Хмельницький національний університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

Крижановського Дмитра Сергійовича
Прізвище, ім'я, по батькові студента(ки)

на здобуття ступеня вищої освіти Бакалавра

Мобільний застосунок для Інтернет-магазину з продажу побутових товарів
Назва теми

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр КвРІПЗ.200249.01.11.ПЗ

Виконав студент IV курсу, група ІПЗ-20-1 _____ Дмитро КРИЖАНОВСЬКИЙ
Підпис Ім'я, ПРІЗВИЩЕ

Керівник канд. техн. наук, доцент _____ Галина РАДЕЛЬЧУК
Науковий ступінь, вчене звання Підпис Ім'я, ПРІЗВИЩЕ

Нормоконтролер доцент _____ Галина РАДЕЛЬЧУК
Посада Підпис Ім'я, ПРІЗВИЩЕ

До захисту допускаю:
Завідувач кафедри інженерії програмного забезпечення _____ Леонід БЕДРАТЮК
Підпис Ім'я, ПРІЗВИЩЕ

7 червня 2024 р.

Хмельницький 2024

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій
Кафедра Інженерії програмного забезпечення
Рівень вищої освіти Перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри ІПЗ
Л. П. Бедратюк
02 01 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Крижановському Дмитру Сергійовичу
Прізвище, ім'я, по батькові студента

1. Тема роботи Мобільний застосунок для Інтернет-магазину з продажу побутових товарів

Керівник роботи Радельчук Галина Іванівна, канд. техн. наук, доцент
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 08.01.2024 р. № 6-КП
2. Строк подання студентом роботи на кафедру 01.06.2024 р.
3. Вихідні дані до роботи Матеріали переддипломної практики
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
Аналіз предметної області та постановка задачі
Проєктування мобільного застосунку
Програмна реалізація та тестування мобільного застосунку
5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)
Три креслення у форматі А3:
1. Діаграма варіантів використання
2. Діаграма послідовності реєстрації
3. Діаграма сценарію використання сервісів AWS на основі AppSync

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Радельчук Г. І., доцент	03.06.24	07.06.24
Антиплагіат	Форкун Ю. В., доцент	30.05.24	06.06.2024

7. Дата видачі завдання « 02 » січня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1 Ознайомлення з тематикою дипломного проектування, визначення та узгодження індивідуальних тем кваліфікаційних робіт (КвР)	01.12 – 31.12.2023	
2 Збір матеріалу за темою КвР; дослідження предметної області, в якій планується використання програмного забезпечення (ПЗ), визначення задач та вимог, розробка технічного завдання	01.01 – 20.02.2024	
3 Проектування програмного забезпечення	21.02 – 20.03 2024	
4 Програмна реалізація з використанням відповідних засобів розробки. Тестування ПЗ	21.03 – 30.04.2024	
5 Написання вступу, загальних висновків, оформлення переліку джерел посилання та додатків. Оформлення пояснювальної записки КвР згідно вимог	01.05 – 25.05.2024	
6 Попередній захист КвР	травень 2024	Згідно графіка
7 Перевірка КвР на плагіат, нормоконтроль, отримання відгуків, рецензій та інших супровідних документів. Брошурування (зшиття) пояснювальної записки	26.05 – 30.05.2024	
8 Здача КвР на кафедру; підготовка КвР для розміщення у репозитарії ХНУ; підготовка до захисту та захист КвР	з 01.06.2024	

Студент _____ Дмитро КРИЖАНОВСЬКИЙ
Підпис Ім'я, ПРІЗВИЩЕ

Керівник роботи _____ Галина РАДЕЛЬЧУК
Підпис Ім'я, ПРІЗВИЩЕ

АНОТАЦІЯ

Тема кваліфікаційної роботи: «Мобільний застосунок для Інтернет-магазину з продажу побутових товарів».

Автор роботи: Крижановський Дмитро Сергійович.

Керівник роботи: Радельчук Галина Іванівна.

Пояснювальна записка: 76 с., 9 рис., 38 табл., 7 дод., 30 джерел.

Графічна частина: три плакати ф. А3.

МОБІЛЬНИЙ ЗАСТОСУНОК, ІНТЕРНЕТ-МАГАЗИН, AWS, C#, SWIFT, GRAPHQL.

Мета кваліфікаційної роботи: розроблення мобільного застосунку, який надаватиме користувачам зручний інструмент для покупок побутових товарів у мережі Інтернет.

У кваліфікаційній роботі проведено аналіз предметної області та визначені функціональні і нефункціональні вимоги до мобільного застосунку. Розроблена загальна архітектура застосунку, спроєктована база даних, налаштовані сервіси AWS, виконана реалізація застосунку. Для розроблення програмного продукту використано мови програмування C#, Swift та мову запитів GraphQL. За допомогою цих засобів розроблено мобільний застосунок для Інтернет-магазину з продажу побутових товарів.

Впровадження розробленого застосунку дозволяє підтримувати конкурентоспроможність бізнесу в умовах динамічного ринку та відповідає потребам сучасного користувача.

07.06.2024
Дата

Підпис

ВІДОМІСТЬ ДОКУМЕНТІВ

№ рядка	Формат	Позначення документа	Найменування документа	К-сть аркушів	№ екз.	Примітка
			Текстові документи			
1	А4	КвРІПЗ.200249.01.06.ПЗ	Пояснювальна записка	76		
2	А4		Завдання на кваліфіка-	1		
			ційну роботу			
3	А4		Анотація	1		
			Графічні документи			
5	А3	КвРІПЗ.200249.01.06.Е8	Діаграма варіантів	1		
			використання			
6	А3	КвРІПЗ.200249.01.06.Е8	Діаграма послідовності	1		
			реєстрації			
7	А3	КвРІПЗ.200249.01.06.Е8	Діаграма сценарію	1		
			використання сервісів			
			AWS на основі AppSync			

Змн.	Арк.	№ докум.	Підпис	Дата	КвРІПЗ.200249.01.11.ВД		
Виконав		Крижановський		07.06	Мобільний застосунок для Інтернет-магазину з продажу побутових товарів Відомість документів	Літ.	Арк. 1 / Аркушів 1
Керівник		Радельчук Г.І.		07.06			
Н. контр.		Радельчук Г.І.		07.06		ХНУ, ІПЗ-20-1	
Зав. каф.		Бедратюк Л.П.		07.06			

ЗМІСТ

ВСТУП

В сучасному світі Інтернет-торгівлі тенденції розвитку надзвичайно динамічні. З кожним роком зростає кількість користувачів, які вибирають онлайн-покупки як зручний та швидкий спосіб придбання товарів. Побутові товари, такі як побутова техніка, предмети домашнього затишку, електроніка та інше, стають об'єктом підвищеного попиту серед споживачів.

Економічна ситуація спонукає продавців активно розвивати Інтернет-торгівлю. Зростання витрат на традиційні продажі через високі орендні тарифи, витрати на персонал та інші фактори, а також зміни у споживчій повединці призводять до того, що все більше покупців шукають більш доступні товари в інтернет середовищі. Інтернет-торгівля активно розвивається, привертаючи все більше споживачів та підприємців. В контексті цього мобільні застосунки стають невід'ємною складовою успішного бізнесу та стратегічним активом, що дозволяє підприємствам зберігати конкурентоспроможність.

Метою кваліфікаційної роботи є розроблення мобільного застосунку, який надаватиме користувачам зручний інструмент для покупок побутових товарів у мережі Інтернет.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- вивчити процеси організації Інтернет-торгівлі;
- проаналізувати сучасні мобільні застосунки-аналоги та провести їх порівняльну характеристику;
- визначити функціональні та нефункціональні вимоги до розроблюваного застосунку;
- обрати архітектуру мобільного застосунку;
- визначити структуру даних та спроєктувати базу даних;
- спроєктувати структуру мобільного застосунку;
- обрати та налаштувати засоби та інструменти реалізації застосунку;
- реалізувати застосунок та розробити керівництво користувача;
- виконати тестування мобільного застосунку та провести аналіз отриманих результатів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз предметної області, її структурних та функціональних особливостей

У сфері електронної торгівлі Інтернет-магазини є ключовим елементом для здійснення покупок товарів і послуг в онлайн-середовищі. Вони можуть працювати в різних галузях, таких як електроніка, побутова техніка, мода, косметика тощо. Розроблення мобільного додатку для інтернет-торгівлі є надзвичайно актуальною у сучасному світі. Ось кілька причин, чому це так:

– користувачі все частіше використовують смартфони та планшети для здійснення покупок в інтернеті. Мобільні пристрої стають основними засобами доступу до Інтернет-магазинів, тому розроблення застосунків для них стає важливою складовою успішного бізнесу;

– мобільні застосунки забезпечують зручний та швидкий доступ до товарів у будь-який час та з будь-якого місця (користувачам не потрібно включати комп'ютер або ноутбук, вони можуть здійснити покупку прямо зі свого смартфона під час подорожей, на роботі або вдома);

– мобільні застосунки дозволяють створювати персоналізовані пропозиції та рекомендації для користувачів на основі їхньої попередньої поведінки та відповідно до їхніх інтересів. Це сприяє збільшенню залученості і підвищує ймовірність здійснення ними покупок;

– у світі електронної торгівлі конкуренція дуже велика, тому Інтернет-магазинам потрібно постійно розвиватися та пристосовуватися до змін, щоб зберігати свою конкурентоспроможність; мобільний застосунок може бути ключовим елементом успішної стратегії розвитку.

Отже, розроблення мобільного застосунку для Інтернет-торгівлі є актуальним, оскільки це відповідає потребам сучасного користувача та допомагає підтримувати конкурентоспроможність бізнесу в умовах динамічного ринку. Головна проблема, яка вирішується за допомогою мобільного застосунку

для Інтернет-магазину – це зручний та швидкий доступ користувачів до асортименту товарів та можливість здійснення покупок у будь-який зручний для них час та місце через мобільні пристрої. Кінцевими користувачами мобільного застосунку Інтернет-магазину з продажу побутових товарів є клієнти, які мають інформаційні потреби щодо доступу до асортименту товарів, їх характеристик, цін та умов придбання. Їхня основна потреба − здійснювати покупки швидко та зручно через мобільні пристрої.

1.2 Аналіз наявного програмно-технічного забезпечення предметної області

На сьогоднішній день існує значна кількість мобільних застосункіув інтернет-магазинів, які пропонують різноманітні товари та послуги, проте лише деякі з них змогли завоювати популярність та успіх на ринку. Оцінка переваг та недоліків мобільного застосунку здійснюється на основі аналізу програмно-технічного забезпечення. Результати аналізу дозволяють зрозуміти сильні та слабкі сторони застосунку, покращити його функціональність та надійність, а також забезпечити успішну конкурентоспроможність. Порівняльний аналіз програмно-технічного забезпечення виконано для найбільш популярних та поширених мобільних застосунків, таких як ROZETKA, Prom.ua та Comfy (рисунок 1.1).

«ROZETKA» – це один з найбільших українських інтернет-магазинів, що пропонує широкий асортимент товарів, від електроніки та побутової техніки до товарів для дому, спортивних товарів, косметики та багато іншого [1]. ROZETKA працює у форматі електронного роздрібного магазину, де користувачі можуть замовляти товари через інтернет та мобільний застосунок.

«Prom.ua» – це великий український онлайн-маркетплейс, який об'єднує тисячі продавців та мільйони покупців [2]. На цьому ресурсі можна знайти широкий асортимент товарів і послуг від різних продавців, включаючи

електроніку, одяг, взуття, косметику, товари для дому, товари для дітей та багато іншого. Prom.ua дозволяє компаніям та індивідуальним продавцям створювати власні магазини на платформі, де вони можуть розміщувати свої товари, встановлювати ціни та вести взаємодію з покупцями.

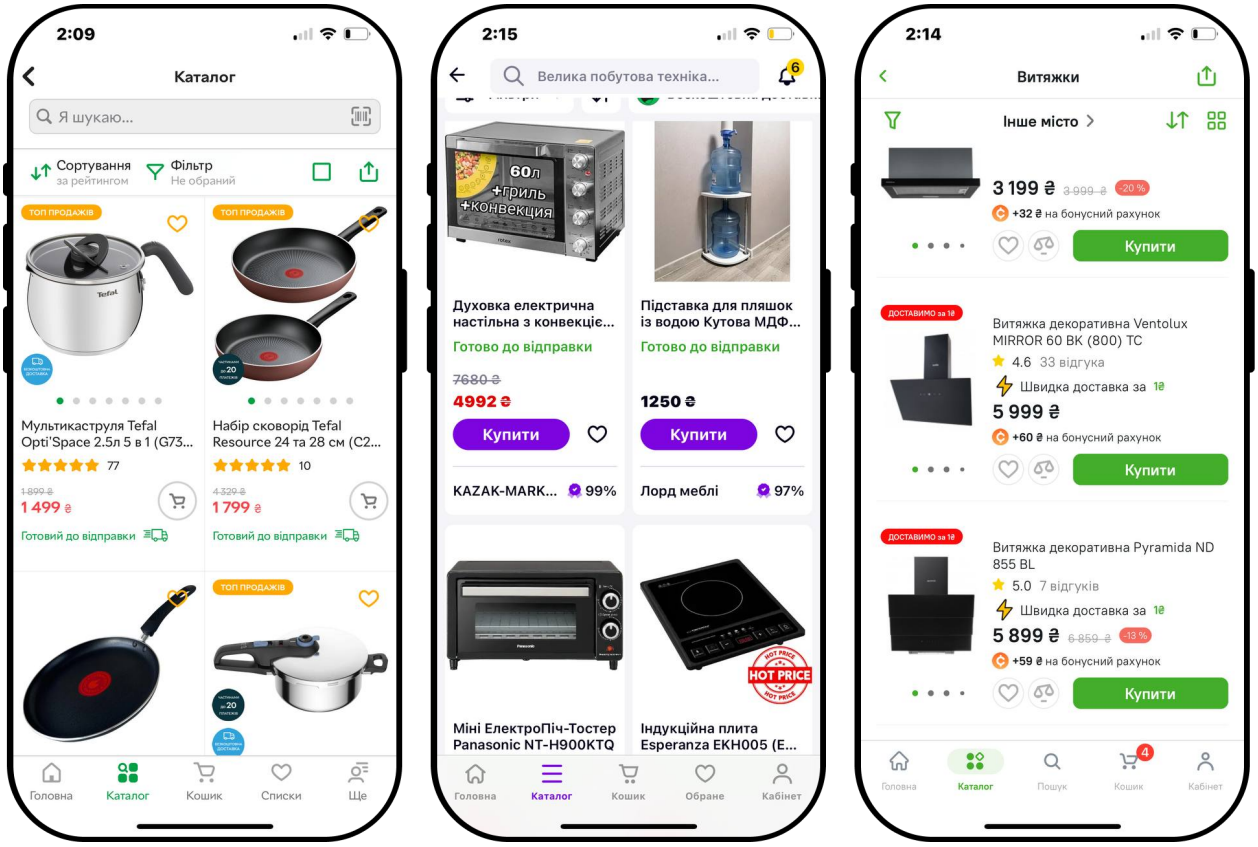


Рисунок 1.1 – Інтерфейси мобільних застосунків ROZETKA, Prom.ua, Comfy

«Comfy» – це один з найбільших інтернет-магазинів в Україні, що спеціалізується на продажі електроніки, побутової техніки, товарів для дому, одягу, взуття та інших товарів різних категорій [3]. Він є популярним каналом для покупців, які шукають якісні товари за доступними цінами та зручними умовами доставки. Споживачі можуть замовляти товари через Інтернет на вебсайті Comfy або в одному з магазинів-салонів, які розташовані у багатьох містах України. Порівняльний аналіз трьох найпопулярніших мобільних застосунків для інтернет-магазинів здійснювався за певними ключовими критеріями. Результати аналізу наведено у таблиці 1.1.

Таблиця 1.1 – Порівняльна таблиця можливостей мобільних застосунків

Критерій для порівняння	ROZETKA	Prom.ua	Comfy
Реєстрація користувача	Так	Так	Так
Профіль користувача	Так	Так	Так
Головне меню	Так	Так	Так
Категорії товарів	Так	Так	Так
Детальний опис товару	Так	Так	Так
Фільтр для пошуку	Так	Так	Так
Список бажаних товарів	Так	Так	Так
Історія замовлень	Так	Так	Так
Чат з продавцем	Так	Так	Ні
Відгуки користувачів	Так	Так	Так
Порівняння товарів	Так	Ні	Так
Акційні пропозиції	Так	Так	Так

Результати проведеного аналізу вказують на те, що сучасний мобільний застосунок для інтернет-магазину повинен включати реєстрацію та авторизацію користувачів, головне меню, профайл користувача, поділ товарів за категоріями, фільтр пошуку, формування кошику замовлень, відгуки користувачів, а також можливість перегляду історії замовлень та додавання торів до списку бажаного.

1.3 Визначення функціональних та нефункціональних вимог до програмного забезпечення

Загальні вимоги до мобільного застосунку поділяються на дві категорії: функціональні та нефункціональні. Функціональні вимоги, повинні визначати, що система, тобто мобільний застосунок, повинен робити з точки зору функціональності та функцій [4]. Нефункціональні вимоги, повинні зосереджуватися на якості та характеристиках мобільного застосунку.

Функціональні вимоги:

- реєстрація нового та авторизація існуючого користувача;
- відновлення паролю та підтвердження електронної адреси;
- каталог товарів та пошук по фільтру за встановленими критеріями;
- детальний опис товару та ознайомлення з додатковими матеріалами (статті, сертифікати якості, документація і т.д.);
- формування кошику замовлень;
- здійснення замовлення обраних товарів;
- перегляд історії замовлень;
- оцінка товарів та відгуки користувачів.

Нефункціональні вимоги:

- користувач повинен автентифікуватися перед доступом до основного функціоналу застосунку за допомогою електронної пошти та пароля;
- сторінки каталогу товарів повинні завантажуватися швидко, ~2 секунд;
- застосунок повинен бути сумісний з операційними системами iOS попередніх версій (починаючи з 15 і вище);
- застосунок повинен бути сумісний з мобільними пристроями iPhone попередніх версій (починаючи з 11 і вище);
- розміри та інтерфейс елементів застосунку повинні відповідати стандартам дизайну для забезпечення комфортного використання на різних пристроях.

1.4 Висновки. Постановка задачі

Проведено аналіз предметної області, обгрунтовано актуальність розроблення мобільного застосунку для Інтернет-магазину з продажу побутової техніки, визначена проблематика та потреби кінцевих користувачів.

Виконано аналіз наявного програмно-технічного забезпечення предметної області, визначено її основні функціональні особливості.

Сформульовано основні функціональні та нефункціональні вимоги до мобільного застосунку.

Метою кваліфікаційної роботи є розроблення мобільного застосунку, який надаватиме користувачам зручний інструмент для покупок побутових товарів у мережі Інтернет.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- вивчити процеси організації Інтернет-торгівлі;
- проаналізувати сучасні мобільні застосунки-аналоги та провести їх порівняльну характеристику;
- визначити функціональні та нефункціональні вимоги до розроблюваного застосуну;
- обрати архітектуру мобільного застосунку;
- визначити структуру даних та спроєктувати базу даних;
- спроєктувати структуру мобільного застосуноку;
- обрати та налаштувати засоби та інструменти реалізації застосунку;
- реалізувати застосунок та розробити керівництво користувача;
- виконати тестування мобільного застосунку та провести аналіз отриманих результатів.

2 ПРОЄКТУВАННЯ МОБІЛЬНОГО ЗАСТОСУНКУ

2.1 Архітектура та функціональна структура застосунку

Розроблення мобільного застосунку – це складний та цікавий процес, який включає в себе кілька етапів. Етап проектування є одним із головних, де створюється архітектура застосунку, вибираються технології та інструменти для розробки. У невеликих застосунках вибір архітектури може здатися не настільки важливим, оскільки їхня логіка зазвичай досить інтуїтивно зрозуміла. Однак, коли потрібно розширити функціонал, невірно обрана архітектура може призвести до необхідності переписування значної частини коду та зміни внутрішньої логіки. Правильно обрана архітектура дозволяє ефективно впроваджувати новий функціонал без великих змін у вже існуючому коді. Вона сприяє легкості розуміння коду для розробників, що у свою чергу спрощує процес супроводу та відлагодження програми.

У розробленні мобільних застосунків існують різні архітектурні підходи, кожен з яких має свої переваги та недоліки. Основні типи архітектур включають монолітну, сервіс-орієнтовану, мікро-сервісну та безсерверну.

Монолітна архітектура – це архітектурний підхід, де весь функціонал додатку розглядається як єдине ціле, об'єднане в одному монолітному блоку коду. Її перевагами є простота розробки, відлагодження та тестування, оскільки весь код знаходиться в одному місці. Проблеми виникаю при збільшені обсягу трафіку або кількості користувачів, що може призвести до складнощів у масштабуванні застосунку. Також через високу залежність компонентів можуть виникнути проблеми при зміні та підтримці застосунку.

Загалом монолітна архітектура програмного забезпечення може бути корисною, якщо команда розробників перебуває на стадії створення продукту, а також ідеально підходить для стартапів, які повинні мати готовий продукт якомога швидше.

Сервіс-орієнтована архітектура (SOA) – це підхід до розробки програмного забезпечення, в якому функції додатку розбиваються на окремі сервіси, які виконують конкретні завдання та взаємодіють між собою за допомогою API (інтерфейсів програмування застосунків). Кожен сервіс є самостійним, має власну логіку, базу даних та може використовувати різні технології при розробці.

Проте збільшення кількості сервісів може призвести до складнощів у їх управлінні та координації взаємодії між ними, також підтримка великої кількості сервісів може призвести до збільшення накладних витрат на комунікацію та передачу даних між ними.

Загалом сервіс-орієнтована архітектура найкраще підходить для складних корпоративних систем або великих застосунків з багатьма функціональними блоками та потребою у гнучкому масштабуванні та готовності до змін.

Мікро-сервісна архітектура – це підхід до розробки програмного забезпечення, в якому функції застосунку розділяються на невеликі самостійні сервіси, кожен з яких відповідає за виконання конкретного завдання та взаємодіє з іншими сервісами через API. Тобто кожен сервіс може бути масштабований незалежно від інших, що дозволяє оптимізувати використання ресурсів. Розроблення окремих сервісів може вестися незалежно, що спрощує розподілене розроблення та дозволяє використовувати різні технології і мови програмування.

Проте збільшення кількості сервісів може призвести до складнощів у їх управлінні та координації взаємодії між ними. Зазвичай збільшується кількість системних витрат на комунікацію та координацію, а також з ростом системи складність тестування та відлагодження.

Мікро-сервісна архітектура ідеально підходить для розробки великих і складних систем, особливо у випадках, коли потрібна висока гнучкість, масштабованість та швидкість реагування на зміни.

Безсерверна архітектура – це підхід до розроблення програмного забезпечення, де код розглядається як набір незалежних функцій, які викликаються лише відповідно до запитів або подій.

У цій архітектурі ресурси автоматично масштабуються залежно від навантаження, що дозволяє ефективно використовувати ресурси та знижує витрати. За допомогою неї розробники можуть швидко розгортати функції та додатки без необхідності управління інфраструктурою. Також плата за без серверні послуги зазвичай залежить від реального використання ресурсів, що дозволяє економити кошти при низькому навантаженні.

Проте перехід до безсерверної архітектури може вимагати від розробників зміни свого мислення та підходу до програмування. Також деякі області, такі як довгочасні обчислення або постійні процеси, можуть бути менш ефективними. Загалом використання без серверної архітектури зазвичай означає залежність від конкретного постачальника хмарних послуг або платформи, що може впливати на міграцію та портативність коду.

Безсерверна архітектура ідеально підходить для виконання короткочасних та інтенсивних обчислень, обробки подій в реальному часі та реалізації допоміжних процесів, що дозволяє ефективно використовувати ресурси, швидко реагувати на зміни навантаження та знижувати витрати на інфраструктуру.

За результатами порівняльного аналізу різних типів архітектур, їх переваг та недоліків, зупиняємо свій вибір на безсерверній архітектурі. Ця архітектура відповідає нашим потребам найкраще, оскільки дозволяє ефективно використовувати обчислювальні ресурси та забезпечує гнучкість і швидкість реагування на зміни у мобільному застосунку.

Amazon Web Services (AWS) є одним з провідних постачальників безсерверних технологій на ринку хмарних послуг. Компанія відома своїм широким набором сервісів, які дозволяють розробникам будувати та впроваджувати без серверні додатки ефективно та надійно. Технологія AWS базується на серверних кластерах (фермах), розташованих по всьому світу.

Amazon пропонує широке різноманіття хмарних сервісів, які можуть задовольнити різноманітні потреби розробників у будь-якій сфері діяльності, серед них: Amplify, Cognito, AppSync, CloudWatch, Lambda, DynamoDB, S3 Storage, API Gateway, Simple Email Service (SES), OpenSearch, Identity and Access Management (IAM).

AWS Amplify – це набір спеціально створених інструментів і можливостей, які дозволяють розробникам веб- та мобільних застосунків швидко та легко створювати наскрізні програми на AWS і надають їм гнучкість у використанні різноманітних послуг AWS у міру розвитку їхніх випадків використання [5].

AWS Cognito – це орієнтована на розробників, економічно ефективна служба ідентифікації клієнтів та управління доступом (CIAM). Він забезпечує безпечне сховище та параметри федерації посвідчень, які можуть масштабуватися до мільйонів користувачів [6]. Cognito підтримує вхід за допомогою постачальників ідентифікації соціальних мереж та постачальників ідентифікації на основі SAML або OIDC, щоб забезпечити задоволеність клієнтів, а також пропонує розширені можливості безпеки для ефективного захисту клієнтів та підприємств.

AWS AppSync – це сервіс який дозволяє, створювати універсальні API для безпечного доступу, змінювати та об`єднувати дані із кількох джерел (рисунок 2.1) [7]. В AppSync реалізовано вбудовані елементи керування авторизацією, з підтримкою ключів API, IAM, Amazon Cognito, OpenID, підтримка JavaScript та TypeScript для написання бізнес-логіки. В AppSync використовується мова запитів для роботи з даними GraphQL, яка дає змогу клієнтським додаткам отримувати дані від різних серверів, змінювати їх і підписуватися на них.

AWS CloudWatch – це сервіс моніторингу та управління, що дає змогу збирати та зберігати журнали ресурсів, додатків і сервісів у режимі, близькому до реального часу [8]. Він надає корисну інформацію, яка дає змогу

оптимізувати продуктивність застосунків, керувати використанням ресурсів та оцінювати працездатність системи загалом.

AWS Lambda – це безсерверний обчислювальний сервіс, який автоматично запускає код у відповідь на різноманітні події, такі як HTTP-запити та автоматично керує основними обчислювальними ресурсами (рисунок 2.2) [9]. Ці події можуть включати зміни стану або оновлення, наприклад, коли користувач додає товар до кошика.

AWS DynamoDB – це повністю керована без серверна база даних NoSQL на основі пари «ключ-значення», створена для запуску високопродуктивних додатків будь-якого масштабу [10]. DynamoDB пропонує вбудований захист, постійне резервне копіювання, автоматичну реплікацію в декількох регіонах, кешування пам'яті та інструменти імпорту та експорту даних (рисунок 2.3).

AWS S3 Storage − це масштабований веб сервіс хмарного зберігання для оперативного резервного копіювання та архівування даних і прикладних програм з високою швидкістю [11]. Сервіс S3 Storage дозволяє зберігати кожен об'єкт разом з пов'язаними метаданими та ідентифікаційним номером, який використовується для доступу. На відміну від сховищ файлів і блоків, для доступу до об'єктів розробники можуть використовувати інтерфейс API RESTful.

AWS API Gateway – це компонент інфраструктури AWS, який знаходиться між клієнтами та сервісами та забезпечує централізовану обробку API-зв’язку між ними [12]. Він також забезпечує дотримання політик безпеки, моніторинг і видимість у локальних, мультихмарних і гібридних середовищах.

Amazon SES – це універсальний постачальник сервісів електронної пошти, який може бути інтегрований у будь-який застосунок для автоматизації надсилання великих обсягів електронної пошти [13].

AWS OpenSearch (раніше відомий як AWS Elasticsearc) – це повністю керований сервіс, який надається AWS для розгортання, керування та масштабування пошукового рішення на основі відкритого програмного забезпечення Elasticsearch та Kibana. OpenSearch є дуже потужним

інструментом для аналізу, візуалізації та пошуку великих обсягів структурованих та неструктурованих даних [14].

AWS IAM (Identity and Access Management) – це служба управління доступом користувачів та ресурсів в Amazon Web Services (AWS) [15]. Ця служба дозволяє адміністраторам створювати та керувати обліковими записами користувачів, надавати їм права доступу до різних AWS ресурсів, а також контролювати їхні дії. За допомогою IAM можна встановлювати різні політики доступу, які визначають, які користувачі мають доступ до конкретних ресурсів, які дії вони можуть виконувати і в яких регіонах ці дії дозволені.

Сценарії використання AppSync та API Gateway є ключовими в розробці сучасних мобільних застосунків на базі AWS [16]. Розроблені варіанти сценаріїв використання сервісів представлені на рисунках 2.1, 2.2. API Gateway має досить простий інтерфейс для налаштування та розгортання API. Він дозволяє швидко створювати та налаштовувати точки доступу до API. API Gateway забезпечує можливість масштабування вашого API відповідно до потреб додатка, підтримує різноманітні типи інтеграцій, включаючи Lambda-функції, сервіси AWS, HTTP-сервери, докер контейнери тощо.

Одна з ключових переваг AppSync − це можливість реалізувати синхронізацію даних між клієнтами у реальному часі. AppSync підтримує GraphQL, що дає гнучкість в роботі з даними та дозволяє клієнтам запитувати лише необхідні для них дані та надає можливість ефективно працювати в офлайн-режимі, зберігаючи та синхронізуючи зміни з сервером, коли відсутнє підключення до мережі.

Вибір між AppSync та API Gateway залежить від конкретних потреб мобільного застосунку. API Gateway може бути кращим вибором для простих RESTful API або для застосунків, які використовують різні типи інтеграцій, тоді як AppSync може бути більш підходящим для застосунків, що потребують реального часу, гнучкості та офлайн-режиму.

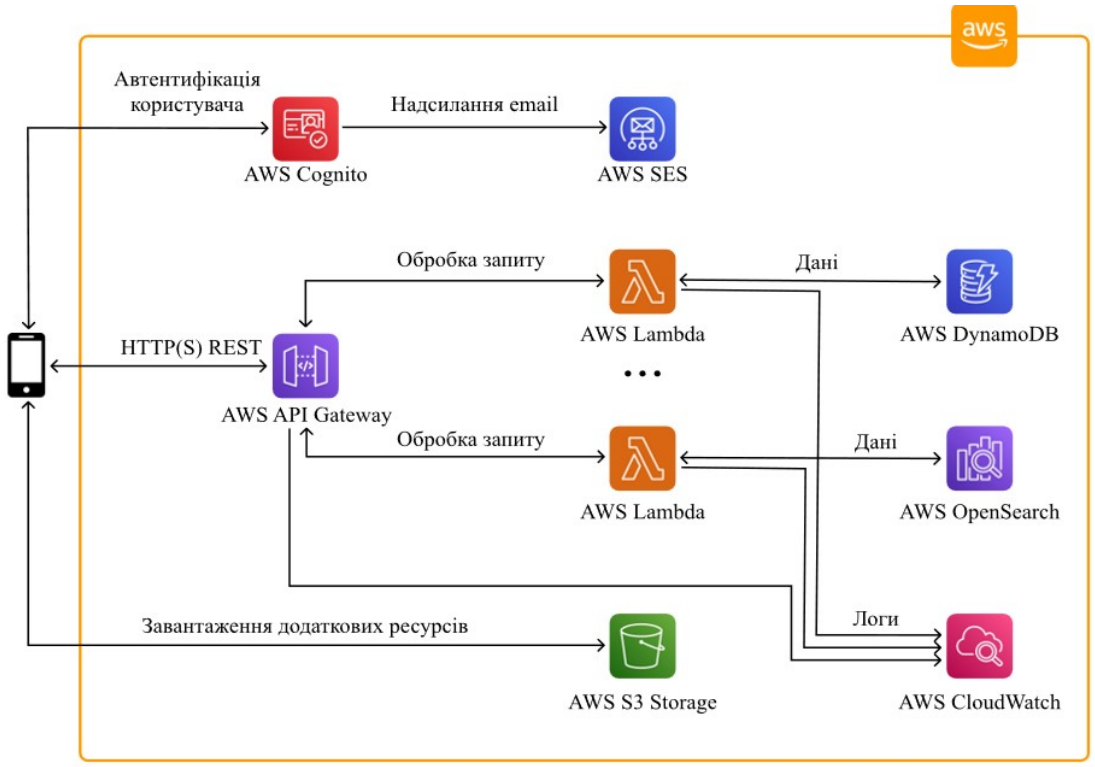


Рисунок 2.1 – Сценарій використання сервісів AWS на основі API Gateway

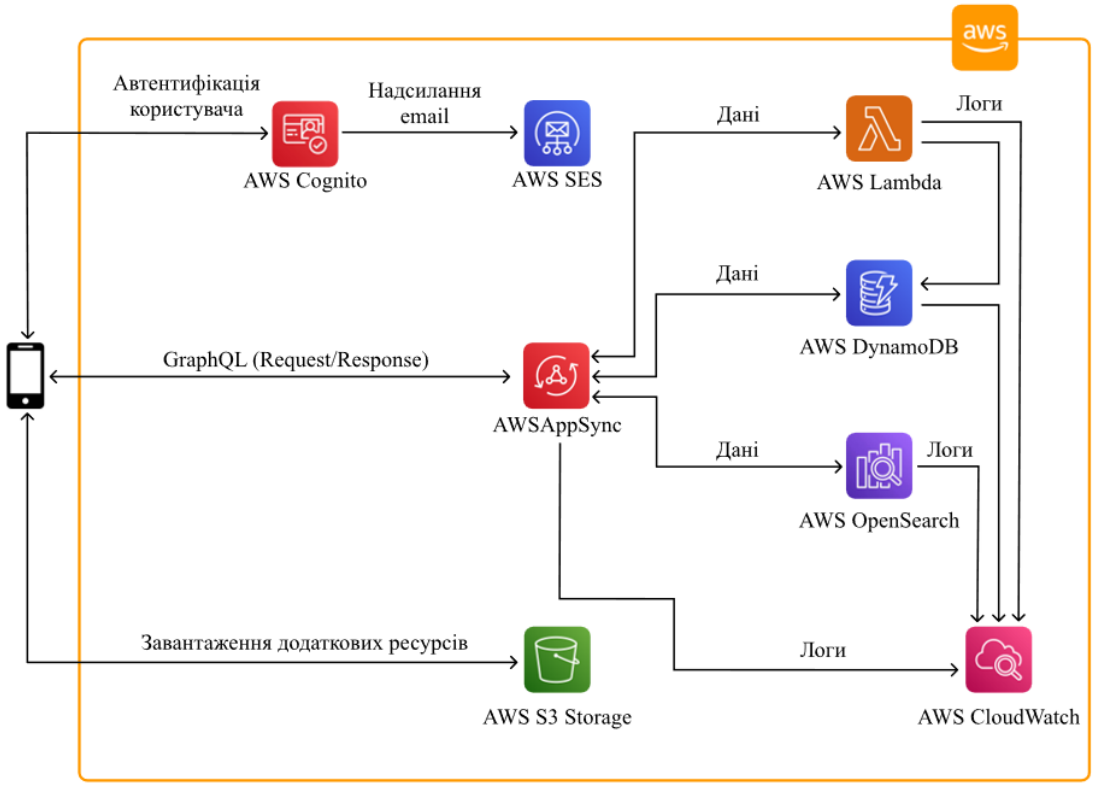


Рисунок 2.2 – Сценарій використання сервісів AWS на основі AppSync

Проаналізувавши наявні переваги обох сценаріїв можна зробити висновок, що для мобільного застосунку Інтернет-магазину з продажу побутових товарів є доцільним використання сценарію на основі AppSync.

2.2 Проєктування структури бази даних

Для створення гнучких та ефективних баз даних використовуються різні патерни проектування, серед яких: One-to-One, One-to-Many, Entity-Attribute-Value (EAV) та інші [17]. Патерн one-to-one встановлює звязки, де кожен об'єкт з одного класу пов'язаний з одним об'єктом з іншого класу. Патерн one-to-many використовується для відображення звязків, де один об'єкт з одного класу може мати багато об'єктів з іншого класу. Патерн EAV застосовується там, де важко передбачити всі можливі типи даних та їх відносини заздалегідь, і коли необхідна велика гнучкість в структурі даних.

Сутності є основними об'єктами які використовуються при проектуванні баз даних. Кожна сутність представляє окремий об'єкт або концепцію, яка має свої властивості і може мати відносини з іншими сутностями.

Сутність Customer представляє собою інформацію про окремого користувача, який здійснює покупки в мобільному застосунку. Основні атрибути сутності Customer наведено у таблиці 2.1.

