для надсилання відповідного повідомлення про замовлення потрібному користувачу.

3.6 Оптимізація роботи застосунку

Попри компактний масштаб проєкту, під час розробки застосунку належну увагу було приділено оптимізації його продуктивності.

Першочергового вдосконалення потребував механізм взаємодії користувача з кошиком покупок. Архітектурне рішення щодо зберігання користувацьких даних на стороні сервера, хоч й забезпечило потенціал для масштабування системи та збереження цілісності даних, створило додаткове навантаження на серверну частину. Особливо помітним це могло бути під час інтенсивної взаємодії з сервером, наприклад, при швидкій зміні кількості товарів у кошику, що генерувало надмірну кількість запитів до бази даних, серед яких актуальним залишався лише останній. Для розв’язання цієї проблеми впроваджено механізм оптимістичного оновлення кешованих даних: зміни відображались у локальному кеші миттєво, тоді як до сервера надсилався єдиний агрегований запит із затримкою, що дозволило знизити навантаження на базу даних.

Аналогічний підхід застосовано для оптимізації системи фільтрації товарів. Для неї розроблено гібридний метод, що поєднує локальне та глобальне сховища змінних, а запит до сервера здійснюється лише після остаточного підтвердження обраних критеріїв, що дозволило уникнути надмірних звернень до бази даних, які виникали при використанні стандартного механізму збереження критеріїв через Expo Router.

Окремої уваги потребувало питання оптимального відображення списків, які є ключовими елементами застосунку (каталог товарів, списки замовлень, бажань та кошик покупок). Для цього Expo рекомендує використовувати спеціальний компонент Flashlist, який реалізує механізм повторного використання компонентів при їх появі в області видимості, замість їх повторного створення. Проте в процесі тестування було виявлено, що дана оптимізація конфліктує з функціоналом swipe-to-delete (проведіть пальцем, щоб видалити), у списку бажань та кошику. Конфлікт проявлявся у випадковій активації жестів на компонентах, що з’являлися в області видимості. Це пов’язано з принципом роботи Flashlist. Разом з цим swipe-to-delete проявляв проблеми в анімаціях при поєднанні з компонентами візуального відгуку та створював додаткове навантаження на застосунок при великій кількості елементів. Після аналізу можливих компромісів було прийнято рішення відмовитись від swipe-to-delete на користь кращої продуктивності та стабільності роботи застосунку.

Загалом синтезувавши низку допоміжних методик, включаючи кешування на стороні сервера засобами Redis, клієнтське кешування через React Query, використання інших спеціалізованих компонентів Expo, бібліотеки Reanimated для анімацій, вдалося досягти оптимального балансу між функціональністю та продуктивністю застосунку, вихідний код якого представлено в репозиторії GitHub [25], а його візуальне відображення – у додатку И.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи досягнуто головної мети – розроблено мобільний застосунок для міні-маркету одягу з використанням React Native та системи керування вмістом Directus.

На початку роботи проведено аналіз впливу мобільних застосунків на сферу комерції. Аналіз літератури та досліджень підтвердив, що мобільні застосунки відкрили нові можливості для бізнесу, ставши потужним інструментом, який перевершив інші форми комерції завдяки підвищеній доступності, інтуїтивній простоті використання та можливості швидкого пошуку. Паралельно з цим вдалося встановити, що успішність застосунку залежить від свідомого врахування вимог безпеки та цільової аудиторії. Також для високовартісних товарів застосунки можуть виявитись менш актуальними, але для роздрібних, як-от одяг, вони є ефективним рішенням.

Аналіз актуального стану мобільної розробки та порівняння наявних підходів обґрунтував вибір React Native як оптимального фреймворку для проєкту. Дослідження технологій, використаних у наявних на ринку застосунках для пошуку одягу, показало, що React Native, хоч і поступається нативній розробці, забезпечує необхідний баланс продуктивності, простоти розробки, можливостей масштабування та сумісності з SDK для обраної headless CMS. Паралельно з цим, аналіз наявних застосунків дозволив виділити спільні функціональні компоненти, які було прийнято за базові вимоги до застосунку та враховано для точнішого визначення критеріїв підбору CMS.

Порівняльний аналіз систем керування вмістом дозволив обрати headless архітектуру як найбільш відповідну для проєкту з високими вимогами до персоналізації, адаптації до технологічних змін та потенційним розширенням на інші платформи. Використання синтезованого методу багатокритеріального аналізу (AHP та TOPSIS) дозволило систематизувати та оптимізувати процес вибору конкретної headless CMS серед наявних альтернатив. У результаті було

обрано систему Directus, яка продемонструвала найбільш збалансоване поєднання таких переваг, як стабільність, висока продуктивність, гнучкість і відсутність суттєвих обмежень, у порівнянні з конкурентами.

Практична реалізація застосунку підтвердила можливість успішної інтеграції headless CMS з мобільним застосунком, незважаючи на обмежену кількість документованих прикладів. За умови інтеграції системи оплати та доставки, застосунок може стати частиною бізнес-процесів невеликого бізнесу.

Перспективними напрямками подальших досліджень можна вважати аналіз можливостей впровадження новітніх систем, таких як Payload CMS та MedusaJS, яким на сьогодні бракує стабільності та швидкодії, через їх активний процес розробки.

ПЕРЕЛІК ПОСИЛАНЬ

1. Bianchi T. Global mobile traffic share by region 2024. *Statista*. URL: https://www.statista.com/statistics/306528/share-of-mobile-internet-traffic-in-global-regions/ (date of access: 17.01.2025);

2. Taylor P. Forecast number of mobile users worldwide 2020-2025. *Statista*.URL: https://www.statista.com/statistics/218984/number-of-global-mobile-users-since-2010/#:~:text=In%202021,%20the%20number%20of,projected%20to%20reach%207.49%20billion (date of access: 17.01.2025);

3. Dubel M. Development trends in the digital distribution of mobile applications. *European journal of economics and management*. 2024. Vol. 10, no. 4. P. 5–16. DOI: 10.46340/eujem.2024.10.4.1 (date of access: 18.01.2025).

4. Global: m-commerce revenue and share in e-commerce 2017-2028. Statista. URL: https://www.statista.com/statistics/1449284/retail-mobile-commerce-revenue-worldwide/?__sso_cookie_checker=failed (date of access: 22.01.2025).

5. Tang A. K. Y. A systematic literature review and analysis on mobile apps in m-commerce: implications for future research. *Electronic commerce research and applications*. 2019. Vol. 37. P. 100885. DOI: 10.1016/j.elerap.2019.100885 (date of access: 18.01.2025).

6. Khrais L. T. Investigation use of social media, mobile apps, and the impacts of enlarging e-commerece. *2020 6th international conference on advanced computing and communication systems (ICACCS)*, Coimbatore, India, 6–7 March 2020. 2020. DOI: 10.1109/icaccs48705.2020.9074341 (date of access: 18.01.2025).

7. Curry D. Ecommerce app benchmarks. Business of Apps. URL: https://www.businessofapps.com/data/ecommerce-app-benchmarks/ (date of access: 17.01.2025).

8. Shwetha A. N., Sumathi R., Prabodh C. P. A survey on mobile application development models. *Mobile application development: practice and experience*.

Singapore, 2023. P. 1–9. DOI: 10.1007/978-981-19-6893-8_1 (date of access: 18.01.2025).

9. Zarichuk O. Comparative analysis of frameworks for mobile application development: native, hybrid, or cross-platform solutions. *Вісник Черкаського державного технологічного університету*. 2023. Vol. 28, no. 4. P. 19–27. DOI: 10.62660/2306-4412.4.2023.19-27 (date of access: 18.01.2025).

10. Ottervig V. The essential history of CMS. Enonic. URL: https://www.enonic.com/blog/the-history-of-cms--what-has-happened (date of access: 17.01.2025).

11. *Semantic Scholar | AI-Powered Research Tool*. URL: https:// www.semanticscholar.org/ (date of access: 06.02.2025).

12. *Academia.edu - Find Research Papers, Topics, Researchers*. URL: https://www.academia.edu/ (date of access: 06.02.2025).

13. *ResearchGate | Find and share research*. URL: https:/ /www.researchgate.net/ (date of access: 07.02.2025).

14. *Google Scholar*. URL: https://scholar.google.com/ (date of access: 06.02.2025).

15. *Android Apps on Google Play*. URL: https://play.google.com/ store/games?hl=en (date of access: 06.02.2025).

16. Banwarie A. Choosing the right framework for Android development: which mobile development frameworks are chosen and why? : Master Thesis. Amsterdam, Netherlands, 2022. 43 p. URL: https://ictinstitute.nl/wp-content/uploads/2022/08/Master_Thesis_A.Banwarie_FINAL.pdf (date of access: 18.01.2025).

