

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування інституту, факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавр

(ступінь вищої освіти)

на тему РОЗРОБКА КРОСПЛАТФОРМНОГО ЗАСТОСУНКУ ДЛЯ ПОКУПКИ
МЕБЛЕВИХ ГАРНІТУР ДЛЯ ДОМУ
DEVELOPMENT OF CROSS-PLATFORM APPLICATION FOR THE
PURCHASE OF HOME FURNITURE SETS

Виконав: студент(ка) 4 курсу, групи КНТ-119
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)
Інженерія програмного забезпечення

ІВАНОВ Д.О.

(ПРИЗВИЩЕ та ініціали)

Керівник КАПЛІЄНКО Т.І.

(ПРИЗВИЩЕ та ініціали)

Рецензент ШИТІКОВА О.В.

(ПРИЗВИЩЕ та ініціали)

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

Факультет ФКНТ
 Кафедра програмних засобів
 Ступінь вищої освіти бакалавр
 Спеціальність 121 Інженерія програмного забезпечення
 (код і найменування)
 Освітня програма (спеціалізація) Інженерія програмного забезпечення
 (назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
 “ ” 2022 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ІВАНОВА Данила Олександровича
 (ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка кроссплатформного застосунку для покупки меблевих гарнітур для дому Development of Cross-Platform Application for the Purchase of Home Furniture Sets

керівник проєкту (роботи) к.т.н., доцент, КАПЛІЄНКО Тетяна Ігорівна,
 (науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові, ,)

затверджені наказом закладу вищої освіти від “ 04 ” квітня 2023 року № 87

2. Строк подання студентом проєкту (роботи) 07 червня 2023 року

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблеми та постановка завдань дослідження. 2. Матеріали і методи. 3. Опис програми. 4. Експлуатація, тестування та експериментальне дослідження програми.

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

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	КАПЛІЄНКО Т.І., доцент		
Нормоконтроль	КАМІНСЬКА Ж.К., асистент		

7. Дата видачі завдання « 03 » лютого 2023 року.

КАЛЕНДАРНИЙ ПЛАН

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

Студент(ка)

_____ Данило ІВАНОВ
(підпис) (ім'я та ПРІЗВИЩЕ)

Керівник проєкту (роботи)

_____ Тетяна КАПЛІЄНКО
(підпис) (ім'я та ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 113 с., 42 рис., 2 табл., 2 дод., 34 джерел.

МОБІЛЬНИЙ ЗАСТОСУНОК, ЗАМОВЛЕННЯ МЕБЛІВ, КРОСПЛАТФОРМЕНІСТЬ, REACT NATIVE.

Об'єкт дослідження – кросплатформні програмні засоби для замовлення та покупки товарів.

Предмет дослідження – мобільні програми для замовлення та покупки меблів.

Мета роботи – створення програмної системи для зручного та швидкого замовлення меблів, також відпрацювання навичок роботи по створенню кросплатформних застосунків, використанню MVC архітектури та мови для запитів Graph QL.

Матеріали, методи та технічні засоби: структурне та об'єктно-орієнтоване програмування, технології React Native. NodeJS та Express, програми Android Studio та Xcode, мови програмування JavaScript, Graph QL, тип бази даних NoSQL(не реляційна) та представник – це Mongo DB, персональний комп'ютер з процесором Intel Core i-7 9750H під управліннями операційної системи Mac OS.

Результати. Створено програмний IOS та Android застосунок, який працює з сервісною частиною, використовуючи мову запитів Graph QL, та дозволяє користувачам замовляти меблі.

Висновки. Розроблено програмну систему для замовлення меблів, яка автоматизує, робить зручним та прискорює процес покупки меблів для користувачів, однією з переваг якої є кросплатформеність.

Галузь використання – продаж та замовлення на розробку продуктів через мобільний застосунок.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 113 pages, 42 figures, 2 tables, 2 appendixes, 34 sources.

PRODUCT STORE, MOBILE APP, FURNITURE ORDER, CROSS-PLATFORM, REACT NATIVE.

The object of the study is software for ordering and purchasing products.

The subject of the research is mobile applications for ordering and buying furniture.

The goal of the work is to create a software system for convenient and fast ordering of furniture, as well as practice the skills of working with cross-platform, microservice architecture, and the GraphQL request language.