Таблиця 2.1 – Основні атрибути сутності Customer

Атрибут	Тип даних	Призначення
id	ID	Унікальний ідентифікатор
firstName	String	Ім’я користувача
lastName	String	Прізвище користувача
email	AWSEmail	Електрона пошта
emailVerified	Boolean	Стан верифікації електронної пошти

Кінець таблиці 2.1

phone	AWSPhone	Мобільний телефон
phoneVerified	Boolean	Стан верифікації мобільного телефону
createdAt	AWSDateTime	Дата створення запису
updatedAt	AWSDateTime	Дата зміни запису

Для опису товару в магазині вводимо наступні сутності: Category, Brand Product, ProductItem, ProductDetail, ProductAttribute. Сутність Category відображає конкретну групу товарів, які мають спільні характеристики, призначення або властивості. Вона дозволяє класифікувати продукти за певними критеріями для зручності управління та організації. Основні атрибути сутності Category наведено у таблиці 2.2.

Таблиця 2.2 – Основні атрибути сутності Category

Атрибут	Тип даних	Призначення
id	ID	Унікальний ідентифікатор
parentId	ID	Ідентифікатор батьківської категорії
name	String	Назва категорії
description	String	Опис категорії
imageUrl	AWSURL	Посилання на логотип
displayOrder	Int	Поряд відображення
createdAt	AWSDateTime	Дата створення запису
updatedAt	AWSDateTime	Дата зміни запису

Сутність Brand відображає конкретного виробника товару, який представляє продукцію певної торгової марки в базі даних. Вона дозволяє ідентифікувати та організовувати інформацію про виробників, що спрощує управління та пошук товарів за брендами. Основні атрибути сутності Brand наведено у таблиці 2.3.

Таблиця 2.3 – Основні атрибути сутності Brand

Атрибут	Тип даних	Призначення
id	ID	Унікальний ідентифікатор
name	String	Назва бренду
description	String	Опис бренду
imageUrl	AWSURL	Посилання на логотип
createdAt	AWSDateTime	Дата створення запису
updatedAt	AWSDateTime	Дата зміни запису

Сутність Product відображає конкретний тип продукту, який можна придбати у магазині. Вона містить посилання на сайт виробника, що дозволяє клієнтам отримувати додаткову інформацію про товар та його характеристики. Основні атрибути сутності Product наведено у таблиці 2.4.

Таблиця 2.4 – Основні атрибути сутності Product

Атрибут	Тип даних	Призначення
id	ID	Унікальний ідентифікатор
brandId	ID	Ідентифікатор бренду
categoryId	ID	Ідентифікатор категорії
ean	String	Європейський номер товару
name	String	Назва товару
siteUrl	AWSURL	Посилання на сайт продавця
imageUrl	AWSURL	Посилання на фото
createdAt	AWSDateTime	Дата створення запису
updatedAt	AWSDateTime	Дата зміни запису

Сутність ProductItem відображає різні модифікації або варіанти одного основного продукту. Кожен варіант може мати свої відмінності у характеристиках, таких як розмір, колір, модель, матеріал, тип пакування тощо. Основні атрибути сутності ProductItem наведено у таблиці 2.5.

Таблиця 2.5 – Основні атрибути сутності ProductItem

Атрибут	Тип даних	Призначення
id	ID	Унікальний ідентифікатор
productId	ID	Ідентифікатор продукту
sku	String	Складський ідентифікатор товару
name	String	Назва товару
price	Float	Вартість за одиницю товару
status	String	Статус
imageUrl	AWSURL	Посилання на фото
createdAt	AWSDateTime	Дата створення запису
updatedAt	AWSDateTime	Дата зміни запису

Сутність ProducDetail відображає групу характеристик продукту, такі як основні характеристики, додаткові функції, комплектація, габаритні розміри, тощо. Основні атрибути сутності ProducDetail наведено у таблиці 2.6.

Таблиця 2.6 – Основні атрибути сутності ProducDetail

Атрибут	Тип даних	Призначення
id	ID	Унікальний ідентифікатор
productId	ID	Ідентифікатор товару
name	String	Назва товару
type	ProductDetailType	Тип групи характеристик
displayOrder	Int	Поряд відображення
createdAt	AWSDateTime	Дата створення запису
updatedAt	AWSDateTime	Дата зміни запису

Сутність ProductAttribute представляє собою набір характеристик або властивостей, які характеризують конкретний товар або його варіанти. Вони допомагають детально описати товар і розрізняти його від інших товарів, а також забезпечують інформацію, яка може бути корисною для клієнтів при виборі товару. Основні атрибути сутності ProductAttribute наведено у таблиці 2.7.

Таблиця 2.7 – Основні атрибути сутності ProductAttribute

Атрибут	Тип даних	Призначення
id	ID	Унікальний ідентифікатор
productId	ID	Ідентифікатор товару
productItemId	ID	Ідентифікатор варіанту товару
productDetailId	ID	Ідентифікатор групи характеристик
attributeId	ID	Ідентифікатор характеристики
displayOrder	Int	Поряд відображення
units	String	Одиниці виміру характеристики
type	DataType	Тип даних характеристики
value	DataValue	Значення характеристики
createdAt	AWSDateTime	Дата створення запису
updatedAt	AWSDateTime	Дата зміни запису

Сутність Attribute представляє собою характеристики або властивості продукту, які можуть бути використані для опису та класифікації товарів або послуг. Ці атрибути допомагають уточнювати інформацію про продукт, забезпечуючи користувачам більш детальні та повні дані. Кожен атрибут може мати різні типи значень, залежно від специфіки продукту або послуги. Основні атрибути сутності Attribute наведено у таблиці 2.8.

Таблиця 2.8 – Основні атрибути сутності Attribute

Атрибут	Тип даних	Призначення
id	ID	Унікальний ідентифікатор
name	String	Назва характеристики
createdAt	AWSDateTime	Дата створення запису
updatedAt	AWSDateTime	Дата зміни запису

Сутність Inventory представляє собою інформацію про наявність та кількість товарів на складі чи в магазині. Ця сутність містить дані про кожний конкретний продукт або його варіанти, включаючи кількість одиниць, доступних для продажу, та стан запасів. Відстеження цих даних дозволяє оптимізувати процеси постачання та продажу. Основні атрибути сутності Inventory наведено у таблиці 2.9.

Таблиця 2.9 – Основні атрибути сутності Inventory

Атрибут	Тип даних	Призначення
id	ID	Унікальний ідентифікатор
productId	ID	Ідентифікатор товару
productItemId	ID	Ідентифікатор варіанту товару
totalQuantity	Int	Загальна кількість товару
salesQuantity	Int	Кількість проданого товару
reservedQuantity	Int	Кількість зарезервованого товару
createdAt	AWSDateTime	Дата створення запису
updatedAt	AWSDateTime	Дата зміни запису

Сутність Discount представляє інформацію про знижки, які можуть бути застосовані до продуктів та їх варіантів у магазині. Знижки можуть бути сезонними, акційними або залежати від певних умов. Основні атрибути сутності Discount наведено у таблиці 2.10.

Таблиця 2.10 – Основні атрибути сутності Discount

Атрибут	Тип даних	Призначення
id	ID	Унікальний ідентифікатор
productId	ID	Ідентифікатор товару
productItemId	ID	Ідентифікатор варіанту товару

Кінець таблиці 2.10

name	String	Назва знижки
createdAt	AWSDateTime	Дата створення запису
updatedAt	AWSDateTime	Дата зміни запису
expiredAt	AWSTimestamp	Часова відмітка дії знижки

Сутність DiscountItem забезпечує можливості використання диференційованих знижок, тобто знижок які залежності від кількості замовленого товару. Це дозволяє стимулювати покупців до більших замовлень за рахунок знижок, що збільшуються зі збільшенням кількості товару. Основні атрибути сутності DiscountItem наведено у таблиці 2.11.

Таблиця 2.11 – Основні атрибути сутності DiscountItem

Атрибут	Тип даних	Призначення
id	ID	Унікальний ідентифікатор
discountId	ID	Ідентифікатор знижки
quantity	Int	Кількість товару
precentage	Int	Процент знижки
createdAt	AWSDateTime	Дата створення запису
updatedAt	AWSDateTime	Дата зміни запису
expiredAt	AWSTimestamp	Часова відмітка дії варіанту знижки

Сутність Wishlist дозволяє користувачам зберігати товари, які вони бажають придбати у майбутньому або які цікавлять їх для подальшого вивчення. Це допомагає покупцям організувати свої інтереси, планувати покупки та отримувати сповіщення про зміни в наявності чи цінах на товари з їхнього списку бажань. Основні атрибути сутності Wishlist наведено у таблиці 2.14.

Таблиця 2.14 – Основні атрибути сутності Wishlist

Атрибут	Тип даних	Призначення
id	ID	Унікальний ідентифікатор
customerId	ID	Ідентифікатор користувача
productItemId	Int	Ідентифікатор варіанта товару
createdAt	AWSDateTime	Дата створення запису
updatedAt	AWSDateTime	Дата зміни запису

Сутність Review представляє собою інформацію про відгуки, коментарі та оцінки, які залишають клієнти щодо продуктів. Відгуки допомагають потенційним покупцям приймати обґрунтовані рішення та покращують довіру до товарів. Основні атрибути сутності Review наведено у таблиці 2.12.

Таблиця 2.12 – Основні атрибути сутності Review

Атрибут	Тип даних	Призначення
id	ID	Унікальний ідентифікатор
customerId	ID	Ідентифікатор користувача
productItemId	Int	Ідентифікатор варіанта товару
rating	Int	Оцінка користувача
comment	String	Відгук користувача
createdAt	AWSDateTime	Дата створення запису
updatedAt	AWSDateTime	Дата зміни запису

Сутність Rating представляє собою інформацію про загальну оцінку чи рейтинг певного продукту, який надається користувачами. Ця сутність дозволяє відображати середню оцінку продукту на основі відгуків та оцінок клієнтів. Основні атрибути сутності Rating наведено у таблиці 2.13.

Таблиця 2.13 – Основні атрибути сутності Rating

Атрибут	Тип даних	Призначення
id	ID	Унікальний ідентифікатор
productId	ID	Ідентифікатор товару
productItemId	Int	Ідентифікатор варіанта товару
averageRating	Int	Середня оцінка товару
rateCount	Int	Кількість оцінок
createdAt	AWSDateTime	Дата створення запису
updatedAt	AWSDateTime	Дата зміни запису

Сутність Cart представляє собою віртуальний список товарів, які користувач вибрав для покупки. Ця сутність дозволяє клієнтам зберігати вибрані товари перед оформленням замовлення та обчислювати суму покупок перед завершенням операції. Основні атрибути сутності Cart наведено у таблиці 2.15.

Таблиця 2.15 – Основні атрибути сутності Cart

Атрибут	Тип даних	Призначення
id	ID	Унікальний ідентифікатор
customerId	ID	Ідентифікатор користувача
totalAmount	Int	Загальна вартість товарів
createdAt	AWSDateTime	Дата створення запису
updatedAt	AWSDateTime	Дата зміни запису
expiredAt	AWSTimestamp	Часова відмітка життя кошика

Сутність CartItem представляє окремий товар, який додано до кошика покупця. Ця сутність містить інформацію про товар, його кількість в кошику та інші додаткові атрибути, необхідні для обробки замовлення. Основні атрибути сутності CartItem наведено у таблиці 2.16.

Таблиця 2.16 – Основні атрибути сутності CartItem

Атрибут	Тип даних	Призначення
id	ID	Унікальний ідентифікатор
cartId	ID	Ідентифікатор кошика
productItemId	ID	Ідентифікатор варіанту товару
quantity	Int	Кількість товару у кошику
createdAt	AWSDateTime	Дата створення запису
updatedAt	AWSDateTime	Дата зміни запису

Для опису замовлення в магазині вводимо наступні сутності: Address, Recipient, Order, OrderItem Сутність Address в контексті системи управління замовленнями представляє інформацію про адресу, на яку необхідно доставити замовлення покупця. Основні атрибути сутності Address наведено у таблиці 2.17.

Таблиця 2.17 – Основні атрибути сутності Address

Атрибут	Тип даних	Призначення
id	ID	Унікальний ідентифікатор
customerId	ID	Ідентифікатор користувача
country	String	Країна
state	String	Область
city	String	Місто
addressLine1	String	Адреса 1
addressLine2	String	Адреса 2
postalCode	String	Поштовий індекс
isDeleted	Boolean	Флаг видалення
createdAt	AWSDateTime	Дата створення запису
updatedAt	AWSDateTime	Дата зміни запису
deletedAt	AWSDateTime	Дата видалення запису

Сутність Recipient представляє інформацію про особу, яка буде отримувати замовлення. Основні атрибути сутності Recipient наведено у таблиці 2.18.

Таблиця 2.18 – Основні атрибути сутності Recipient

Атрибут	Тип даних	Призначення
id	ID	Унікальний ідентифікатор
customerId	ID	Ідентифікатор користувача
firstName	String	Ім’я одержувача
lastName	String	Прізвище одержувача
email	String	Електрона пошта
phone	String	Номер телефону
isDeleted	Boolean	Флаг видалення
createdAt	AWSDateTime	Дата створення запису
updatedAt	AWSDateTime	Дата зміни запису
deletedAt	AWSDateTime	Дата видалення запису

Сутність Order представляє інформацію про конкретне замовлення покупця, включаючи деталі товарів, їх кількість, ціни, інформацію про отримувача, адресу доставки, статус замовлення та інші важливі атрибути. Основні атрибути сутності Order наведено у таблиці 2.19.

Таблиця 2.19 – Основні атрибути сутності Order

Атрибут	Тип даних	Призначення
id	ID	Унікальний ідентифікатор
customerId	ID	Ідентифікатор користувача
addressId	ID	Ідентифікатор адреси доставки
recipientId	ID	Ідентифікатор одержувача
totalAmount	Float	Загальна вартість

Кінець таблиці 2.19

number	Int	Номер
status	OrderStatus	Статус
comment	String	Коментар до замовлення
createdAt	AWSDateTime	Дата створення запису
updatedAt	AWSDateTime	Дата зміни запису

Сутність OrderItem представляє окремий товар, який входить до складу замовлення. Кожен елемент замовлення містить інформацію про конкретний товар, кількість одиниць, ціну та інші деталі, які характеризують цей товар в рамках замовлення. Основні атрибути сутності OrderItem наведено у таблиці 2.20.

Таблиця 2.20 – Основні атрибути сутності OrderItem

Атрибут	Тип даних	Призначення
id	ID	Унікальний ідентифікатор
orderId	ID	Ідентифікатор замовлення
productItemId	ID	Ідентифікатор варіанту товару
price	ID	Ціна за одиницю
amount	Float	Загальна вартість з урахуванням знижки
quantity	Int	Кількість замовленого товару
discount	OrderStatus	Знижка
createdAt	AWSDateTime	Дата створення запису
updatedAt	AWSDateTime	Дата зміни запису

Для реалізації пошуку у мобільному застосунку за фільтрами, що визначаються певними категоріями товарів, вводимо наступні сутності: CategoryFilterField, CategoryFieldOption. Сутність CategoryFilterField представляє собою атрибут категорії товарів, за допомогою якого можна

встановлювати фільтри під час пошуку товарів. Основні атрибути сутності CategoryFilterField наведено у таблиці 2.21.

Таблиця 2.21 – Основні атрибути сутності CategoryFilterField

Атрибут	Тип даних	Призначення
id	ID	Унікальний ідентифікатор
categoryId	ID	Ідентифікатор категорії
attributeId	ID	Ідентифікатор характеристики
type	DataType	Тип даних характеристики
displayOrder	Int	Порядок відображення
createdAt	AWSDateTime	Дата створення запису
updatedAt	AWSDateTime	Дата зміни запису

Сутність CategoryFieldOption представляє варіант вибору для конкретного поля фільтрації категорії товарів. Ця сутність дозволяє створювати різні опції для поля фільтра, щоб користувачі могли вибирати бажані параметри для фільтрації товарів у відповідності з їхніми вимогами та уподобаннями. Основні атрибути сутності CategoryFieldOption наведено у таблиці 2.22.

Таблиця 2.22 – Основні атрибути сутності CategoryFieldOption

Атрибут	Тип даних	Призначення
id	ID	Унікальний ідентифікатор
fieldId	ID	Ідентифікатор поля пошуку
name	String	Назва
value	DataValue	Значення
range	DataRange	Діапазон значень
displayOrder	Int	Порядок відображення
createdAt	AWSDateTime	Дата створення запису
updatedAt	AWSDateTime	Дата зміни запису

Сутність StaticPage представляє собою набір статичних сторінок, які мають постійний зміст та не змінюються часто. Ці сторінки зазвичай призначені для відображення інформації, яка не потребує частого оновлення і залишається сталою протягом тривалого часу. Основні атрибути сутності StaticPage наведено у таблиці 2.23.

Таблиця 2.23 – Основні атрибути сутності StaticPage

Атрибут	Тип даних	Призначення
id	ID	Унікальний ідентифікатор
type	ID	Тип контенту
content	String	Зміст сторінки
createdAt	AWSDateTime	Дата створення запису
updatedAt	AWSDateTime	Дата зміни запису

Для моделювання структури даних та взаємодії звязків між ними використано діаграми зв'язків (Entity-Relationship diagrams, ER diagrams) [18]. Вони використовуються для графічного зображення сутностей, атрибутів і зв'язків між цими сутностями. Основна мета діаграм зв'язків – це надати зрозуміле та структуроване відображення бази даних. Вони допомагають розробникам чітко уявити структуру даних, що дозволяє правильно спроектувати схему бази даних та візуалізують, як сутності взаємодіють між собою, що дозволяє краще розуміти логіку системи.

У рамках даної роботи було розроблено багато діаграм зв'язків, проте так як загалом вони дуже схожі, буде розглянута лише одна з них, тоді як решта наведена у додатку Б.

Діаграма зв'язків «Товар і його атрибути» представлена на рисунку 2.3. Кожен товар належить певному бренду, зв'язок встановлюється через поле brandId у таблиці Products, що вказує на відповідний бренд у таблиці Brands.

Варіанти продуктів пов'язані з товарами через поле productId у таблиці Products. Товар має групу характеристик, зв'язок з якими встановлюється також через поле productId. Кожна група характеристик містить атрибути, зв'язок між якими здійснюється через поле productDetailId. Кожен атрибут пов'язаний з товаром або його варіантом через поля productId та productItemId, а з атрибутом у таблиці Attributes – через поле attributeId.

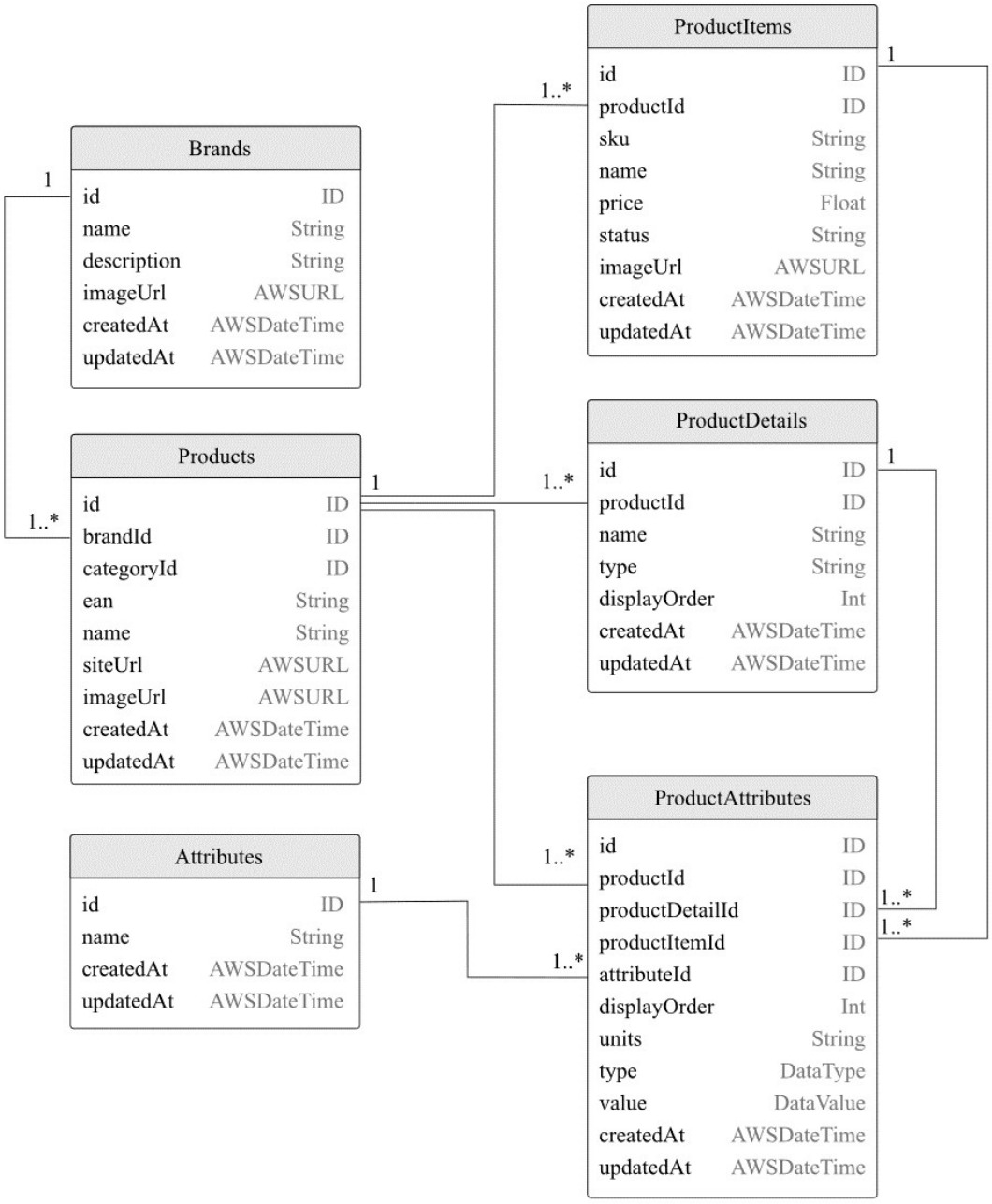


Рисунок 2.3 – Діаграма звязків «Товар і його атрибути»

У процесі проєктування бази даних для Інтернет-магазину було визначено та спроєктовано низку важливих сутностей, які відображають основні об'єкти та концепції системи. Для покращення гнучкості та ефективності бази даних були застосовані певні патерни проєктування. Для відображення взаємозв'язків між сутностями та їх атрибутами були створені відповідні діаграми зв'язків.

2.3 Проєктування функціональних можливостей застосунку

Проєктування функціональних можливостей мобільного застосунку здійснюється на основі аналізу діаграми варіантів використання (рисунок 2.4), яка розроблена з урахуванням вимог до мобільного застосунку. Цей підхід дозволяє чітко визначити, які функції та можливості повинен мати застосунок для задоволення потреб користувачів та досягнення поставлених цілей. Діаграма варіантів використання відображає різні сценарії взаємодії користувачів із застосунком та ілюструє послідовність дій, які користувачі можуть виконувати [19]. Вона також допомагає виявити можливі точки покращення та оптимізації функціоналу, забезпечуючи більш точне та ефективне планування розробки.

Опис усіх варіантів використання здійснюємо за допомогою діаграм послідовності. Діаграма послідовності – це графічний інструмент, який використовується для моделювання та візуалізації послідовності взаємодій між різними об'єктами чи компонентами в системі в рамках конкретного сценарію (варіанту використання) [20]. Діаграма дозволяє показати порядок виконання операцій, передачу повідомлень та взаємодію між об'єктами в зрозумілій формі. В контексті використання безсерверної архітектури акторами в діаграмі послідовності виступають: користувач, мобільний застосунок та сервіси AWS. До прикладу розглянемо діаграму постідовності варіанту «Реєстрація» (рисунок 2.5). Решта аналогічних діаграм наведена у додатку А.

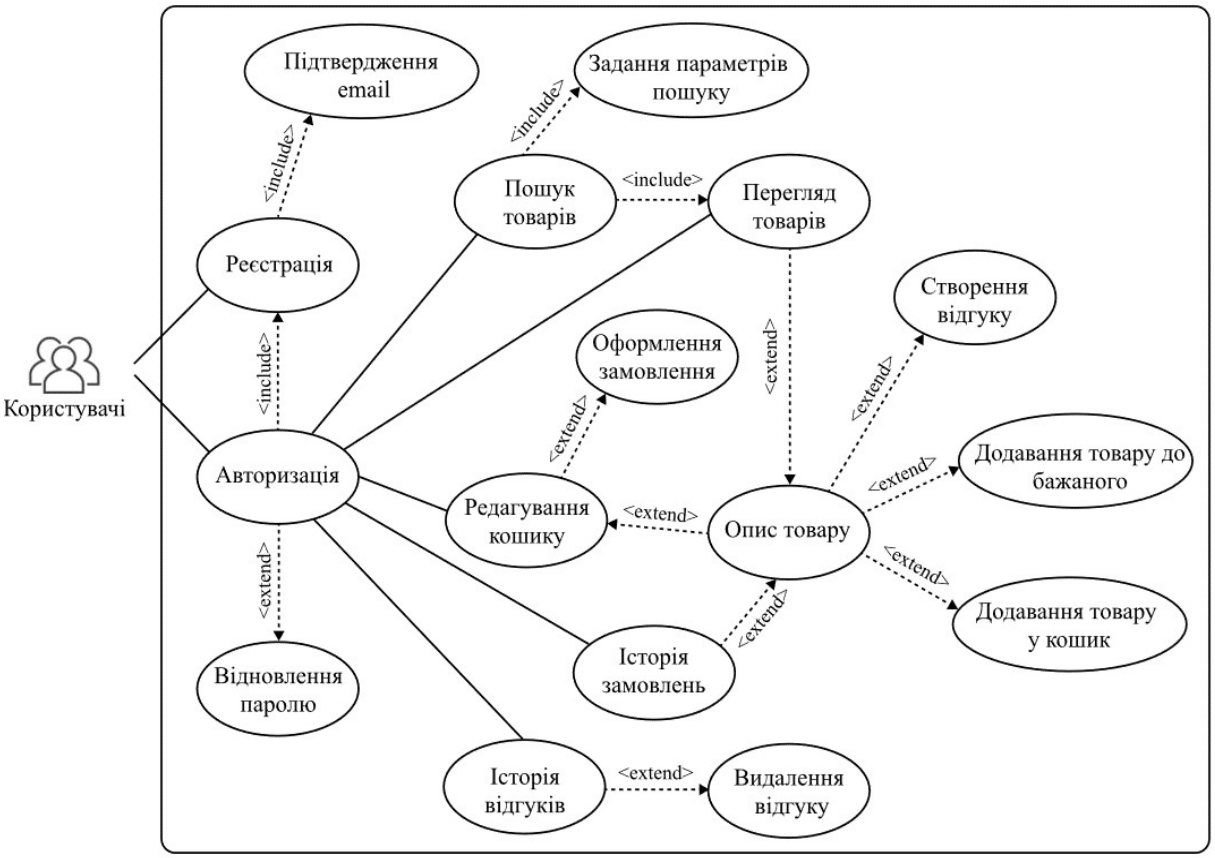


Рисунок 2.4 – Діаграма варіантів використання

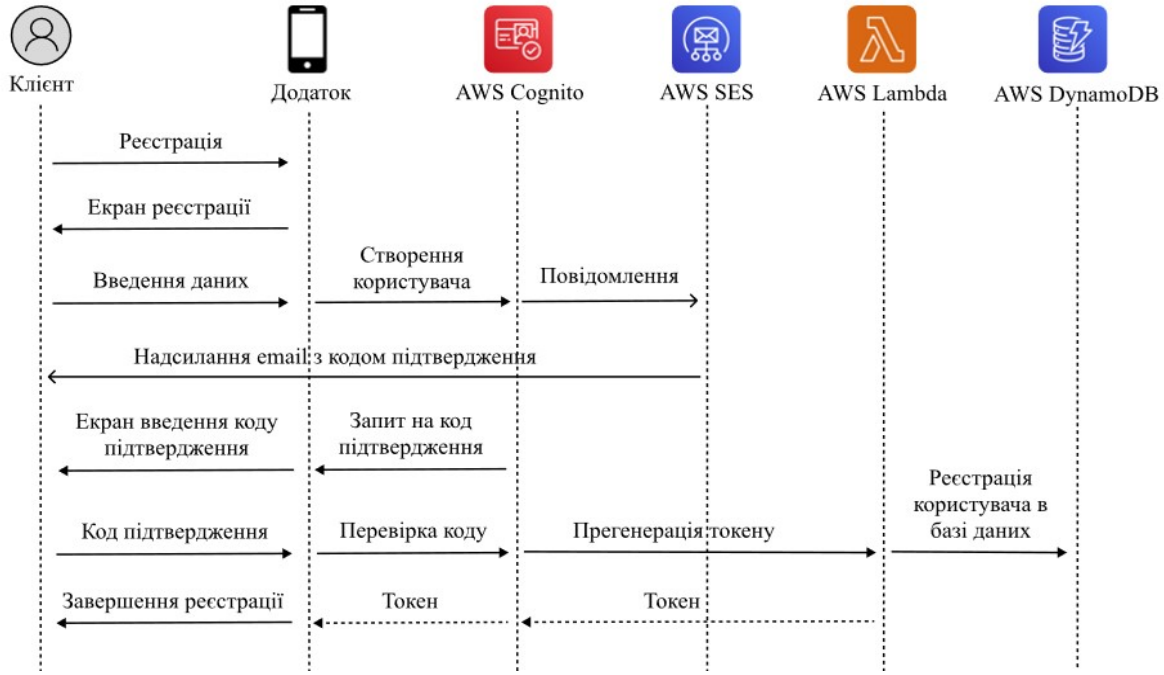


Рисунок 2.5 – Діаграма послідовності варіанту «Реєстрація»

Варіант «Реєстрація» реалізує можливість створення акаунта для нового користувача. Користувач має заповнити обов`язкові поля форми реєстрації (ім`я, прізвище, email). Сервіс AWS Cognito верифікує email шляхом надсилання коду підтвердження на email користувача через сервіс AWS SES. Користувач вводить код підтвердження з надісланого email. Застосунок надсилає код в сервіс Cognito де відбувається його перевірка. Уразі успіху сервіс AWS Cognito генерує токен для користувача. AWS Lambda перехоплює подію генерації токена та реєструє користувача в базі даних. На цьому варіант «Реєстрація» завершується.

Результати аналізу варіантів використання та розглянуті сценарії виступають як важлива основа при реалізації мобільного застосунку для Інтернет-магазину. Ці сценарії дозволять створити зручний та функціональний застосунок, який відповідатиме потребам сучасного споживача.

2.4 Проєктування інтерфейсу користувача

Основними принципами проектування інтерфейсу користувача для мобільного застосунку є простота та зрозумілість, контекстуальність, зручна навігація, привабливий дизайн, відповідність стандартам платформи та тестування з користувачами. У цьому процесі важливо створювати зручне та привабливе середовище для користувачів, де вони легко зможуть знаходити необхідні функції та взаємодіяти з застосунком. Дизайн повинен бути спрощеним, сучасним та відповідати вимогам конкретної мобільної платформи. Крім того, тестування з реальними користувачами допомагає виявляти проблеми та вдосконалювати інтерфейс для досягнення кращого досвіду взаємодії. Візуальними представленнями дизайну інтерфейсу користувача у проектуванні є макети. Вони використовуються для показу структури, розміщення елементів та взаємодії в рамках мобільного застосунку.

Figma – це вебплатформа для дизайну, яка включає в себе засоби для створення макетів, прототипів та спільної роботи команди над дизайном інтерфейсу користувача [21]. У ній можна створювати різні типи макетів, від простих скетчів до деталізованих прототипів. Програма також дозволяє створювати макети для різних платформ та взаємодіяти з елементами інтерфейсу за допомогою векторних об'єктів та компонентів.

Усі макети екранів мобільного застосунку наведено у додатку В. До прикладу можна розглянути макети екранів каталогу товарів, що зображено на рисунку 2.6.

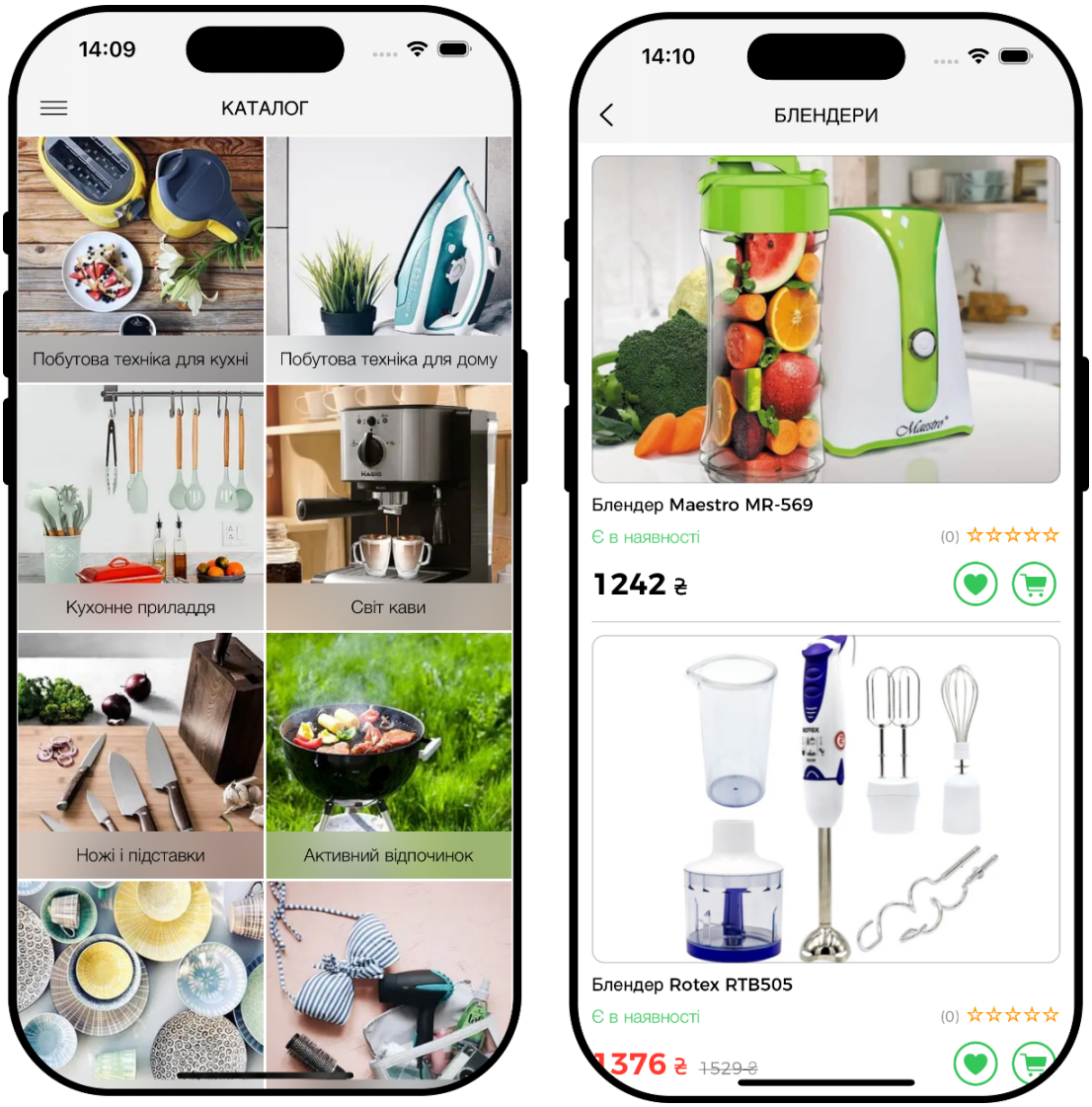


Рисунок 2.6 – Макети екранів каталогу товарів

На екрані каталогу товарів користувач зустріне перелік категорій у вигляді окремих плиток, кожна з яких містить малюнок та назву категорії. Після вибору категорії відображається перелік товарів, що входять до цієї категорії. Кожен товар представлений у вигляді окремого інформаційного блоку, що містить фотографію товару, його назву, ціну, рейтинг та статус наявності. Для зручності користувача в блоку товару є кнопки для додавання товару у кошик та додавання його до списку бажаного.

Розроблені макети інтерфейсів є важливим етапом у процесі проектування мобільного додатку. Вони дозволили візуалізувати та протестувати інтерфейс заздалегідь, визначити оптимальне розташування елементів, вигляд та функціонал, що забезпечує високу користувацьку зручність та ефективність використання.

2.5 Аналіз та вибір технологій і методів реалізації застосунку

Взаємодія між клієнтом та сервером відбувається через API, який визначає правила та протоколи взаємодії між програмами та сервісами. Основні підходи до реалізації взаємодії через API – це REST (Representational State Transfer) та GraphQL. REST базується на принципах CRUD (створення, читання, оновлення, видалення) через HTTP методи: GET, POST, PUT та DELETE. GraphQL, розроблений Facebook, є мовою запитів та середовищем виконання запитів, що дозволяє клієнтам отримувати лише необхідні дані, зменшуючи кількість запитів та відповіді. GraphQL спрощує розробку API, оскільки клієнти можуть запитувати різні дані з одного запиту, що зменшує кількість мережевих взаємодій та ефективно використовує кешування даних.