17. *Google Trends*. URL: https://trends.google.com/trends/ (date of access: 07.02.2025).

18. *GitHub*. URL: https://github.com/ (date of access: 07.02.2025).

19. *Stack Overflow Trends*. URL: https://trends.stackoverflow.co/ (date of access: 07.02.2025).

20. *APKMirror | Free and safe Android APK downloads*. URL: https://www.apkmirror.com/ (date of access: 28.03.2025).

21. Kumar R. A comprehensive review of MCDM methods, applications, and emerging trends. *Decision making advances*. 2024. Vol. 3, no. 1. P. 185–199. DOI: 10.31181/dma31202569 (date of access: 18.01.2025).

22. Gavade R. K. Multi-Criteria Decision Making: an overview of different selection problems and methods. *International journal of computer science and information technologies*. 2014. Vol. 5, no. 4. P. 5643–5646. URL: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=dd1549a5ee01f68fb6c92f6dba7f4134082aaf8c (date of access: 18.01.2025).

23. Misra S. K., Ray A. Integrated AHP-TOPSIS model for software selection under multi-criteria perspective. *Driving the economy through innovation and entrepreneurship*. India, 2013. P. 879–890. DOI: 10.1007/978-81-322-0746-7_72 (date of access: 18.01.2025).

24. Saaty T. L., Vargas L. G. Models, Methods, Concepts & Applications of the Analytic Hierarchy Process. Boston, MA : Springer US, 2012. 345 p. DOI: 10.1007/978-1-4614-3597-6 (date of access: 05.02.2025).

25. Zhelobetskyi A. MyShop. GitHub. URL: https://github.com/Arsenii-Zhelobetskyi/myShop (date of access: 18.01.2025).

ДОДАТОК А

Реалізація алгоритму AHP-TOPSIS для вибору headless CMS

```
import numpy as np
from fractions import Fraction
from prettytable import PrettyTable
class AHP_TOPSIS_Method:
    def __init__(self, goal, criteria_names, alt, criteria_comp,
alt_scores, benefits_criteria):
        self.goal = goal
        self.criteria_names = criteria_names
        self.alt = alt
        self.benefits_criteria = benefits_criteria  # критерії,
які потрібно є позитивними
        self.criteria_comp = criteria_comp  # порівняння критеріїв
        self.alt_scores = alt_scores  # Оцінки альтернатив
        self.m = len(alt_scores)  # Кількість альтернатив
        self.n = len(alt_scores[0])  # Кількість критеріїв

    # < AHP part
    def fill_comp(self, C):
        for i in range(self.n):
            for j in range(self.n):
                if (i < j):
                    C[j][i] = 1 / C[i][j]

    def norm_criteria(self, C):
        N = np.zeros((self.n, self.n))
        for i in range(self.n):
            sum = 0
            for j in range(self.n):
                sum += C[j][i]
            for j in range(self.n):
                N[j][i] = C[j][i] / sum

        return N

    def calc_weight_criteria(self, N):
        sum = 0
        W = np.zeros(self.n)
        for i in range(self.n):
            for j in range(self.n):
                sum += N[i][j]
            W[i] = sum / self.n
            sum = 0
        return W
```

```
    def check_inconsistency(self, C, W):
        consistency_vector = C @ W
        for i in range(len(consistency_vector)):
            consistency_vector[i] = consistency_vector[i] / W[i]
        lambda_max = np.sum(consistency_vector) /
len(consistency_vector)
        CI = (lambda_max - self.n) / (self.n - 1)
        RI_values = [0, 0, 0.58, 0.9, 1.12, 1.24, 1.32, 1.41,
1.45, 1.49, 1.51, 1.48, 1.56, 1.57, 1.58]
        RI = RI_values[self.n - 1]
        CR = CI / RI
        if CR < 0.1:
            print("Критерій узгодженості прийнятний")
        else:
            print("Критерій узгодженості не прийнятний")

    # AHP part >

    # < TOPSIS part
    def norm_alts(self, X):
        R = np.zeros((self.m, self.n))
        for i in range(self.n):
            sum = 0
            for j in range(self.m):
                sum += X[j][i] ** 2
            sum = sum ** 0.5
            for j in range(self.m):
                R[j][i] = X[j][i] / sum
        return R

    def calc_weight_norm_alts(self, R, W):
        V = np.zeros((self.m, self.n))
        for i in range(self.m):
            for j in range(self.n):
                V[i][j] = R[i][j] * W[j]
        return V

    def calc_ideal_sol(self, V, benefits_criteria):
        I_pos = np.zeros(self.n)
        I_neg = np.zeros(self.n)
        for i in range(self.n):
            if benefits_criteria[i]:
                I_pos[i] = np.max(V[:, i])
                I_neg[i] = np.min(V[:, i])
            else:
                I_pos[i] = np.min(V[:, i])
                I_neg[i] = np.max(V[:, i])
        return I_pos, I_neg

    def calc_sep(self, V, I_pos, I_neg):
        S_pos = np.zeros(self.m)
        S_neg = np.zeros(self.m)
        for i in range(self.m):
```

```
            sum_ideal = 0
            sum_anti_ideal = 0
            for j in range(self.n):
                sum_ideal += (V[i][j] - I_pos[j]) ** 2
                sum_anti_ideal += (V[i][j] - I_neg[j]) ** 2
            S_pos[i] = sum_ideal ** 0.5
            S_neg[i] = sum_anti_ideal ** 0.5
        return S_pos, S_neg

    def calc_rel_closeness(self, S_pos, S_neg):
        rel_closeness = np.zeros(self.m)
        for i in range(self.m):
            rel_closeness[i] = S_neg[i] / (S_pos[i] + S_neg[i])
        for i, alt in enumerate(self.alt):
            print(f"{alt} = {rel_closeness[i]:.4f}")
        print("Найкраща альтернатива: ",
self.alt[np.argmax(rel_closeness)])

    # TOPSIS part >

    def print_table(self, headers, sidebars, criteria_comp):
        fractions = [[Fraction(value).limit_denominator() for
value in row] for row in criteria_comp]
        table = PrettyTable(max_table_width=300)
        table.field_names = [""] + headers
        for i, row in enumerate(fractions):
            table.add_row([sidebars[i]] + [str(f) for f in row])
        print(table)

    def calculate(self):
        # AHP part
        # self.fill_comp(self.criteria_comp)
        print("Пріоритети критеріїв:")
        self.print_table(self.criteria_names, self.criteria_names,
self.criteria_comp)
        print("Результати порівняння критеріїв:")
        self.print_table(self.criteria_names, self.alt,
self.alt_scores)
        norm_criteria = self.norm_criteria(self.criteria_comp)
        criteria_weights =
self.calc_weight_criteria(norm_criteria)
        self.check_inconsistency(self.criteria_comp,
criteria_weights)

        # TOPSIS part
        norm_alts = self.norm_alts(self.alt_scores)
        weighted_norm_alts = self.calc_weight_norm_alts(norm_alts,
criteria_weights)
        ideal_sol, anti_ideal_sol =
self.calc_ideal_sol(weighted_norm_alts, self.benefits_criteria)
        S_ideal, S_anti_ideal = self.calc_sep(weighted_norm_alts,
ideal_sol, anti_ideal_sol)
        print("Результати:")
```

```
        self.calc_rel_closeness(S_ideal, S_anti_ideal)


def myData():
    goal = "Decide on software"
    criteria_names = [
        'Стабільність', 'Активність спільноти', 'Тривалість
існування',"Управління зв\'язками","Можливості
розгортання","Бізнес-логіка","Інтеграція з комерцією","Керування
користувачами","Керування доступом до
API","Документація","Простота","Швидкодія","Вартість","Відсутність
обмежень"
    ]
    benefits_criteria = [True, True, True, True, True, True, True,
True, True, True, True, True, True, True]

    criteria_comp = [
        [1, 3, 9, 3, 3, 3, 7, 3, 3, 5, 3, 3, 1, 1],
        [1 / 3, 1, 5, 1 / 9, 1 / 9, 1 / 9, 1 / 5, 1 / 9, 1 / 9, 1
/ 5, 1 / 5, 1 / 3, 1 / 9, 1 / 9],
        [1 / 9, 1 / 5, 1, 1 / 9, 1 / 9, 1 / 9, 1 / 5, 1 / 9, 1 /
9, 1 / 5, 1 / 5, 1 / 5, 1 / 9, 1 / 9],
        [1 / 3, 9, 9, 1, 1, 1, 3, 1, 1, 3, 5, 3, 1 / 9, 1 / 9],
        [1 / 3, 9, 9, 1, 1, 1, 1 / 3, 1 / 5, 1 / 5, 1 / 3, 1 / 3,
3, 1 / 9, 1 / 9],
        [1 / 3, 9, 9, 1, 1, 1, 9, 1, 1, 3, 5, 3, 1 / 3, 1 / 7],
        [1 / 7, 5, 5, 1 / 3, 3, 1 / 9, 1, 1 / 3, 1 / 3, 1 / 5, 3,
1, 1 / 7, 1 / 7],
        [1 / 3, 9, 9, 1, 5, 1, 3, 1, 1, 3, 3, 3, 1 / 7, 1 / 7],
        [1 / 3, 9, 9, 1, 5, 1, 3, 1, 1, 5, 5, 3, 1 / 5, 1 / 5],
        [1 / 5, 5, 5, 1 / 3, 3, 1 / 3, 5, 1 / 3, 1 / 5, 1, 1, 1, 1
/ 7, 1 / 5],
        [1 / 3, 5, 5, 1 / 5, 3, 1 / 5, 1 / 3, 1 / 3, 1 / 5, 1, 1,
1, 1 / 5, 1 / 5],
        [1 / 3, 3, 5, 1 / 3, 1 / 3, 1 / 3, 1, 1 / 3, 1 / 3, 1, 1,
1, 1 / 3, 1 / 3],
        [1, 9, 9, 9, 9, 3, 7, 7, 5, 7, 5, 3, 1, 1],
        [1, 9, 9, 9, 9, 7, 7, 7, 5, 5, 5, 3, 1, 1]

    ]
    alt = [
        "PayloadCMS","Directus", "Strapi","Contetnfull","Sanity",
"MedusaJS","Vendure"
    ]
    alt_scores = [
        [1, 3, 3, 7, 9, 7, 5, 5, 5, 3, 3, 1, 5, 9],
        [9, 7, 9, 9, 9, 9, 3, 9, 9, 5, 7, 9, 9, 9],
        [3, 9, 5, 3, 9, 5, 3, 5, 5, 7, 5, 3, 9, 7],
        [9, 3, 7, 1, 1, 1, 1, 1, 1, 1, 5, 9, 1, 1],
        [9, 5, 5, 3, 3, 3, 3, 1, 1, 5, 5, 9, 7, 3],
        [1, 3, 1, 9, 7, 7, 9, 5, 5, 1, 3, 7, 5, 9],
        [9, 1, 3, 7, 7, 5, 9, 3, 3, 1, 1, 7, 1, 7]
```

```
    ]
    method = AHP_TOPSIS_Method(goal, criteria_names, alt,
criteria_comp, alt_scores, benefits_criteria)
    method.calculate()


if __name__ == '__main__':
    myData()
```