Materials, methods and technical means: structural and object-oriented programming, React Native, NodeJS and Express technologies, Android Studio and Xcode programs, JavaScript programming languages, GraphQL, NoSQL database type (non-realistic) and the representative is Mongo DB, a personal computer with an Intel Core i-7 9750H processor running the Mac OS operating system.

The results: An IOS and Android software application has been created that works with the service part using the GraphQL request language and allows users to order furniture.

Conclusions: Developed a software system for ordering furniture that automates, makes it convenient and speeds up the process of buying furniture for users, learned how to work with cross-platform functionality and what problems may arise when developing for different platforms.

The field of use is sales and ordering of product development through a mobile application.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК	8
ВСТУП.....	9
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ.....	10
1.1 Актуальність проблеми.....	10
1.2 Кросплатформна розробка	10
1.3 Огляд існуючих програм та комерційних продуктів з використанням кросплатформної розробки.....	12
1.3.1 Мобільний застосунок «Puma» з використанням кросплатформної розробки.....	12
1.3.2 Мобільний застосунок «Bolt Food» з використанням кросплатформної розробки	14
1.3.3 Мобільний застосунок «IKEA».....	16
1.4 Висновки за розділом 1	17
2 ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ	19
2.1 Опис та види провідних технологій для розробки кросплатформних застосунків(фреймворків).....	19
2.1.1 Flutter	20
2.1.2 Xamarin.....	22
2.1.3 React Native	24
2.2 Порівняльна таблиця найвідоміших платформ для створення кросплатформних застосунків	28
2.2 Основні відомості про розробку мобільного застосунку з використанням React Native.....	29
2.3 Node JS	33
2.4 Express	34
2.5 Graph QL	36
2.6 MVC	39
2.7 Mongo DB.....	40
2.8 Вибір програмної мови для розробки комерційної системи	41

2.9 Висновки за розділом	44
3 ОСНОВНІ РІШЕННЯ З РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ	46
3.1 Основні рішення щодо розроблення серверної частини	46
3.1.1 Сутність User	48
3.1.2 Сутність PaymentMethod	53
3.1.3 Сутність ShippingAddress	55
3.1.4 Сутність Furniture	57
3.1.5 Сутність Basket	59
3.1.6 Сутність Order.....	61
3.2 Основні рішення щодо розробки інтерфейсу	62
3.2.1 Основні рішення щодо використання додаткових бібліотек	62
3.2.2 Основні рішення з впорядкування елементів інтерфейсу	64
3.2.3 Основні рішення щодо логіки у компонентах інтерфейсу	69
3.3 Основні рішення щодо розробки БД	70
4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ	73
4.1 Призначення та умови застосування програми	73
4.1.1 Умови застосування	73
4.2 Експлуатація програмного забезпечення	74
4.2.1 Розгортання проєкту.....	74
4.2.2 Звертання до програми	74
4.3.3 Вхідні та вихідні дані	76
4.3.4 Повідомлення.....	77
4.3 Тестування системи.....	78
4.4 Висновки з розділу	79
ВИСНОВКИ	80
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ	82
Додаток А Текст програми	86
Додаток В Слайди презентації	105

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API	– Application Programming Interface;
IDE	– Integrated Development Environment;
IOS	– iPhone Operation System;
MVC	– Model View Controller;
JS	– Java Script;
UI	– User Interface;
БД	– база даних;
ПЗ	– програмне забезпечення;
ПК	– персональний комп'ютер;
ООП	– Об'єктно-орієнтоване програмування;
ОС	– операційна система.

ВСТУП

Розробка мобільних застосунків – дуже швидко зростаюча область програмування, адже користування мобільними пристроями набагато легше ніж персональними комп'ютерами. Цей пристрій виконує багато корисних функцій: планування подій завдяки календарю, записування важливих даних у нотатки, швидка комунікація тощо. Розповсюдження застосунків з метою надання послуг, торгівлі та просто способів розважитись збільшується з кожним роком.

У нашому часі існують дві мобільні операційні системи (ОС) Android та iPhone Operation System(IOS). Мобільний розробник – людина, яка розробляє застосунки для однієї з мобільних операційних систем, тобто ділять на 2 види: Android Developer або IOS Developer, але є програмісти, які завдяки деяким технологіям, можуть розробляти для двох платформ одночасно. Їх називають – кросплатформні розробники. В даний час вони дуже вигідні для комерційних продуктів, тому що замість двох можна найняти одного для обох систем.