GraphQL є строго типізованим протоколом, де всі операції перевіряються за схемою GraphQL. Схема визначає форму даних та операції, які можна виконувати, і складається з типів об'єктів, скалярів, типів введення, інтерфейсів, переліків та об'єднань. Вона визначає три типи операцій верхнього рівня: Query

для вибірки даних (лише для читання), mutation для операцій зміни даних та subscription, що є довготривалим з'єднанням для отримання даних.

AWS SDK (Amazon Web Services Software Development Kit) – це набір інструментів для розробки програм, що взаємодіють з послугами AWS [22]. SDK включає бібліотеки, приклади коду та документацію для спрощення інтеграції з AWS. AWS SDK забезпечує розробникам інструменти для розробки, тестування та розгортання програм на AWS, спрощуючи роботу з хмарними застосунками.

.NET – це платформа розробки від Microsoft, яка підтримує крос-платформенну розробку. Одним з головних компонентів .NET є мова програмування C# [23]. Вона створена для розробки на платформі .NET і має зрозумілий синтаксис, що дозволяє ефективно створювати додатки різного типу. C# популярна завдяки потужній функціональності та широкому спектру бібліотек, особливо у корпоративній розробці. У AWS Lambda можна розробляти лямбда-функції на C#.

Swift – це мова програмування від Apple для платформ iOS, macOS, watchOS та tvOS [24]. Вона має чистий синтаксис, що робить код зрозумілішим. Swift підтримує об'єктно-орієнтоване, функціональне та протокольне програмування, що робить його потужним і гнучким інструментом. UIKit - це набір інструментів для створення інтерфейсу користувача в додатках для iOS. Він включає елементи інтерфейсу, такі як кнопки, текстові поля та таблиці, дозволяючи створювати стильні та функціональні додатки.

CocoaPods – це менеджер залежностей для розробки під iOS та macOS. Він дозволяє легко встановлювати та керувати залежностями в проектах. Однією з ключових переваг CocoaPods є простота використання - розробники можуть додавати нові залежності за допомогою простих команд, а CocoaPods автоматично завантажує та встановлює необхідні файли.

OpenSearch − це відкрите програмне забезпечення для пошуку та аналітики, засноване на Elasticsearch та Kibana. Воно призначене для швидкого

пошуку великих обсягів даних у реальному часі, забезпечуючи розширені можливості для індексації та пошуку у розподілених системах.

Google Analytics – це інструмент для відстеження та аналізу поведінки користувачів у мобільних застосунках [25]. Він надає важливі дані для вдосконалення застосунку, що забезпечує кращий користувацький досвід та результативність. Використання Google Analytics допомагає виокремити ключові тренди та розуміти, як користувачі взаємодіють з вашим застосунком, що є критично важливим для ефективної стратегії розвитку.

Отже, на основі аналізу сучасних засобів для розроблення мобільного застосунку визначено оптимальний технологічний стек. Для реалізації цілей обрано: GraphQL для взаємодії клієнта з сервером; AWS SDK для хмарних послуг та інтеграції з AWS-інфраструктурою; .NET та C# для написання лямбда-функцій; Google Analytics для відстеження використання застосунку; Swift для швидкого і ефективного розроблення функціональності та інтерфейсу користувача. Такий підхід дозволяє створювати надійні та продуктивні мобільні застосунки, які відповідають сучасним технологіям.

2.6 Висновки

На основі порівняльного аналізу обрано безсерверну архітектуру для розроблення мобільного застосунку. Розглянуто сценарії використання сервісів AWS та обрано найбільш відповідний до потреб застосунку варіант.

Визначено та спроектовано низку важливих сутностей, які відображають основні об'єкти та концепції системи. Застосовано патерни проєктування та побудовано діаграми зв'язків.

Розроблено та проаналізовано діаграму варіантів використання. Розглянуто можливі варіанти використання, побудовано діаграми послідовності у контексті використання сервісів AWS.

Розроблено макети мобільних інтерфейсів, що дозволило візуалізувати та протестувати інтерфейси заздалегідь, визначити оптимальне розташування елементів, вигляд та функціонал.

На основі проведеного аналізу сучасних засобів та технологій для розроблення мобільного застосунку визначено оптимальний технологічний стек. Обрано набір технологій, інструментів, фреймворків, мов програмування та інших компонентів, які будуть використовуватися для розроблення застосунку.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ МОБІЛЬНОГО ЗАСТОСУНКУ

3.1 Реалізація сервісів мобільного застосунку

3.1.1 Налаштування сервісу Cognito

Робота з мобільним застосунком передбачає наявність механізмів аутентифікації та авторизації користувачів, що можна легко реалізувати за допомогою сервісу AWS Cognito. Сервіс Cognito забезпечує безпечну аутентифікацію користувачів за допомогою різних методів, таких як електронна пошта, номер телефону або соціальні мережі, а також управління доступом до ресурсів за допомогою ролей та політик авторизації. Процес створення нового пула користувачів здійснюється в декілька етапів, на кожному із яких налаштовуємо певну групу параметрів. Основні налаштування параметрів, які були здійснені в процесі створення наведено в таблиці 3.1.

Таблиця 3.1 – Параметри налаштування пула користувачів

Назва параметра	Значення
Тип провайдера	Cognito user pool
Параметри входу	User name
Режим політики паролів	Cognito defaults
Двофакторна аутентифікація	No MFA
Спосіб доставки повідомлень	Email only
Самореєстрація	Enable
Атрибути для перевірки	Email address
Обов'язкові атрибути	Email
Назва пула користувачів	eShop

Наступним кроком є налаштування пула ідентифікацій. Пул ідентифікацій – це компонент, який дозволяє управляти ідентичністю користувачів та

керувати доступом до AWS ресурсів. Його основна мета – це надання користувачам або групам користувачів обмеженого доступу до різних AWS послуг або функцій, в залежності від їхніх ролей та налаштувань. Основні налаштування параметрів пула ідентифікацій, наведено в таблиці 3.2.

Таблиця 3.2 – Параметри налаштування пула ідентифікацій

Назва параметра	Значення
Доступ користувача	Authenticated access, Guest access
Аутентифіковані джерела ідентичності	Amazon Cognito user pool
Аутентифікована роль	eShop.Cognito.IdentityPool.Auth
Гостьова роль	eShop.Cognito.IdentityPool.Unauth
Налаштування ролі	Use default authenticated role
Відображення підтверджень	Use default mappings
Назва пулу ідентифікації	eShop
Базова аунтентифікація	Activate basic flow

3.1.2 Налаштування сервісу DynamoDB

Для зберігання сутностей визначених у розділі проектування структури бази даних використовуємо сервіс DynamoDB. Даний сервіс надає гнучкий підхід до зберігання даних без жорсткої схеми, що дозволяє додавати та змінювати атрибути для кожної сутності без необхідності зміни всієї структури бази даних. DynamoDB – це повністю керований сервіс бази даних NoSQL від Amazon Web Services (AWS). Він забезпечує гнучкість, швидкість та масштабованість для зберігання та обробки структурованих даних.

Для підвищення швидкості виконання запитів до бази даних та поліпшення її продуктивності використовуються індекси. Вони дозволяють швидше виконувати пошук та фільтрацію даних за вказаними умовами, такими як значення атрибутів чи діапазони значень. Використання індексів допомагає

оптимізувати запити та зменшувати час на обробку, що робить роботу з базою даних більш ефективною та продуктивною.

Для отримання змін, які відбуваються в таблицях DynamoDB в реальному часі використовується сервіс DynamoDB Stream. Цей сервіс дозволяє реагувати на додавання, оновлення або видалення записів у таблиці, виконувати певні дії на основі цих змін та інтегрувати DynamoDB з іншими сервісами чи застосунками для автоматизації обробки даних або реалізації реактивної логіки. Наприклад, можна використати DynamoDB Stream для сповіщення про зміни в базі даних і відразу відправити ці зміни на аналіз, або автоматично оновити дані в інших системах.

Для автоматичного видалення застарілих даних або даних, які більше не потрібні після певного часу, використовуємо механізм TTL (Time-To-Live), який дозволяє визначити час життя об'єкта в базі даних DynamoDB. Встановлюючи TTL для певного об'єкта, ви вказуєте системі, через який час цей об'єкт повинен бути автоматично видалений з бази даних.

Для керування доступом користувачів та сервісів до бази даних, забезпечуючи безпеку та контроль над обмеженнями дій з даними використовуємо політики. Політики - це набір правил і обмежень, які визначають доступ до ресурсів бази даних DynamoDB. Вони визначають, які дії (наприклад, читання, запис, оновлення, видалення) і які ресурси (таблиці, індекси) можуть бути виконані користувачами чи ролями в системі. Політики для DynamoDB встановлюються за допомогою AWS Identity and Access Management (IAM).

3.1.3 Налаштування сервісу S3 Storage

Для зберігання різноманітних типів інформації, таких як медіафайли та документи використовуємо сервіс Amazon S3 (Simple Storage Service) (рисунок 3.3). Даний сервіс використовується для зберігання, керування та розподілу

великих обсягів даних в хмарному середовищі. Основні налаштування параметрів створення сховища в S3 наведено у таблиці 3.3.

Таблиця 3.3 – Параметри налаштування сховища в S3

Назва параметра	Значення
AWS Регіон	Europe (Stockholm) eu-north-1
Тип сховища	General purpose
Назва сховища	eshop-hosted-content
Список контролю доступу	Yes
Блокувати весь публічний доступ	Off
Шифрування	Amazon S3 managed keys (SSE-S3)

Правила і дозволи, які користувачі можуть мати для доступу до об'єктів у S3, визначаються відповідними політиками та списком контролю доступу (ACL). Політики встановлюють права на доступ до об'єктів, такі як читання, запис, видалення тощо, для конкретних користувачів, груп користувачів або за визначеними умовами. Ці політики дозволяють гнучко керувати доступом на різних рівнях.

ACL (Access Control List) - це механізм керування доступом до об'єктів, який визначає права доступу до конкретного об'єкта для різних користувачів. ACL може містити різні рівні доступу, що дозволяє точно налаштовувати, які користувачі чи групи мають право взаємодіяти з об'єктом у хмарному сховищі.

Приклад політики для доступу до об’єктів у папці public наведено нижче:

```
{
    "Version": "2012-10-17",
    "Statement": [{
            "Action": [ "s3:GetObject" ],
            "Resource": [
                "arn:aws:s3:::eshop-hosted-content/public/*"
            ],
            "Effect": "Allow"
        }
    ]
}
```

3.1.4. Налаштування поштового сервісу SES

Для надсилання електронних листів на поштові скриньки користувачів, використовуємо поштовий сервіс Simple Email Service (SES). Даний сервіс, надає можливість відправляти електронні листи з використанням інфраструктури Amazon. Він дозволяє легко і безпечно надсилати повідомлення, включаючи підтвердження реєстрації, деталі замовлення, сповіщення та інші.

Для отримання запитів, питань і скарг від клієнтів та зворотної комунікації, створюємо окрему поштову скриньку. Забезпечення доступ до облікового запису без необхідності розголошення основного пароля реалізуємо через пароль застосунку. Паролі застосунків (App Passwords) – це спеціальні, 16-значні коди, які використовуються для забезпечення безпеки доступу в сервісах, які не підтримують двофакторну аутентифікацію (2FA) через SMS або додаток аутентифікації.

Створену поштову скриньку реєструємо в SES та підтверджуємо її електрону адресу. Після підтвердження ідентифікатора електронної пошти, її можна використовувати у сервісі Cognito для відправки кодів підтвердження реєстрації та зміни паролю. Додатково розроблено шаблон електронного листа з інформацією про замовлення, що надсилається користувачу.

3.1.5 Налаштування сервісу OpenSearch

Можливість пошуку за такими параметрами, як ціна, категорія товару, виробник, наявність на складі та інші важливі характеристики реалізується за допомогою концепції наскрізного пошуку. Наскрізний пошук використовує можливості пошуку за індексом даних, що дозволяє швидко знаходити потрібні товари за будь-якими вказаними атрибутами без зайвого перебору всього асортименту.

Для реалізації наскрізного пошуку товарів за певними критеріями та атрибутами використовуємо сервіс OpenSearch. OpenSearch – це відкрита, розподілена система пошуку та аналізу даних, яка базується на технологіях Elasticsearch, Kibana і Logstash. Вона надає можливості для швидкого і надійного пошуку, агрегації та візуалізації даних, включаючи текстовий, числовий, географічний та інший вид пошуку. OpenSearch дозволяє легко масштабувати систему для обробки великих обсягів даних. Для аналізу та візуалізації даних використовуємо OpenSearch Dashboard, який дозволяє створювати зручні та інформативні панелі керування для аналізу даних та моніторингу їх стану. Для взаємодії з OpenSearch використовується програмний інтерфейс, що дозволяє виконувати індексацію, пошук та агрегацію даних. OpenSearch надає клієнти для різних мов програмування та платформ, включаючи C#, Java, Python, Ruby та інші.

3.1.6 Налаштування сервісу Lambda

Реалізація бізнес логіки відбувається через написання функцій, які виконують необхідні операції та обробляють дані відповідно до потреб застосунку. Вони включають в себе обробку вхідних подій, взаємодію з іншими AWS сервісами (наприклад, S3, DynamoDB, Cognito тощо), виконання розрахунків, обробку запитів від клієнтів та інші бізнес-логічні завдання. AWS Lambda дозволяє ефективно використовувати обчислювальні, що робить його ідеальним інструментом для реалізації бізнес-логіки в хмарному середовищі. Реалізація Lambda-функцій може бути здійснена на мовах програмування, таких як Python, Node.js, Java, C# та інші, з використанням можливостей викликів інших сервісів та бібліотек для роботи з даними та логікою додатку.

Програмна реалізація усіх Lambda-функцій виконана окремим рішенням eShop.Lambda у середовищі Microsoft Visual Studio на мові C# із використанням набору інструментів AWS SDK Toolkit, що надає розробникам доступ до AWS

API та сервісів, дозволяє легко і швидко інтегрувати AWS функціональність у свої застосунки та сервіси. Перелік усіх проектів рішення eShop.Lambda та їх призначення наведено у таблиці 3.4.

Таблиця 3.4 – Перелік проектів рішення eShop.Lambda та їх призначення

Назва проекту	Призначення
eShop.Lambda.Common.	Містить перелік моделей сутностей спільних для усіх проектів Lambda-функцій
eShop.Lambda.Console	Кросплатформений консольний застосунко для оновлення та завантаження даних в DynamoDB, медіаресурсів у S3 Storage та реіндексації товарів у OpenSearch
eShop.Lambda.*	Реалізація відповідних Lambda-функції

Структура проектів усіх Lambda-функцій подібна один до одного. Кожен проект містить головний клас Function, у якому реалізована бізнес-логіка функції та файл конфігурації. Кожен клас функції містить метод, що приймає вхідні дані, виконує обробку цих даних та повертає результат обробки. Lambda функція може мати різні тригери, такі як виклики API Gateway, події S3 або DynamoDB, або розгортатися як реакція на інші події в AWS середовищі. Коли тригер викликає функцію, AWS Lambda запускає код функції та обробляє вхідні дані згідно з логікою, описаною в коді. В окремих проектах eShop.Lambda.Orders та eShop.Lambda.Reviews присутня папка Models у якій містяться класи, специфічні для кожної функції (класи вхідних параметрів та результати виконання).

Після тестування та відладки, код функції упаковується в архів, наприклад, у форматі ZIP, разом з файлом конфігурації та пакетом залежностей. Цей архів завантажується на AWS Lambda через консоль керування або за

допомогою інструментів розробки. Проект eShop.Lambda.Console не є проектом Lambda-функції, а носить утилітарний характер. Його призначення – розгортання даних в сервісах DynamoDB, S3 Storage та OpenSearch. Формат даних для розгортання файл .xlsx.

3.1.7 Налаштування сервісу AppSync

Для створення API з доступом до різних джерел таких як бази даних DynamoDB, AWS Lambda, OpenSearch та інші, використовуємо сервіс AWS AppSync. Він забезпечує реальний час спілкування між клієнтом і сервером, автоматичну синхронізацію даних і можливість роботи в автономному режимі. Цей сервіс особливо корисний для розробки додатків з вимогами до реального часу, таких як месенджери, колаборативні інструменти та інші програми, де важлива швидка та точна синхронізація даних. Основні параметри налаштування для сервісу AppSync наведено в таблиці 3.5.

Таблиця 3.5 – Параметри налаштування сервісуAppSync

Назва параметра	Значення
Тип API	GraphQL
Назва API	eShop
Приватні функції	Off
Ведення журналу	On
Трасування	Off
Запити інтроспекції	On
Глибину запиту	Off
Лічильник розв`язувачів	Off
Основний режим авторизації	Amazon Cognito User Pool
Додатковий режим авторизації	AWS Identity and Access Management (IAM)

Джерела даних у AWS AppSync визначають, звідки сервіс буде отримувати дані. Це можуть бути бази даних, зовнішні API, Lambda функції та інші. Для AppSync джерела даних є ключовим елементом, оскільки вони визначають, як дані будуть доступні та оброблятимуться через API.

Для кожної сутності визначеної у розділі проектування структури бази даних розроблено п`ять базових функцій (Get, List, Create, Update, Delete), які використовують відповідні джерела даних.

Структура даних та операції, які можна виконувати через API визначаються GraphQL схемою. Вона включає типи даних, запити (queries) для отримання даних та мутації (mutations) для зміни даних. Фрагмент схеми із визначення типу даних:

```
type Product @aws_iam @aws_cognito_user_pools {
    id: ID!
    brandId: ID!
    brand: Brand!
    categoryId: ID!
    ean: String!
    name: String!
    siteUrl: AWSURL
    imageUrl: AWSURL
    items: [ProductItem!]
    createdAt: AWSDateTime!
    updatedAt: AWSDateTime!
}

type ProductConnection @aws_iam @aws_cognito_user_pools {
    items: [Product!]
    nextToken: String
}
```

З'єднання з джерелами даних і виконання необхідних операцій для отримання або зміни даних здійснюється за допомогою розв’язувачів. Кожен розв’язувач під'єднується до конкретного типу запиту або мутації у GraphQL схемі і визначає, як саме ці операції повинні бути виконані. Реалізації GraphQL API завершується виконанням тестування та перевіркою на правильність інтеграції з джерелами даних.

3.1.8 Інтеграція сервісів із використанням Amplify

Інтеграція налаштованих сервісів AWS у мобільний застосунок Інтернет-магазину здійснюється за допомогою AWS Amplify. Даний сервіс призначений для швидкої і зручної розробки хмарних застосунків та надає набір інструментів і бібліотек, що спрощують процес розробки. AWS Amplify CLI є одним із інструментів для роботи через інтерфейс командного рядка із застосунка Terminal. Послідовність команд необхідних для інтеграція сервісів наведено у таблиці 3.6.

Таблиця 3.6 – Послідовність команд Amplify CLI

Команда	Призначення
$ amplify init	Ініціалізація нового проекту Amplify
$ amplify import auth	Імпорт налаштувань сервісу Cognito
$ amplify import storage	Імпорт налаштувань сервісу S3 Storage
$ amplify codegen add --apiId	Генерація GraphQL (запити, мутації)
$ amplify push.	Збереження конфігурації проекту

У результаті виконання перерахованих команд створюється файл awsconfiguration.json та файл GraphQL клієнта, на основі схеми, визначеної в AppSync. Файл awsconfiguration.json містить ідентифікатори ресурсів та інші параметри, необхідні для взаємодії з AWS. Файл GraphQL клієнта містить автоматично згенеровані моделі даних та запити, які відповідають схемі, що дозволяє уникнути необхідності ручного написання цього коду.

3.2 Реалізація API клієнта

API клієнт є важливою складовою розробки мобільного застосунку, оскільки дозволяє отримувати, відправляти та оброблювати дані, керувати

автентифікацією, забезпечувати безпеку даних та оптимізувати продуктивність застосунку в цілому через кешування та оптимізацію запитів. Реалізацію API клієнта представлено у класі ApiClient. Основні методи та їх призначення наведено у таблиці 3.7.

Таблиця 3.7 – Основні методи ApiClient та їх призначення

Назва методу	Призначення
getUserAttributes	Отримує атрибути користувача, такі як ім'я, електронна адреса, тощо.
getCart	Отримує вміст кошика користувача.
createCartItem	Створює новий елемент в кошику з вказаними параметрами, такими як ідентифікатор товару та кількість.
updateCartItem	Оновлює існуючий елемент в кошику з новою кількістю або іншими параметрами.
deleteCartItem	Видаляє елемент з кошика користувача.
createWishlist	Створює новий список бажань для користувача.
deleteWishlist	Видаляє список бажань користувача.
getWishlists	Отримує список бажань користувача.
getCategoriesByParent	Отримує категорії за батьківським ідентифікатором.
getProductsByCategory	Отримує товари за категорією.
getProductDetails	Отримує детальну інформацію про конкретний товар.
getProduct	Отримує інформацію про товар за його ідентифікатором.
createOrder	Створює нове замовлення з вказаними параметрами.

Кінець таблиці 3.7

getOrders	Отримує список замовлень користувача.
getRecipients	Отримує список отримувачів для доставки замовлень.
getRecipient	Отримує інформацію про конкретного отримувача за його ідентифікатором.
createRecipient	Створює нового отримувача для доставки замовлень.
updateRecipient	Оновлює інформацію про існуючого отримувача.
deleteRecipient	Видаляє інформацію про отримувача.
getAddresses	Отримує адреси користувача.
getAddress	Отримує інформацію про конкретну адресу за її ідентифікатором.
createAddress	Створює нову адресу для користувача.
updateAddress	Оновлює інформацію про існуючу адресу.
deleteAddress	Видаляє інформацію про адресу.
createReview	Створює відгук на товар.
updateReview	Оновлює існуючий відгук на товар.
deleteReview	Видаляє відгук на товар.
getReview	Отримує відгук на товар за його ідентифікатором.
getReviews	Отримує список відгуків на товар.
getProductItems	Отримує список товарів.
getProductItemsWithDiscount	Отримує список товарів зі знижкою.
getBrands	Отримує список брендів товарів.
getProductByBrand	Отримує товари за певним брендом.
getCategories	Отримує список категорій товарів.
searchProductItems	Проводить пошук товарів за вказаними параметрами фільтрації.

Фрагмент коду ApiClient для створення замовлення представлено нижче:

```
public func createOrder(cartId: String,
                        recipientId: String,
                        addressId: String,
                        comment: String?,
                        completion: @escaping (Order?, ApiError?) -
> Void) {

    let input = CreateOrderByCartInput(cartId: cartId, addressId:
addressId, recipientId: recipientId, comment: comment)
    let mutation = CreateOrderByCartMutation(input: input)
    appSync.perform(mutation: mutation, queue: .global(),
optimisticUpdate: nil, conflictResolutionBlock: nil) { result,
error in
        if let error = error {
            completion(nil, ApiError(error.localizedDescription))
            return
        }

        if let result = result {
            let apiError = (result.errors?.isEmpty ?? true) ? nil :
ApiError(result.errors!)
            if let snapshot =
result.data?.createOrderByCart?.snapshot {
                let order = Order(snapshot: snapshot)
                completion(order, apiError)
            } else {
                completion(nil, apiError)
            }
        } else {
            completion(nil, nil)
        }
    }
}
```

Функція createOrder приймає ідентифікатор кошика, отримувача та адреси, а також необов'язковий коментар. Вона створює вхідні дані для мутації, яка виконується за допомогою appSync. У випадку успішного виконання мутації, функція створює об'єкт замовлення з отриманих даних.

Для представлення помилок, пов'язаних з виконанням запитів до API використовується клас ApiError. Основна його функція полягає у створенні об'єктів, які містять інформацію про помилку, що виникла під час виконання

певного запиту. Це може бути корисно для оброблення помилок і відображення відповідного повідомлення користувачеві або в іншій логіці програми.

3.3 Реалізація мобільного застосунку

Програмна реалізація проекту мобільного застосунку інтернет магазину з продажу побутових товарів, виконана для платформи iOS, в середовищі розробки Xcode, з використання фреймворку UIKit, на мові програмування Swift. Для розробки екранів мобільного застосунку використано шаблон Model-View-Controller (MVC) [26]. Він дозволяє розділити логіку на три основні частини: модель, представлення і контролер. Модель представляє дані та бізнес-логіку, представлення відповідає за відображення інформації користувачеві, контролер керує взаємодією між моделлю та представленням.

Структура проекту мобільного застосунку представлена двома окремими частинами: eShop.Mobile – це основний проект застосунку та Pods – це проект бібліотеки залежностей, яка підключається до основного проекту. Основні залежності використані в проекті представлені в таблиці 3.8.

Таблиця 3.8 – Опис залежностей та їх призначення

Назва залежності	Призначення
AWSS3	Компонент AWS SDK призначений для роботи з Amazon Simple Storage Service (S3)
AWSAppSync	Компонент AWS SDK призначений для роботи з GraphQL API
AWSCognitoIdentityProvider	Компонент AWS SDK призначений для роботи з Amazon Cognito
Firebase/Crashlytics	Бібліотека для моніторингу та відстеження виникнення помилок
Firebase/Analytics	Бібліотека для збору та аналізу даних

Структура проекту eShop.Mobile представлена у вигляді набору каталогів та файлів. Основні елементи структури проекту та їх призначення представлено у таблиці 3.9.

Таблиця 3.9 – Основні елементи структури проекту та їх призначення

Назва елементу	Призначення
Каталог Api	Містить усі необхідні класи моделей, схему GraphQL та клієнт для роботи з AppSync
Каталог Extensions	Містить класи розширення функціоналу
Каталог Components	Містить класи окремих самодостатніх компонентів
Каталог Controllers	Мітить класи контролерів екранів застосунку
Каталог Resources	Ресурси застосунку (шрифти, малюнки)
Каталог Amplify	Файли конфігурації сервісів Amazon
Каталог GoogleService	Файли конфігурації для Google сервісів
Файл AppDelegate.swift	Відповідає за обробку подій на рівні програми, таких як запуск програми та інші
Файл AppManager.Swift	Містить функціонал управління застосунком
Файл AppInfo.swift	Містить інформацію про застосунок (назва застосунку, версія, операційна система та тощо)
Файл AppLog.swift	Містить функціонал логування у системний журнал подій
Файл Info.plist	Файл конфігурації та налаштувань застосунку

До прикладу екрани реєстрації реалізовано контролерами: SignUpRegisterViewController та SignUpCompleteViewController. Вони реалізують процедуру створення нового користувача та верифікації його електронної адреси. Розмітка екранів виконана відповідно до розроблених макетів за допомогою наступних UI компонентів: UIScrollView, UIView. UIImageView. UILabel, UIButton, UITextField, UIToolbar. Основні методи

SignUpRegisterViewController: «cancel», що повертає користувача на початковий екран; «createAccount», що здійснює валідацію заповнених полів та створює акаунт користувача та «confirmUser», що здійснює перемикання користувача на екран підтвердження електроної адреси.

Контролер реалізує делегат UITextFieldDelegate, а саме метод textFieldShouldReturn:

```
extension SignUpRegisterViewController: UITextFieldDelegate {
    func textFieldShouldReturn(_ textField: UITextField) -> Bool {
        switch textField {
        case firstNameInputField:
lastNameInputField.becomeFirstResponder()
        case lastNameInputField:
phoneNumberInputField.becomeFirstResponder()
        case phoneNumberInputField:
emailAddressInputField.becomeFirstResponder()
        case emailAddressInputField:
passwordInputField.becomeFirstResponder()
        case passwordInputField:
confirmPasswordInputField.becomeFirstResponder()
        case confirmPasswordInputField:
textField.resignFirstResponder()
        default: textField.resignFirstResponder()
        }
        return true
    }
}
```

В даному методі реалізовано алгоритм послідовного переходу з одного поля на інше при натисканні кнопки Return на клавіатурі. Додатково реалізовано алгоритм адаптивного показу клавіатури, який дозволяє уникати перекриття полів вводу клавіатурою. Основними методи SignUpCompleteViewController є: «cancel», що працює аналогічно як і у SignUpRegisterViewController; «submit», що здійснює валідацію коду для підтведження, підтверджує користувача та завершує процедуру реєстрації; «resendCode», що здійснює повторне надсилання коду на електрону пошту вказану на екрані реєстрації.

Також можна розглянути екран каталогу товарів, що реалізовано контролером CategoryViewController. Розмітка екрана виконана відповідно до

макета. В даному контролері використано компонент UICollectionView, що одночасно відображає категорії та товари. Моделью для сутності категорії виступає клас CategoryViewModel, а представленням CategoryViewCell. Моделью для сутності товару виступає клас ProductViewModel, а представленням ProductViewCell. Основними методами CategoryViewController є «reloadCategories», що завантажує усі підкатегорії для певної категорії та «reloadProducts», що завантажує усі товари для певної категорії.

Контролер реалізує протоколи UICollectionViewDataSource та UICollectionViewDelegate для елементу UICollectionView. Додатково реалізовано протокол ProductViewCellDelegate, що представлений двума методами: «addToCart» для додавання товару до кошика користувача та «addToWishList» для додавання товару до списку бажаного.

Вибір користувачем певного товару призводить до показу екрану детальної інформації про товар. Екран детальної інформації про товар реалізовано контролером ProductViewController. Розмітка екрана виконана відповідно до макета. Основним компонентом контролера вистує UICollectionView з використанням UICollectionViewCompositionalLayout, - це новий спосіб визначення макетів для колекційних переглядів у iOS, введений у iOS 13 [27]. Він дозволяє швидко і зручно створювати складні макети для колекційних відображень, використовуючи композиційні аспекти, такі як групи, розподілення та розташування елементів:

```
private lazy var layoutForPhotos: NSCollectionLayoutSection = {

    let itemSize = NSCollectionLayoutSize(widthDimension:
.fractionalWidth(1), heightDimension: .fractionalWidth(0.75))
    let item = NSCollectionLayoutItem(layoutSize: itemSize)
    item.contentInsets = NSDirectionalEdgeInsets(top: 10, leading:
10, bottom: 5, trailing: 10)

    let groupSize = NSCollectionLayoutSize(widthDimension:
.fractionalWidth(0.94), heightDimension: .estimated(1))
    let group = NSCollectionLayoutGroup.vertical(layoutSize:
groupSize, subitems: [item])
```

```
    let section = NSCollectionLayoutSection(group: group)
    section.contentInsets = NSDirectionalEdgeInsets(top: 0,
leading: 0, bottom: 0, trailing: 0)
    section.interGroupSpacing = -10
    section.orthogonalScrollingBehavior = .continuous

    return section
}()
```

Основні типи моделей та представлень наведено у таблиці 3.10.

Таблиця 3.10 – Моделі та представлення контролера ProductViewController

Модель	Представлення	Опис
ProductAttributeViewModel	ProductAttributeViewCell	Атрибут товару
ProductDetailViewModel	ProductDetailViewCell	Група атрибутів
ProductInfoViewModel	ProductInfoViewCell	Стислий опис
ProductOptionViewModel	ProductOptionViewCell	Варіант товару
ProductPhotoViewModel	ProductPhotoViewCell	Фотографія товару
ProductPriceViewModel	ProductPriceViewCell	Ціна товару
ProductReviewViewModel	ProductReviewViewCell	Відгук на товар
ProductStatusViewModel	ProductStatusViewCell	Статус товару
ProductTitleViewModel	ProductTitleViewCell	Назва товару

Основними методами контролера ProductViewController є: «reloadProduct» для завантаження товару; «reloadDetails» для завантаження усіх характеристик; «reloadReviews» для завантаження усіх відгуків; «addToCart» для додавання товару до кошика користувача; «addToWishlist» для додавання товару до списку бажаного користувача; «addReview» для додавання відгуку користувача про товар; «presentPhoto» для відображення переліку усіх фотографій товару.

Повний код програми наведено у додатку Г.

3.4. Керівництво користувача

Робота із мобільним застосунком починається із процедур реєстрації та авторизації. Процедура реєстрації виконується новим користувачем для створення облікового запису та верифікації електронної пошти. Процедура аутентифікації виконується користувачем з існуючим обліковим записом для входу у мобільний застосунок.

Навігація у мобільному застосунку побудована на основі головного меню, у якому представлені основні розділи та функції. В загальному випадку сценарій взаємодії користувача із застосунком Інтернет-магазину можна представити наступною послідовністю: Пошук товарів, формування кошику замовлень і оформлення замовлення.

Пошук товарів у мобільному застосунку може бути здійснений різними способами. У розділі «Каталог товарів» користувач може переглядати товари за їх категоріями, а у розділі «Виробники» за брендами виробників. «Акційні пропозиції» показують перелік товарів із зниженими цінами, тоді як у «Пошук» користувач може скористатися різними фільтрами для пошуку, починаючи від назви та ціни і закінчуючи значенням властивостей із опису товару.

Формування кошику замовлень у мобільному застосунку передбачає додавання обраних товарів користувачем у спеціальний список. Кожен товар має власну карточку із зображенням, описом та ціною, що дозволяє зручно вибирати та додавати їх до кошика. Після додавання товарів, користувач має можливість переглянути вміст кошика, змінити кількість або видалити товари, а також перевірити загальну вартість покупки перед оформленням замовлення.

Оформлення замовлення у застосунку зазвичай включає введення даних для адреси доставки та контактну інформацію про отримувача. Після введення необхідної інформації і підтвердження замовлення, система автоматично обробляє дані, резервуючи товари та готуючи їх до відправлення. Користувач отримує підтвердження замовлення на свою електронну пошту та через

повідомлення у застосунку. Перегляд історії покупок користувач може здійснити у розділі «Мої замовлення».

Відгуки користувача у мобільному застосунку дозволяють користувачам ділитися своїм досвідом використання товарів або послуг. Користувачі можуть залишати відгуки про якість товарів, швидкість доставки та рівень обслуговування. Це допомагає іншим користувачам приймати обґрунтовані рішення під час покупок та додатково стимулює продавців до покращення якості своїх товарів та послуг.

Розділ «Про нас» у застосунку надає користувачам додаткову інформацію про компанію чи магазин. Тут представлені основні принципи, цінності та місія, що допомагає користувачам краще розуміти та довіряти магазину.

Розділ «Гарантія та сервіс» містить інформацію про гарантійні умови на товари, правила повернення та обміну, контактну інформацію для звернень з питань обслуговування та ремонту. Це дає користувачам впевненість у підтримці після покупки та допомагає вирішувати можливі проблеми швидко і ефективно.

Розділ «Доставка та оплата» надає детальну інформацію про умови доставки товарів, способи оплати та терміни відправлення. Користувачі можуть обирати зручний для них спосіб отримання замовлення та оплати, що дозволяє покупцям максимально адаптувати процес покупок до своїх потреб та вимог.

3.5. Технічні характеристики мобільного застосунку

Технічні характеристики для мобільного застосунку є важливими, оскільки вони визначають його оптимальну роботу на конкретних пристроях та ОС. Знання цих характеристик допомагає користувачеві зрозуміти, як оптимально використовувати програму на своєму пристрої, враховуючи його можливості. Це дозволяє уникнути можливих конфліктів або непорозумінь з використанням застосунку, забезпечуючи комфортну та ефективну роботу. Для мобільного застосунку «eShop» технічні характеристики надаються у таблиці 3.11:

Таблиця 3.11 – Технічні характеристики мобільного застосунку eShop

Назва характеристики	Значення
Операційна система	iOS 15.0 або вище
Модель пристрою	iPhone XS або вище
Обсяг оперативної пам'яті	Наявність вільної пам’яті у розмірі 4 ГБ
Інтернет-з'єднання	Підтримка Wi-Fi та мобільного Інтернету (3G/4G/5G), залежно від можливостей пристрою та мережі провайдера
Мережеві вимоги	Підтримка мережевих протоколів, таких як HTTP, HTTPS, WebSocket тощо для забезпечення зв'язку з сервером та іншими сервісами
Додаткові вимоги	На телефоні встановлено App Store

3.6 Тестування мобільного застосунку

3.6.1. Аналіз методів тестування мобільного застосунку

Тестування мобільного застосунку – це процес перевірки його функціональності, взаємодії з користувачем та стабільності на різних пристроях та платформах. Це важлива складова розробки, яка допомагає виявити та виправити помилки, забезпечуючи надійну та задоволену роботу застосунку для користувачів. Виявлення помилок на ранніх етапах розробки допомагає зменшити витрати на їх виправлення. Існують різні методи тестування мобільних застосунків, серед яких ручне тестування, автоматизоване тестування та тестування з використанням реальних пристроїв та емуляторів.

Ручне тестування мобільного застосунку – це процес, під час якого тестувальники вручну перевіряють функціональність та коректність роботи програми. Під час ручного тестування, тестувальники взаємодіють з застосунком, вводять дані, перевіряють реакцію на різні введення та переконуються у правильному відображенні даних на екрані. Це дозволяє побачити роботу програми з точки зору користувача.