ДОДАТОК Б

Застосування алгоритму AHP-TOPSIS для вибору headless CMS

Пріоритети критеріїв:

	Стабільніст	Активність спільнот	Тривалість існуванн	Управління зв'язкам	Можливості розгортанн	Бізнес-логік	Інтеграція з комерціє	Керування користувачам	Керування доступом до AP	Документаці	Простот	Швидкоді	Вартіст	Відсутність обмежен
Стабільність	1	3	9	3	3	3	7	3	3	5	3	3	1	1
Активність спільноти	1/3	1	5	1/9	1/9	1/9	1/5	1/9	1/9	1/5	1/5	1/3	1/9	1/9
Тривалість існування	1/9	1/5	1	1/9	1/9	1/9	1/5	1/9	1/9	1/5	1/5	1/5	1/9	1/9
Управління зв'язками	1/3	9	9	1	1	1	3	1	1	3	5	3	1/9	1/9
Можливості розгортання	1/3	9	9	1	1	1	1/3	1/5	1/5	1/3	1/3	3	1/9	1/9
Бізнес-логіка	1/3	9	9	1	1	1	9	1	1	3	5	3	1/3	1/7
Інтеграція з комерцією	1/7	5	5	1/3	3	1/9	1	1/3	1/3	1/5	3	1	1/7	1/7
Керування користувачами	1/3	9	9	1	5	1	3	1	1	3	3	3	1/7	1/7
Керування доступом до API	1/3	9	9	1	5	1	3	1	1	5	5	3	1/5	1/5
Документація	1/5	5	5	1/3	3	1/3	5	1/3	1/5	1	1	1	1/7	1/5
Простота	1/3	5	5	1/5	3	1/5	1/3	1/3	1/5	1	1	1	1/5	1/5
Швидкодія	1/3	3	5	1/3	1/3	1/3	1	1/3	1/3	1	1	1	1/3	1/3
Вартість	1	9	9	9	9	3	7	7	5	7	5	3	1	1
Відсутність обмежень	1	9	9	9	9	7	7	7	5	5	5	3	1	1

Результати порівняння критеріїв:

	Стабільність	Активність спільноти	Тривалість існування	Управління зв'язками	Можливості розгортання	Бізнес-логіка	Інтеграція з комерцією	Керування користувачами	Керування доступом до API	Документація	Простота	Швидкодія	Вартість	Відсутність обмежень
PayloadCMS	1	3	3	7	9	7	5	5	5	3	3	1	5	9
Directus	9	7	9	9	9	9	3	9	9	5	7	9	9	9
Strapi	3	9	5	3	9	5	3	5	5	7	5	3	9	7
Contetnfull	9	3	7	1	1	1	1	1	1	1	5	9	1	1
Sanity	9	5	5	3	3	3	3	1	1	5	5	9	7	3
MedusaJS	1	3	1	9	7	7	9	5	5	1	3	7	5	9
Vendure	9	1	3	7	7	5	9	3	3	1	1	7	1	7

Критерій узгодженості не прийнятний
Результати:
PayloadCMS = 0.5665
Directus = 0.9144
Strapi = 0.6485
Contetnfull = 0.2791
Sanity = 0.4838
MedusaJS = 0.5726
Vendure = 0.4577
Найкраща альтернатива: Directus

ДОДАТОК В

Файл конфігурації headless CMS та суміжних сервісів

```
services:
  db:
    image: postgis/postgis:16-3.5
    restart: always
    environment:
      POSTGRES_USER: ${POSTGRES_USER}
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
      POSTGRES_DB: ${POSTGRES_DB}
    ports: #comment this for production
      - "5432:5432" #comment this for production
    volumes:
      - ./postgres:/var/lib/postgresql/data
  cache:
    image: redis:latest
  directus:
    image: directus/directus:latest
    ports: #comment this for production
      - 8055:8055 #comment this for production
    volumes:
      - ./directus/database:/directus/database
      - ./directus/uploads:/directus/uploads
      - ./directus/extensions:/directus/extensions
      - ./directus/templates:/directus/templates
    depends_on:
      - cache
      - db
    environment:
      SECRET: ${SECRET_PHRASE}
      ADMIN_EMAIL: ${ADMIN_EMAIL}
      ADMIN_PASSWORD: ${ADMIN_PASSWORD}

      DB_CLIENT: ${DB_CLIENT}
      DB_HOST: ${POSTGRES_HOST}
      DB_PORT: ${POSTGRES_PORT}
      DB_DATABASE: ${POSTGRES_DB}
      DB_USER: ${POSTGRES_USER}
      DB_PASSWORD: ${POSTGRES_PASSWORD}

      CACHE_ENABLED: ${CACHE_ENABLED}
      CACHE_AUTO_PURGE: ${CACHE_AUTO_PURGE}
      CACHE_STORE: ${CACHE_STORE}
      REDIS: ${REDIS_URL}

      MAX_RELATIONAL_DEPTH: 100
```

```
      USER_REGISTER_URL_ALLOW_LIST :
${USER_REGISTER_URL_ALLOW_LIST}
      PASSWORD_RESET_URL_ALLOW_LIST:
${PASSWORD_RESET_URL_ALLOW_LIST}

      EMAIL_TRANSPORT: ${EMAIL_TRANSPORT}

      EMAIL_FROM: ${EMAIL_FROM}
      EMAIL_SMTP_HOST: ${EMAIL_SMTP_HOST}
      EMAIL_SMTP_PORT: ${EMAIL_SMTP_PORT}
      EMAIL_SMTP_USER: ${EMAIL_SMTP_USER}
      EMAIL_SMTP_PASSWORD: ${EMAIL_SMTP_PASSWORD}
  # nginx config
  # nginx:
  # image: nginx:latest
  # depends_on:
  #   - directus
  # ports:
  #   - "80:80"
  #   - "443:443"
  # volumes:
  #   - ./nginx/nginx.conf:/etc/nginx/conf.d/default.conf
  #   -
./nginx/cloudflare_origin.pem:/etc/nginx/cloudflare_origin.pem:ro
  #   -
./nginx/cloudflare_origin.key:/etc/nginx/cloudflare_origin.key:ro
  # restart: always

#sudo docker compose -f docker-compose-dev.yml up
#sudo docker system prune --all -volumes
```