Головна особливість в розробці мобільних застосунків це розміри та форми самих пристроїв, бо смартфони, планшети, які мають або маленький екран, або невеликий, або нестандартний (квадратний, прямокутний) екран. Тому дуже важливим для команди розробників розробити такий дизайн, який буде задовольняти користувачів і слідувати нормам легкого користування. Питання про оптимізацію важливе для цієї розробки, бо в наш час мобільні пристрої удосконалюються швидко, але вони ще не спроможні змагатися з персональними комп'ютерами у потужності.

Отже, розробка мобільних застосунків сьогодні широко використовується як потужний спосіб надати застосунок, який може замінити використання інших засобів для торгівлі та надання послуг, користувачам мобільних пристроїв.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність проблеми

За останні 20 років поширення мобільних пристроїв привело до того, що майже у всієї молоді є смартфон. Кожен користувач знаходить в ньому що йому потрібно. Діти можуть грати в ігри, учні знаходити матеріал завдяки сервісам пошуку, читати книги, власники бізнесу керувати своє виробництво та всім іноді потрібні звичайні застосунки, де можна викликати таксі або кур'єра з їжею, замовити столик, щось купити тощо [1]. Для надання цих послуг потрібна велика команда розробників, менеджерів, дизайнерів, щоб зробити корисний та зручний продукт для користувачів смартфона.

Метою роботи є створення комерційного зручного застосунка з використанням кросплатформної розробки для продажі меблів, який буде інтуїтивно зрозумілим та корисним. Наявність програми для смартфонів – невід'ємна частина сучасного ведення бізнесу та управління процесами, бо можна захватити велику кількість потенціальних користувачів. Згідно зі статистикою опублікованою в Data portal [2], 67% дорослих людей в усьому світі використовують смартфони щодня.

Якщо порівнювати це рішення з вебсайтом, то мобільний застосунок має більше широкий і зручний функціонал, адаптивний дизайн та більше легкий для користувачів.

1.2 Кросплатформна розробка

Кросплатформна розробка програм – це розробка програмного забезпечення, яке може працювати в декількох операційних системах. Найбільш поширеними ОС є Android та пристрої Apple(iPhone, iPad тощо). Головна відмінність цієї розробки від рідної є те, що застосунки рідної розробки призначенні спеціально для однієї ОС. Використовують такі мови

для рідної розробки: для Android це Kotlin або Java, а для IOS Swift або Objective C.

Користь від кросплатформної розробки:

- швидке випускання програм;
- менший бюджет;
- використання однієї мови для двох ОС;
- швидкий пошук помилок у роботі програми.

Але є мінуси цього підходу:

- це більш складний код, бо розробник використовує елементи, з кожних операційних систем, та нерідні частини в цих же компонентах інтерфейсу;
- можливість некоректної роботи програм, бо важко услідкувати за всіма особливостями архітектури різних операційних систем, це може вплинути на оптимізацію застосунка;
- програма використовує лише основні функції пристрою.

1.2.1 Процедура розробки кросплатформних застосунків

Розробка мобільних застосунків для обох ОС починаються з стандартного кроку – це вибір стратегії, на ньому команда вирішує цілі програми для бізнесу, можливості та функції. Це допоможе розробникам зрозуміти, що потрібно використовувати саме кросплатформну розробку для цього проєкту, також на цьому кроці потрібно вирішувати вимоги програми, вони потрібні для створення архітектури та вибору потрібних технологій, пакетів для застосунку.

Наступний крок, це створення каркасу для перевірки дизайнерами, щоб візуалізувати програму у вигляді моделі.

Далі, розробники розробляють програму, після того як вони зрозуміють концепцію та елементи дизайну готові.

Також, як будь-який комерційний, цифровий продукт перед завантаженням до магазинів обох платформ, застосунок повинен пройти стандартний процес кодування і тестування.

1.3 Огляд існуючих програм та комерційних продуктів з використанням кросплатформної розробки.

Зараз на ринку є велика кількість програмних засобів, яка використовує кросплатформну розробку, на жаль, зрозуміти це рідний застосунок чи ні дуже важко. Тому для аналізу було використано застосунки, які указані в документації технології для цієї розробки [3]. Ціновий діапазон цих рішень дуже суттєво відрізняється – від безкоштовних до рішень з ліцензією.