Автоматизоване тестування мобільних застосунків використовує програмні скрипти для автоматичного виконання тестових сценаріїв. Такі скрипти можуть перевіряти функціональність, взаємодію з базою даних, реакцію на різні події та інші аспекти роботи застосунку без прямого втручання користувача.

Тестування з використанням реальних пристроїв та емуляторів полягає в перевірці роботи застосунку на різних апаратних платформах та у різних умовах. Використання реальних пристроїв дозволяє точніше відтворювати умови реального використання застосунку, а емулятори забезпечують можливість тестування на різних версіях операційних систем без необхідності наявності фізичного пристрою. Такий підхід допомагає виявляти проблеми, які можуть виникнути на різних пристроях.

На основі проведеного аналізу можна зробити висновок про доцільність використання певного методу тестування для кожної окремої частини проєкту. Тестування AWS-сервісів доцільно виконувати ручним методом тестування, оскільки деякі аспекти AWS-інфраструктури, такі як конфігурації, права доступу та безпека, можуть бути складними для автоматизації через їхню велику різноманітність та специфіку. Також певні важливі функції AWS, наприклад, резервне копіювання даних або відновлення з резерву, можуть вимагати виконання складних сценаріїв тестування, які краще виконувати вручну.

Тестування мобільного застосунку ефективно виконувати через поєднання автоматизованого методу тестування із тестуванням на реальних пристроях та емуляторах. Автоматизоване тестування дозволяє швидко перевіряти основні функції та сценарії безперервно, тоді як тестування на реальних пристроях та емуляторах дозволяє виявляти проблеми, що можуть виникнути через різні характеристики апаратної та програмної платформи, такі як відмінності в розмірах екранів та версіях операційних систем. Таке поєднання дозволяє забезпечити більшу впевненість у якості та надійності застосунку на різних пристроях та умовах використання.

3.6.2 Тестування хмарних сервісів AWS

Для ручного тестування хмарних сервісів AWS використовуємо інструмент CloudWatch (рисунок 3.1), який дозволяє моніторити різні метрики та логи, допомагає виявляти та аналізувати проблеми у реальному часі. Такий підхід забезпечує більш глибоке тестування AWS сервісів та дозволяє вчасно виявляти та вирішувати проблеми, що можуть виникнути під час експлуатації.

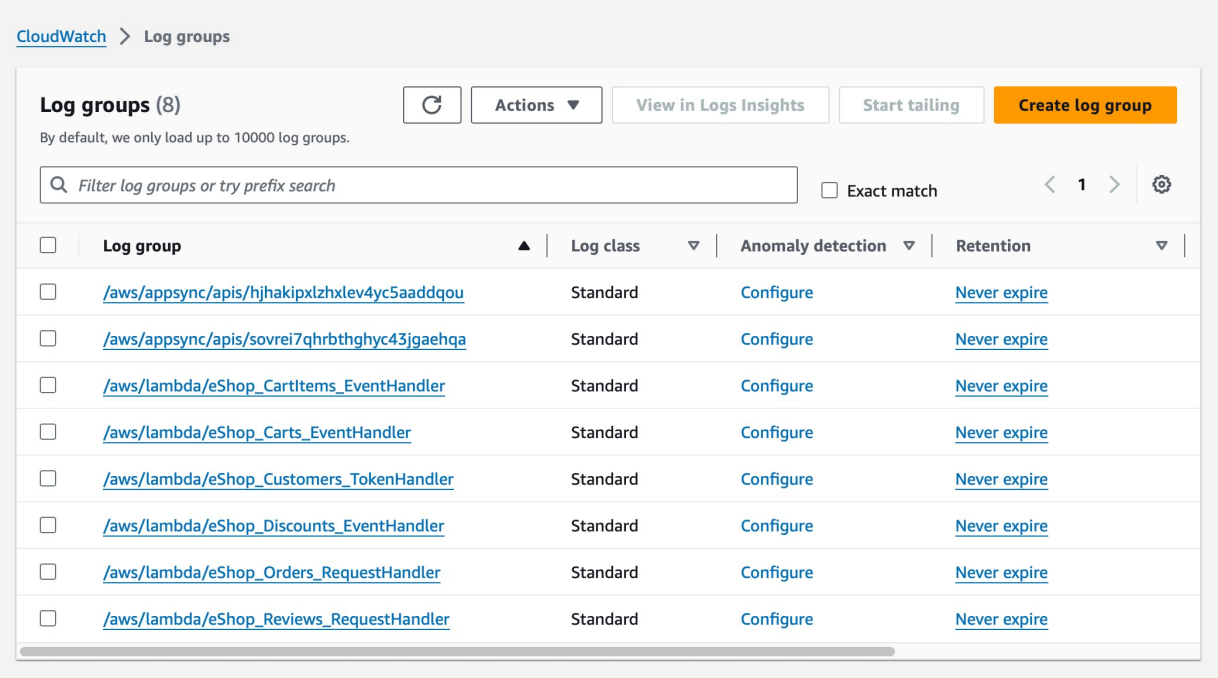


Рисунок 3.1 – Вебінтерфейс сервісу CloudWatch

Кожен окремий сервіс AWS має власний набір інструментів для моніторингу та тестування. Наприклад сервіс AppSync має окремий вебінтерфейс для тестування GraphQL запитів, а OpenSearch для тестування пошукових запитів. Оптимальним рішенням для тестування Lambda функцій є використання автоматизованих методів тестування на основі Unit тестів, які дозволяють розробникам перевіряти правильність роботи конкретних функцій, методів або класів, що сприяє виявленню та виправленню помилок на різних етапах розробки. Приклад Unit тесту для функції створення відгука наведено таким чином:

```
public class FunctionTest
{
    [Fact]
    public async void TestReviewCreate(){
        var mockProductItem = new ProductItem {
            Id = Guid.NewGuid().ToString(),
            ProductId = Guid.NewGuid().ToString()
        };

        var mockRatingId =
$"{mockProductItem.ProductId}.{mockProductItem.Id}";
        var mockRating = new Rating
        {
            Id = mockRatingId,
            ProductId = mockProductItem.ProductId,
            ProductItemId = mockProductItem.ProductId,
            AverageRating = 0,
            RateCount = 0
        };

        var mockDynamoDBContext = new Mock<IDynamoDBContext>();
        mockDynamoDBContext.Setup(x =>
x.LoadAsync<ProductItem>(mockProductItem.Id,
default)).ReturnsAsync(mockProductItem);
        mockDynamoDBContext.Setup(x =>
x.LoadAsync<Rating>(mockRating.Id,
default)).ReturnsAsync(mockRating);
        var productItem = await
mockDynamoDBContext.Object.LoadAsync<ProductItem>(mockProductItem.I
d);

        var context = new TestLambdaContext();
        var function = new Function(mockDynamoDBContext.Object);
        var request = new ReviewRequest
        {
            Create = new ReviewCreateInput
            {
                CustomerId = Guid.NewGuid().ToString(),
                ProductItemId = productItem.Id,
                Rating = 5
            }
        };

        var response = await function.RequestHandler(request,
context);
        Assert.NotNull(response);
        Assert.Null(response.Error);
        Assert.NotNull(response.Review);
        Assert.Equal(response.Review.Rating,
request.Create.Rating);
    }
}
```

3.6.3 Тестування функціоналу мобільного застосунку

Тестування функціоналу мобільного застосунку проводимо за допомогою автоматизованих методів тестування. Для автоматизованого тестування на платформі iOS використовують Unit тести, які дозволяють перевірити окремі частини коду на коректність та відповідність очікуванням. Unit тести для поділяються на три типи: модульні, інтеграційні та тести користувацького інтерфейсу [28]. Модульні тести перевіряють окремі частини коду, такі як класи чи методи. Інтеграційні тести перевіряють взаємодію між різними компонентами програми. Тести UI перевіряють відображення та взаємодію з інтерфейсом. Представлений фрагмент коду модульного тесту, який перевіряє коректність отриманих даних про продукт з сервера:

```
import XCTest
import AWSAppSync
@testable import eShop_Mobile

final class APIGetProductTest: XCTestCase {
    var queryData: GetProductQuery.Data? = nil

    override func setUpWithError() throws {
        let responseData = try
ResourceManager.resourceContent(with: "apiGetProductResponse.json")
        let jsonObject = (try JSONSerialization.jsonObject(with:
responseData)) as! JSONObject
        queryData = try GetProductQuery.Data(jsonObject:
jsonObject)
    }
    func testGetProduct() throws {
        XCTAssertNotNil(queryData)
        XCTAssertNotNil(queryData!.getProduct)
        let snapshot = queryData!.getProduct!.snapshot
        let product = Product(snapshot: snapshot)
        XCTAssertNotNil(product)
        XCTAssertEqual(product.id, "1ee2db22-5aa0-4620-b754-
0706bbe7092b")
        XCTAssertEqual(product.name, "Сушка для посуду Maestro MR-
1027")
        XCTAssertNotNil(product.brand)
        XCTAssertNotNil(product.items)
        XCTAssertNotNil(product.imageUrl)
    }}
```

3.6.4 Тестування за допомогою симулятора

Тестування мобільного застосунку за допомогою симуляторів та емуляторів є ефективним способом перевірки його функціональності та сумісності без необхідності володіння реальними пристроями [29]. Симулятор відтворює виконання коду без прив'язки до середовища запуску коду. Емулятор відтворює принципи роботи системи пристрою зі збереженням ключових властивостей та аспектів роботи. Емуляція відтворює код у необхідному для нього середовищі. На iOS зараз доступні лише симулятори (рисунок 3.2), а на Android – емулятори.

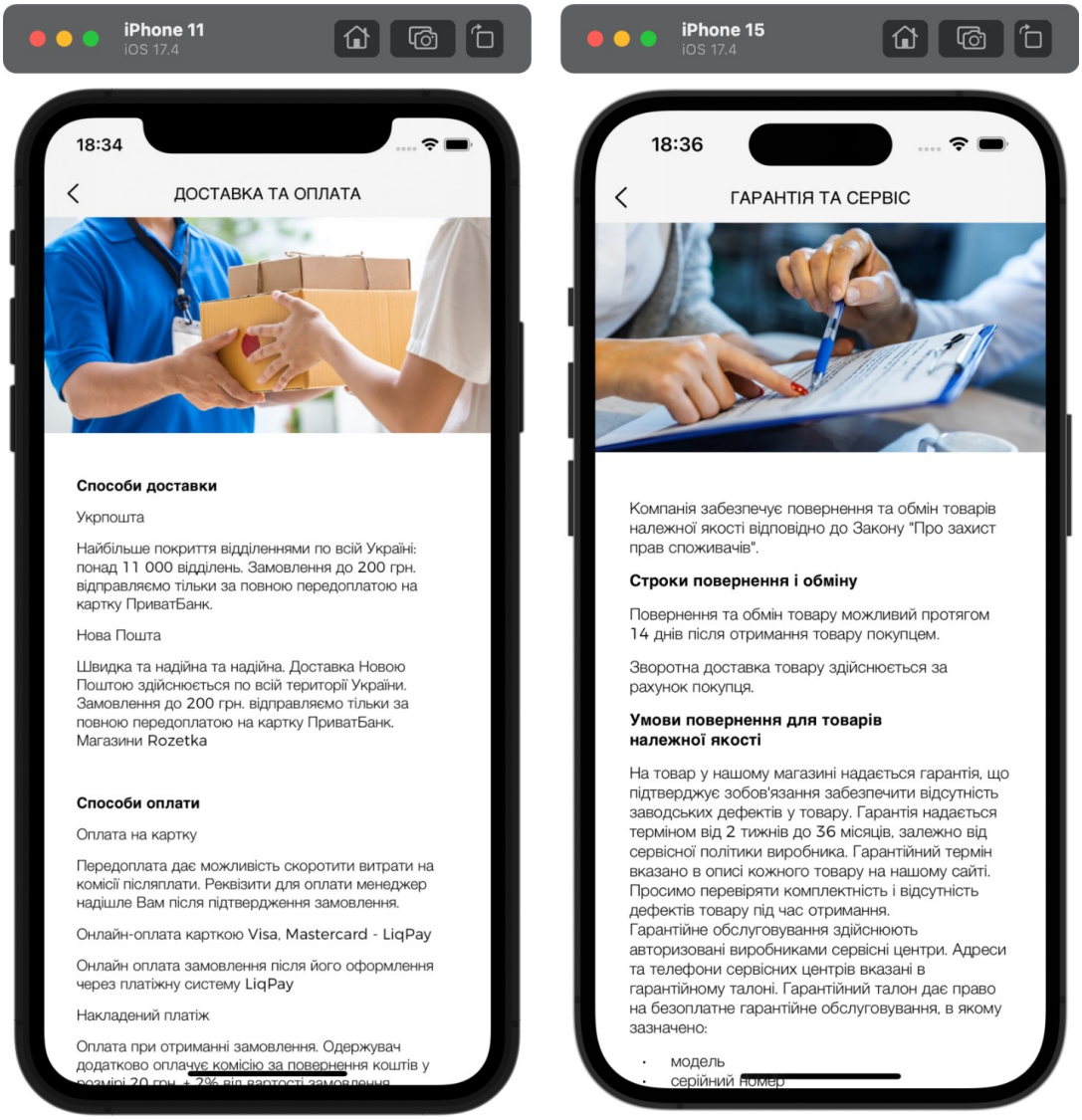


Рисунок 3.2 – Симулятори iPhone 11 та 15, iOS 17.4

Симулятори iOS імітують роботу реальних пристроїв, дозволяючи виконувати тести на різних версіях операційної системи та моделях пристроїв. Симулятор є вбудованим інструментом в середовище розробки Xcode для платформи iOS та macOS. Меню Simulator дозволяє обирати різні параметри налаштування, такі як масштабування розміру, встановлення місця або маршруту, імітація системних повідомлень, швидкість анімації та інше.

Недивлячись на те, що тестування на симуляторі дозволяє ефективно перевірити основний функціонал програми, все ж таки необхідно виконувати тестування на реальному пристрої. Це обумовлено тим, що навіть після успішного тестування на симуляторі можуть виникнути певні проблеми або «баги», які проявляться тільки на реальних пристроях через їх унікальні характеристики та реальні умови використання. Такий підхід дозволяє підвищити надійність та якість програмного забезпечення перед його релізом.

Для повноцінного тестування мобільного застосунку важливо використовувати як симулятори, так і реальні пристрої. Комбінація обох методів дозволяє забезпечити високу якість та надійність програмного продукту перед його випуском.

3.6.5 Аналіз результатів тестування мобільного застосунку

Тестування мобільного застосунку було проведено на пристроях iPhone 11, 13 та 15 з різними версіями операційної системи. Результати тестування представлені в таблиці 3.12.

Тестування проведено успішно, в процесі тестування було виявлено вразливість, яка полягає у тому що користувач міг ввести номер телефону в форматі, який не відповідав вимогам API. Крім того при виході із застосунку не зберігався «username» авторизованого користувача, що призводило до необхідності повторного його введення при повторному вході користувача у застосунок.

Таблиця 3.12 – Результати тестування

Функція	Пристрій		
	iPhone 11, iOS 15.8.2	iPhone 13, iOS 16.7.7	iPhone 15, iOS 17.4.1
Реєстрація нового користувача	+	+	+
Авторизація існуючого користувача	+	+	+
Налаштування пошукових фільтрів	+	+	+
Пошук товарів	+	+	+
Формування кошику	+	+	+
Створення замовлення	+	+	+
Історія замовлень	+	+	+
Створення та редагування відгуку	+	+	+
Підтвердження замовлення	+	+	+

Також було окремо протестовано реакцію застосунку на помилки. Для цього були неповністю заповнені форми створення нової витрати і нового документу. В результаті тестування встановлено, що додаток веде себе очікувано – повідомляє користувачеві, що потрібно заповнити відсутню інформацію, збоїв і помилок в роботі не зафіксовано. Це забезпечує користувачам більш інтуїтивний і безпечний досвід роботи з застосунком.

Додатково проведено тестування на користування (usability testing), яке спрямоване на оцінку зручності використання застосунку для користувачів [30]. Участь у тестуванні взяла група з 3 учасників, яким запропонували випробувати основні функції програми та оцінити їх зручність за п'ятибальною шкалою. Кожному учаснику було надано низку завдань, що відображають типові сценарії використання застосунку, такі як створення нового замовлення, перегляд історії замовлень і т.д. Результати тестування представлені у таблиці 3.13.

Таблиця 3.13 – Результати Usability тестів

Функція	Учаснки 1	Учаснки 2	Учаснки 3
Реєстрація нового користувача	4	5	5
Авторизація існуючого користувача	4	4	5
Налаштування пошукових фільтрів	3	4	5
Пошук товарів	4	4	5
Створення замовлення	3	4	4
Історія замовлень	5	5	5
Створення та редагування відгуку	3	5	4
Загальне враження	4	4	5

Учасники виявили декілька незручностей в роботі з застосунком. Після завершення процесу реєстрації користувачів переносить на екран входу у застосунок, а не відразу у головне меню. У розділ каталогу товари відображаються лише для батьківської категорії та адреса доставки не враховує сервіси поштові відділення Нової пошти.

На основі проведених тестувань були визначені можливості для покращення роботи застосунку. Можна покращити завантаження товарів у розділі каталогу на основі індексованих даних із OpenSearch. Підключити API Нової пошти для пошуку адрес відділень за назвою міста. Також підключити платіжні системи для онлайн оплати замовлень.

3.7 Висновки

Створено та налаштовано сервіс авторизації та аутентифікації користувачів мобільного застосунку.

За допомогою сервісу DynamoDB розроблено базу даних мобільного застосунку для зберігання сутностей визначених у розділі проектування.

Створено та налаштовано сховище S3 Storage для зберігання медіа даних мобільного застосунку.

Налаштовано домен OpenSearch для реалізація наскрізного пошуку товарів за такими параметрами, як ціна, категорія товару, виробник та інші важливі характеристики.

Реалізовано низку Lambda-функцій, що містять у собі бізнес-логіку мобільного застосунку.

Реалізовано API мобільного застосунку на основі GraphQL.

Виконано інтеграцію налаштованих сервісів AWS у мобільний застосунок за допомогою AWS Amplify.

Розроблено API клієнт для комфортної роботи мобільного застосунку з GraphQL запитами.

Розроблено екрани мобільного застосунку, передбачені варіантами використання розглянутими у розділі проектування.

Розроблено керівництво користувача та описано технічні характеристики мобільного застосунку.

На основі аналізу методів тестування визначено оптимальні методи для тестування окремих частин проєкту.

Виконано тестування хмарних сервісів та функціональних можливостей мобільного застосунку.

Виконано аналіз результатів тестування, виявлено недоліки та сформовано перелік для подальшого удосконалення застосунку.

ВИСНОВКИ

На основі аналізу предметної області та існуючих програмно-технічно рішень обгрунтована актуальність розроблення мобільного застосунку, сформульвано основні функціональні та нефункціональні вимоги.

На основі порівняльного аналізу обрано безсерверну архітектуру для розроблення мобільного застосунку. Розглянуто сценарії використання сервісів AWS та обрано найбільш відповідний до потреб застосунку варіант.

Визначено та спроектовано низку важливих сутностей, які відображають основні об'єкти та концепції системи. Розроблено та проаналізовано діаграму варіантів використання. Розглянуто можливі варіанти використання та побудовано діаграми послідовності у контексті використання сервісів AWS.

На основі аналізу сучасних засобів та технологій для розроблення мобільного застосунку визначено оптимальний набір технологій, інструментів, фреймворків, мов програмування та інших компонентів, яку будуть залучені для розробки мобільного застосунку.

Налаштовано сервіс авторизації та аутентифікації користувачів, розроблено базу даних мобільного застосунку, налаштовано сховище медіа-даних, налаштовано домен для реалізації наскрізного пошуку, реалізовано низку Lambda-функції для бізнес логіки, реалізовано API на основі GraphQL.

Виконано інтеграцію налаштованих сервісів у мобільний застосунок, розроблено API клієнт для роботи з GraphQL-запитами та екрани мобільного застосунку відповідно до макетів.

Розроблено керівництво користувача та описано технічні характеристики мобільного застосунку.

Визначено оптимальні методи для тестування окремих частин проєкту. Виконано тестування сервісів та функціоналу мобільного застосунку. Проведено аналіз результатів тестування, виявлено недоліки та сформовано перелік для подальшого удосконалення застосунку.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Про нас | ROZETKA. URL: https://rozetka.com.ua/ua/pages/about/ (дата звернення: 21/02/2024)
2. Про маркетплейс Prom.ua. URL: https://prom.ua/ua/about_us (дата звернення: 21/02/2024)
3. Про компанію COMFY. URL: https://comfy.ua/ua/aboutus.html (дата звернення: 21/02/2024)
4. Функціональні функції. URL: https://visuresolutions.com/uk/blog/functional-requirements/ (дата звернення: 22/02/2024)
5. Amplify Documentation. URL: https://docs.aws.amazon.com/amplify (дата звернення: 22/02/2024)
6. Cognito Developer. URL: https://docs.aws.amazon.com/cognito (дата звернення: 22/02/2024)
7. AppSync Documentation. URL: https://docs.aws.amazon.com/appsync/ (дата звернення: 22/02/2024)
8. CloudWatch Developer Guide. URL: https://docs.aws.amazon.com/cloudwatch/ (дата звернення: 22/02/2024)
9. Lambda Documentation. URL: https://docs.aws.amazon.com/lambda/ (дата звернення: 22/02/2024)
10. DynamoDB Developer Guide. URL: https://docs.aws.amazon.com/amazondynamodb (дата звернення: 22/02/2024)
11. S3 Storage Documentation. URL: https://docs.aws.amazon.com/AmazonS3 (дата звернення: 22/02/2024)
12. API Gateway Developer Guide. URL: https://docs.aws.amazon.com/apigateway (дата звернення: 22/02/2024)
13. SES Developer Guide. URL: https://docs.aws.amazon.com/ses (дата звернення: 22/02/2024)

14. OpenSearch Documentation. URL: https://docs.aws.amazon.com/opensearch-service (дата звернення: 22/02/2024)

15. IAM User Guide. URL: https://docs.aws.amazon.com/IAM (дата звернення: 22/02/2024)

16. Practical AWS Diagram tutorial and exampes. URL: https://www.visual-paradigm.com/guide/cloud-services-architecture/what-is-aws-architecture/ (дата звернення: 22/02/2024)

17. Design Patterns for Relational Databases. URL: https://www.geeksforgeeks.org/design-patterns-for-relational-databases/ (дата звернення: 18/03/2024)

18. Діаграма сутність-зв'язок. URL: https://studfile.net/preview/5462334/page:12/ (дата звернення: 18/03/2024)

19. Варіанти використання та сценарії (Use Cases and Scenarios). URL: https://www.maxzosim.com/use-cases-and-scenarios/ (дата звернення: 18/03/2024)

20. Діаграма послідовності (Sequence Diagrams). URL: https://www.maxzosim.com/sequence-diagrams/ (дата звернення: 18/03/2024)

21. Figma Learn. URL: https://help.figma.com/hc/en-us (дата звернення: 18/03/2024)

22. AWS SDK for .NET. URL: https://docs.aws.amazon.com/sdk-for-net/ (дата звернення: 24/03/2024)

23. Мова програмування C#. URL: https://learn.microsoft.com/en-us/dotnet/csharp/ (дата звернення: 24/03/2024)

24. Мова програмування Swift. URL: https://book.swift.org.ua/book/mova-programuvannya-swift/ (дата звернення: 24/03/2024)

25. Firebase Crashlytics. URL: https://firebase.google.com/docs/crashlytics (дата звернення: 24/03/2024)

26. Understanding the Role of MVC. URL: https://medium.com/@viniciusnadin/understanding-the-role-of-mvc-in-ios-

development-how-apple-utilizes-the-mvc-architecture-in-ui-4aa128ed59ad (дата звернення: 10/04/2024)

27. UICollectionViewCompositionalLayout. URL: https://developer.apple.com/documentation/uikit/uicollectionviewcompositionallayout (дата звернення: 10/04/2024)

28. Тестування додатків на iOS. URL: https://qagroup.com.ua/publications/testuvannia-dodatkiv-na-ios/ (дата звернення: 22/02/2024)

29. Емулятор проти Симулятора – різниця між ними. URL: https://www.guru99.com/uk/real-device-vs-emulator-testing-ultimate-showdown.html (дата звернення: 10/04/2024)

30. Що таке юзабіліті тестування. URL: https://mate.academy/blog/qa/usability-testing/ (дата звернення: 10/04/2024)

ДОДАТОК А
(обов’язковий)

ДІАГРАМИ ПОСЛІДОВНОСТІ ДЛЯ КОЖНОГО ВАРІАНТУ ВИКОРИСТАННЯ

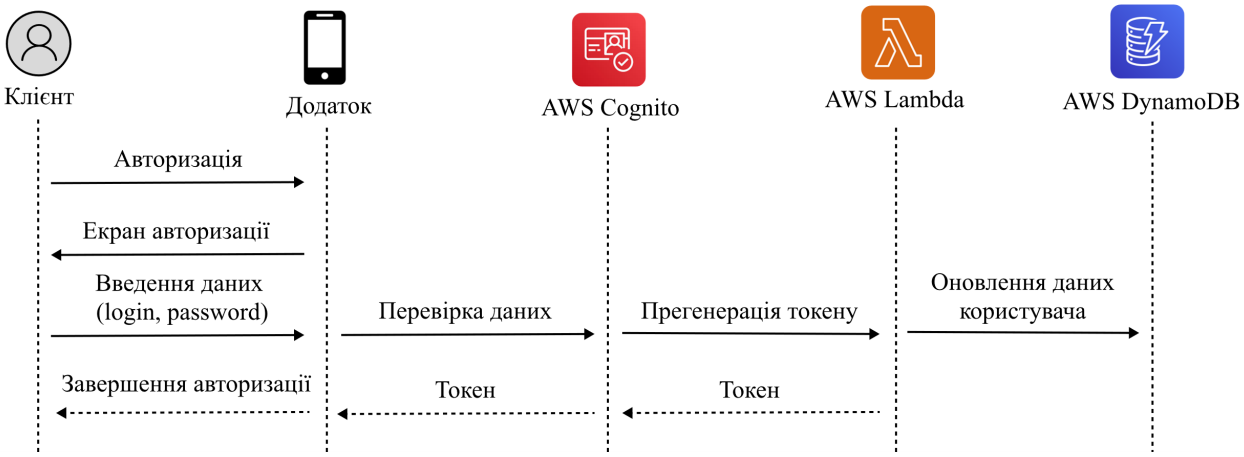


Рисунок А.1 – Діаграма послідовності для варіанту «Авторизація»

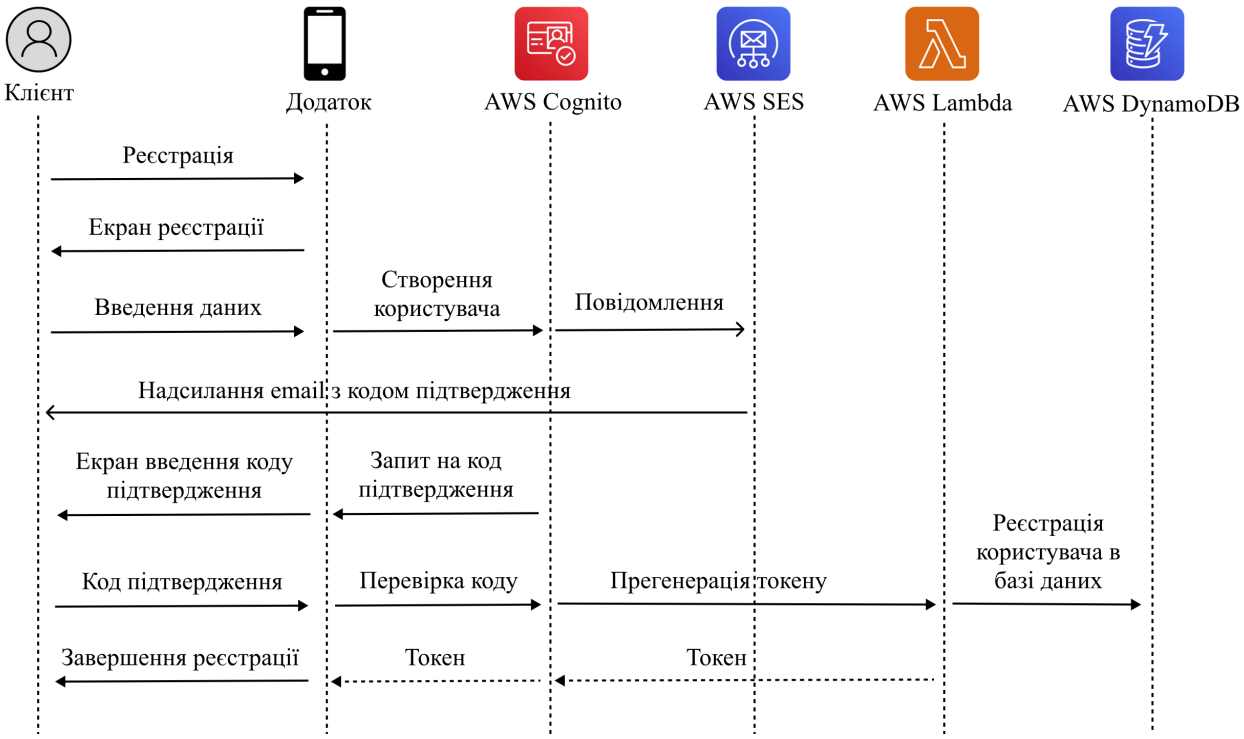


Рисунок А.2 – Діаграма послідовності для варіанту «Реєстрація»

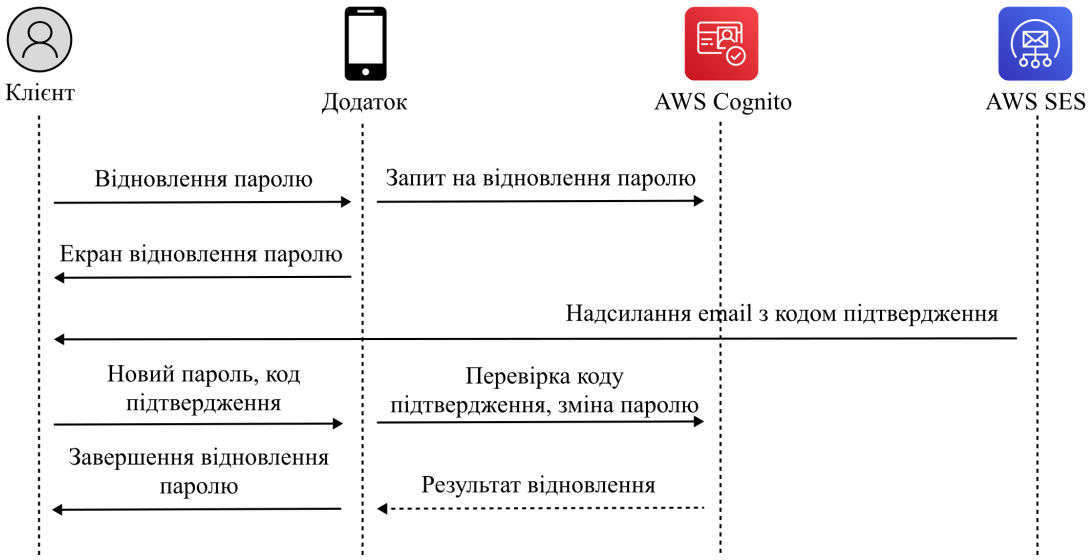


Рисунок А.3 – Діаграма послідовності для варіанту «Відновлення паролю»

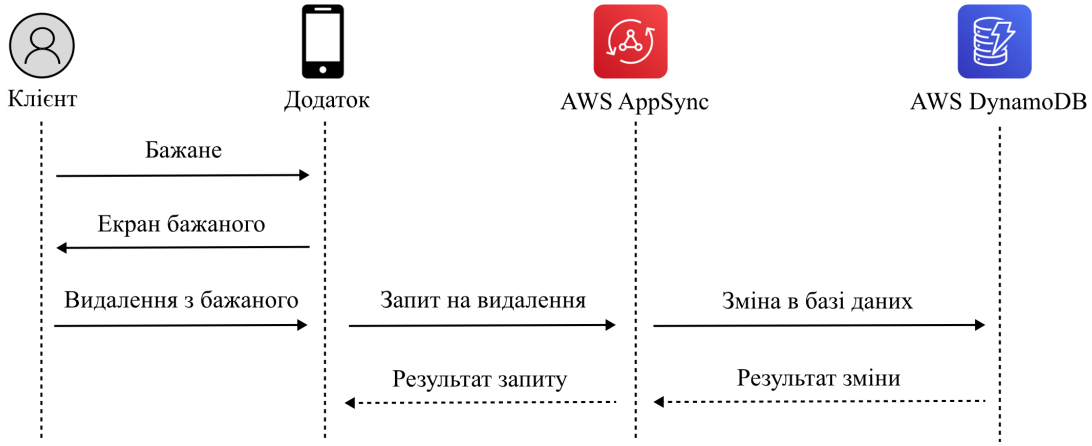


Рисунок А.4 – Діаграма послідовності для варіанту «Видалення з бажаного»

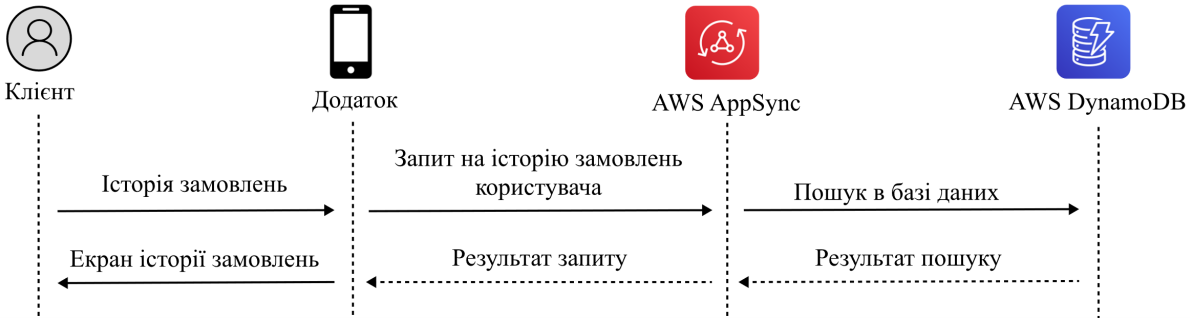


Рисунок А.5 – Діаграма послідовності для варіанту «Історія замовлень»

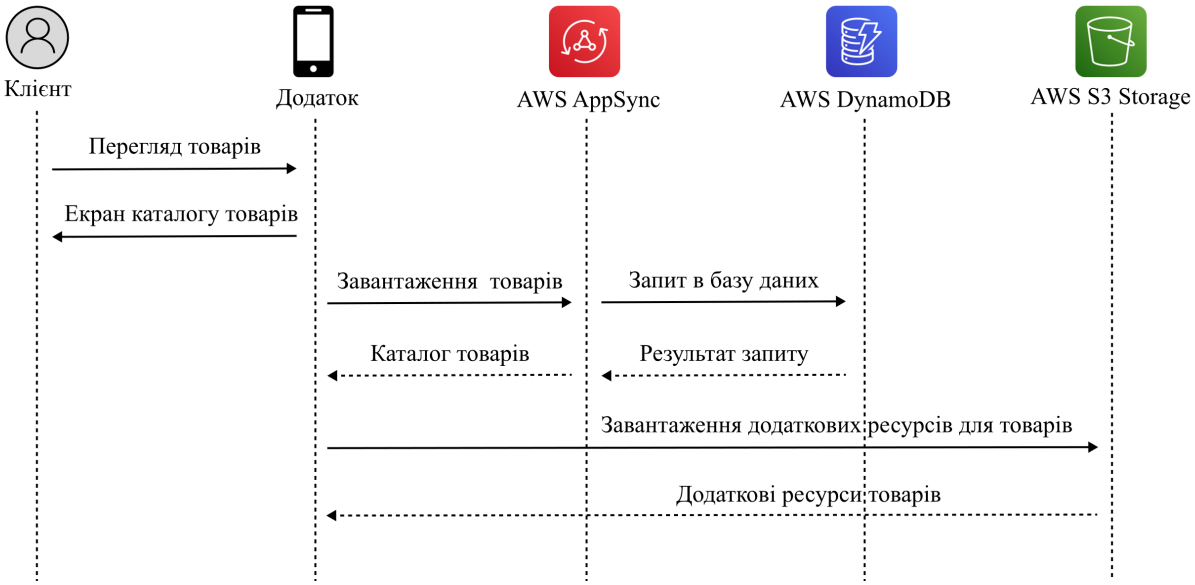


Рисунок А.6 – Діаграма послідовності для варіанту «Перегляд товарів»