ДОДАТОК Г

Приклади API функцій на читання та відправку даних

```
import { Product } from '~/types/product';
import client from '~/lib/directus';
import { readItems, readItem } from '@directus/sdk';
import { isDirectusError } from '~/lib/utils';
import { Products } from '~/types/products';

export async function getProducts( pageNumber: number = 1,
type_id: string | undefined, categoriesNames_ids: string[] |
undefined, sortBy: string | undefined, priceRange: number[] |
undefined,
    searchQuery: string | '',
  ) {
    try {
      const products = await client.request(
        readItems('products', {
          fields: [ 'id', 'price', 'title', 'desc',
'variants.images.images.directus_files_id',],
          limit: 6,
          page: pageNumber,
          search: searchQuery,
          filter: {
            _and: [
              {
                categories: {
                  products_categories_id: {
                    name_id: { _in: categoriesNames_ids },
                  },
                },
              },
              {
                categories: {
                  products_categories_id: {
                    type_id: { _eq: type_id },
                  },
                },
              },
              {
                price: {
                  _between: priceRange,
                },
              },
            ],
          },
          sort: [sortBy || ''],
```

```
          deep: {
            variants: {
              images: {
                images: {
                  _limit: 1,
                },
              },
            },},})   );

      const flatProducts: Products = products
        ? products.map((product) => ({
            id: product.id,
            title: product.title,
            price: parseFloat(product.price),
            desc: product.desc,
            image:
              product.variants[0]?.images.images[0]?.directus_file
s_id || null,
          }))
        : [];

      return { products: flatProducts, nextPage: pageNumber };
    } catch (error: any) {
      if (isDirectusError(error)) {
        throw new Error(error.errors[0].message);
      }
      throw new Error(error?.message || 'An unknown error
occurred');
    }
  }


import client from '~/lib/directus';
import * as SecureStore from 'expo-secure-store';
import { createItem, deleteItems, readMe } from '@directus/sdk';
import { isDirectusError } from '~/lib/utils';
import { Orders } from '~/types/orders';
import { OrdersHistoryItem } from '~/types/types';
export async function createOrder({
  items,
  total_price,
  total_quantity,
  deleteFromCart_ids,
}: {
  items: {
    product_variant: number;
    price_per_item: number;
    total_price: number;
    quantity: number;
  }[];
  total_price: number;
  total_quantity: number;
  deleteFromCart_ids?: number[];
```

```
}) {
  try {
    const user_id = await SecureStore.getItemAsync('user_id');
    const result = await client.request(
      createItem('orders', {
        user_id,
        items,
        total_price,
        total_quantity,
      }),
    );
    if (deleteFromCart_ids) {
      await client.request(deleteItems('users_carts',
deleteFromCart_ids));
    }
    return result;
  } catch (error: any) {
    if (isDirectusError(error)) {
      throw new Error(error.errors[0].message);
    }
    throw new Error(error?.message || 'An unknown error
occurred');
  }
}
```

ДОДАТОК Д

Приклади використання API функцій за допомогою React Query

```
import { useInfiniteQuery } from '@tanstack/react-query';
import { getProducts } from '~/services/apiProducts';
import filterStore from '~/store/filterStore';
export function useProducts() {
  const type_id = filterStore((state) => state.type_id);
  const sortBy= filterStore((state) => state.sortBy);
  const categoriesNames_ids = filterStore((state) =>
state.categoriesNames_ids);
  const minPrice =filterStore((state) => state.minPrice);
  const maxPrice=filterStore((state) => state.maxPrice);
  const searchQuery=filterStore((state) => state.searchQuery);
  const priceRange= minPrice !==undefined && maxPrice!==undefined
? [minPrice,maxPrice] : undefined;
  const { data, isPending,isError, refetch, isFetching,
hasNextPage, fetchNextPage } =
    useInfiniteQuery({
      queryKey: [ 'products', `category-${categoriesNames_ids}`,
`type-${type_id}`, `sortBy-${sortBy}`, `priceRange-${priceRange}`,
`searchQuery-${searchQuery}`,],
      queryFn: ({ pageParam = 1 }) =>
        getProducts(pageParam, type_id,categoriesNames_ids,
sortBy, priceRange,searchQuery),
      initialPageParam: 1,
      getNextPageParam: (lastPage) => {
        if (lastPage.products.length === 0) return undefined;
        return lastPage.nextPage + 1;},
      retry:3,});
  return {
    data, isPending, isError, refetch, isFetching, hasNextPage,
fetchNextPage,};
}

import { useMutation, useQueryClient } from '@tanstack/react-
query';
import { toast } from 'sonner-native';
import { createOrder as createOrderApi } from
'~/services/apiOrders';
export function useCreateOrder() {
  const queryClient = useQueryClient();
  const { mutate: createOrder, isPending: isCreateOrderPending } =
useMutation({
    mutationFn: createOrderApi,
    onSuccess: () => {
      toast.success('Order created successfully', {
```

```
        description: 'Go check my orders',
        duration: 6000,
        closeButton: true,
        richColors: true,});
      queryClient.invalidateQueries({ queryKey: ['cart'] });
    },
    onError: (error) => {
      toast.error('Some problem has happened !', {
        description: error.message,
        duration: 6000,
        closeButton: true,
        richColors: true,});},});
return { createOrder, isCreateOrderPending };}
```

ДОДАТОК Е

Інтерфейс і модулі Directus

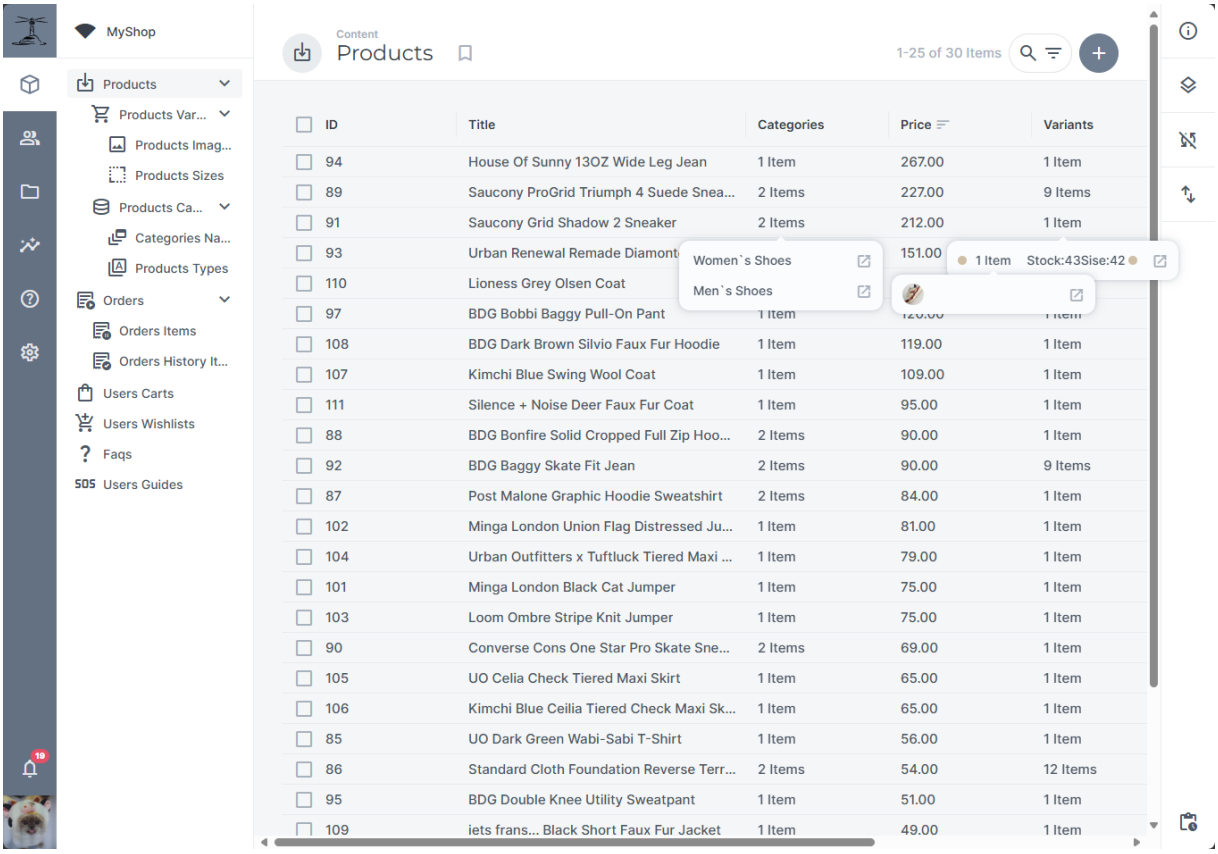


Рисунок Е.1 – Модуль Content (загальний вигляд для керування сутностями)