1.3.1 Мобільний застосунок «Puma» з використанням кросплатформної розробки

Puma SE – це мобільний застосунок для покупок в Інтернет-магазині Puma [4]. Він надає можливість зручного та швидкого пошуку та замовлення товарів з магазину через мобільний телефон або планшет. За допомогою застосунку, користувачі можуть оглядати нові колекції з використанням зручних та легких з точки зору користувача фільтрів пошуку, дізнаватись про акції та знижки, розглядати детальні фото та описи товарів, а також використання кошика для замовлення з доставкою додому. Застосунок надає можливість здійснювати покупку якісних та стильного одягу і взуття для фанатів бренду Puma.

Переваги:

- мінімалізм в дизайні, немає нічого зайвого;
- система корзини;
- система оплати;

- цікава функція «Віртуальна примірювальна»

Недоліки:

- робота тільки з інтернетом.

Це приклад стандартного комерційного застосунку з наданням можливості покупки товарів. Система оплати дозволяє заплатити за товар миттєво, також магазини в яких публікується застосунки сильно слідкують за всіма транзакціями, якщо щось не так вони швидко допоможуть.

Робота програми продемонстрована на рисунку 1.1. та 1.2.

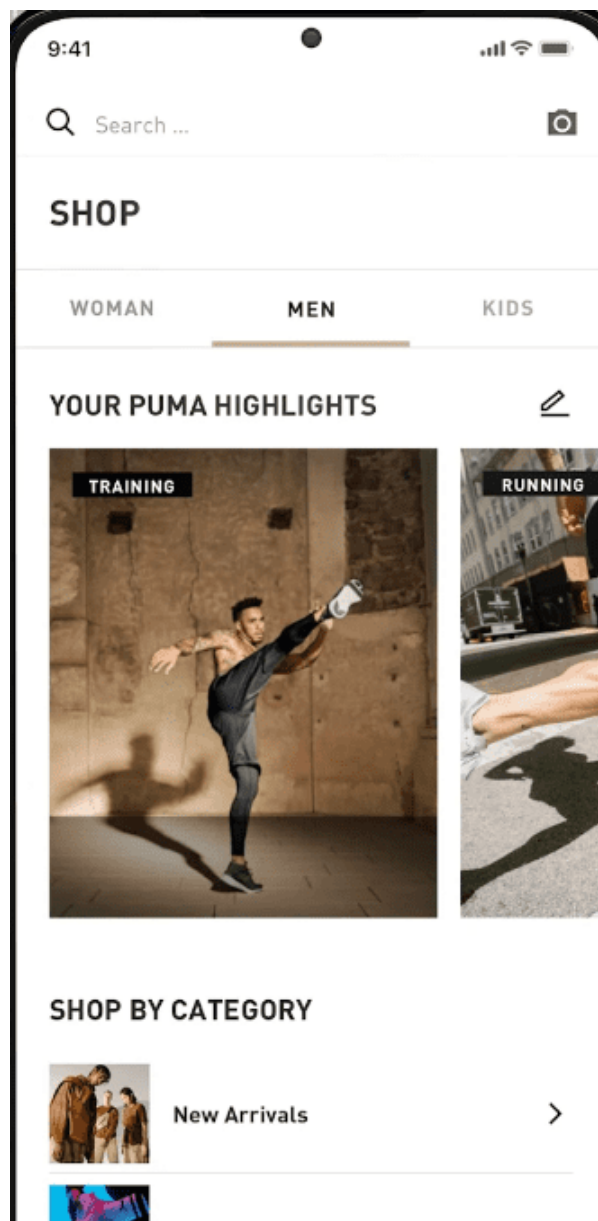


Рисунок 1.1 – Головний екран застосунок«Puma»