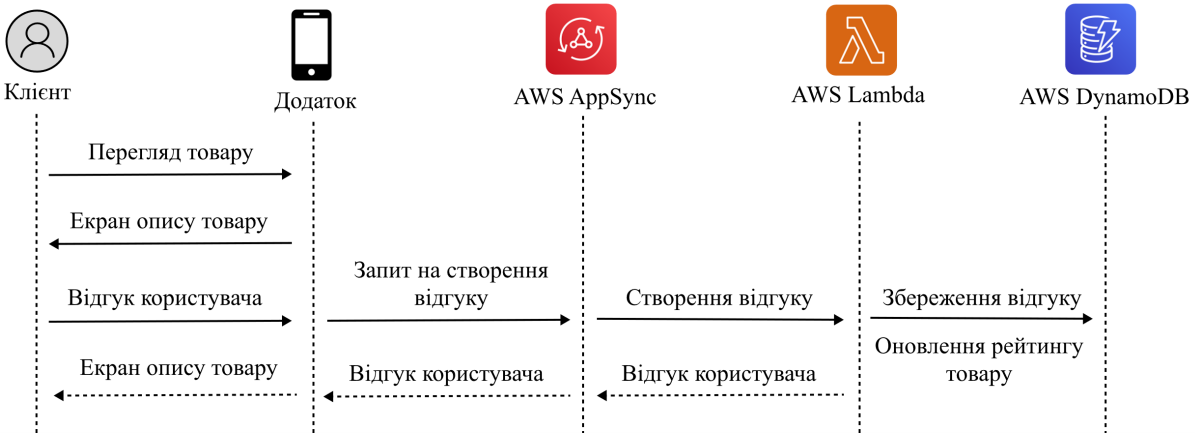


Рисунок А.7 – Діаграма послідовності для варіанту «Створення відгуку»

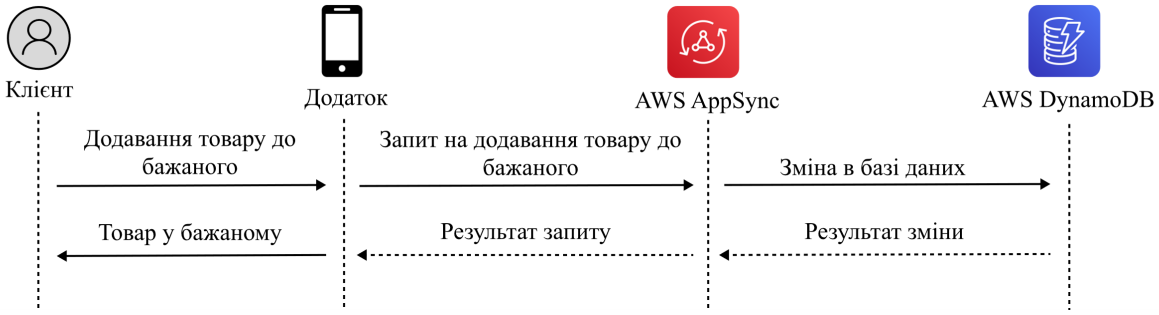


Рисунок А.8 – Діаграма послідовності для варіанту «Додавання товару до бажаного»

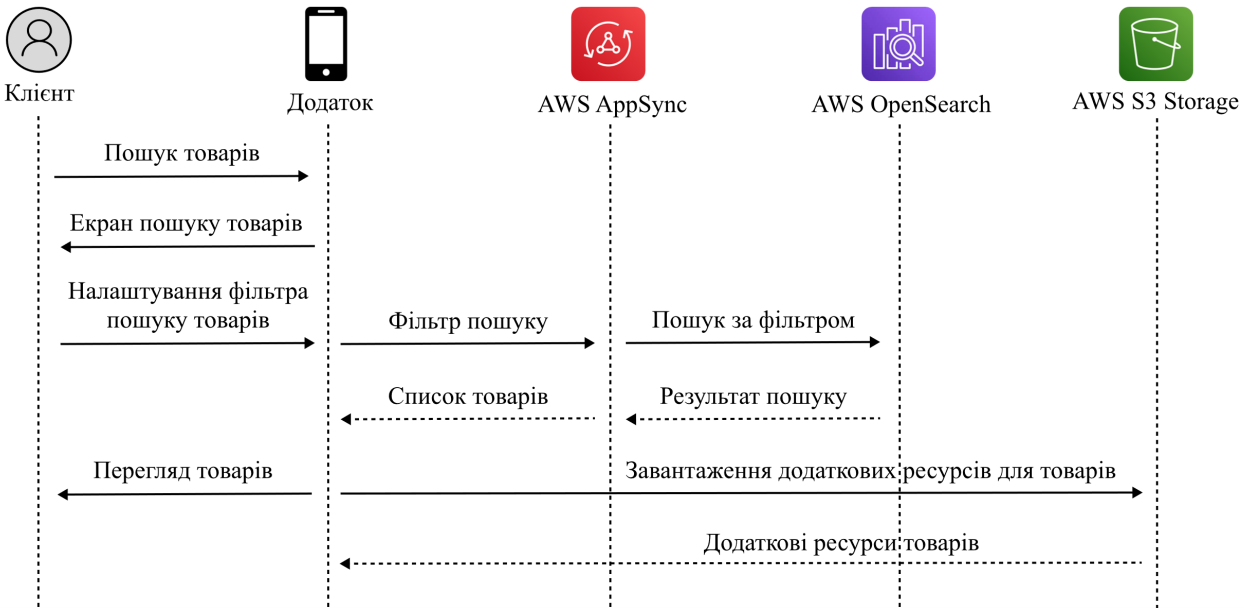


Рисунок А.9 – Діаграма послідовності для варіанту «Пошук товарів»

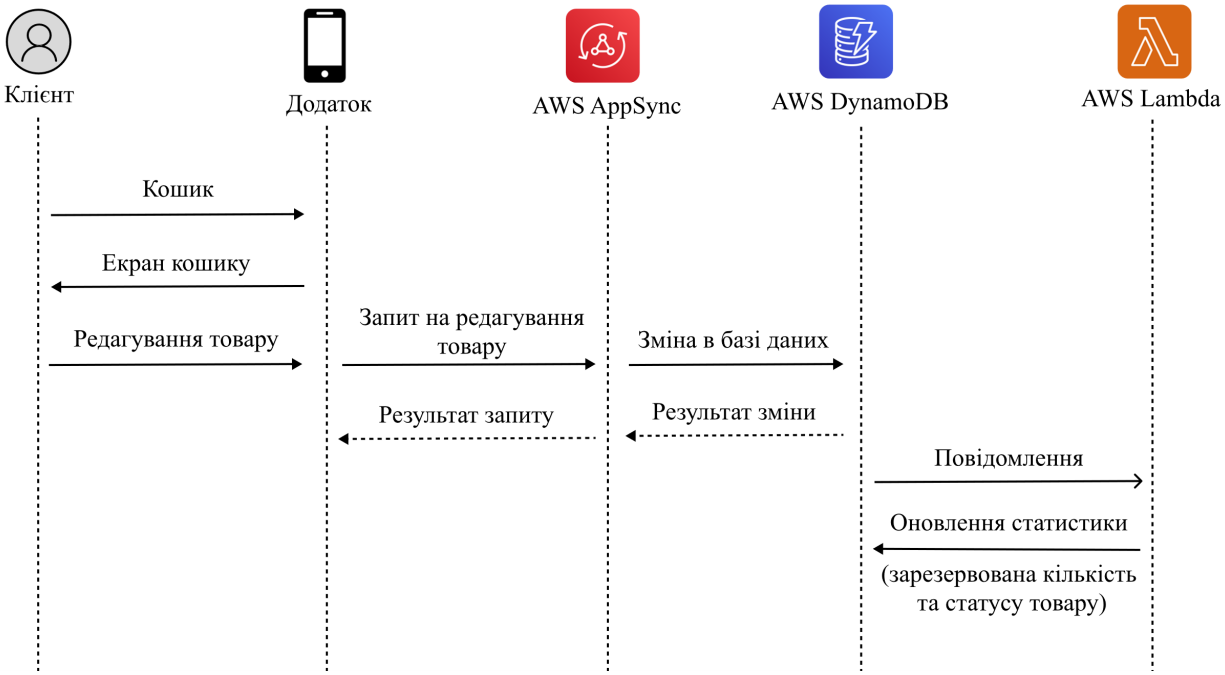


Рисунок А.10 – Діаграма послідовності для варіанту «Редагування в корзині»

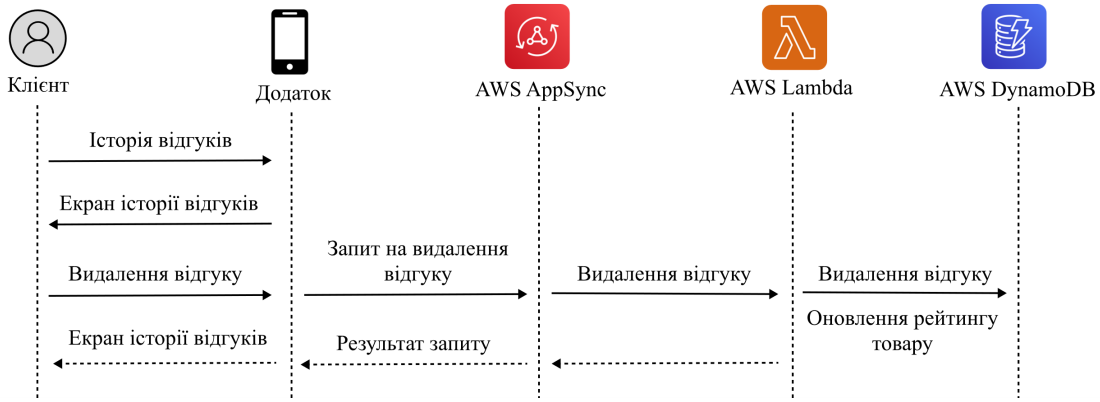


Рисунок А.11 – Діаграма послідовності для варіанту «Видалення відгуку»

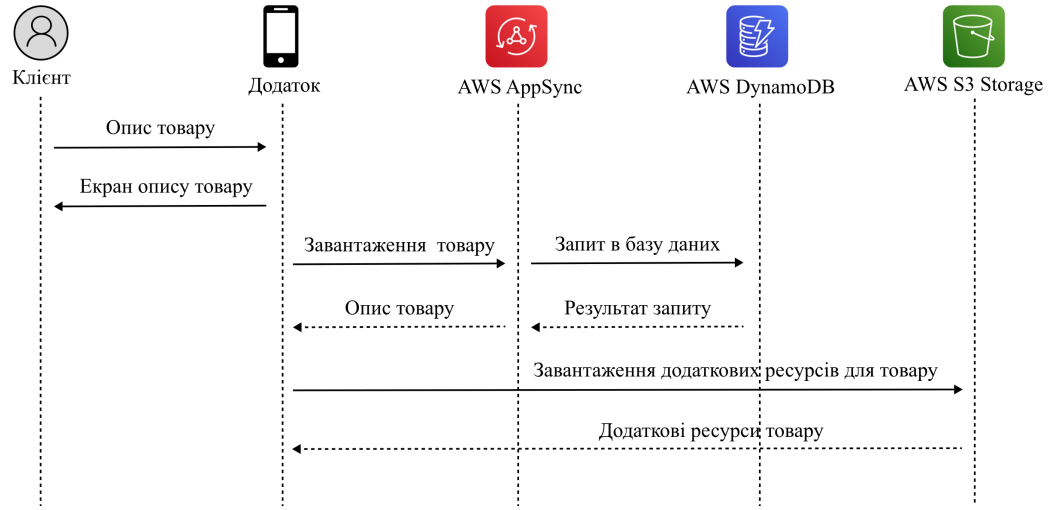


Рисунок А.12 – Діаграма послідовності для варіанту «Опис товару»

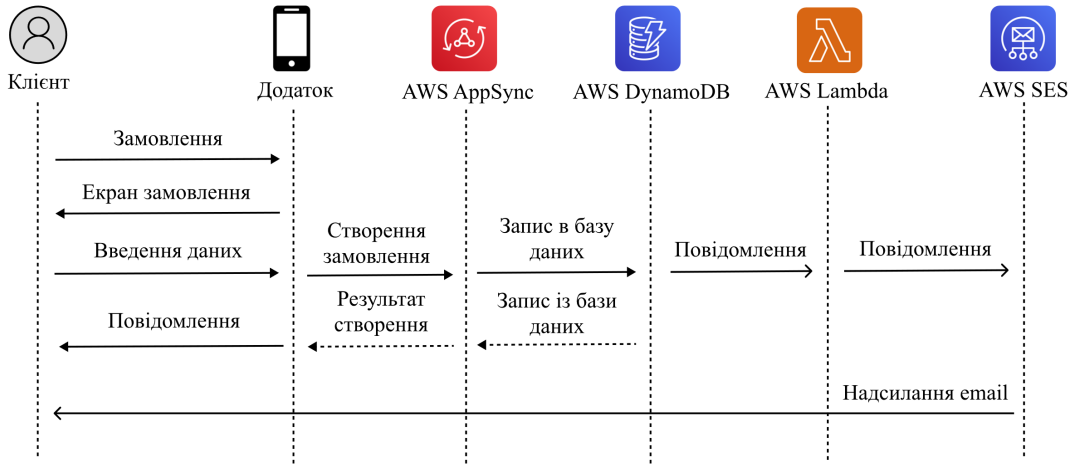


Рисунок А.13 – Діаграма послідовності для варіанту «Створення замовлення»

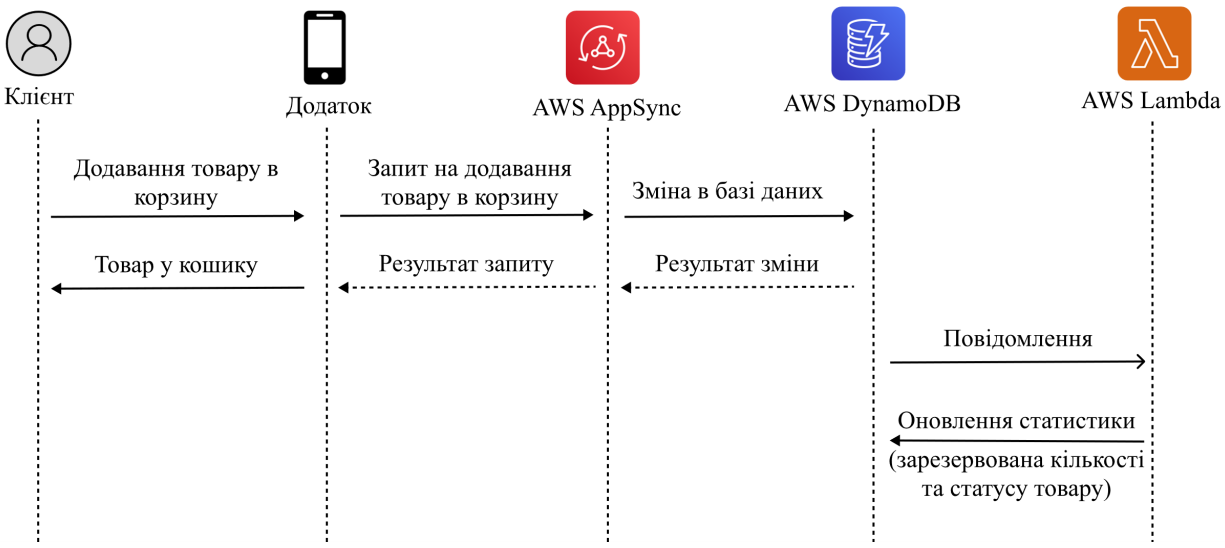


Рисунок А.14 – Діаграма послідовності для варіанту «Додавання товару в кошик»

ДОДАТОК Б
(обов’язковий)

ДІАГРАМИ ЗВЯЗКІВ ДЛЯ ВІДОБРАЖЕННЯ СТРУКТУРИ БАЗИ ДАНИХ

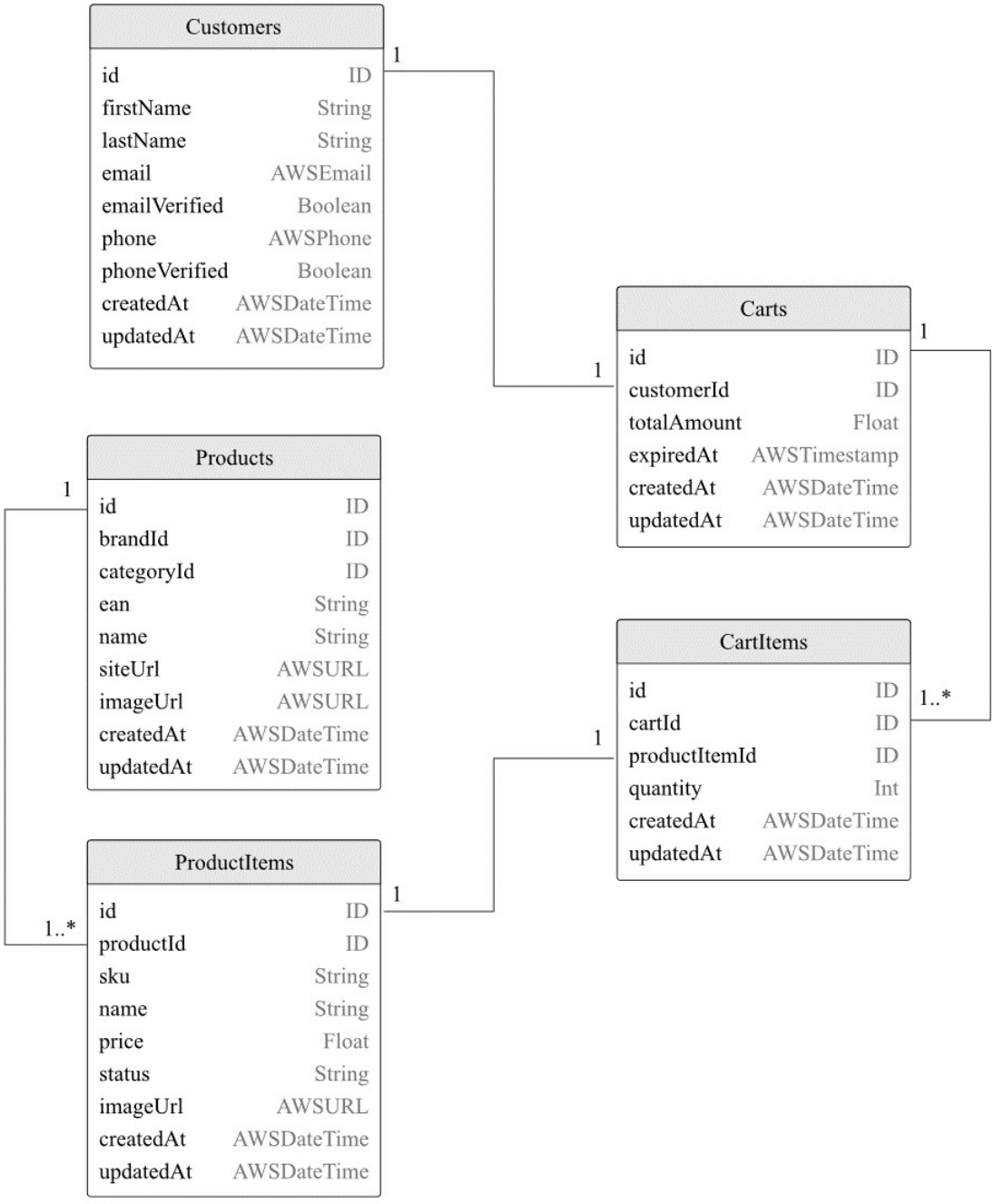


Рисунок Б.1 – Діаграма зв’язків «Користувач і його кошик»

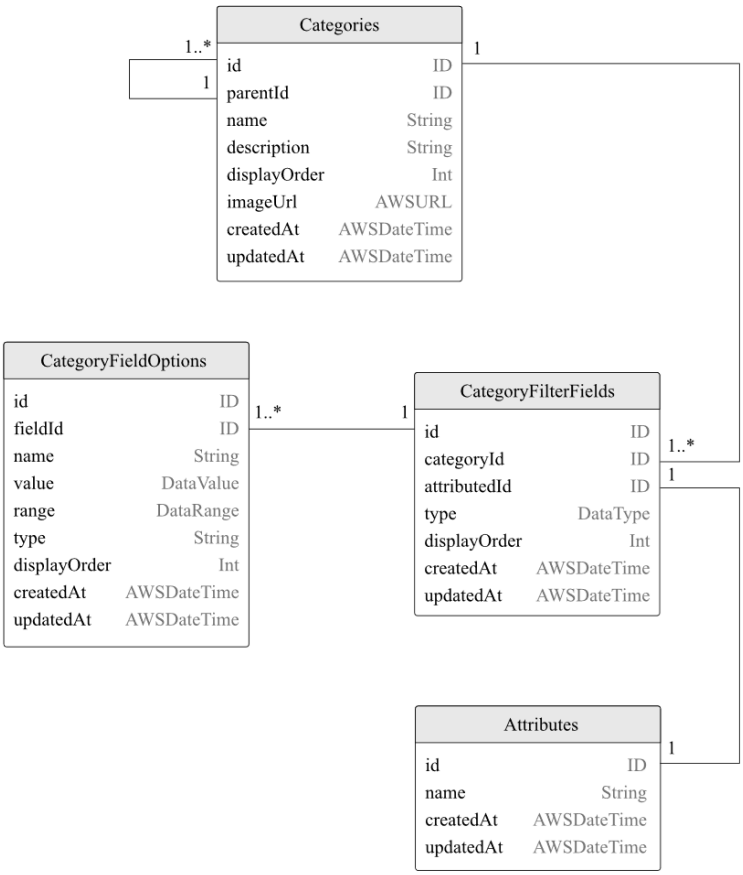


Рисунок Б.2 – Діаграма зв’язків «Категорія та пошукові фільтри»

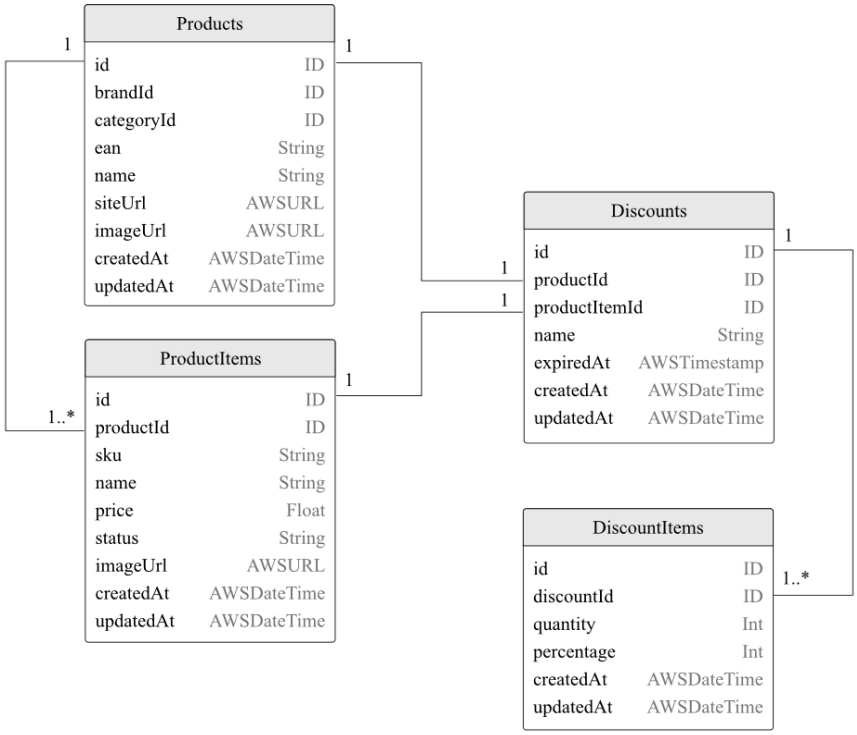


Рисунок Б.3 – Діаграма звязків «Товар і його знижки»

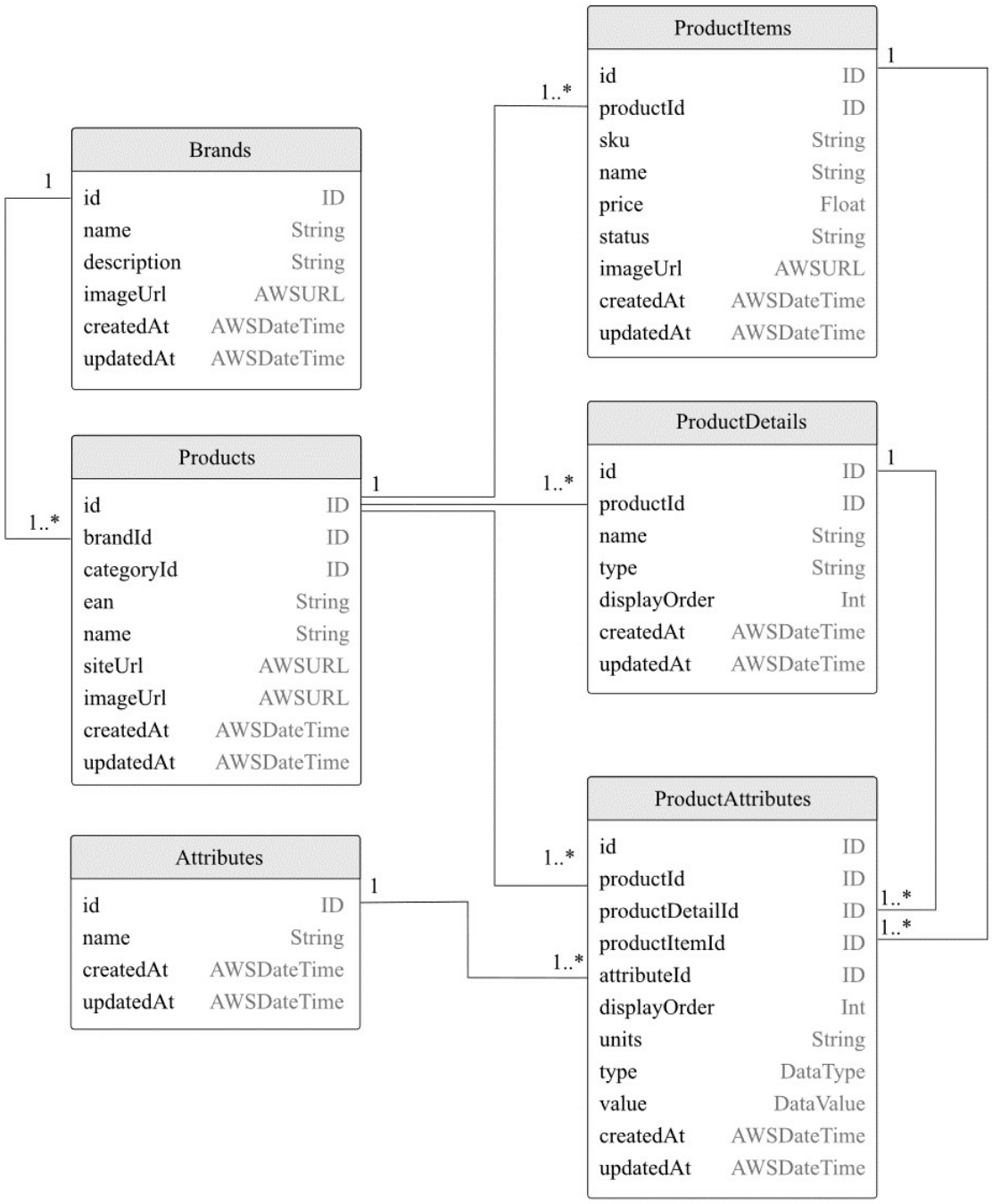


Рисунок Б.4 – Діаграма звязків «Товар і його атрибути»

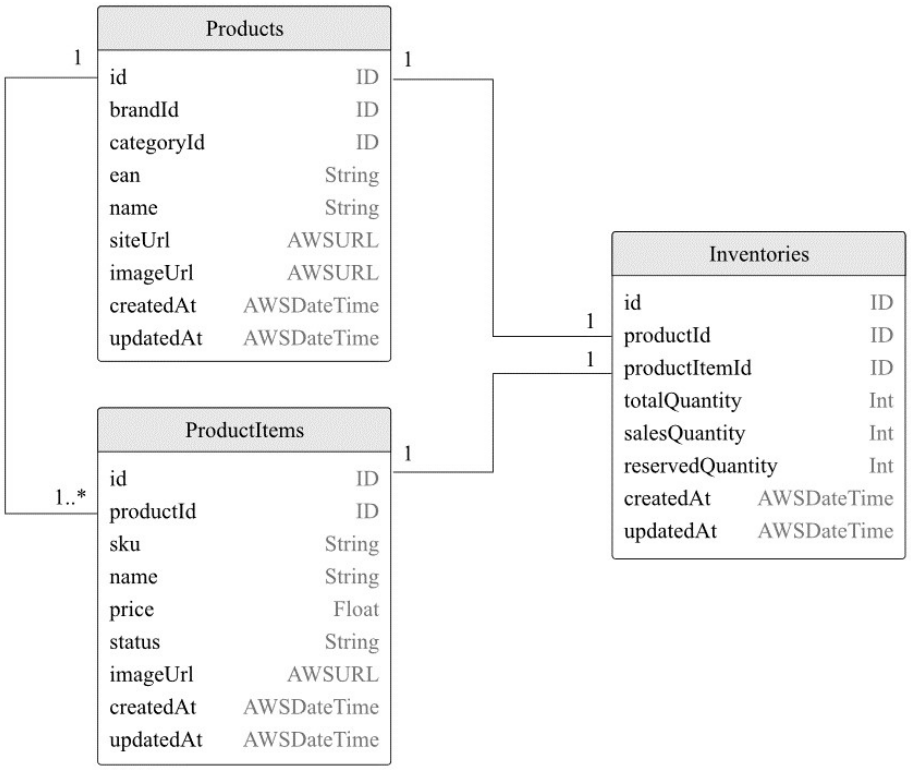


Рисунок Б.5 – Діаграма звязків «Товар і його запаси»

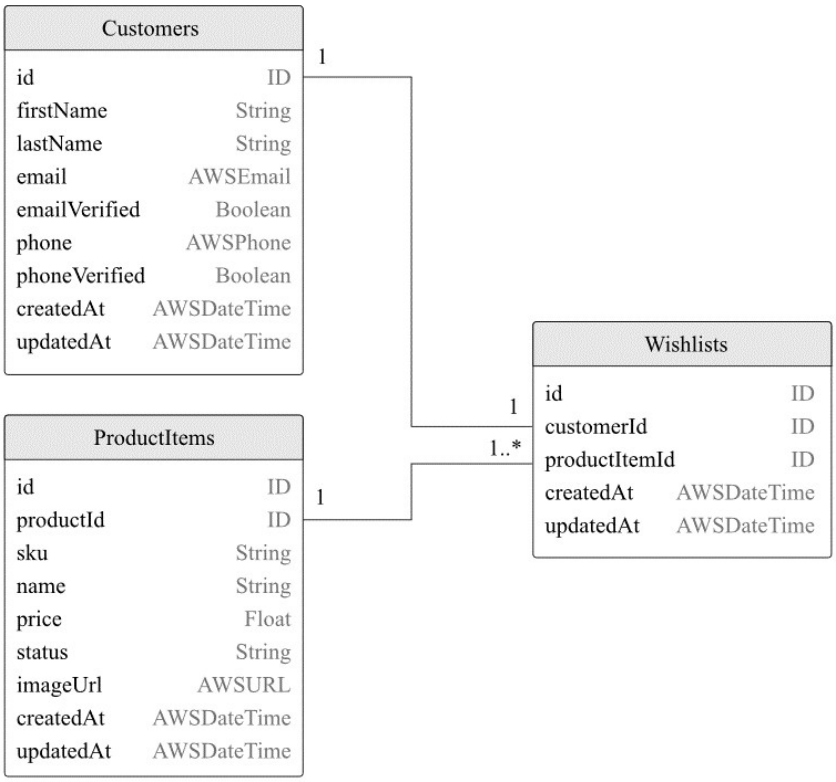


Рисунок Б.6 – Діаграма звязків «Користувач та його список бажань»

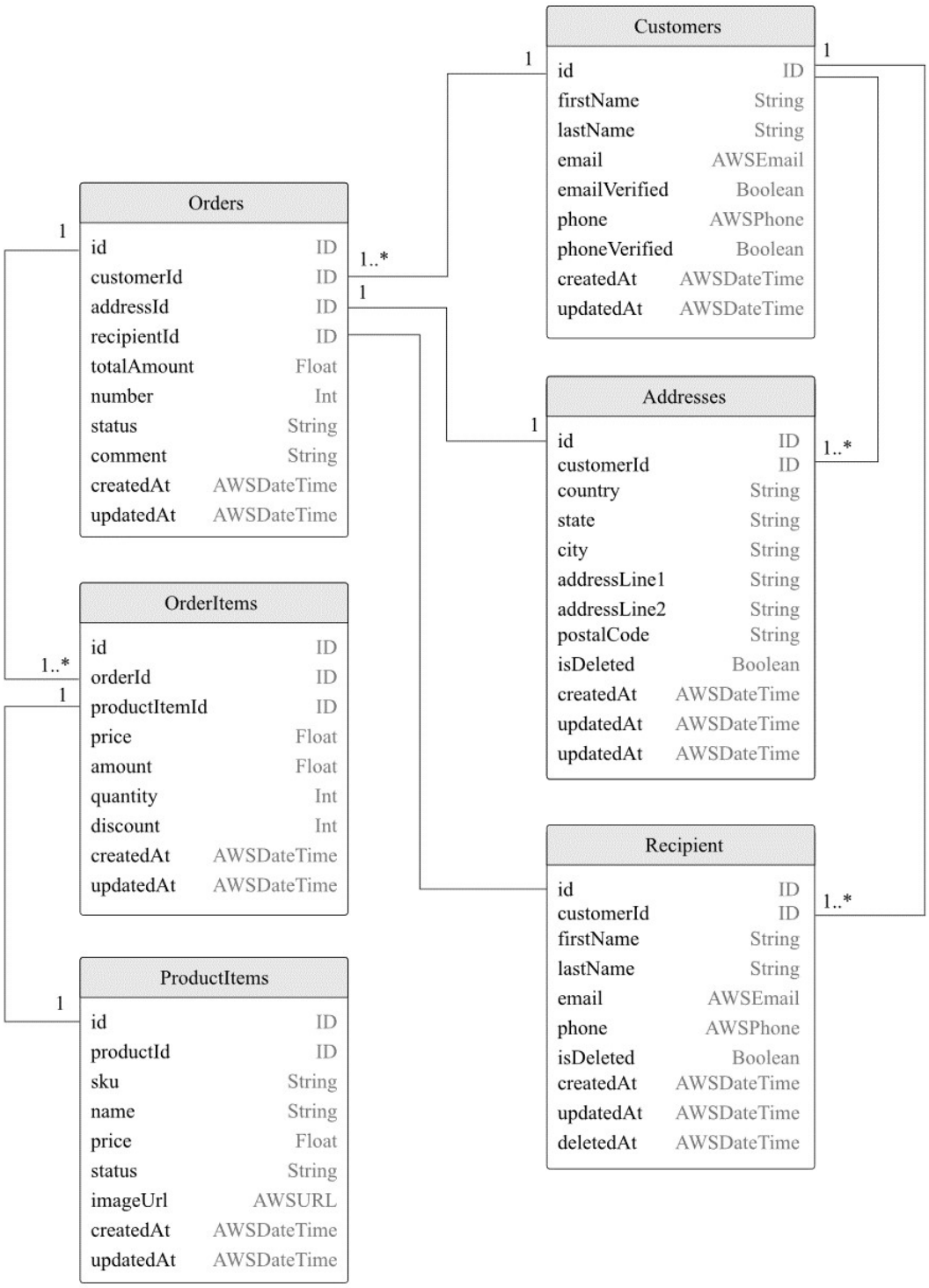


Рисунок Б.7– Діаграма звязків «Користувача і замовлення»

ДОДАТОК В
(обов'язковий)