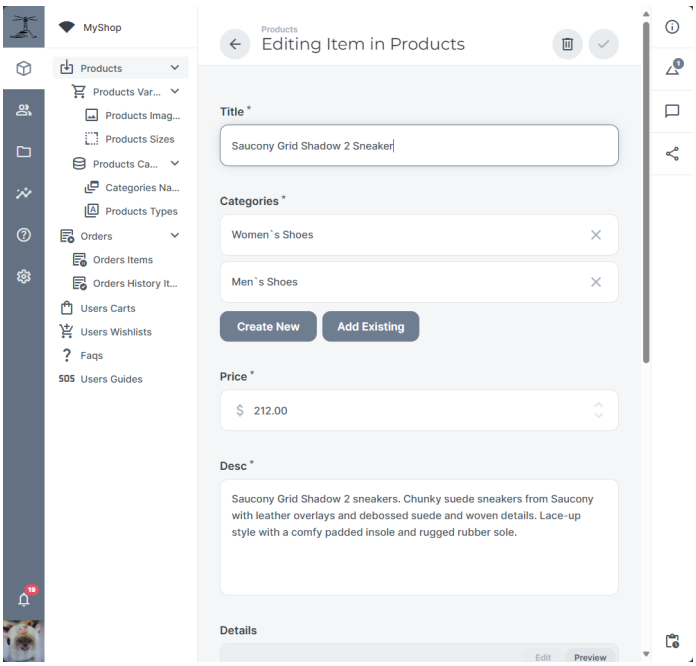


Рисунок Е.2 – Модуль Content (вигляд під час редагування сутності)