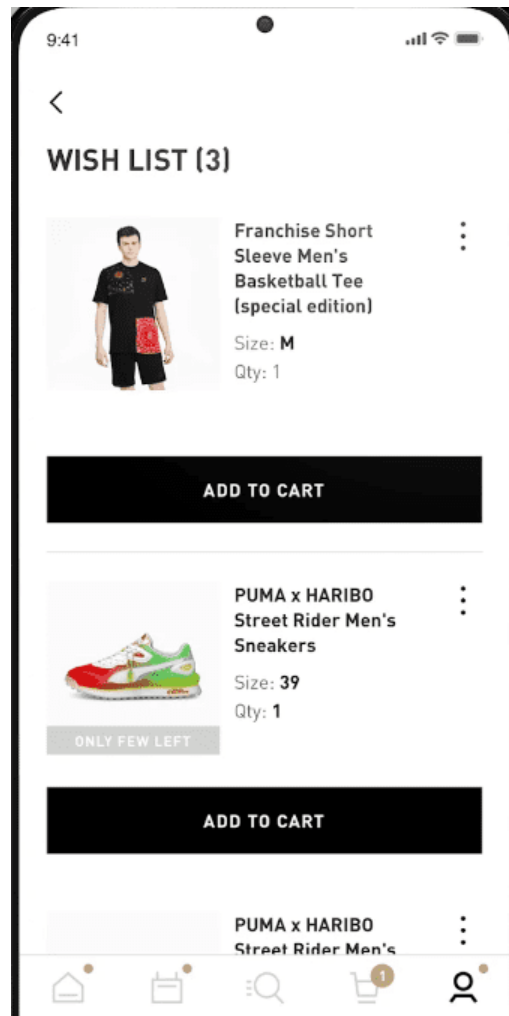


Рисунок 1.2 – Лист бажання у застосунку «Puma»

1.3.2 Мобільний застосунок «Bolt Food» з використанням кросплатформної розробки

Bolt Food – це мобільний застосунок, що дозволяє замовляти їжу з ресторанів та кафе, що працюють з доставкою [5]. За допомогою Bolt Food користувачі можуть знайти ресторани близько до себе, переглянути меню, вибрати страви та напої, та замовити доставку на вказану адресу. Застосунок пропонує зручний та швидкий спосіб оплати, можливість залишити спеціальні коментарі для поварів, та стежити за доставкою.

Переваги:

- зручний, зрозумілий інтерфейс для користувачів та кур'єрів;
- онлайн відстеження замовлення;

- без готівка оплата через застосунок;
- великий вибір товарів, бо компанія дозволяє стати партнером системи;

- можливість працювати як кур'єр.

Недоліки:

- робота тільки з інтернетом;
- немає можливості чату з кур'єром.

Це стандартний комерційний застосунок, який надає послуги користувачам для доставки їжі, надає можливість брати замовлення кур'єрам та користувачам замовляти їжу з магазинів, кафе, ресторанів, з якими працює Bolt. Завдяки комісії дозволяє заробляти кошти для бізнесу. Використання цього застосунку є не дуже важким для звичайних користувачів телефонів і є зручними для замовлення їжі додому або в офіс.

Робота програми продемонстрована на рисунку 1.3.

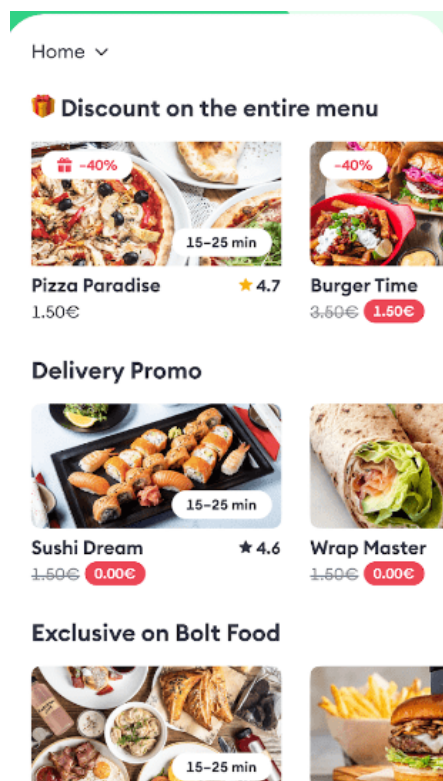


Рисунок 1.3 – Головне меню в «Bolt Food»

1.3.3 Мобільний застосунок «IKEA»

IKEA – це мобільний застосунок, який є офіційним програмним забезпеченням від компанії IKEA, що дозволяє покупати меблі чи різні товари для дому [6]. Користувач може швидко шукати, вибирати колір або стиль продукту, переглядати та покупати товари, для цього потрібно лише смартфон та підключення до інтернету.

Переваги:

- зрозумілий інтерфейсу пошуку необхідним товарів;
- віртуальний огляд товарів;
- багато інструментів для створення дизайну кімнати.

Недоліки:

- розмір застосунку;
- необхідне постійне підключення до інтернету;
- не кросплатформний;
- не з всіх країн можна користуватись.