ЕКРАНИ МОБІЛЬНОГО ЗАСТОСУНКУ

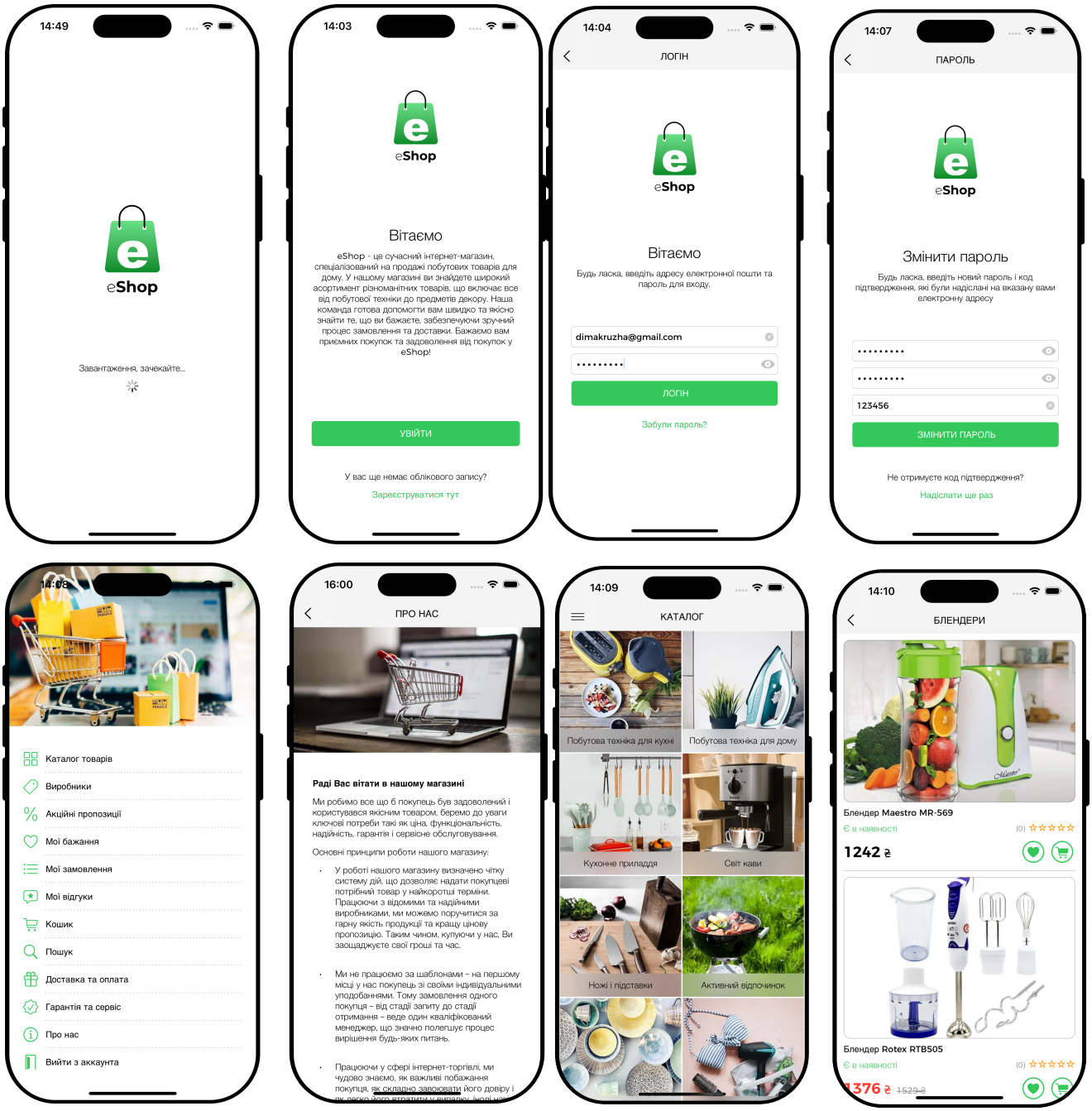


Рисунок В.1 – Перша частина екранів мобільного застосунку

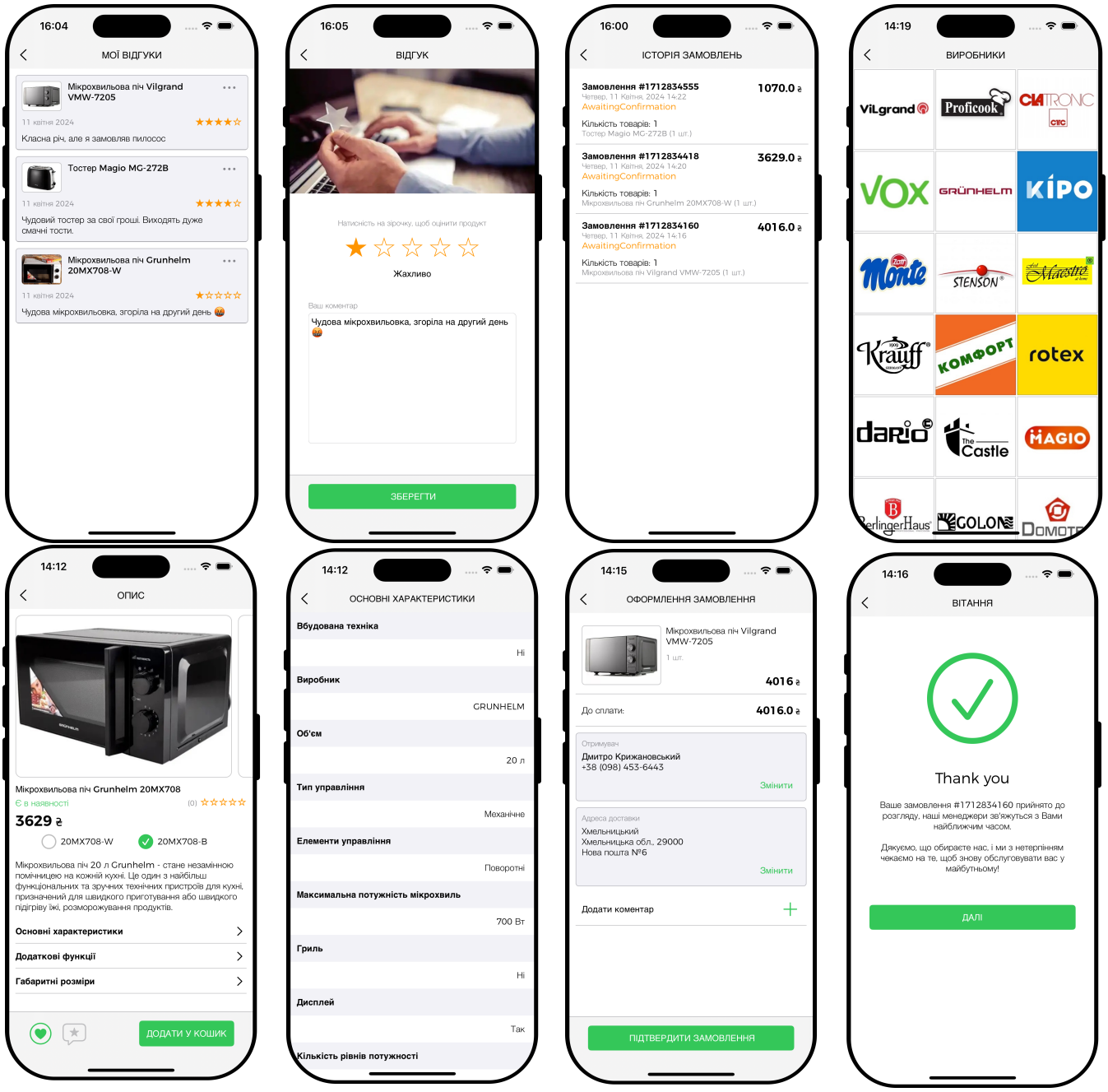


Рисунок В.2 – Друга частина екранів мобільного застосунку

ДОДАТОК Г
(обов'язковий)

КОД (ЛІСТИНГ) ЗАСТОСУНКУ

Код файлу AppDelegate:

```
//
//  AppDelegate.swift
//  eShop.Mobile
//
//  Created by Dmitro Kryzhanovsky on 27.02.2024.
//

import UIKit
import FirebaseCore
import FirebaseCrashlytics
import AWSCognitoIdentityProvider

@main
class AppDelegate: UIResponder, UIApplicationDelegate {

    lazy var window: UIWindow? = {
        let window = UIWindow()
        window.backgroundColor = .systemBackground
        return window
    }()

    lazy var navigationController: UINavigationController = {
        let controller = UINavigationController(nibName: nil, bundle: nil)

        let navigationBaAppearance = UINavigationBarAppearance()
        navigationBaAppearance.configureWithTransparentBackground()
        navigationBaAppearance.backgroundEffect = UIBlurEffect(style:
.systemMaterial)

        controller.navigationBar.standardAppearance = navigationBaAppearance
        controller.navigationBar.compactAppearance = navigationBaAppearance
        controller.navigationBar.scrollEdgeAppearance = navigationBaAppearance
        controller.navigationBar.backgroundColor = .clear
        controller.navigationBar.tintColor = .label
        controller.navigationBar.isHidden = true
        return controller
    }()

    func application(_ application: UIApplication, didFinishLaunchingWithOptions
launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {
        FirebaseApp.configure()

        AppLog.info("Bundle: \(AppInfo.bundleInfo)")
        AppLog.info("Device: \(AppInfo.deviceInfo)")
        AppLog.info("System: \(AppInfo.systemInfo)")
        AppLog.info("Locale: \(AppInfo.localeInfo)")

        let launchScreen = LaunchScreenController()
        window?.rootViewController = launchScreen
        window?.makeKeyAndVisible()

        //AppManager.shared.clearCache()
```

```
        checkSession()
        return true
    }

    func checkSession() {
        guard let currentUser = AppManager.shared.userPool.currentUser() else {
            presentView(with: nil)
            return
        }

        AppManager.shared.userPool.delegate = self

        currentUser.clearSession()
        currentUser.getSession().continueWith(block: { task in
            DispatchQueue.main.async {
                if task.result != nil {
                    self.presentView(with: task.result)
                } else {
                    currentUser.signOut()
                    self.presentView(with: nil)
                }
            }
        })
    }

    func presentView(with session: AWSCognitoIdentityUserSession?) {
        navigationController.view.alpha = 0
        navigationController.pushViewController(WelcomeViewController(),
animated: false)
        if let _ = session {
            navigationController.pushViewController(MainViewController(),
animated: false)
        }
        navigationController.viewControllers.forEach({ $0.loadViewIfNeeded() })

        window?.rootViewController = navigationController
        window?.makeKeyAndVisible()

        UIView.animate(withDuration: 0.3, animations: {
            self.navigationController.view.alpha = 1
        })
    }
}

//MARK: AWSCognitoIdentityInteractiveAuthenticationDelegate
extension AppDelegate: AWSCognitoIdentityInteractiveAuthenticationDelegate {
}

//MARK: About
extension AppDelegate {
    @objc static func showAbout() {
        guard let delegate = UIApplication.shared.delegate as? AppDelegate else
{ return }
        let imageAttachment = AppImage.Logo.app?.makeAttachment(maxPixelSize:
Int(72 * AppFont.scale))
        delegate.navigationController.presentAlert(imageAttachment:
imageAttachment, attrubutedMessage: aboutInfo)
    }

    static var aboutInfo: NSAttributedString {
        let attributedInfo = NSMutableAttributedString()
```

```
        attributedInfo.append(NSAttributedString(string:
AppString.Common.version.localized() + " ",
                                                 attributes: [.font:
AppFont.Alert.message, .foregroundColor: UIColor.label]))
        attributedInfo.append(NSAttributedString(string:
"\(AppInfo.bundleVersion) (\(AppInfo.bundleBuild))\n",
                                                 attributes: [.font:
AppFont.Alert.message, .foregroundColor: UIColor.secondaryLabel]))
        return attributedInfo
    }
}
```

Код файлу MainViewController:

```
//
//  MainViewController.swift
//  eShop.Mobile
//
//  Created by Dmitro Kryzhanovsky on 20.03.2024.
//  Copyright © 2024 Home. All rights reserved.
//

import Foundation
import UIKit

class MainViewController: UIViewController {

    init() {
        super.init(nibName: nil, bundle: nil)
    }

    required init?(coder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }

    private lazy var viewControllers: [UIViewController] = {
        return [menuNavigationController, viewNavigationController]
    }()

    private lazy var menuViewController: MenuViewController = {
        let controller = MenuViewController()
        controller.delegate = self
        return controller
    }()

    private lazy var menuNavigationController: UINavigationController = {
        return UIKit.UINavigationController(rootViewController:
menuViewController)
    }()

    private lazy var viewNavigationController: UINavigationController = {
        let controller = UINavigationController(rootViewController:
CategoryViewController(category: nil, delegate: self))

        let navigationBaAppearance = UINavigationBarAppearance()
        navigationBaAppearance.configureWithTransparentBackground()
        navigationBaAppearance.backgroundEffect = UIBlurEffect(style:
.systemMaterial)

        controller.navigationBar.standardAppearance = navigationBaAppearance
        controller.navigationBar.compactAppearance = navigationBaAppearance
        controller.navigationBar.scrollEdgeAppearance = navigationBaAppearance
```

```
        controller.navigationBar.backgroundColor = .clear
        controller.navigationBar.tintColor = .label
        controller.navigationBar.isHidden = false
        return controller
    }()

    private lazy var pageViewController: UIPageViewController = {
        let controller = UIPageViewController(transitionStyle: .scroll,
navigationOrientation: .horizontal, options: nil)
        controller.view.backgroundColor = .clear
        controller.delegate = self
        controller.dataSource = self
        return controller
    }()

    override func viewDidLoad() {
        super.viewDidLoad()
        view.clipsToBounds = true
        view.backgroundColor = .systemBackground
        navigationItem.leftBarButtonItem = UIBarButtonItem(image:
AppImage.Chevron.left, style: .plain, target: self, action: nil)
        navigationController?.navigationBar.isHidden = true
        //navigationController?.interactivePopGestureRecognizer?.isEnabled =
false

        view.addSubview(pageViewController.view)
        addChild(pageViewController)
        pageViewController.didMove(toParent: self)

        pageViewController.view.translatesAutoresizingMaskIntoConstraints =
false

        NSLayoutConstraint.activate([
            pageViewController.view.topAnchor.constraint(equalTo:
view.topAnchor),
            pageViewController.view.leftAnchor.constraint(equalTo:
view.leftAnchor),
            pageViewController.view.rightAnchor.constraint(equalTo:
view.rightAnchor),
            pageViewController.view.bottomAnchor.constraint(equalTo:
view.bottomAnchor)
        ])

        reloadData()
    }

    private func reloadData() {
        viewControllers.compactMap({ $0 as? UINavigationController })
            .forEach({
                $0.navigationBar.tintColor = UIColor.label
                $0.topViewController?.loadViewIfNeeded()
            })

        selectPage(with: 0)
    }

    private func selectPage(with index: Int, animated: Bool = true) {
        guard let selectedViewController = index < viewControllers.count ?
viewControllers[index] : nil else { return }
        let selectedViewControllerIndex = viewControllers.firstIndex(of:
selectedViewController)!
```

```
        var currentViewControllerIndex = -1
        if let currentViewController = pageViewController.viewControllers?.first
{
            currentViewControllerIndex = viewControllers.firstIndex(of:
currentViewController)!
        }

        let navigationDirection = (currentViewControllerIndex >
selectedViewControllerIndex)
            ? UIPageViewController.NavigationDirection.reverse
            : UIPageViewController.NavigationDirection.forward

        pageViewController.setViewControllers([selectedViewController],
direction: navigationDirection, animated: animated, completion: nil)
    }

    private func presentCatalog() {
        if let viewController =
viewNavigationController.viewControllers.first(where: { $0 is
CategoryViewController }) {
            viewNavigationController.popToViewController(viewController,
animated: false)
        } else {
            let viewController = CategoryViewController(category: nil, delegate:
self)
            viewNavigationController.pushViewController(viewController,
animated: false)
        }
        selectPage(with: 1)
    }

    private func presentWishlist() {
        if let viewController =
viewNavigationController.viewControllers.first(where: { $0 is
WishlistViewController }) {
            viewNavigationController.popToViewController(viewController,
animated: false)
        } else {
            let viewController = WishlistViewController()
            viewNavigationController.pushViewController(viewController,
animated: false)
        }
        selectPage(with: 1)
    }

    private func presentCart() {
        if let viewController =
viewNavigationController.viewControllers.first(where: { $0 is CartViewController
}) {
            viewNavigationController.popToViewController(viewController,
animated: false)
        } else {
            let viewController = CartViewController()
            viewNavigationController.pushViewController(viewController,
animated: false)
        }
        selectPage(with: 1)
    }

    private func presentHistory() {
```

```
        if let viewController =
viewNavigationController.viewControllers.first(where: { $0 is
OrderHistoryViewController }) {
            viewNavigationController.popToViewController(viewController,
animated: false)
        } else {
            let viewController = OrderHistoryViewController()
            viewNavigationController.pushViewController(viewController,
animated: false)
        }
        selectPage(with: 1)
    }

    private func presentReviews() {
        if let viewController =
viewNavigationController.viewControllers.first(where: { $0 is
ReviewListViewController }) {
            viewNavigationController.popToViewController(viewController,
animated: false)
        } else {
            let viewController = ReviewListViewController()
            viewNavigationController.pushViewController(viewController,
animated: false)
        }
        selectPage(with: 1)
    }

    private func presentSearch() {
        let viewController = SearchViewController()
        viewNavigationController.viewControllers.removeAll(where: { $0 is
SearchViewController })
        viewNavigationController.pushViewController(viewController, animated:
false)
        selectPage(with: 1)
    }

    private func presentBrand() {
        if let viewController =
viewNavigationController.viewControllers.first(where: { $0 is
BrandViewController }) {
            viewNavigationController.popToViewController(viewController,
animated: false)
        } else {
            let viewController = BrandViewController()
            viewNavigationController.pushViewController(viewController,
animated: false)
        }
        selectPage(with: 1)
    }

    private func presentSpecials() {
        if let viewController =
viewNavigationController.viewControllers.first(where: { $0 is
SpecialViewController }) {
            viewNavigationController.popToViewController(viewController,
animated: false)
        } else {
            let viewController = SpecialViewController()
            viewNavigationController.pushViewController(viewController,
animated: false)
        }
        selectPage(with: 1)
```

```
    }

    private func presentStaticPage(with type: PageType) {
        if let viewController =
viewNavigationController.viewControllers.first(where: { ($0 as?
StaticPageViewController)?.type == type }) {
            viewNavigationController.popToViewController(viewController,
animated: false)
        } else {
            let viewController = StaticPageViewController(with: type)
            viewNavigationController.pushViewController(viewController,
animated: false)
        }
        selectPage(with: 1)
    }

    private func confirmSignOut() {
        presentQuestion(title: "Sign Out".localized(), message:
AppString.Messages.confirmSignOut, actions: [
            UIAlertAction(title: AppString.Button.yes.localized(), style:
.destructive, handler: { _ in self.performSignOut() }),
            UIAlertAction(title: AppString.Button.no.localized(), style:
.cancel, handler: nil)
        ])
    }

    private func performSignOut() {
        AppManager.shared.clearCache()
        AppManager.shared.userPool.currentUser()?.signOut()
        navigationController?.popViewController(animated: true)
    }
}

//MARK: UIPageViewControllerDataSource
//
extension MainViewController: UIPageViewControllerDataSource {
    func pageViewController(_ pageViewController: UIPageViewController,
viewControllerBefore viewController: UIViewController) -> UIViewController? {
        if let pageIndex = viewControllers.firstIndex(of: viewController),
pageIndex > 0 {
            return viewControllers[pageIndex - 1]
        }
        return nil
    }

    func pageViewController(_ pageViewController: UIPageViewController,
viewControllerAfter viewController: UIViewController) -> UIViewController? {
        if let pageIndex = viewControllers.firstIndex(of: viewController),
pageIndex < viewControllers.count - 1 {
            return viewControllers[pageIndex + 1]
        }
        return nil
    }
}

//MARK: UIPageViewControllerDelegate
//
extension MainViewController: UIPageViewControllerDelegate {
    func pageViewController(_ pageViewController: UIPageViewController,
didFinishAnimating finished: Bool, previousViewControllers: [UIViewController],
transitionCompleted completed: Bool) {
        guard completed else { return }
```

```
        guard let viewController = pageViewController.viewControllers?.first
else { return }
        if let pageIndex = self.viewControllers.firstIndex(of: viewController) {
            selectPage(with: pageIndex)
        }
    }
}

//MARK: MenuViewControllerDelegate
//
extension MainViewController: MenuViewControllerDelegate {
    func menu(viewController: MenuViewController, didSelect item:
MenuItemViewType) {
        selectPage(with: 0, animated: false)

        switch item {
        case .catalog: presentCatalog(); break
        case .brand: presentBrand(); break
        case .cart: presentCart(); break
        case .wishlist: presentWishlist(); break
        case .orderHistory: presentHistory(); break
        case .reviewHistory: presentReviews(); break
        case .signOut: confirmSignOut(); break
        case .search: presentSearch(); break
        case .about: presentStaticPage(with: .about); break
        case .specials: presentSpecials(); break
        case .delivery: presentStaticPage(with: .delivery); break
        case .warranty: presentStaticPage(with: .warranty); break
        }
    }
}

//MARK: CategoryViewControllerDelegate
//
extension MainViewController: CategoryViewControllerDelegate {
    func category(viewController: CategoryViewController, didHome: AnyObject?) {
        selectPage(with: 0)
    }
}
```

Код файлу MenuViewController:

```
//
//  MenuViewController.swift
//  eShop.Mobile
//
//  Created by Dmitro Kryzhanovsky on 19.03.2024.
//  Copyright © 2024 Home. All rights reserved.
//

import Foundation
import UIKit

protocol MenuViewControllerDelegate: AnyObject {
    func menu(viewController: MenuViewController, didSelect item:
MenuItemViewType)
}

extension MenuViewControllerDelegate {
    func menu(viewController: MenuViewController, didSelect item:
MenuItemViewType) {}
```

```
}

class MenuViewController: UIViewController {

    weak var delegate: MenuViewControllerDelegate? = nil
    var items: [MenuItemViewModel] = []

    init() {
        super.init(nibName: nil, bundle: nil)
        self.title = AppString.View.Menu.title.localized()
    }

    required init?(coder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }

    struct Sections {
        public static var header = 0
        public static var content = 1
    }

    private lazy var layoutForHeader: NSCollectionLayoutSection = {
        let itemSize = NSCollectionLayoutSize(widthDimension:
.fractionalWidth(1), heightDimension: .fractionalWidth(0.65))
        let item = NSCollectionLayoutItem(layoutSize: itemSize)
        item.contentInsets = NSDirectionalEdgeInsets.zero

        let groupSize = NSCollectionLayoutSize(widthDimension:
.fractionalWidth(1), heightDimension: .estimated(1))
        let group = NSCollectionLayoutGroup.horizontal(layoutSize: groupSize,
subitems: [item])

        let section = NSCollectionLayoutSection(group: group)
        section.contentInsets = NSDirectionalEdgeInsets(top: 0, leading: 0,
bottom: 0, trailing: 0)
        section.interGroupSpacing = 0
        return section
    }()

    private lazy var layoutForDefault: NSCollectionLayoutSection = {
        let itemSize = NSCollectionLayoutSize(widthDimension:
.fractionalWidth(1), heightDimension: .estimated(20))
        let item = NSCollectionLayoutItem(layoutSize: itemSize)
        item.contentInsets = NSDirectionalEdgeInsets.zero

        let groupSize = NSCollectionLayoutSize(widthDimension:
.fractionalWidth(1), heightDimension: .estimated(1))
        let group = NSCollectionLayoutGroup.horizontal(layoutSize: groupSize,
subitems: [item])

        let section = NSCollectionLayoutSection(group: group)
        section.contentInsets = NSDirectionalEdgeInsets(top: 10, leading: 10,
bottom: 0, trailing: 10)
        section.interGroupSpacing = 0
        return section
    }()

    private lazy var collectionView: UICollectionView = {
        let configuration = UICollectionViewCompositionalLayoutConfiguration()
        configuration.interSectionSpacing = 20
        let layout = UICollectionViewCompositionalLayout(sectionProvider: {
(sectionIndex, enviroment) in
```

```
            if sectionIndex == Sections.header {
                return self.layoutForHeader
            } else {
                return self.layoutForDefault
            }
        }, configuration: configuration)
        let view = UICollectionView(frame: .zero, collectionViewLayout: layout)
        view.backgroundColor = .clear
        view.isScrollEnabled = true
        view.showsVerticalScrollIndicator = false
        view.showsHorizontalScrollIndicator = false
        view.indicatorStyle = .default
        view.bounces = true
        view.bouncesZoom = false
        view.alwaysBounceVertical = true
        view.alwaysBounceHorizontal = false
        view.clipsToBounds = false
        view.backgroundColor = .clear
        view.register(MenuHeaderViewCell.self, forCellWithReuseIdentifier:
MenuHeaderViewCell.defaultReuseIdentifier)
        view.register(MenuItemViewCell.self, forCellWithReuseIdentifier:
MenuItemViewCell.defaultReuseIdentifier)
        view.delegate = self
        view.dataSource = self
        return view
    }()

    override func viewDidLoad() {
        super.viewDidLoad()
        view.clipsToBounds = true
        view.backgroundColor = .systemBackground
        navigationController?.navigationBar.isHidden = true

        view.addSubview(collectionView)
        collectionView.translatesAutoresizingMaskIntoConstraints = false

        NSLayoutConstraint.activate([
            collectionView.topAnchor.constraint(equalTo: view.topAnchor),
            collectionView.leadingAnchor.constraint(equalTo:
view.leadingAnchor),
            collectionView.trailingAnchor.constraint(equalTo:
view.trailingAnchor),
            collectionView.bottomAnchor.constraint(equalTo: view.bottomAnchor)
        ])

        if let windowScene = UIApplication.shared.connectedScenes.first as?
UIWindowScene,
           let safeAreaInsetTop = windowScene.windows.first?.safeAreaInsets.top
{
            collectionView.contentInset.top = -safeAreaInsetTop
        }

        reloadData()
    }

    private func reloadData() {
        self.items = [
            MenuItemViewModel(type: .catalog),
            MenuItemViewModel(type: .brand),
            MenuItemViewModel(type: .specials),
            MenuItemViewModel(type: .wishlist),
            MenuItemViewModel(type: .orderHistory),
```

```
            MenuItemViewModel(type: .reviewHistory),
            MenuItemViewModel(type: .cart),
            MenuItemViewModel(type: .search),
            MenuItemViewModel(type: .delivery),
            MenuItemViewModel(type: .warranty),
            MenuItemViewModel(type: .about),
            MenuItemViewModel(type: .signOut, isSeparatorHidden: true),
        ]
        self.collectionView.reloadData()
    }
}

//MARK: UICollectionViewDataSource
//
extension MenuViewController: UICollectionViewDataSource {
    func numberOfSections(in collectionView: UICollectionView) -> Int {
        return 2
    }

    func collectionView(_ collectionView: UICollectionView,
numberOfItemsInSection section: Int) -> Int {
        if section == Sections.header {
            return 1
        } else {
            return self.items.count
        }
    }

    func collectionView(_ collectionView: UICollectionView, cellForItemAt
indexPath: IndexPath) -> UICollectionViewCell {
        if indexPath.section == Sections.header {
            let reuseIdentifier = MenuHeaderViewCell.defaultReuseIdentifier
            let cell = collectionView.dequeueReusableCell(withReuseIdentifier:
reuseIdentifier, for: indexPath) as! MenuHeaderViewCell
            return cell
        } else {
            let reuseIdentifier = MenuItemViewCell.defaultReuseIdentifier
            let cell = collectionView.dequeueReusableCell(withReuseIdentifier:
reuseIdentifier, for: indexPath) as! MenuItemViewCell
            cell.model = self.items[indexPath.item]
            return cell
        }
    }
}

//MARK: UICollectionViewDelegate
//
extension MenuViewController: UICollectionViewDelegate {
    func collectionView(_ collectionView: UICollectionView, didSelectItemAt
indexPath: IndexPath) {
        collectionView.deselectItem(at: indexPath, animated: true)
        delegate?.menu(viewController: self, didSelect:
items[indexPath.item].type)
    }
}
```