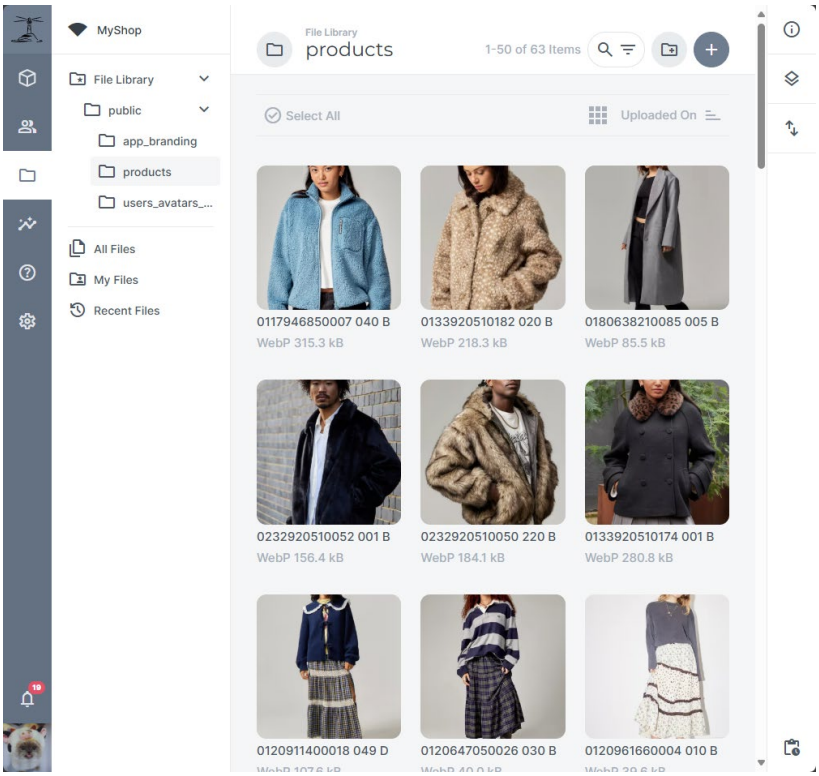


Рисунок Е.3 – Модуль Files Library для керування медіафайлами

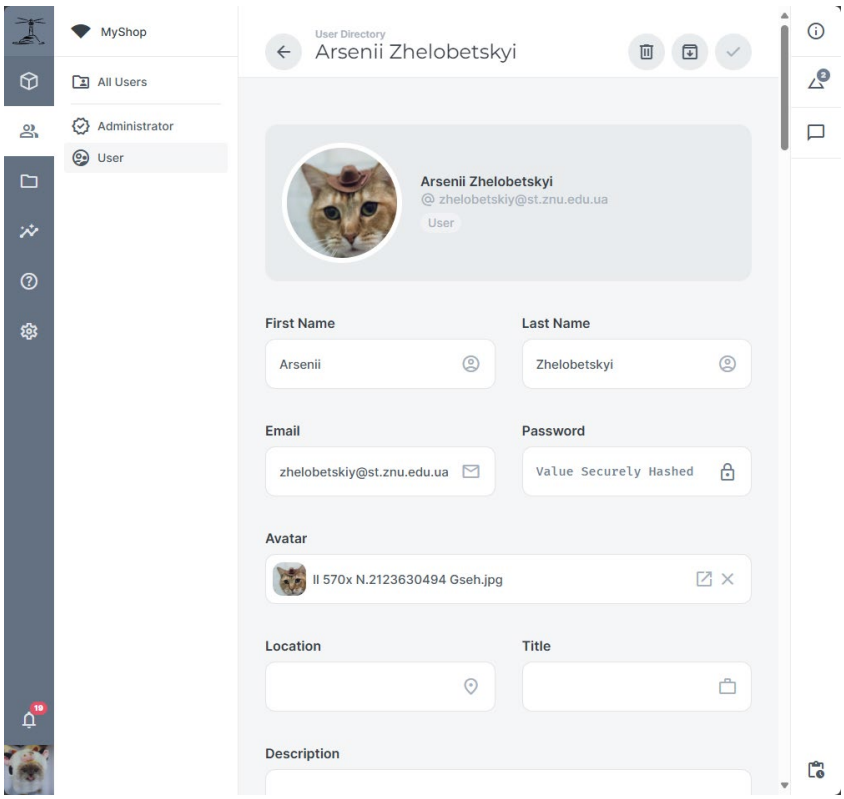


Рисунок Е.4 – Модуль User’s directory для керування користувачами

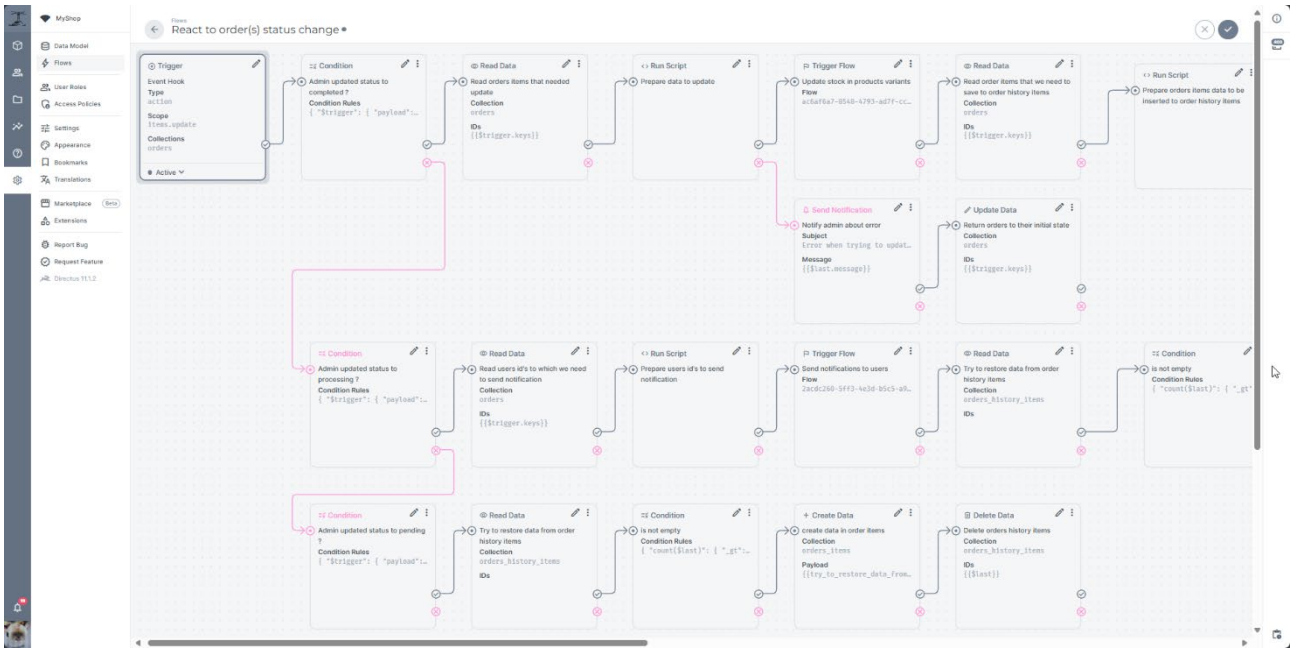


Рисунок Е.5 – Flow для автоматизації управління замовленнями в модулі Flows

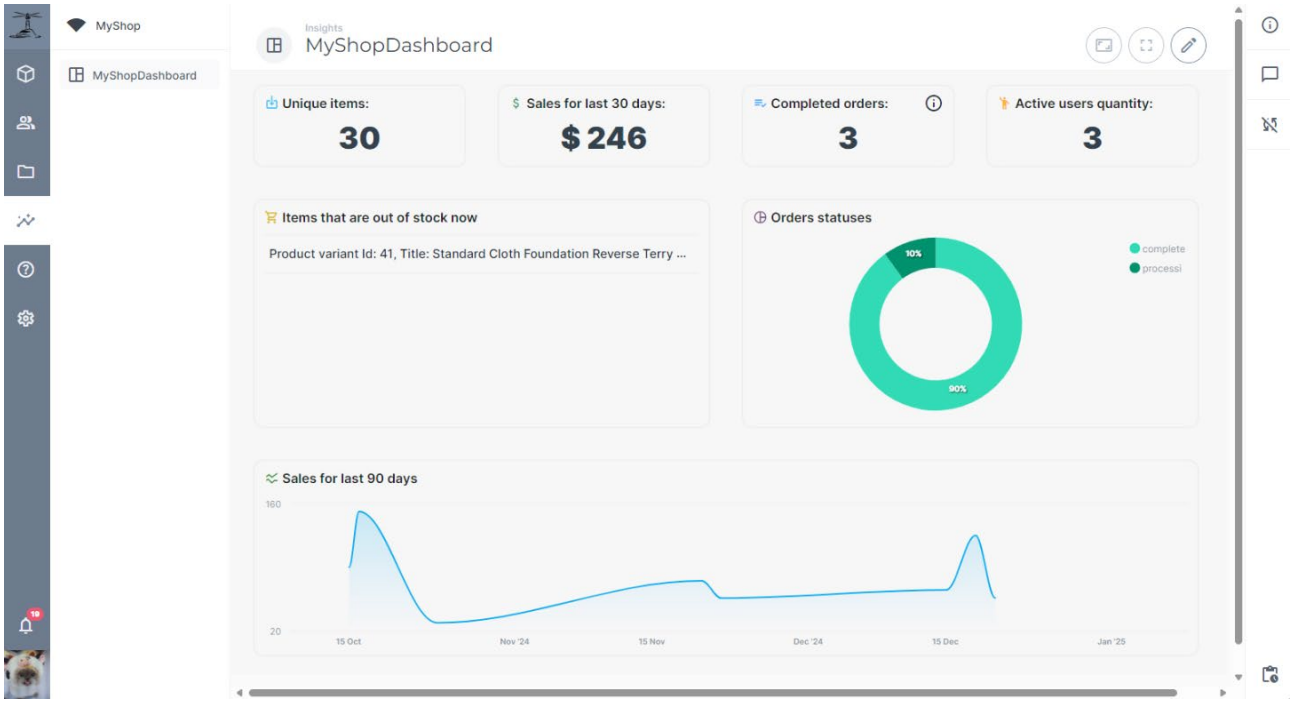


Рисунок Е.6 – Статистика системи, створена за допомогою модуля Insights

ДОДАТОК Ж

Приклад React Native компоненти

```
import { SlidersVertical } from 'lucide-react-native';
import { cssInterop } from 'nativewind';
import { useState } from 'react';
import { View } from 'react-native';
import { SafeAreaView } from 'react-native-safe-area-context';
import FilterModal from '~/components/home/FilterModal';
import Products from '~/components/home/Products';
import SearchInput from '~/components/home/SearchInput';
import { Button } from '~/components/ui/button';
import { Input } from '~/components/ui/input';
import { useProducts } from '~/hooks/useProducts';
cssInterop(SlidersVertical, {
  className: 'style',
  fill: {
    target: 'style',
    nativeStyleToProp: {
      color: 'fill',},
  },
});
export default function HomeScreen() {
  const [modalVisible, setModalVisible] = useState(false);
  const {isPending, isFetching, isError, hasNextPage,
fetchNextPage, data, refetch,
  } = useProducts();
  const products = data?.pages.flatMap((page) => page.products);
  const loadMoreProducts = isError ? undefined : hasNextPage ? ()
=> fetchNextPage() : () => refetch();
  return (
    <SafeAreaView className="flex-1 gap-5 bg-background px-4">
      <View className="mt-5 flex-row gap-2">
        {
          modalVisible &&
          <FilterModal
          modalVisible={modalVisible}
          setModalVisible={setModalVisible}
          />
        }
        <SearchInput />
        <Button onPress={() => setModalVisible(true)}>
          <SlidersVertical className="text-primary-foreground" />
        </Button>
      </View>
      <Products
        products={products}
```

```
          refetch={refetch}
          loadMoreProducts={loadMoreProducts}
          isPending={isPending}
          isFetching={isFetching}
        />
      </SafeAreaView>
    );
}
```

ДОДАТОК И

Візуальне представлення інтерфейсу застосунку

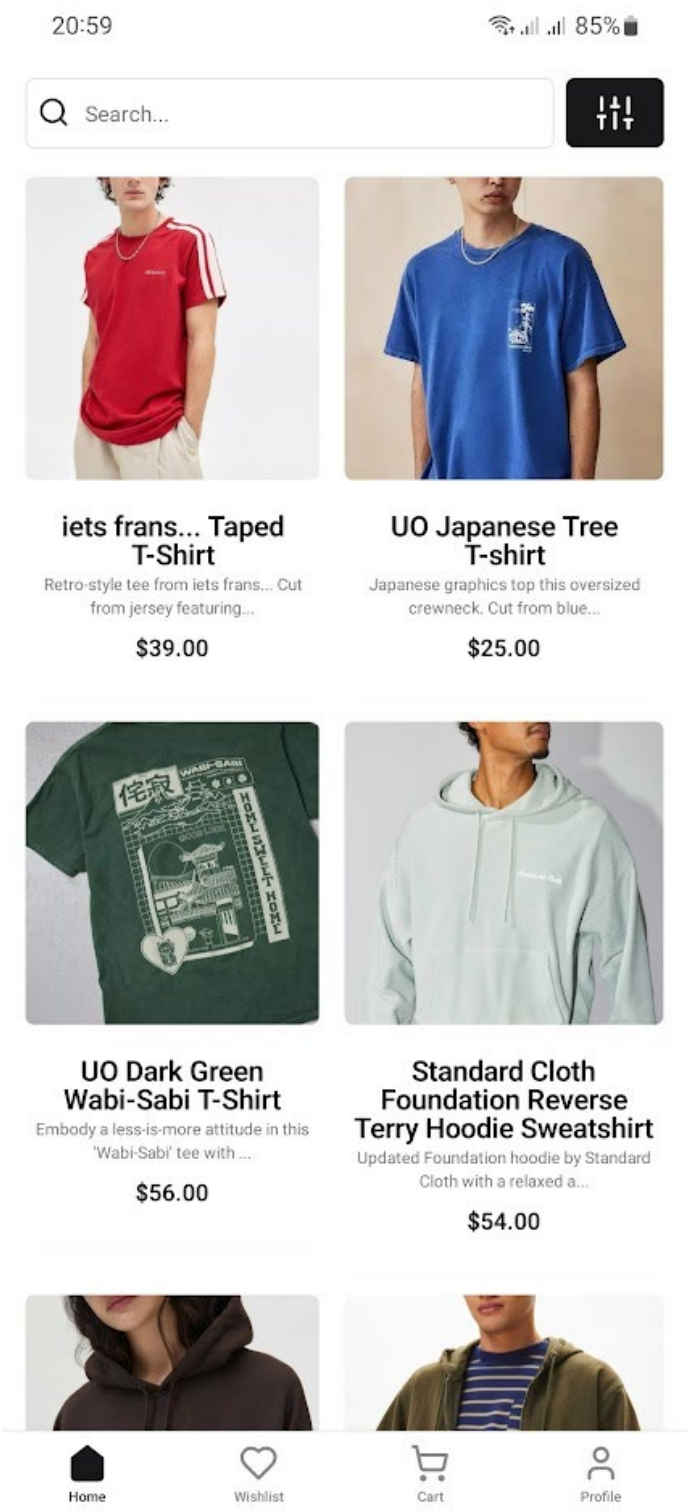


Рисунок И.1 – Головна сторінка застосунку

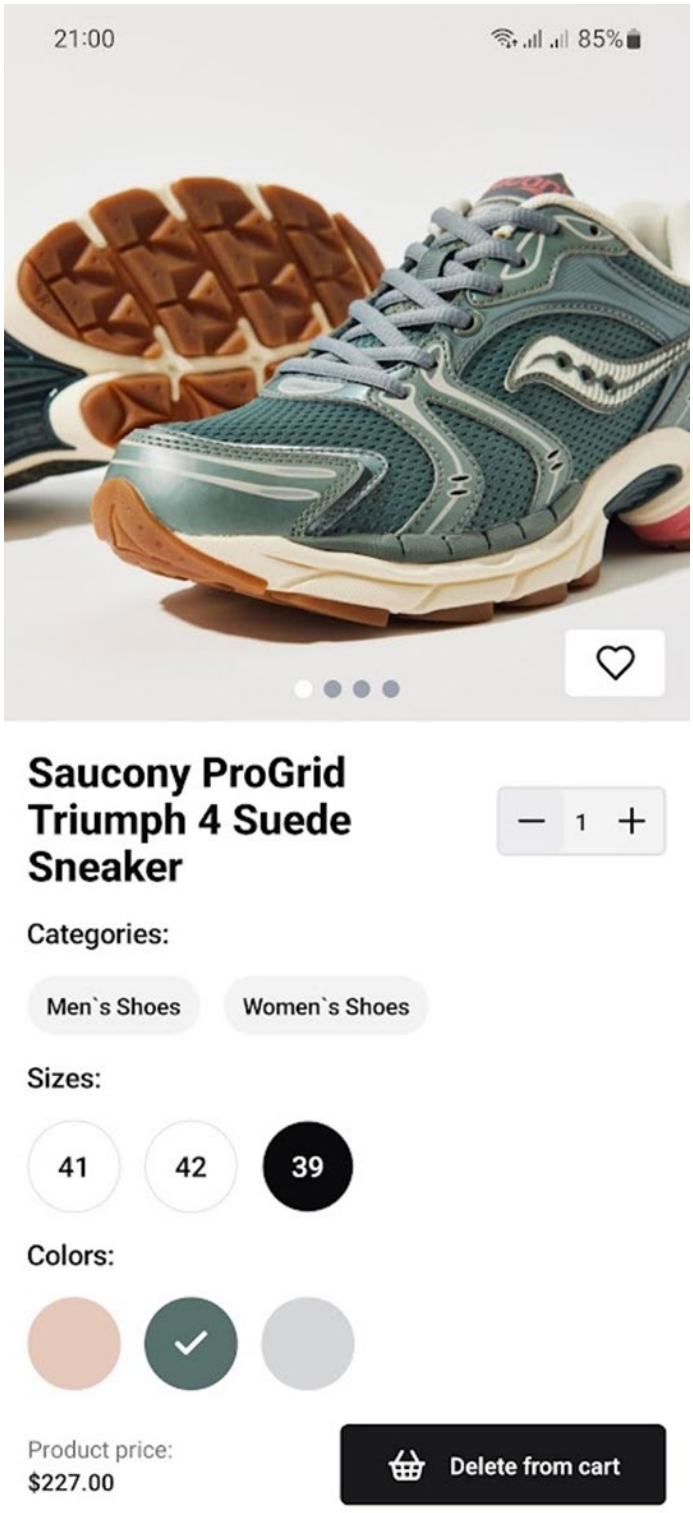


Рисунок И.2 – Сторінка товару

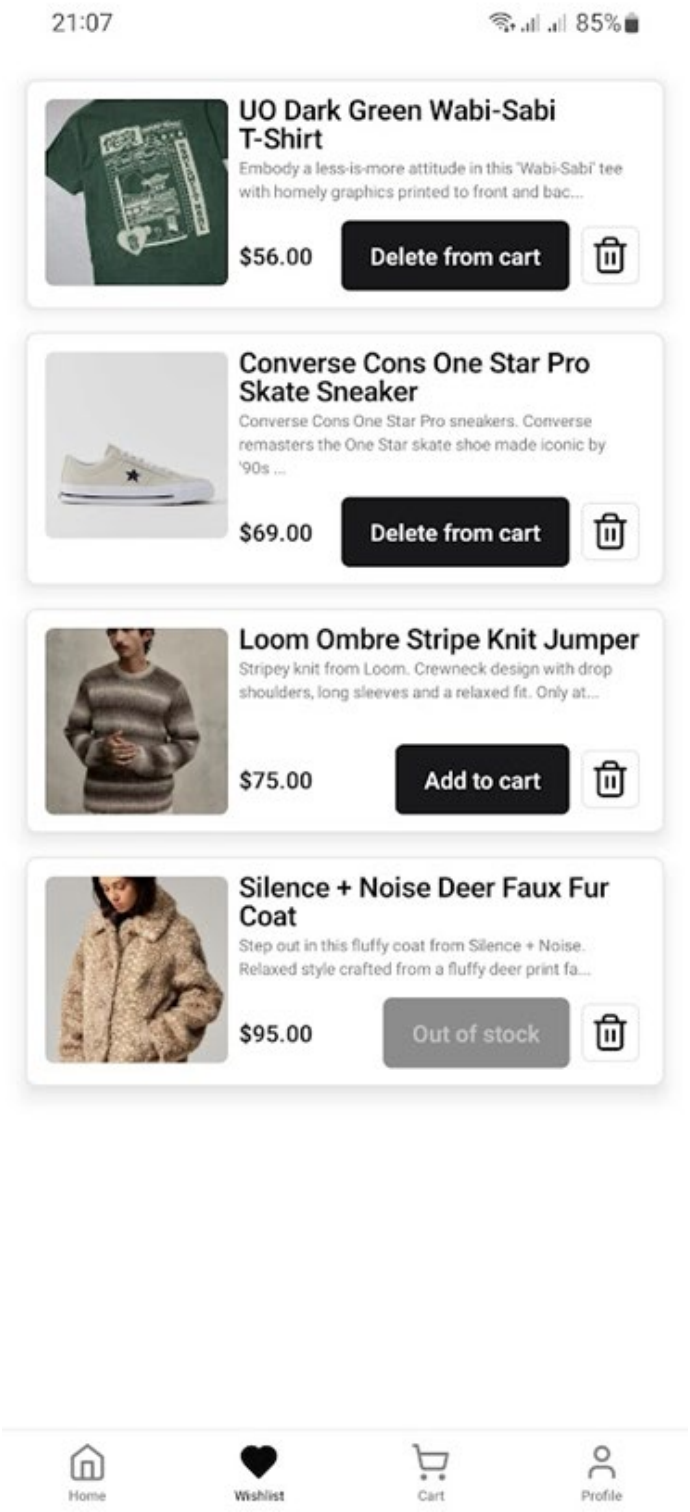