Код файлу ApiClient:

```
//
//  ApiClient.swift
//  eShop.Mobile
//
//  Created by Dmitro Kryzhanovsky on 04.03.2024.
```

```
//  Copyright © 2024 Home. All rights reserved.
//

import Foundation
import AWSAppSync
import AWSCognitoIdentityProvider

class ApiClient {

    private let appSync: AWSAppSyncClient
    private let userPool: AWSCognitoIdentityUserPool

    public init(userPool: AWSCognitoIdentityUserPool, userPoolsAuthProvider:
AWSCognitoUserPoolsAuthProvider, cacheConfiguration:
AWSAppSyncCacheConfiguration) {
        let appSyncClientConfiguration = try!
AWSAppSyncClientConfiguration(appSyncServiceConfig: AWSAppSyncServiceConfig(),

userPoolsAuthProvider: userPoolsAuthProvider,

cacheConfiguration: cacheConfiguration)

        let appSync = try! AWSAppSyncClient(appSyncConfig:
appSyncClientConfiguration)
        appSync.apolloClient?.cacheKeyForObject = { $0["id"] }
        self.userPool = userPool
        self.appSync = appSync
    }

    private func getUserAttributes(completion: @escaping
([AWSCognitoIdentityProviderAttributeType]?, ApiError?) -> Void) {
        userPool.currentUser()?.getDetails().continueOnSuccessWith { (task) ->
AnyObject? in
            if let error = task.error {
                completion(nil, ApiError(error.localizedDescription))
                return nil
            }

            if let userAttributes = task.result?.userAttributes {
                completion(userAttributes, nil)
                return nil
            }

            completion(nil, nil)
            return nil
        }
    }

    public func clearCache() throws {
        try appSync.clearCaches()
    }

    public func getCart(cachePolicy: CachePolicy = .returnCacheDataAndFetch,
completion: @escaping (Cart?, ApiError?) -> Void) {
        getUserAttributes { [self] (userAttributes, error) in
            if let error = error {
                completion(nil, error)
                return
            }

            guard let customerId = userAttributes?.first(where: { $0.name ==
"sub" })?.value else {
```

```
                completion(nil, nil)
                return
            }

            let query = GetCartByCustomerQuery(customerId: customerId)
            appSync.fetch(query: query, cachePolicy: cachePolicy, queue:
.global()) { result, error in
                if let error = error {
                    completion(nil, ApiError(error.localizedDescription))
                    return
                }

                if let result = result {
                    let apiError = (result.errors?.isEmpty ?? true) ? nil :
ApiError(result.errors!)
                    if let snapshot = result.data?.getCartByCustomer?.snapshot {
                        let cart = Cart(snapshot: snapshot)
                        completion(cart, apiError)
                    } else {
                        completion(nil, apiError)
                    }
                } else {
                    completion(nil, nil)
                }
            }
        }
    }

    public func createCartItem(productItemId: String, quantity: Int = 1,
completion: @escaping (CartItem?, ApiError?) -> Void) {
        getUserAttributes { [self] (userAttributes, error) in
            if let error = error {
                completion(nil, error)
                return
            }

            guard let customerId = userAttributes?.first(where: { $0.name ==
"sub" })?.value else {
                completion(nil, nil)
                return
            }

            let input = CreateCartItemForCustomerInput(customerId: customerId,
productItemId: productItemId, quantity: quantity)
            let mutation = CreateCartItemForCustomerMutation(input: input)
            appSync.perform(mutation: mutation, queue: .global(),
optimisticUpdate: nil, conflictResolutionBlock: nil) { result, error in
                if let error = error {
                    completion(nil, ApiError(error.localizedDescription))
                    return
                }

                if let result = result {
                    let apiError = (result.errors?.isEmpty ?? true) ? nil :
ApiError(result.errors!)
                    if let snapshot =
result.data?.createCartItemForCustomer?.snapshot {
                        let cartItem = CartItem(snapshot: snapshot)
                        completion(cartItem, apiError)
                    } else {
                        completion(nil, apiError)
                    }
```

```
                } else {
                    completion(nil, nil)
                }
            }
        }
    }

    public func updateCartItem(cartItemId: String, quantity: Int, completion:
@escaping (CartItem?, ApiError?) -> Void) {
        let input = UpdateCartItemInput(id: cartItemId, quantity: quantity)
        let mutation = UpdateCartItemMutation(input: input)
        appSync.perform(mutation: mutation, queue: .global(), optimisticUpdate:
nil, conflictResolutionBlock: nil) { result, error in
            if let error = error {
                completion(nil, ApiError(error.localizedDescription))
                return
            }

            if let result = result {
                let apiError = (result.errors?.isEmpty ?? true) ? nil :
ApiError(result.errors!)
                if let snapshot = result.data?.updateCartItem?.snapshot {
                    let cartItem = CartItem(snapshot: snapshot)
                    completion(cartItem, apiError)
                } else {
                    completion(nil, apiError)
                }
            } else {
                completion(nil, nil)
            }
        }
    }

    public func deleteCartItem(cartItemId: String, completion: @escaping
(CartItem?, ApiError?) -> Void) {
        let input = DeleteCartItemInput(id: cartItemId)
        let mutation = DeleteCartItemMutation(input: input)
        appSync.perform(mutation: mutation, queue: .global(), optimisticUpdate:
nil, conflictResolutionBlock: nil) { result, error in
            if let error = error {
                completion(nil, ApiError(error.localizedDescription))
                return
            }

            if let result = result {
                let apiError = (result.errors?.isEmpty ?? true) ? nil :
ApiError(result.errors!)
                if let snapshot = result.data?.deleteCartItem?.snapshot {
                    let cartItem = CartItem(snapshot: snapshot)
                    completion(cartItem, apiError)
                } else {
                    completion(nil, apiError)
                }
            } else {
                completion(nil, nil)
            }
        }
    }

    public func createWishlist(productItemId: String, completion: @escaping
(Wishlist?, ApiError?) -> Void) {
        getUserAttributes { [self] (userAttributes, error) in
```

```
            if let error = error {
                completion(nil, error)
                return
            }

            guard let customerId = userAttributes?.first(where: { $0.name ==
"sub" })?.value else {
                completion(nil, nil)
                return
            }

            let input = CreateWishlistInput(customerId: customerId,
productItemId: productItemId)
            let mutation = CreateWishlistMutation(input: input)
            appSync.perform(mutation: mutation, queue: .global(),
optimisticUpdate: nil, conflictResolutionBlock: nil) { result, error in
                if let error = error {
                    completion(nil, ApiError(error.localizedDescription))
                    return
                }

                if let result = result {
                    let apiError = (result.errors?.isEmpty ?? true) ? nil :
ApiError(result.errors!)
                    if let snapshot = result.data?.createWishlist?.snapshot {
                        let wishlist = Wishlist(snapshot: snapshot)
                        completion(wishlist, apiError)
                    } else {
                        completion(nil, apiError)
                    }
                } else {
                    completion(nil, nil)
                }
            }
        }
    }

    public func deleteWishlist(wishlistId: String, completion: @escaping
(Wishlist?, ApiError?) -> Void) {
        let input = DeleteWishlistInput(id: wishlistId)
        let mutation = DeleteWishlistMutation(input: input)
        appSync.perform(mutation: mutation, queue: .global(), optimisticUpdate:
nil, conflictResolutionBlock: nil) { result, error in
            if let error = error {
                completion(nil, ApiError(error.localizedDescription))
                return
            }

            if let result = result {
                let apiError = (result.errors?.isEmpty ?? true) ? nil :
ApiError(result.errors!)
                if let snapshot = result.data?.deleteWishlist?.snapshot {
                    let wishlist = Wishlist(snapshot: snapshot)
                    completion(wishlist, apiError)
                } else {
                    completion(nil, apiError)
                }
            } else {
                completion(nil, nil)
            }
        }
    }
```

```
    public func getCategoriesByParent(parentId: String?, completion: @escaping
(Connection<Category>?, ApiError?) -> Void) {
        let query = ListCategoriesByParentQuery(parentId: parentId ??
UUID.zero.uuidString)
        appSync.fetch(query: query, cachePolicy: .returnCacheDataAndFetch,
queue: .global()) { result, error in
            if let error = error {
                completion(nil, ApiError(error.localizedDescription))
                return
            }

            if let result = result {
                let apiError = (result.errors?.isEmpty ?? true) ? nil :
ApiError(result.errors!)
                if let snapshot = result.data?.listCategoriesByParent?.snapshot
{
                    let categoryConnection = Connection<Category>(snapshot:
snapshot)
                    completion(categoryConnection, apiError)
                } else {
                    completion(nil, apiError)
                }
            } else {
                completion(nil, nil)
            }
        }
    }

    public func getWishlists(completion: @escaping (Connection<Wishlist>?,
ApiError?) -> Void) {
        getUserAttributes { [self] (userAttributes, error) in
            if let error = error {
                completion(nil, error)
                return
            }

            guard let customerId = userAttributes?.first(where: { $0.name ==
"sub" })?.value else {
                completion(nil, nil)
                return
            }

            let query = ListWishlistsByCustomerQuery(customerId: customerId)
            appSync.fetch(query: query, cachePolicy: .returnCacheDataAndFetch,
queue: .global()) { result, error in
                if let error = error {
                    completion(nil, ApiError(error.localizedDescription))
                    return
                }

                if let result = result {
                    let apiError = (result.errors?.isEmpty ?? true) ? nil :
ApiError(result.errors!)
                    if let snapshot =
result.data?.listWishlistsByCustomer?.snapshot {
                        let wishlistConnection = Connection<Wishlist>(snapshot:
snapshot)
                        completion(wishlistConnection, apiError)
                    } else {
                        completion(nil, apiError)
                    }
```

```
                } else {
                    completion(nil, nil)
                }
            }
        }
    }

    public func getProductsByCategory(categoryId: String, completion: @escaping
(Connection<Product>?, ApiError?) -> Void) {
        let query = ListProductsByCategoryQuery(categoryId: categoryId)
        appSync.fetch(query: query, cachePolicy: .returnCacheDataAndFetch,
queue: .global()) { result, error in
            if let error = error {
                completion(nil, ApiError(error.localizedDescription))
                return
            }

            if let result = result {
                let apiError = (result.errors?.isEmpty ?? true) ? nil :
ApiError(result.errors!)
                if let snapshot = result.data?.listProductsByCategory?.snapshot
{
                    let productConnection = Connection<Product>(snapshot:
snapshot)
                    completion(productConnection, apiError)
                } else {
                    completion(nil, apiError)
                }
            } else {
                completion(nil, nil)
            }
        }
    }

    public func getProductDetails(productId: String, completion: @escaping
(Connection<ProductDetail>?, ApiError?) -> Void) {
        let query = ListProductDetailsByProductQuery(productId: productId)
        appSync.fetch(query: query, cachePolicy: .returnCacheDataAndFetch,
queue: .global()) { result, error in
            if let error = error {
                completion(nil, ApiError(error.localizedDescription))
                return
            }

            if let result = result {
                let apiError = (result.errors?.isEmpty ?? true) ? nil :
ApiError(result.errors!)
                if let snapshot =
result.data?.listProductDetailsByProduct?.snapshot {
                    let productDetailsConnection =
Connection<ProductDetail>(snapshot: snapshot)
                    completion(productDetailsConnection, apiError)
                } else {
                    completion(nil, apiError)
                }
            } else {
                completion(nil, nil)
            }
        }
    }
```

```
    public func getProduct(productId: String, cachePolicy: CachePolicy =
.returnCacheDataAndFetch, completion: @escaping (Product?, ApiError?) -> Void) {
        let query = GetProductQuery(id: productId)
        appSync.fetch(query: query, cachePolicy: cachePolicy, queue: .global())
{ result, error in
            if let error = error {
                completion(nil, ApiError(error.localizedDescription))
                return
            }

            if let result = result {
                let apiError = (result.errors?.isEmpty ?? true) ? nil :
ApiError(result.errors!)
                if let snapshot = result.data?.getProduct?.snapshot {
                    let product = Product(snapshot: snapshot)
                    completion(product, apiError)
                } else {
                    completion(nil, apiError)
                }
            } else {
                completion(nil, nil)
            }
        }
    }

    public func getStaticPage(type: PageType, cachePolicy: CachePolicy =
.returnCacheDataAndFetch, completion: @escaping (StaticPage?, ApiError?) ->
Void) {
        let filter = ListStaticPageFilterInput(type: TableStringFilterInput(eq:
type.rawValue))
        let query = ListStaticPagesQuery(filter: filter)
        appSync.fetch(query: query, cachePolicy: cachePolicy, queue: .global())
{ result, error in
            if let error = error {
                completion(nil, ApiError(error.localizedDescription))
                return
            }

            if let result = result {
                let apiError = (result.errors?.isEmpty ?? true) ? nil :
ApiError(result.errors!)
                if let snapshot = result.data?.listStaticPages?.snapshot {
                    let staticPage = (Connection<StaticPage>(snapshot:
snapshot).items ?? []).first
                    completion(staticPage, apiError)
                } else {
                    completion(nil, apiError)
                }
            } else {
                completion(nil, nil)
            }
        }
    }

    public func createOrder(cartId: String, recipientId: String, addressId:
String, comment: String?, completion: @escaping (Order?, ApiError?) -> Void) {
        let input = CreateOrderByCartInput(cartId: cartId, addressId: addressId,
recipientId: recipientId, comment: comment)
        let mutation = CreateOrderByCartMutation(input: input)
        appSync.perform(mutation: mutation, queue: .global(), optimisticUpdate:
nil, conflictResolutionBlock: nil) { result, error in
            if let error = error {
```

```
                completion(nil, ApiError(error.localizedDescription))
                return
            }

            if let result = result {
                let apiError = (result.errors?.isEmpty ?? true) ? nil :
ApiError(result.errors!)
                if let snapshot = result.data?.createOrderByCart?.snapshot {
                    let order = Order(snapshot: snapshot)
                    completion(order, apiError)
                } else {
                    completion(nil, apiError)
                }
            } else {
                completion(nil, nil)
            }
        }
    }

    public func getOrders(cachePolicy: CachePolicy = .returnCacheDataAndFetch,
completion: @escaping (Connection<Order>?, ApiError?) -> Void) {
        getUserAttributes { [self] (userAttributes, error) in
            if let error = error {
                completion(nil, error)
                return
            }

            guard let customerId = userAttributes?.first(where: { $0.name ==
"sub" })?.value else {
                completion(nil, nil)
                return
            }

            let query = ListOrdersByCustomerQuery(customerId: customerId)
            appSync.fetch(query: query, cachePolicy: cachePolicy, queue:
.global()) { result, error in
                if let error = error {
                    completion(nil, ApiError(error.localizedDescription))
                    return
                }

                if let result = result {
                    let apiError = (result.errors?.isEmpty ?? true) ? nil :
ApiError(result.errors!)
                    if let snapshot =
result.data?.listOrdersByCustomer?.snapshot {
                        let orders = Connection<Order>(snapshot: snapshot)
                        completion(orders, apiError)
                    } else {
                        completion(nil, apiError)
                    }
                } else {
                    completion(nil, nil)
                }
            }
        }
    }

    public func getRecipients(cachePolicy: CachePolicy =
.returnCacheDataAndFetch, completion: @escaping (Connection<Recipient>?,
ApiError?) -> Void) {
        getUserAttributes { [self] (userAttributes, error) in
```

```
            if let error = error {
                completion(nil, error)
                return
            }

            guard let customerId = userAttributes?.first(where: { $0.name ==
"sub" })?.value else {
                completion(nil, nil)
                return
            }

            let filter = ListRecipientFilterInput(isDeleted:
TableBooleanFilterInput(eq: false))
            let query = ListRecipientsForCustomerQuery(customerId: customerId,
filter: filter)
            appSync.fetch(query: query, cachePolicy: cachePolicy, queue:
.global()) { result, error in
                if let error = error {
                    completion(nil, ApiError(error.localizedDescription))
                    return
                }

                if let result = result {
                    let apiError = (result.errors?.isEmpty ?? true) ? nil :
ApiError(result.errors!)
                    if let snapshot =
result.data?.listRecipientsForCustomer?.snapshot {
                        let recipients = Connection<Recipient>(snapshot:
snapshot)
                        completion(recipients, apiError)
                    } else {
                        completion(nil, apiError)
                    }
                } else {
                    completion(nil, nil)
                }
            }
        }
    }

    public func getRecipient(recipientId: String, cachePolicy: CachePolicy =
.returnCacheDataAndFetch, completion: @escaping (Recipient?, ApiError?) -> Void)
{
        let query = GetRecipientQuery(id: recipientId)
        appSync.fetch(query: query, cachePolicy: cachePolicy, queue: .global())
{ result, error in
            if let error = error {
                completion(nil, ApiError(error.localizedDescription))
                return
            }

            if let result = result {
                let apiError = (result.errors?.isEmpty ?? true) ? nil :
ApiError(result.errors!)
                if let snapshot = result.data?.getRecipient?.snapshot {
                    let recipient = Recipient(snapshot: snapshot)
                    completion(recipient, apiError)
                } else {
                    completion(nil, apiError)
                }
            } else {
                completion(nil, nil)
```

```
            }
        }
    }

    public func createRecipient(recipient: Recipient, completion: @escaping
(Recipient?, ApiError?) -> Void) {
        getUserAttributes { [self] (userAttributes, error) in
            if let error = error {
                completion(nil, error)
                return
            }

            guard let customerId = userAttributes?.first(where: { $0.name ==
"sub" })?.value else {
                completion(nil, nil)
                return
            }

            let input = CreateRecipientInput(customerId: customerId, firstName:
recipient.firstName, lastName: recipient.lastName, email: recipient.email,
phone: recipient.phone)
            let mutation = CreateRecipientMutation(input: input)
            appSync.perform(mutation: mutation, queue: .global(),
optimisticUpdate: nil, conflictResolutionBlock: nil) { result, error in
                if let error = error {
                    completion(nil, ApiError(error.localizedDescription))
                    return
                }

                if let result = result {
                    let apiError = (result.errors?.isEmpty ?? true) ? nil :
ApiError(result.errors!)
                    if let snapshot = result.data?.createRecipient?.snapshot {
                        let recipient = Recipient(snapshot: snapshot)
                        completion(recipient, apiError)
                    } else {
                        completion(nil, apiError)
                    }
                } else {
                    completion(nil, nil)
                }
            }
        }
    }

    public func updateRecipient(recipient: Recipient, completion: @escaping
(Recipient?, ApiError?) -> Void) {
        let input = UpdateRecipientInput(id: recipient.id, firstName:
recipient.firstName, lastName: recipient.lastName, email: recipient.email,
phone: recipient.phone)
        let mutation = UpdateRecipientMutation(input: input)
        appSync.perform(mutation: mutation, queue: .global(), optimisticUpdate:
nil, conflictResolutionBlock: nil) { result, error in
            if let error = error {
                completion(nil, ApiError(error.localizedDescription))
                return
            }

            if let result = result {
                let apiError = (result.errors?.isEmpty ?? true) ? nil :
ApiError(result.errors!)
                if let snapshot = result.data?.updateRecipient?.snapshot {
```

```
                    let recipient = Recipient(snapshot: snapshot)
                    completion(recipient, apiError)
                } else {
                    completion(nil, apiError)
                }
            } else {
                completion(nil, nil)
            }
        }
    }

    public func deleteRecipient(recipientId: String, completion: @escaping
(Recipient?, ApiError?) -> Void) {
        let input = DeleteRecipientInput(id: recipientId)
        let mutation = DeleteRecipientMutation(input: input)
        appSync.perform(mutation: mutation, queue: .global(), optimisticUpdate:
nil, conflictResolutionBlock: nil) { result, error in
            if let error = error {
                completion(nil, ApiError(error.localizedDescription))
                return
            }

            if let result = result {
                let apiError = (result.errors?.isEmpty ?? true) ? nil :
ApiError(result.errors!)
                if let snapshot = result.data?.deleteRecipient?.snapshot {
                    let recipient = Recipient(snapshot: snapshot)
                    completion(recipient, apiError)
                } else {
                    completion(nil, apiError)
                }
            } else {
                completion(nil, nil)
            }
        }
    }

    public func getAddresses(cachePolicy: CachePolicy =
.returnCacheDataAndFetch, completion: @escaping (Connection<Address>?,
ApiError?) -> Void) {
        getUserAttributes { [self] (userAttributes, error) in
            if let error = error {
                completion(nil, error)
                return
            }

            guard let customerId = userAttributes?.first(where: { $0.name ==
"sub" })?.value else {
                completion(nil, nil)
                return
            }

            let filter = ListAddressFilterInput(isDeleted:
TableBooleanFilterInput(eq: false))
            let query = ListAddressesForCustomerQuery(customerId: customerId,
filter: filter)
            appSync.fetch(query: query, cachePolicy: cachePolicy, queue:
.global()) { result, error in
                if let error = error {
                    completion(nil, ApiError(error.localizedDescription))
                    return
                }
```

```
                if let result = result {
                    let apiError = (result.errors?.isEmpty ?? true) ? nil :
ApiError(result.errors!)
                    if let snapshot =
result.data?.listAddressesForCustomer?.snapshot {
                        let addresses = Connection<Address>(snapshot: snapshot)
                        completion(addresses, apiError)
                    } else {
                        completion(nil, apiError)
                    }
                } else {
                    completion(nil, nil)
                }
            }
        }
    }

    public func getAddress(addressId: String, cachePolicy: CachePolicy =
.returnCacheDataAndFetch, completion: @escaping (Address?, ApiError?) -> Void) {
        let query = GetAddressQuery(id: addressId)
        appSync.fetch(query: query, cachePolicy: cachePolicy, queue: .global())
{ result, error in
            if let error = error {
                completion(nil, ApiError(error.localizedDescription))
                return
            }

            if let result = result {
                let apiError = (result.errors?.isEmpty ?? true) ? nil :
ApiError(result.errors!)
                if let snapshot = result.data?.getAddress?.snapshot {
                    let address = Address(snapshot: snapshot)
                    completion(address, apiError)
                } else {
                    completion(nil, apiError)
                }
            } else {
                completion(nil, nil)
            }
        }
    }

    public func createAddress(address: Address, completion: @escaping (Address?,
ApiError?) -> Void) {
        getUserAttributes { [self] (userAttributes, error) in
            if let error = error {
                completion(nil, error)
                return
            }

            guard let customerId = userAttributes?.first(where: { $0.name ==
"sub" })?.value else {
                completion(nil, nil)
                return
            }

            let input = CreateAddressInput(customerId: customerId, country:
address.country, state: address.state, city: address.city, addressLine1:
address.addressLine1, addressLine2: address.addressLine2, postalCode:
address.postalCode)
            let mutation = CreateAddressMutation(input: input)
```

```
            appSync.perform(mutation: mutation, queue: .global(),
optimisticUpdate: nil, conflictResolutionBlock: nil) { result, error in
                if let error = error {
                    completion(nil, ApiError(error.localizedDescription))
                    return
                }

                if let result = result {
                    let apiError = (result.errors?.isEmpty ?? true) ? nil :
ApiError(result.errors!)
                    if let snapshot = result.data?.createAddress?.snapshot {
                        let address = Address(snapshot: snapshot)
                        completion(address, apiError)
                    } else {
                        completion(nil, apiError)
                    }
                } else {
                    completion(nil, nil)
                }
            }
        }
    }

    public func updateAddress(address: Address, completion: @escaping (Address?,
ApiError?) -> Void) {
        let input = UpdateAddressInput(id: address.id, country: address.country,
state: address.state, city: address.city, addressLine1: address.addressLine1,
addressLine2: address.addressLine2, postalCode: address.postalCode)
        let mutation = UpdateAddressMutation(input: input)
        appSync.perform(mutation: mutation, queue: .global(), optimisticUpdate:
nil, conflictResolutionBlock: nil) { result, error in
            if let error = error {
                completion(nil, ApiError(error.localizedDescription))
                return
            }

            if let result = result {
                let apiError = (result.errors?.isEmpty ?? true) ? nil :
ApiError(result.errors!)
                if let snapshot = result.data?.updateAddress?.snapshot {
                    let address = Address(snapshot: snapshot)
                    completion(address, apiError)
                } else {
                    completion(nil, apiError)
                }
            } else {
                completion(nil, nil)
            }
        }
    }

    public func deleteAddress(addressId: String, completion: @escaping
(Address?, ApiError?) -> Void) {
        let input = DeleteAddressInput(id: addressId)
        let mutation = DeleteAddressMutation(input: input)
        appSync.perform(mutation: mutation, queue: .global(), optimisticUpdate:
nil, conflictResolutionBlock: nil) { result, error in
            if let error = error {
                completion(nil, ApiError(error.localizedDescription))
                return
            }
```

```
            if let result = result {
                let apiError = (result.errors?.isEmpty ?? true) ? nil :
ApiError(result.errors!)
                if let snapshot = result.data?.deleteAddress?.snapshot {
                    let address = Address(snapshot: snapshot)
                    completion(address, apiError)
                } else {
                    completion(nil, apiError)
                }
            } else {
                completion(nil, nil)
            }
        }
    }

    public func createReview(review: Review, completion: @escaping (Review?,
ApiError?) -> Void) {
        getUserAttributes { [self] (userAttributes, error) in
            if let error = error {
                completion(nil, error)
                return
            }

            guard let customerId = userAttributes?.first(where: { $0.name ==
"sub" })?.value else {
                completion(nil, nil)
                return
            }

            let input = CreateReviewInput(customerId: customerId, productItemId:
review.productItemId, rating: review.rating, comment: review.comment)
            let mutation = CreateReviewMutation(input: input)
            appSync.perform(mutation: mutation, queue: .global(),
optimisticUpdate: nil, conflictResolutionBlock: nil) { result, error in
                if let error = error {
                    completion(nil, ApiError(error.localizedDescription))
                    return
                }

                if let result = result {
                    let apiError = (result.errors?.isEmpty ?? true) ? nil :
ApiError(result.errors!)
                    if let snapshot = result.data?.createReview?.snapshot {
                        let review = Review(snapshot: snapshot)
                        completion(review, apiError)
                    } else {
                        completion(nil, apiError)
                    }
                } else {
                    completion(nil, nil)
                }
            }
        }
    }

    public func updateReview(review: Review, completion: @escaping (Review?,
ApiError?) -> Void) {
        let input = UpdateReviewInput(id: review.id, rating: review.rating,
comment: review.comment)
        let mutation = UpdateReviewMutation(input: input)
        appSync.perform(mutation: mutation, queue: .global(), optimisticUpdate:
nil, conflictResolutionBlock: nil) { result, error in
```

```
            if let error = error {
                completion(nil, ApiError(error.localizedDescription))
                return
            }

            if let result = result {
                let apiError = (result.errors?.isEmpty ?? true) ? nil :
ApiError(result.errors!)
                if let snapshot = result.data?.updateReview?.snapshot {
                    let review = Review(snapshot: snapshot)
                    completion(review, apiError)
                } else {
                    completion(nil, apiError)
                }
            } else {
                completion(nil, nil)
            }
        }
    }

    public func deleteReview(reviewId: String, completion: @escaping (Review?,
ApiError?) -> Void) {
        let input = DeleteReviewInput(id: reviewId)
        let mutation = DeleteReviewMutation(input: input)
        appSync.perform(mutation: mutation, queue: .global(), optimisticUpdate:
nil, conflictResolutionBlock: nil) { result, error in
            if let error = error {
                completion(nil, ApiError(error.localizedDescription))
                return
            }

            if let result = result {
                let apiError = (result.errors?.isEmpty ?? true) ? nil :
ApiError(result.errors!)
                if let snapshot = result.data?.deleteReview?.snapshot {
                    let review = Review(snapshot: snapshot)
                    completion(review, apiError)
                } else {
                    completion(nil, apiError)
                }
            } else {
                completion(nil, nil)
            }
        }
    }

    public func getReview(productItemId: String, cachePolicy: CachePolicy =
.returnCacheDataAndFetch, completion: @escaping (Review?, ApiError?) -> Void) {
        getUserAttributes { [self] (userAttributes, error) in
            if let error = error {
                completion(nil, error)
                return
            }

            guard let customerId = userAttributes?.first(where: { $0.name ==
"sub" })?.value else {
                completion(nil, nil)
                return
            }

            let reviewId = "\(customerId).\(productItemId)"
            let query = GetReviewQuery(id: reviewId)
```

```
            appSync.fetch(query: query, cachePolicy: cachePolicy, queue:
.global()) { result, error in
                if let error = error {
                    completion(nil, ApiError(error.localizedDescription))
                    return
                }

                if let result = result {
                    let apiError = (result.errors?.isEmpty ?? true) ? nil :
ApiError(result.errors!)
                    if let snapshot = result.data?.getReview?.snapshot {
                        let review = Review(snapshot: snapshot)
                        completion(review, apiError)
                    } else {
                        completion(nil, apiError)
                    }
                } else {
                    completion(nil, nil)
                }
            }
        }
    }

    public func getReviews(productItemId: String, cachePolicy: CachePolicy =
.returnCacheDataAndFetch, completion: @escaping (Connection<Review>?, ApiError?)
-> Void) {
        let filter = ListReviewFilterInput(comment:
TableStringFilterInput(attributeExists: true))
        let query = ListReviewsForProductItemQuery(productItemId: productItemId,
filter: filter)
        appSync.fetch(query: query, cachePolicy: cachePolicy, queue: .global())
{ result, error in
            if let error = error {
                completion(nil, ApiError(error.localizedDescription))
                return
            }

            if let result = result {
                let apiError = (result.errors?.isEmpty ?? true) ? nil :
ApiError(result.errors!)
                if let snapshot =
result.data?.listReviewsForProductItem?.snapshot {
                    let reviews = Connection<Review>(snapshot: snapshot)
                    completion(reviews, apiError)
                } else {
                    completion(nil, apiError)
                }
            } else {
                completion(nil, nil)
            }
        }
    }

    public func getReviews(cachePolicy: CachePolicy = .returnCacheDataAndFetch,
completion: @escaping (Connection<Review>?, ApiError?) -> Void) {
        getUserAttributes { [self] (userAttributes, error) in
            if let error = error {
                completion(nil, error)
                return
            }
```

```
            guard let customerId = userAttributes?.first(where: { $0.name ==
"sub" })?.value else {
                completion(nil, nil)
                return
            }

            let filter = ListReviewFilterInput(comment:
TableStringFilterInput(attributeExists: true))
            let query = ListReviewsForCustomerQuery(customerId: customerId,
filter: filter)
            appSync.fetch(query: query, cachePolicy: cachePolicy, queue:
.global()) { result, error in
                if let error = error {
                    completion(nil, ApiError(error.localizedDescription))
                    return
                }

                if let result = result {
                    let apiError = (result.errors?.isEmpty ?? true) ? nil :
ApiError(result.errors!)
                    if let snapshot =
result.data?.listReviewsForCustomer?.snapshot {
                        let reviews = Connection<Review>(snapshot: snapshot)
                        completion(reviews, apiError)
                    } else {
                        completion(nil, apiError)
                    }
                } else {
                    completion(nil, nil)
                }
            }
        }
    }

    public func getProductItems(cachePolicy: CachePolicy =
.returnCacheDataAndFetch, completion: @escaping (Connection<ProductItem>?,
ApiError?) -> Void) {
        let query = ListProductItemsQuery()
        appSync.fetch(query: query, cachePolicy: cachePolicy, queue: .global())
{ result, error in
            if let error = error {
                completion(nil, ApiError(error.localizedDescription))
                return
            }

            if let result = result {
                let apiError = (result.errors?.isEmpty ?? true) ? nil :
ApiError(result.errors!)
                if let snapshot = result.data?.listProductItems?.snapshot {
                    let productItems = Connection<ProductItem>(snapshot:
snapshot)
                    completion(productItems, apiError)
                } else {
                    completion(nil, apiError)
                }
            } else {
                completion(nil, nil)
            }
        }
    }
```

```
    public func getProductItemsWithDiscount(cachePolicy: CachePolicy =
.returnCacheDataAndFetch, completion: @escaping (Connection<ProductItem>?,
ApiError?) -> Void) {
        let query = ListProductItemsWithDiscountQuery()
        appSync.fetch(query: query, cachePolicy: cachePolicy, queue: .global())
{ result, error in
            if let error = error {
                completion(nil, ApiError(error.localizedDescription))
                return
            }

            if let result = result {
                let apiError = (result.errors?.isEmpty ?? true) ? nil :
ApiError(result.errors!)
                if let snapshot =
result.data?.listProductItemsWithDiscount?.snapshot {
                    let productItems = Connection<ProductItem>(snapshot:
snapshot)
                    completion(productItems, apiError)
                } else {
                    completion(nil, apiError)
                }
            } else {
                completion(nil, nil)
            }
        }
    }

    public func getBrands(cachePolicy: CachePolicy = .returnCacheDataAndFetch,
completion: @escaping (Connection<Brand>?, ApiError?) -> Void) {
        let query = ListBrandsQuery(filter: nil, limit: 100)
        appSync.fetch(query: query, cachePolicy: cachePolicy, queue: .global())
{ result, error in
            if let error = error {
                completion(nil, ApiError(error.localizedDescription))
                return
            }

            if let result = result {
                let apiError = (result.errors?.isEmpty ?? true) ? nil :
ApiError(result.errors!)
                if let snapshot = result.data?.listBrands?.snapshot {
                    let brands = Connection<Brand>(snapshot: snapshot)
                    completion(brands, apiError)
                } else {
                    completion(nil, apiError)
                }
            } else {
                completion(nil, nil)
            }
        }
    }

    public func getProductByBrand(brandId: String, cachePolicy: CachePolicy =
.returnCacheDataAndFetch, completion: @escaping (Connection<Product>?,
ApiError?) -> Void) {
        let query = ListProductsByBrandQuery(brandId: brandId)
        appSync.fetch(query: query, cachePolicy: cachePolicy, queue: .global())
{ result, error in
            if let error = error {
                completion(nil, ApiError(error.localizedDescription))
                return
```

```
            }

            if let result = result {
                let apiError = (result.errors?.isEmpty ?? true) ? nil :
ApiError(result.errors!)
                if let snapshot = result.data?.listProductsByBrand?.snapshot {
                    let products = Connection<Product>(snapshot: snapshot)
                    completion(products, apiError)
                } else {
                    completion(nil, apiError)
                }
            } else {
                completion(nil, nil)
            }
        }
    }

    public func getCategories(cachePolicy: CachePolicy =
.returnCacheDataAndFetch, completion: @escaping (Connection<Category>?,
ApiError?) -> Void) {
        let query = ListCategoriesQuery()
        appSync.fetch(query: query, cachePolicy: cachePolicy, queue: .global())
{ result, error in
            if let error = error {
                completion(nil, ApiError(error.localizedDescription))
                return
            }

            if let result = result {
                let apiError = (result.errors?.isEmpty ?? true) ? nil :
ApiError(result.errors!)
                if let snapshot = result.data?.listCategories?.snapshot {
                    let categories = Connection<Category>(snapshot: snapshot)
                    completion(categories, apiError)
                } else {
                    completion(nil, apiError)
                }
            } else {
                completion(nil, nil)
            }
        }
    }

    public func searchProductItems(filter: SearchFilter, from: Int = 0, size:
Int = 20, cachePolicy: CachePolicy = .returnCacheDataAndFetch, completion:
@escaping ([ProductItem]?, ApiError?) -> Void) {
        let attributes = filter.attributes?.compactMap({
SearchAttributeInput(name: $0.name, type: $0.type,
                values: $0.values?.compactMap({ DataValueInput(ival: $0.ival,
sval: $0.sval, fval: $0.fval, bval: $0.bval) }),
                ranges: $0.ranges?.compactMap({
                    guard let lower = $0.lower, let upper = $0.upper else {
return nil }
                    return DataRangeInput(lower: lower, upper: upper)
                }))
            })

        var input = SearchProductsInput(brands: filter.brands, attributes:
attributes, status: filter.status, name: filter.name)
        if let priceLower = filter.price?.lower, let priceUpper =
filter.price?.upper {
            input.price = DataRangeInput(lower: priceLower, upper: priceUpper)
```

```
        }

        var categories: [String] = []
        if let subcategories = filter.subcategories, !subcategories.isEmpty {
            categories.append(contentsOf: subcategories)
        }

        if let category = filter.category {
            categories.append(category)
        }

        if !categories.isEmpty {
            input.categories = categories
        }

        let query = SearchProductsQuery(filter: input, from: from, size: size)
        appSync.fetch(query: query, cachePolicy: cachePolicy, queue: .global())
{ result, error in
            if let error = error {
                completion(nil, ApiError(error.localizedDescription))
                return
            }

            if let result = result {
                let apiError = (result.errors?.isEmpty ?? true) ? nil :
ApiError(result.errors!)
                if let snapshot = result.data?.searchProducts {
                    let productItems = snapshot.compactMap({ $0?.snapshot
}).map({ ProductItem(snapshot: $0)})
                    completion(productItems, apiError)
                } else {
                    completion(nil, apiError)
                }
            } else {
                completion(nil, nil)
            }
        }
    }
}
```

ДОДАТОК Д
(обов'язковий)

ПРЕЗЕНТАЦІЙНІ МАТЕРІАЛИ

Хмельницький національний університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

«Мобільний застосунок для Інтернет -магазину з продажу побутових товарів»

Виконав: студент гр. ІПЗ-20-1
Крижановський Д. С.

Керівник: канд. техн. наук., доцент
Радельчук Г. І.

Мета та завдання кваліфікаційної роботи

Мета роботи – розроблення мобільного застосунку, який надаватиме користувачам зручний інструмент для покупок побутових товарів у мережі Інтернет.

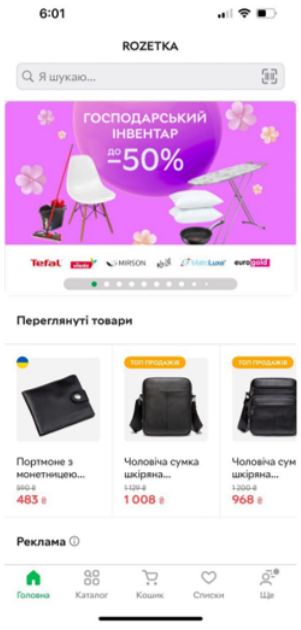
Завдання роботи

- вивчити процеси організації Інтернет-торгівлі;
- проаналізувати сучасні мобільні застосунки-аналоги та провести їх порівняльну характеристику ;
- визначити функціональні та нефункціональні вимоги до розроблюваного застосунку;
- обрати архітектуру мобільного застосунку;
- визначити структуру даних та спроєктувати базу даних;
- спроєктувати структуру мобільного застосунку;
- обрати та налаштувати засоби та інструменти реалізації застосунку;
- реалізувати застосунок та розробити керівництво користувача;
- виконати тестування мобільного застосунку та провести аналіз отриманих результатів.

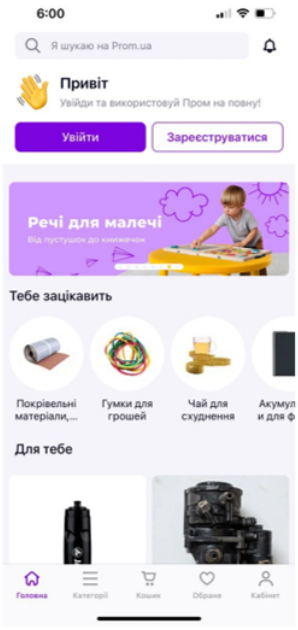
Актуальність

- Мобільні пристрої, такі як смартфони та планшети, стають основними засобами доступу до Інтернет-магазинів, тому мобільні застосунки відіграють важливу роль у забезпеченні зручності та доступності для клієнтів.
- Мобільні застосунки забезпечують зручний та швидкий доступ до товарів у будь-який час та з будь-якого місця (користувачам не потрібно включати комп'ютер або ноутбук, вони можуть здійснити покупку зі свого смартфона).
- Мобільні застосунки дозволяють створювати персоналізовані пропозиції та рекомендації для користувачів на основі їхньої попередньої поведінки та відповідно до їхніх інтересів.

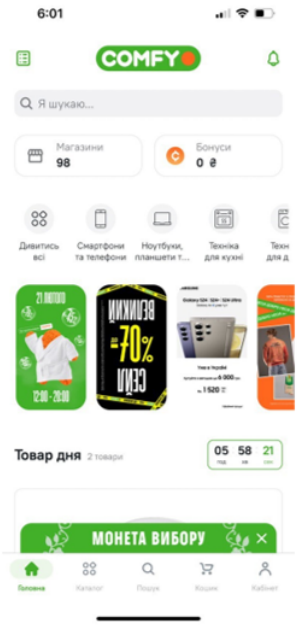
Аналоги



ROZETKA



Prom.ua



Comfy

Порівняльна таблиця

Критерій для порівняння	ROZETKA	Prom.ua	Comfy
Реєстрація користувача	+	+	+
Профіль користувача	+	+	+
Головне меню	+	+	+
Категорії товарів	+	+	+
Детальний опис товару	+	+	+
Фільтр для пошуку	+	+	+
Список бажаних товарів	+	+	+
Історія замовлень	+	+	+
Чат з продавцем	+	+	-
Відгуки користувачів	+	+	+
Порівняння товарів	+	-	+
Акційні пропозиції	+	+	+

Функціональні вимоги

- ❑ реєстрація нового та авторизація існуючого користувача;
- ❑ відновлення паролю та підтвердження електронної адреси;
- ❑ каталог товарів та пошук по фільтру за встановленими критеріями;
- ❑ детальний опис товару та ознайомлення з додатковими матеріалами;
- ❑ формування кошику замовлень;
- ❑ здійснення замовлення обраних товарів;
- ❑ перегляд історії замовлень;
- ❑ оцінка товарів та відгуки користувачів.

Нефункціональні вимоги

- ❑ користувач повинен автентифікуватися перед доступом до основного функціоналу застосунку за допомогою електронної пошти та пароля;
- ❑ сторінки каталогу товарів повинні завантажуватися швидко, ~2 секунд;
- ❑ застосунок повинен бути сумісний з операційними системами iOS попередніх версій;
- ❑ застосунок повинен бути сумісний з мобільними пристроями iPhone попередніх версій;
- ❑ розміри та інтерфейс елементів застосунку повинні відповідати стандартам дизайну для забезпечення комфортного використання на різних пристроях.

Архітектура

Безсерверна архітектура – це підхід до розроблення програмного забезпечення, де розробникам не потрібно вести управління фізичними серверами або віртуальними машинами. У цій архітектурі код розглядається як набір незалежних функцій, які викликаються лише відповідно до запитів або подій.

Amazon Web Services є одним з провідних постачальників безсерверних технологій на ринку хмарних послуг. Компанія відома своїм широким набором сервісів, які дозволяють розробникам будувати та впроваджувати безсерверні застосунки ефективно та надійно. Технологія AWS базується на серверних кластерах (фермах), розташованих по всьому світу.