Рисунок И.3 – Сторінка із списком бажань

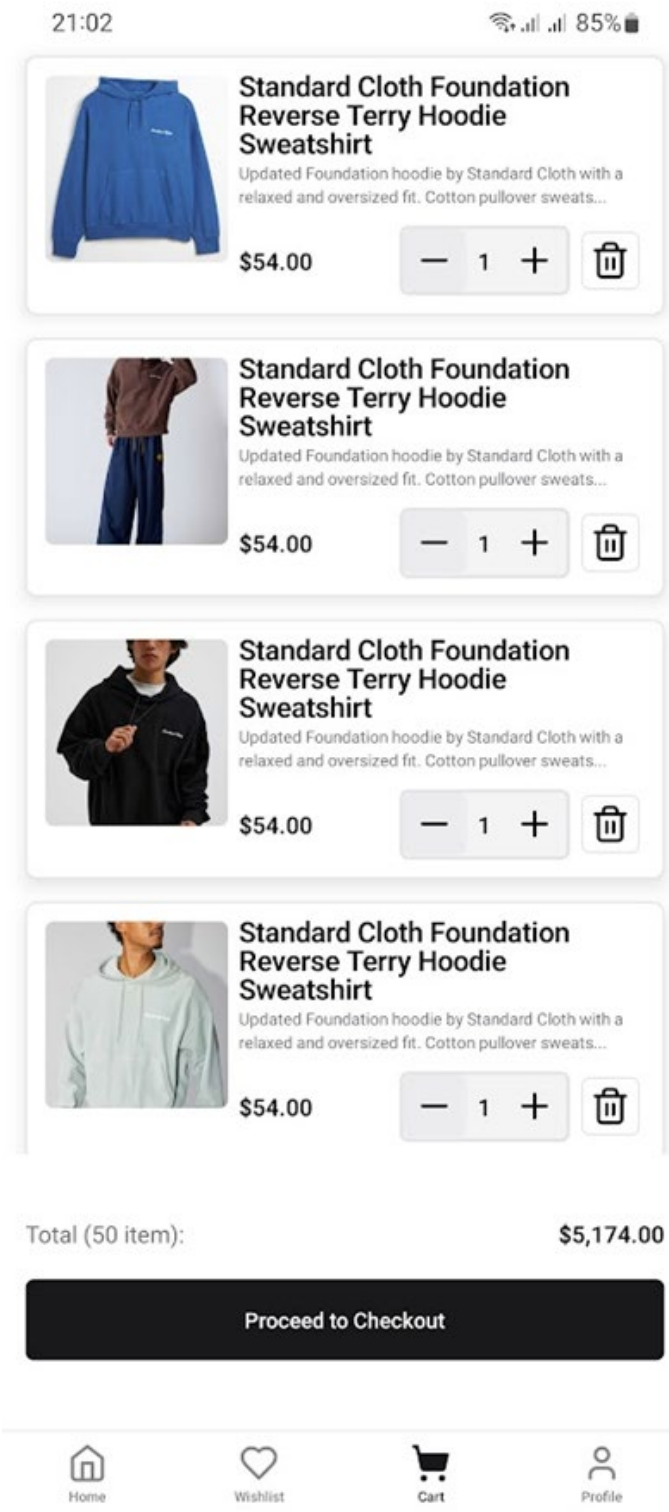


Рисунок И.4 – Сторінка із кошиком

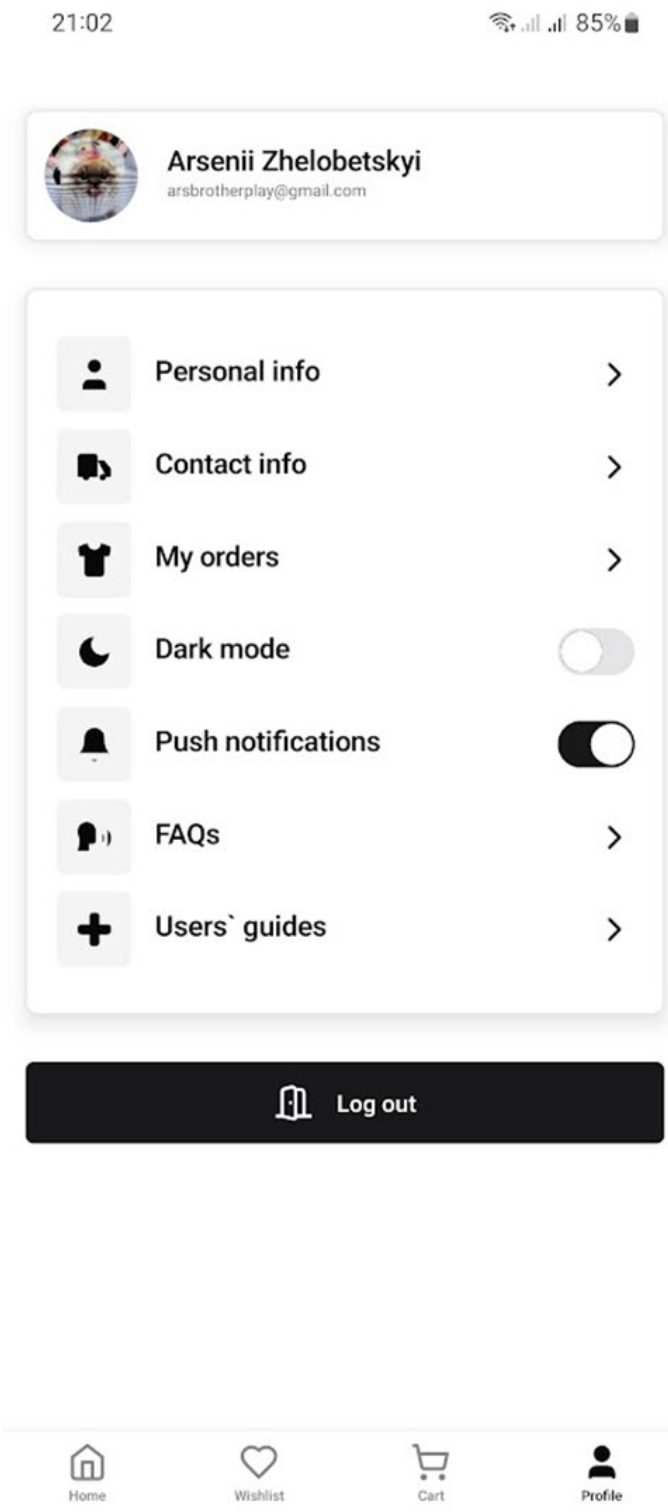


Рисунок И.5 – Сторінка профілю

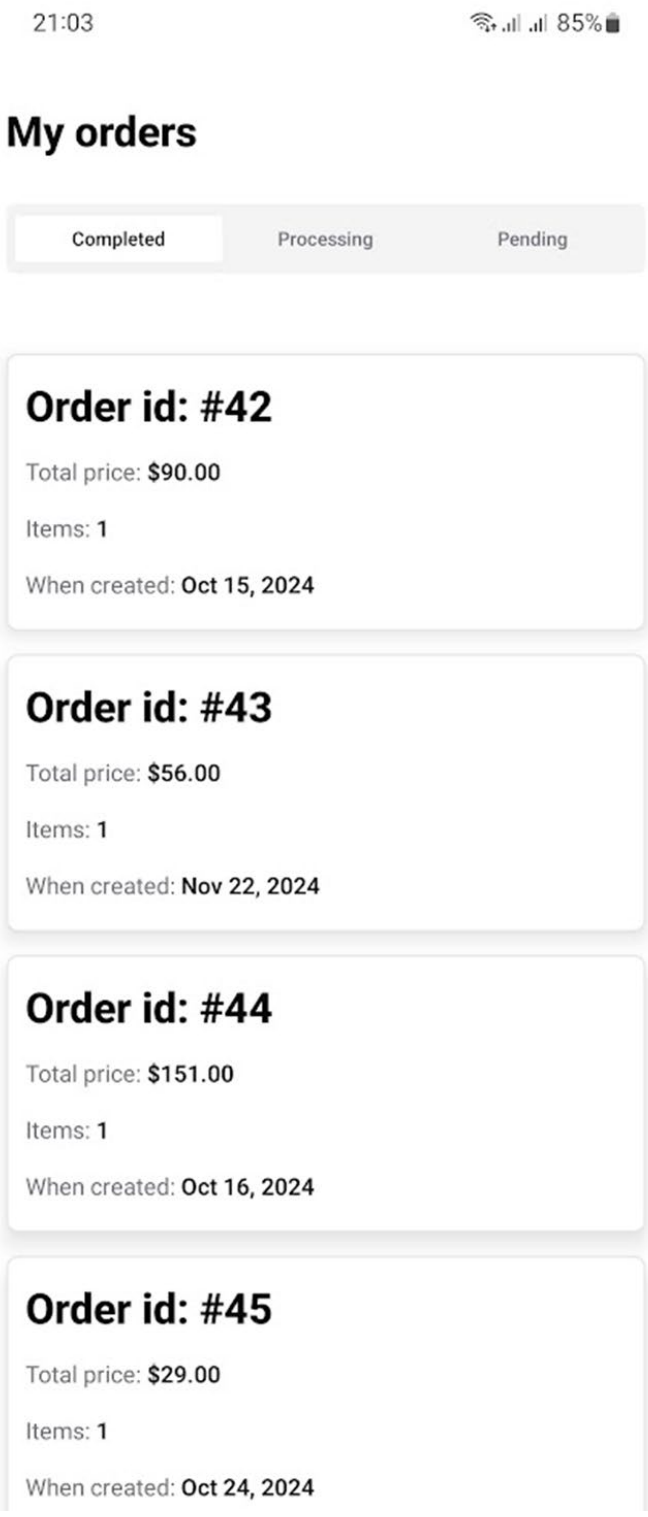


Рисунок И.6 – Сторінка замовлень