Amazon Web Services

Cognito – сервіс для аутентифікації та авторизації користувачів

Simple Email Service – поштовий сервіс

S3 Storage – сервіс для зберігання різноманітних типів інформації

Lambda – сервіс обчислень, який автоматично запускає код у відповідь на різноманітні події

DynamoDB – сервіс баз даних NoSQL

API Gateway – сервіс для забезпечення централізованої обробки API-зв'язку

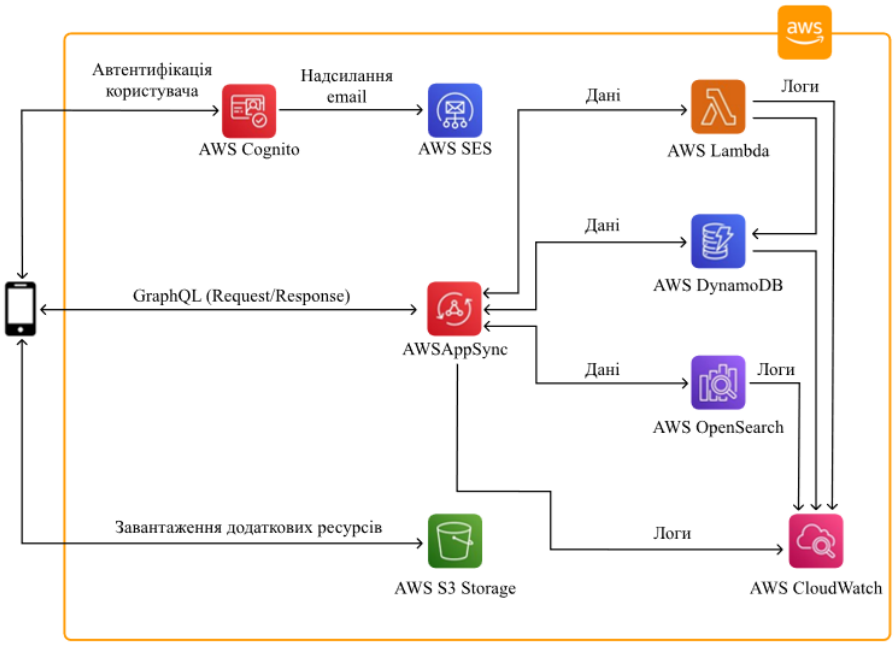
OpenSearch – сервіс пошуку та аналізу даних

AppSync – сервіс який дозволяє, створювати універсальні API

CloudWatch – сервіс моніторингу та управління

IAM – служба управління доступом користувачів та ресурсів

Сценарії



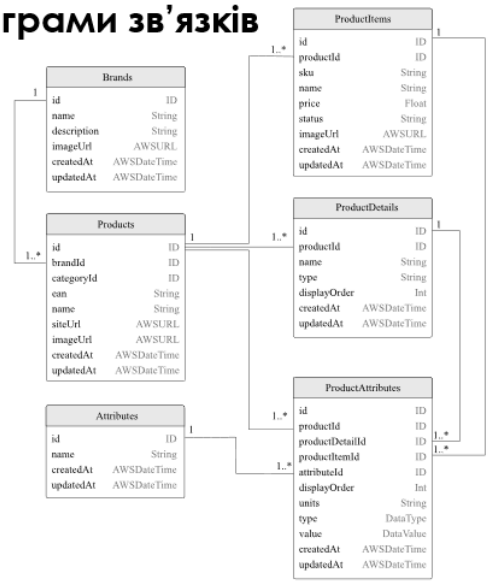
Сценарій використання сервісів AWS на основі AppSync

Проєктування бази даних

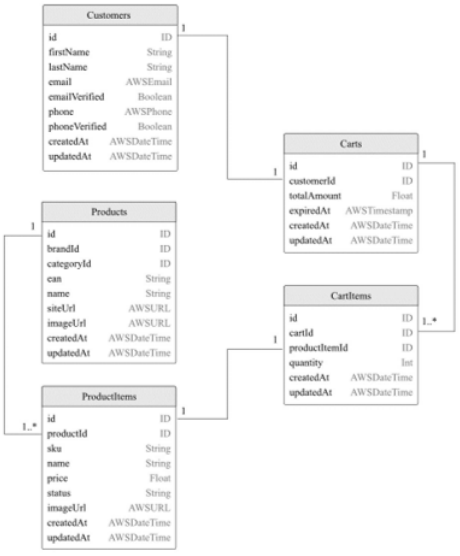
Основні сутності бази даних:

- ❑ **Customer** – користувачі застосунку
- ❑ **Product** – товар
- ❑ **Product Item** – варіант товару
- ❑ **Product Detail** – опис товару
- ❑ **Product Attribute** – атрибути товару
- ❑ **Brand** – виробник товару
- ❑ **Category** – категорія товару
- ❑ **Category Filter Field** – пошуковий фільтр
- ❑ **Category Field Option** – варіант вибору для фільтра
- ❑ **Inventory** – товарні запаси
- ❑ **Discount** – знижка на товар
- ❑ **Discount Item** – величина знижки
- ❑ **Cart** – кошик користувача
- ❑ **Cart Item** – елемент кошику
- ❑ **Wishlist** – список бажаного
- ❑ **Review** – відгук користувача
- ❑ **Rating** – рейтинг товару
- ❑ **Order** – замовлення
- ❑ **Order Item** – елемент замовлення
- ❑ **Address** – адреса доставки
- ❑ **Recipient** – отримувач замовлення
- ❑ **Attribute** - атрибут

Діаграми зв'язків

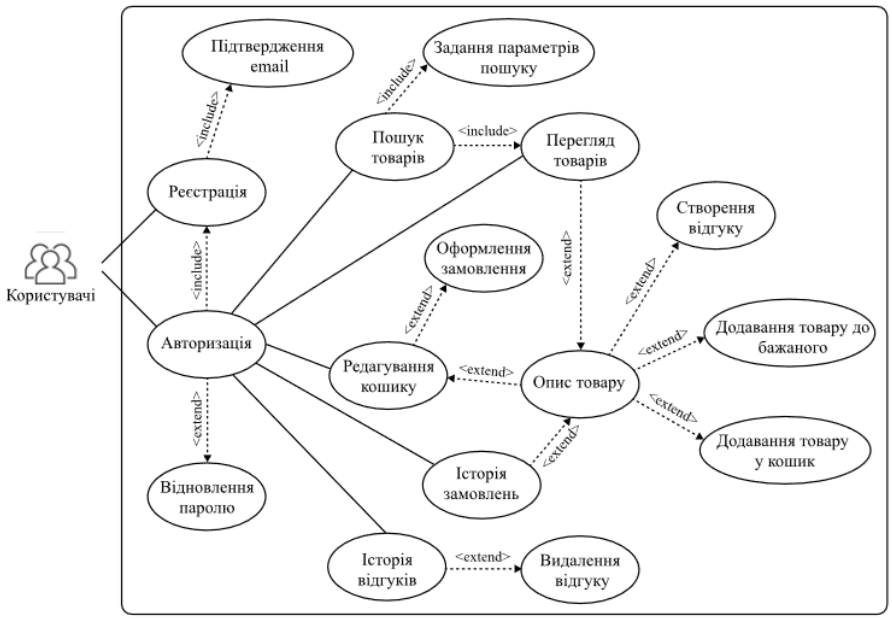


Товар і його атрибути



Користувач і його кошик

Діаграма варіантів використання

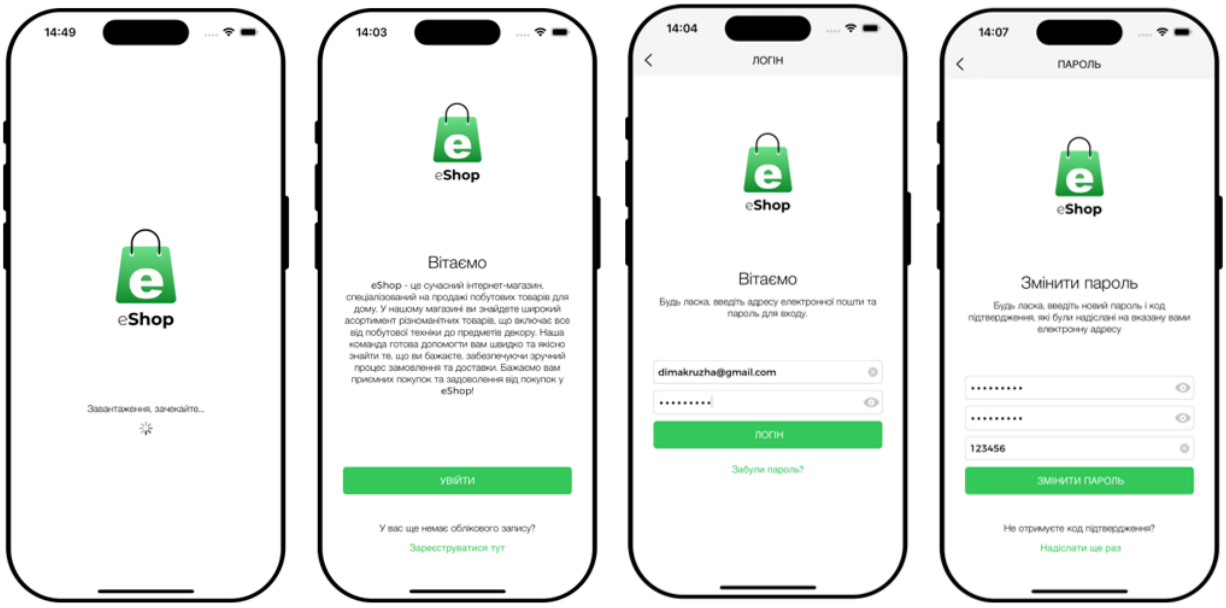


Реалізація

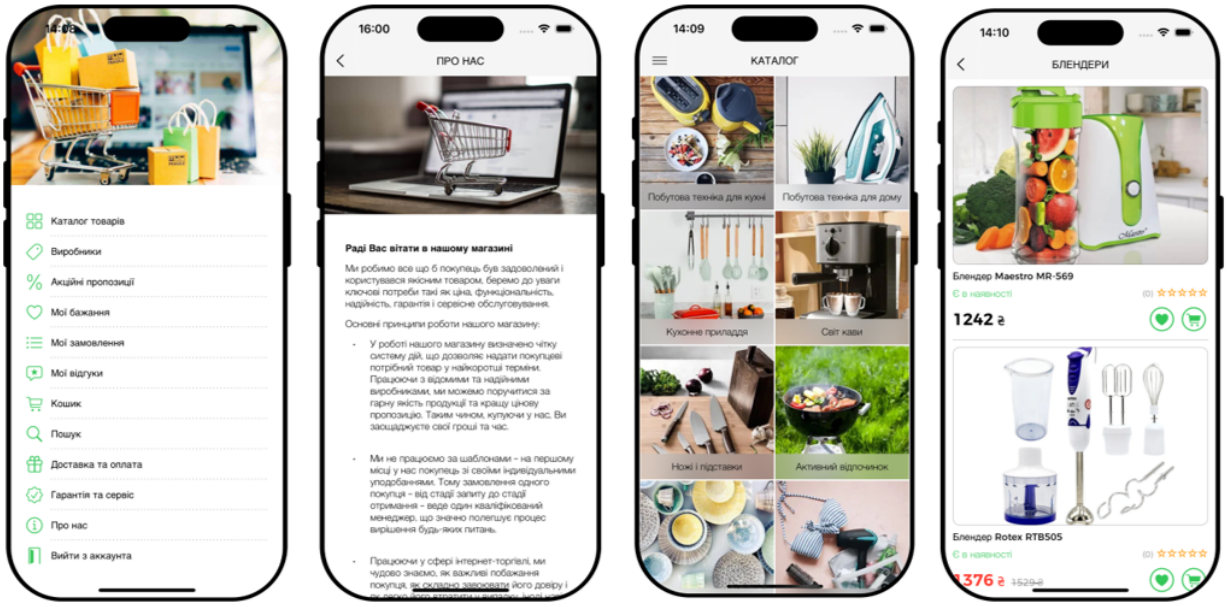
- Програмна реалізація усіх Lambda-функцій виконана окремим рішенням eShop.Lambda у середовищі Microsoft Visual Studio мовою C# з використанням набору інструментів AWS SDK Toolkit, що надає розробникам доступ до AWS API та сервісів, дозволяє легко і швидко інтегрувати AWS функціональність у свої застосунки та сервіси;

- Програмна реалізація проєкту мобільного застосунку Інтернет-магазину з продажу побутових товарів виконана для платформи iOS, у середовищі розробки Xcode, з використання фреймворку UIKit мовою програмування Swift. Для розроблення екранів мобільного застосунку використано шаблон Model-View-Controller (MVC).

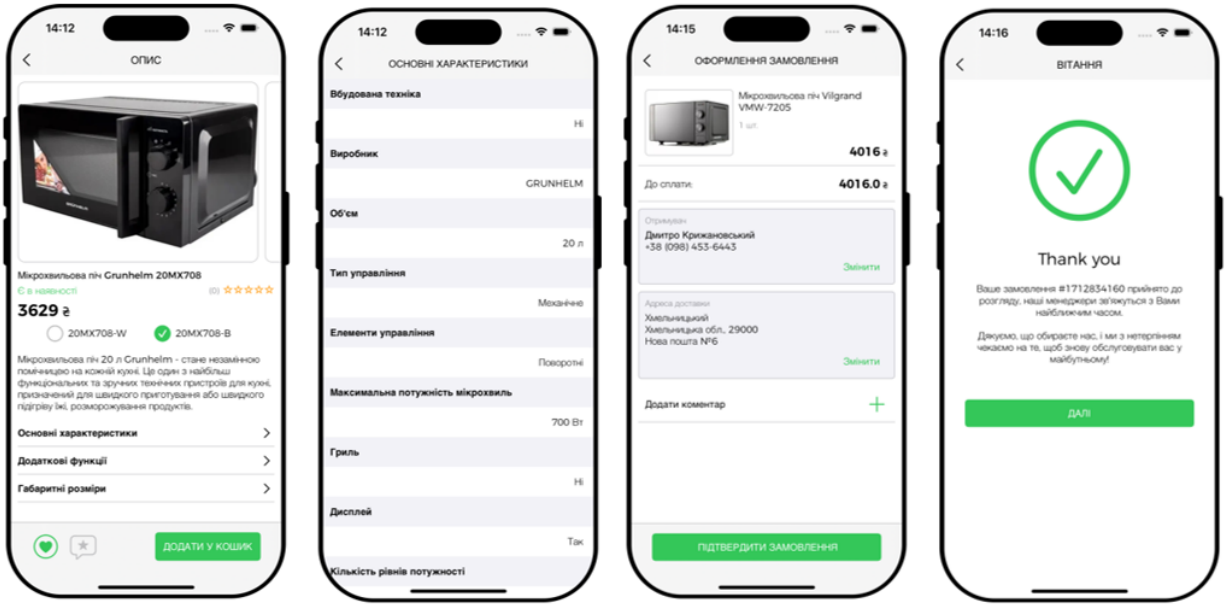
Екрани мобільного застосунку



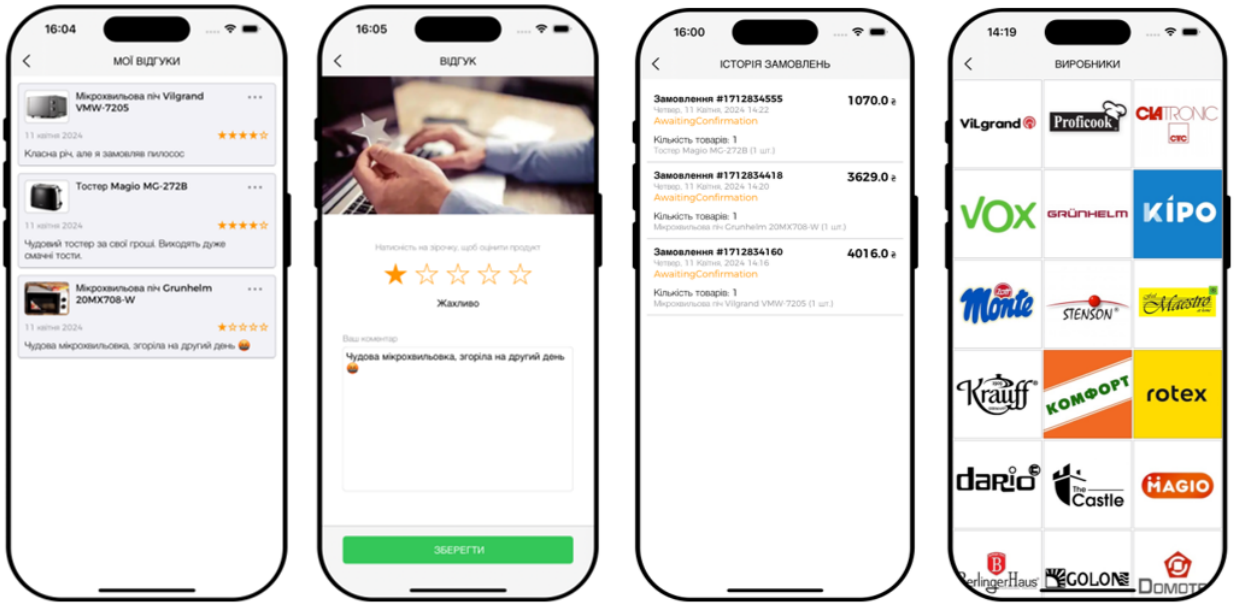
Екрани мобільного застосунку



Екрани мобільного застосунку



Екрани мобільного застосунку



Технічні характеристики

Назва характеристики	Значення
Операційна система	iOS 15.0 або вище
Модель пристрою	iPhone XS або вище
Обсяг оперативної пам'яті	Наявність вільної пам'яті у розмірі 4 ГБ на смартфоні для коректного встановлення та роботи застосунка
Інтернет-з'єднання	Підтримка Wi-Fi та мобільного Інтернету (3G/4G/5G), залежно від можливостей пристрою та мережі провайдера
Мережеві вимоги	Підтримка мережевих протоколів, таких як HTTP, HTTPS, WebSocket тощо для забезпечення зв'язку з сервером та іншими сервісами
Додаткові вимоги	На телефоні встановлено App Store

Тестування

- Для ручного тестування хмарних сервісів AWS використано сервіс CloudWatch, який дозволяє моніторити різні метрики та логи, допомагає виявляти та аналізувати проблеми у реальному часі.

- Тестування Lambda-функцій було здійснено з використанням автоматизованих методів тестування на основі Unit-тестів, які дозволяють перевірити правильність роботи конкретних функцій, методівабо класів.

- Тестування функціоналу мобільного застосунку було здійснено за допомогою автоматизованих методів тестування та симуляторів. Для автоматизованого тестування на платформі iOS використовувались Unit-тести, які дозволяють перевірити окремі частини коду на коректність та відповідність очікуванням.

Висновки

- ❑ Обґрунтована актуальність розроблення мобільного застосунку, сформульовано основні функціональні та нефункціональні вимоги.

- ❑ Обґрунтовано вибір безсерверної архітектури для розроблення мобільного застосунку. Розглянуто сценарії використання сервісів AWS та обрано найбільш відповідний до потреб застосунку варіант.

- ❑ Визначено та спроектовано низку сутностей, які відображають основні об'єкти та концепції системи. Розроблено та проаналізовано діаграму варіантів використання. Побудовано діаграми послідовності у контексті використання сервісів AWS.

Висновки

- ❑ Визначено оптимальний набір технологій, інструментів, фреймворків, мов програмування та інших компонентів, які найкраще підходять для розроблення мобільного застосунку.

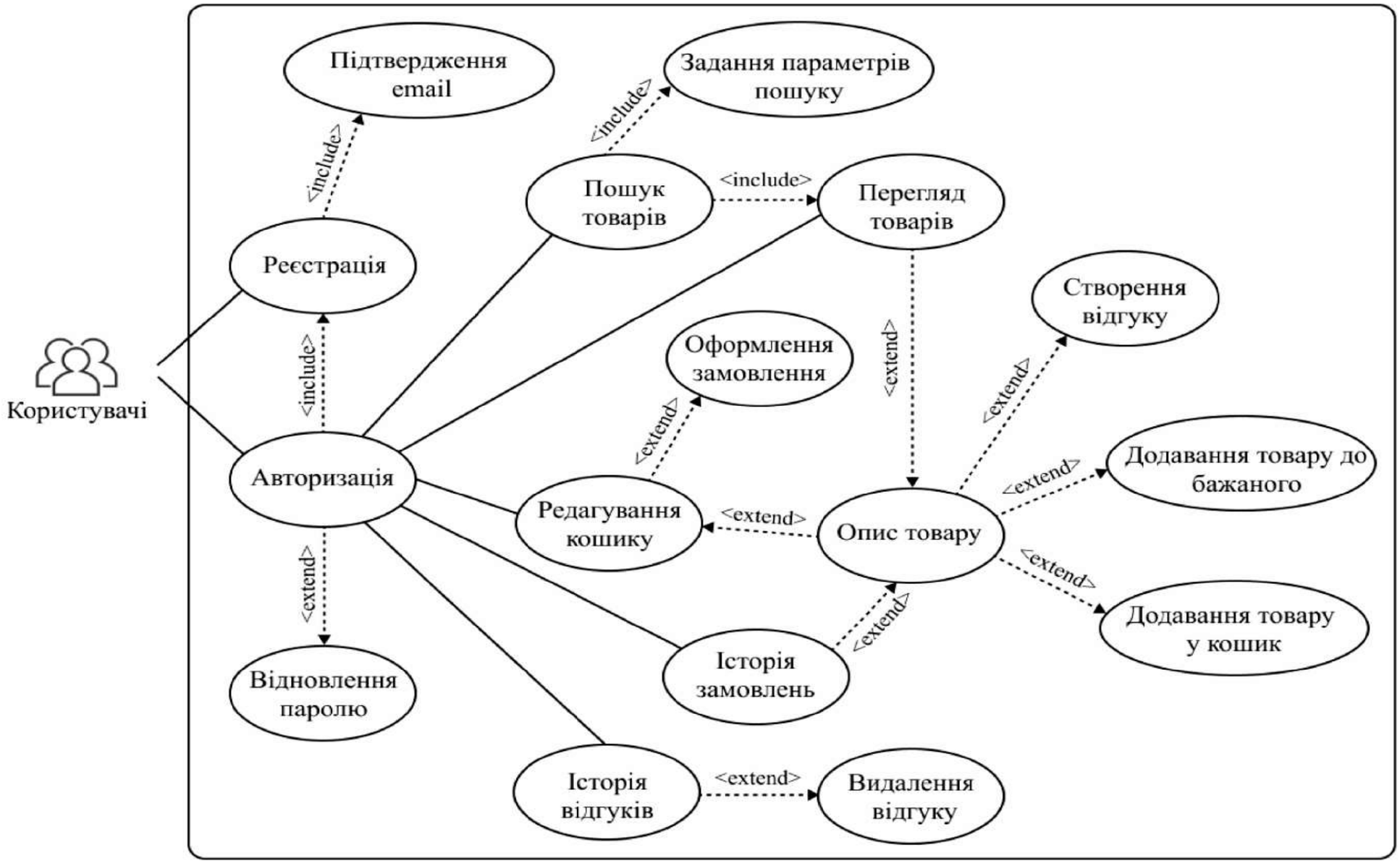
- ❑ Налаштовано сервіс авторизації та аутентифікації користувачів, розроблено базу даних мобільного застосунку, налаштовано сховище медіа-даних, налаштовано домен для реалізації наскрізного пошуку, реалізовано низку Lambda-функцій для бізнес-логіки, реалізовано API на основі GraphQL.

Висновки

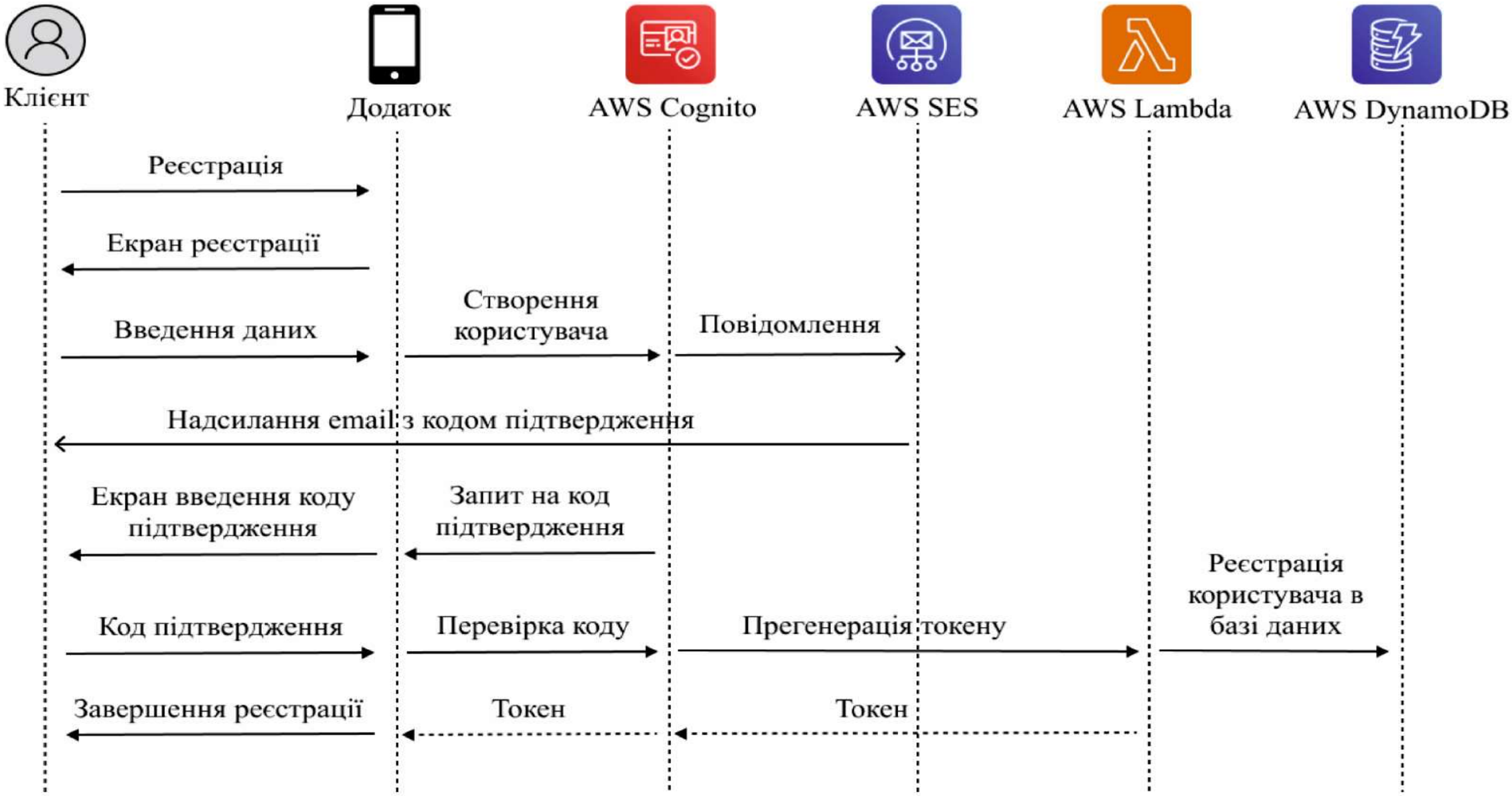
- ❑ Виконано інтеграцію налаштованих сервісів у мобільний застосунок, розроблено API клієнт для роботи з GraphQL-запитами та екрани мобільного застосунку.
- ❑ Розроблено керівництво користувача та описано технічні характеристики мобільного застосунку.
- ❑ Визначено оптимальні методи для тестування окремихчастин проєкту. Виконано тестування сервісів та функціоналу мобільного застосунку. Проведено аналіз результатів тестування, виявлено недоліки та сформовано перелік для подальшого удосконалення застосунку.

Дякую за увагу!

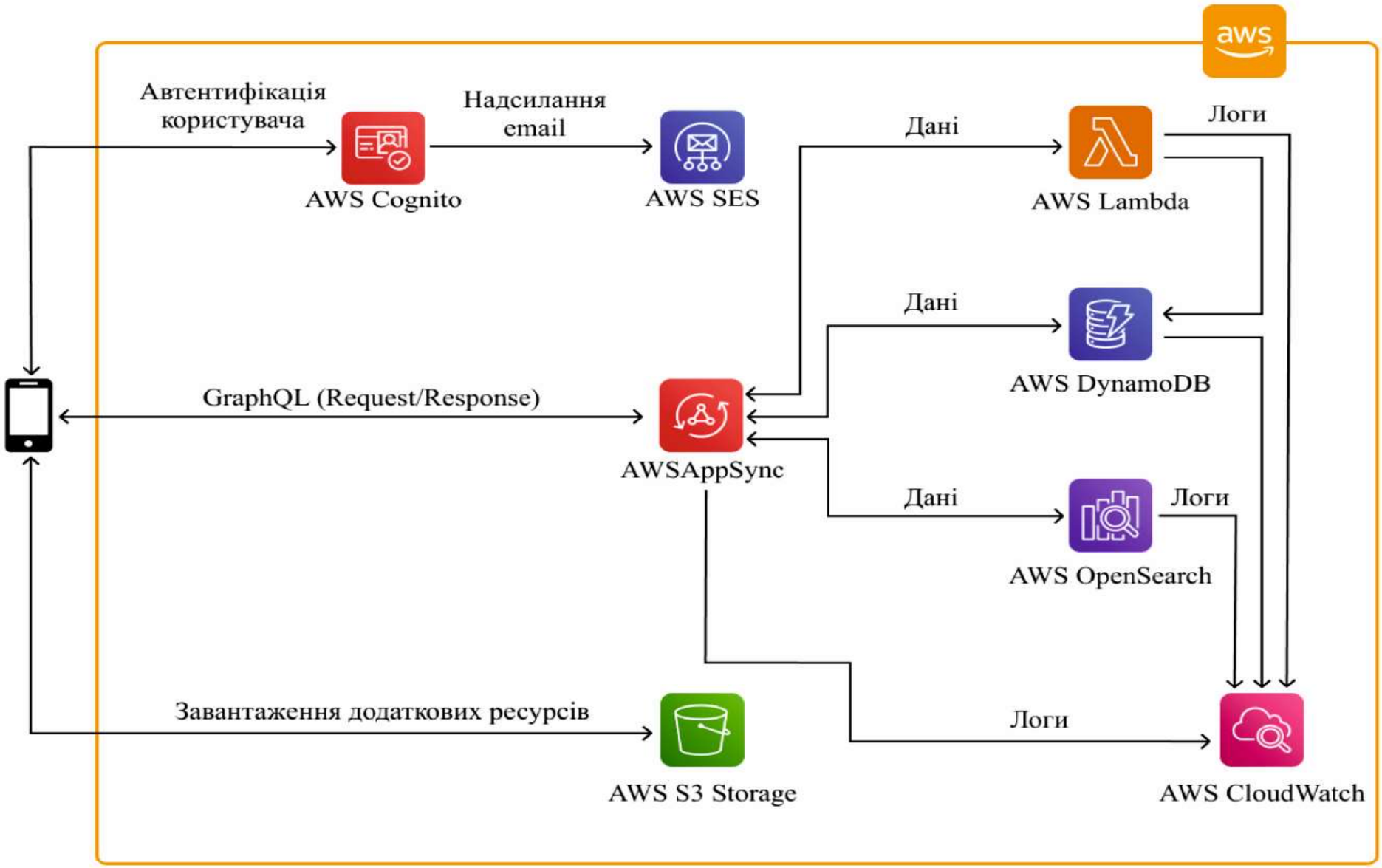
ГРАФІЧНА ЧАСТИНА



					КвРІПЗ.200249.01.11.Е8		
Зм.	Арк.	№ докум.	Підпис	Дата	Мобільний застосунок для Інтернет-магазину з продажу побутових товарів	Літера	Маса / Масштаб
Розробив		Крижановський Д.С.					
Керівник		Радельчук Г.І.				Аркуш 1	Аркушів 3
Консульт.							
Н. Контр.		Радельчук Г.І.			Діаграма варіантів використання	ХНУ.ІПЗ-20-1	
Зав. каф.		Бедратюк Л.П.					



					КвРІПЗ.200249.01.11.Е8	
Зм.	Арк.	№ докум.	Підпис	Дата	Мобільний застосунок для Інтернет-магазину з продажу побутових товарів	Літера / Маса / Масштаб
Розробив		Крижановський Д.С.				
Керівник		Радельчук Г.І.				Аркуш 2 / Аркушів 3
Консульт.						
Н. Контр.		Радельчук Г.І.			Діаграма послідовності реєстрації	ХНУ.ІПЗ-20-1
Зав. каф.		Бедратюк Л.П.				



					КвРІПЗ.200249.01.11.Е8			
Зм.	Арк.	№ докум.	Підпис	Дата	Мобільний застосунок для Інтернет-магазину з продажу побутових товарів	Літера	Маса	Масштаб
Розробив		Крижановський Д.С.						
Керівник		Радельчук Г.І.				Аркуш 3	Аркушів 3	
Консульт.								
Н. Контр.		Радельчук Г.І.			Діаграма сценарію використання сервісів AWS на основі AppSync	ХНУ.ІПЗ-20-1		
Зав. каф.		Бедратюк Л.П.						

СУПРОВІДНІ ДОКУМЕНТИ

Завідувачу кафедри інженерії програмного забезпечення проф. Бедратюку Л. П.

здобувача вищої освіти
Крижановського Д. С.
Прізвище, ініціали

факультет ІТ, 4 курс, група ІПЗ-20-1

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності в Хмельницькому національному університеті», згідно з яким виявлення академічного плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів дисциплінарної та академічної відповідальності, ознайомлений. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на наявність академічного плагіату оповіщений та надаю свою згоду на обробку й збереження університетом моєї роботи в інституційному репозитарії Хмельницького національного університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та/або Anti-Plagiarism) і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення текстових збігів у роботах.

Робота надається для перевірки в електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

03.06.2024 р.
дата

підпис

Wed Jun 05 10:55:20 EEST 2024, Форкун Юрій Вікторович, Хмельницький національний університет, ХНУ

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 1.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 14%**

	Документ		Сумарний збіг по Базі Даних	
ID: 128467 Назва: БКР Мобільний застосунок для Інтернет-магазину з продажу побутових товарів Додано в БД: 2024-06-05 Автора: КРИЖАНОВСЬКИЙ Дмитро Керівники: Радельчук Г.І,, канд. техн. наук, доцент Консультанти: Опоненти:	Символи	Лексеми	Символи	Лексеми
	79959	1199	2830 (4%)	37 (3%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

UNICHECK
by Turnitin

Ім'я користувача:
ІПЗ

Дата перевірки:
05.06.2024 15:09:05 EEST

Дата звіту:
05.06.2024 19:26:09 EEST

ID перевірки:
1016322652

Тип перевірки:
Doc vs Internet + Library

ID користувача:
100012953

Назва документа: БКР_Мобільний застосунок для Інтернет-магазину з продажу побутових товарів_Крижанов...

Кількість сторінок: 76 Кількість слів: 13026 Кількість символів: 108106 Розмір файлу: 2.01 MB ID файлу: 1016121077

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

4.76%
Схожість

Найбільша схожість: 1.27% з джерелом з Бібліотеки (ID файлу: 1016111291)

3.59% Джерела з Інтернету | 395 ..... Сторінка 78

2.59% Джерела з Бібліотеки | 242 ..... Сторінка 81

0% Цитат

Не знайдено жодних цитат

Не знайдено жодних посилань

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи | 7

Підозріле форматування | 12 сторінок

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ
освітнього ступеня «Бакалавр»

Здобувач Крижановський Дмитро Сергійович

Тема Мобільний застосунок для Інтернет-магазину з продажу побутових товарів

Спеціальність 121 – Інженерія програмного забезпечення

Обсяг кваліфікаційної роботи:

Кількість листів креслень три; кількість сторінок записки 76

1.Короткий зміст пояснювальної записки та прийнятих рішень У кваліфікаційній роботі проведено та описано дослідження предметної області щодо використання мобільних застосунків для Інтернет-магазинів з продажу побутових товарів. Дослідження відбувалося шляхом огляду застосунків-аналогів. Згідно з отриманими даними було розроблено функціональні і нефункціональні вимоги до застосунку та доведено актуальність його розроблення. Було спроєктовано базу даних, обрано та налаштовано сервіси AWS, на основі чого реалізовано мобільний застосунок Інтернет-магазину. Також було проведено тестування сервісів та функціоналу мобільного застосунку. Проведено аналіз результатів тестування, виявлено недоліки та сформовано перелік задач для подальшого удосконалення застосунку.

2. Висновок про відповідність роботи поставленому завданню Кваліфікаційна робота виконана відповідно до поставленого завдання та з дотриманням усіх вимог.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки та передових методів роботи У вступі до кваліфікаційної роботи описано тематику, виділено мету та актуальність роботи, а також сформульовано завдання на кваліфікаційну роботу. У першому розділі проведено аналіз предметної області, розглянуто існуючі застосунки-аналоги та визначені функціональні і нефункціональні вимоги до мобільного застосунку. У другому розділі обрано архітектуру мобільного застосунку, спроєктовано базу даних, розглянуто сценарії використання сервісів AWS та обрано найбільш відповідний до потреб застосунку варіант. У третьому розділі описано налаштування сервісів та їх підключення, реалізацію мобільного застосунку, проведено тестування як сервісів, так і застосунку, проаналізовано результати тестування та сформульовано список можливостей для покращення програмного продукту.

4. Позитивні сторони роботи Тематика кваліфікаційної роботи є актуальною, оскільки з кожним роком зростає кількість користувачів, які вибирають онлайн-покупки як зручний та швидкий спосіб придбання товарів. Також, так як було реалізовано API мобільного застосунку на основі GraphQL, то є можливість розроблення застосунку не тільки на платформі iOS, а й на Android.

5. Негативні сторони роботи У мобільному застосунку адресу доставки потрібно вводити вручну, що може не сподобатись користувачам, які звикли до API Нової пошти для пошуку адрес відділень за назвою міста.

6. Оцінка графічного оформлення та пояснювальної записки Графічне оформлення подано у вигляді ілюстративних рисунків, зокрема діаграм та зображень описуваних елементів. Оформлення пояснювальної записки відповідає вимогам чинних стандартів.

7. Відгук про кваліфікаційну роботу в цілому Кваліфікаційна робота заслуговує позитивної оцінки. Матеріал пояснювальної записки подано доступно та зрозуміло, що дозволяє комплексно зрозуміти викладений матеріал та виділити головні досягнення, описані у роботі. Також додатки і графічний матеріал дають можливість візуалізувати деталі та принципи функціонування системи.

8. Інші зауваження

9. Оцінка кваліфікаційної роботи Кваліфікаційна робота виконана у повному обсязі, відповідає поставленій задачі та заслуговує на оцінку "добре".

РЕЦЕНЗЕНТ (прізвище, ім'я, по-батькові, посада, місце роботи) Мартинюк Валерій Володимирович, доктор технічних наук, професор, зав. кафедри автоматизації, комп'ютерно-інтегрованих технологій та робототехніки ХНУ

"7" 06 2024 р. _____
(підпис)

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ
КАФЕДРИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатами звіту/звітів перевірки роботи, продукованими програмно-технічним засобом (ами), на наявність текстових збігів:

Назва кваліфікаційної роботи: «Мобільний застосунок для Інтернет-магазину з продажу побутових товарів»
Автор: Крижановський Дмитро Сергійович
Освітня програма: Освітньо-професійна програма «Інженерія програмного забезпечення»
Спеціальність: 121 – Інженерія програмного забезпечення
Науковий керівник: Радельчук Галина Іванівна, кандидат технічних наук, доцент
Після аналізу звіту/звітів зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	**відповідає**
2	Виявлені запозичення не є академічним плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої й електронної версії роботи	
3	Виявлені запозичення не є академічним плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнуті. Робота може бути допущена до захисту після того, як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття текстових запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
5	Інше:	

Підтвердження:

Запозичення, виявлені у роботі, є законними і не є плагіатом, оскільки:

1) у тексті кваліфікаційної роботи системою перевірки на плагіат Unicheck виявлено схожість з деякими документами у частині загальновживаних обов'язкових словосполучень у стандартних бланках (титулка, відомість документів), у структурі змісту, назвах розділів/підрозділів, у назвах публікацій переліку джерел посилання;

2) в якості запозичень системою Unicheck було зафіксовано деякі послідовності вихідного коду і посилання на бібліотеки, які є стандартними мовними конструкціями програмування та не можуть розглядатися як об'єкт авторських прав і, відповідно, їх порушення;

3) запозичення, виявлені в тексті роботи, є фрагментарними.

Максимальний обсяг запозичень, визначений системою Anti-Plagiarism, складає 1.0%. Обсяг запозичень, визначений системою Unicheck виявлення збігів ідентичності/схожості, складає 4.76% і адресується до 395 джерел з Інтернету і 242 джерел з бібліотеки, що, з урахуванням наведених обґрунтувань, відповідає характеру теми і свідчить на користь кваліфікаційної роботи.

Дата 05.06.2024 р.

Завідувач кафедри — Леонід БЕДРАТЮК

Гарант освітньої програми — Леонід БЕДРАТЮК

Керівник кваліфікаційної роботи — Галина РАДЕЛЬЧУК