Це стандартний комерційний застосунок, який має велику кількість можливостей. Головним недоліком є не кросплатформність, розробка потребує щонайменше двох фахівців, які використовують різні мови розробки, вони розробляють на різні платформи. Підтримка такого проєкту коштує велику кількість грошей та займає більше часу ніж кросплатформна розробка. Користь такої розробки – більша продуктивність та адаптація до кожної платформи окремо.

Робота програми продемонстрована на рисунку 1.4.

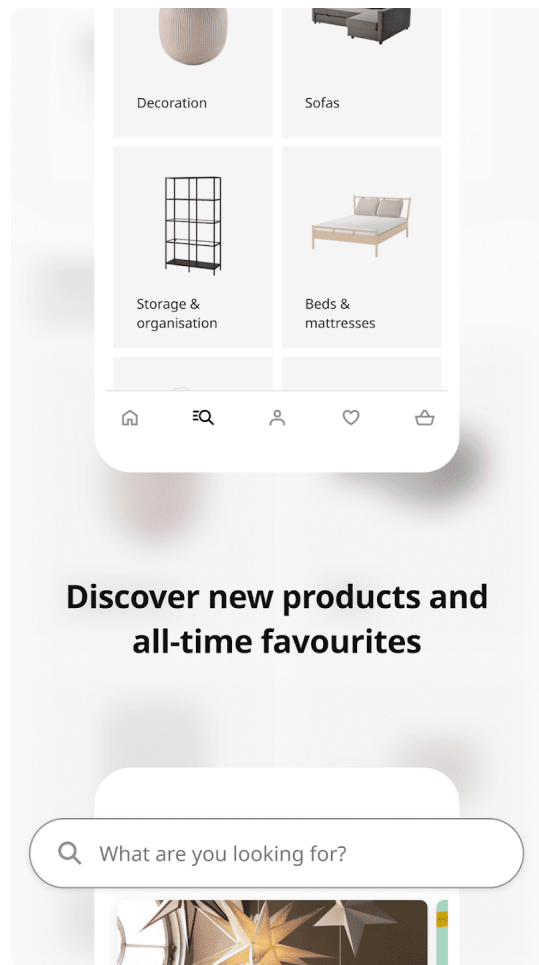


Рисунок 1.4 – Навігація в «ІКЕА»

1.4 Висновки за розділом 1

Мобільна розробка, а саме кросплатформна, має велике значення для успіху бізнесу. Вона допомагає знайти нових користувачів, завдяки наданням легкого користування продуктом і в значній мірі спрощує використання послуг бізнесу.

Виходячи з виконаного огляду існуючих систем і програмних засобів з використанням кросплатформної розробки, виявлені основні можливості, якими повинен володіти комерційна програма для обох мобільних

операційних систем, а саме Android та IOS з використанням кросплатформної розробки:

- система оплати;
- система знижок;
- зрозумілий дизайн для користувачів;
- система відгуків;
- можливість додавати у корзину, товари які сподобались користувачу.

2 ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ

2.1 Опис та види провідних технологій для розробки кросплатформних застосунків(фреймворків)

Фреймворк – це програмне середовище та набір інструментів, які спрощує та прискорює створення програмного забезпечення. Зазвичай фреймворки підтримується іншими розробниками у великих компаніях, де вони були створенні. Під використанням фреймворка мають на увазі: розробник пише код, який реалізує бізнес логіку застосунку. Середньостатистичний фреймворк надає розробнику такі можливості:

- використання компонентів, пакетів, класів тощо декілька разів, це позбавляє необхідність повторювати код щоразу;
- прискорює процес розробки [7];
- тримає увагу розробника на логіці застосунку, що є найголовнішим для бізнесу;
- відокремлює логіку від уявлення [7];
- багато функціоналу вже є у фреймворку, не потрібно щось створювати з нуля;
- велика допомога з сторони користувачів, багато розробників створюють свої бібліотеки на основі фреймворка, що спрощує використання цього інструмента.

Головна ціль фреймворку, дати розробнику комфортне середовище для проєкту з великим та масштабованим функціоналом [8]. Якщо кожен розробник робив свої застосунки з нуля без допомоги сучасних бібліотек та фреймворків, створення цифрових продуктів зайняло би набагато більше часу і зробити їх стало би набагато складніше. Фреймворк сильно допомагає розробникам, бо вони беруть на себе всі основні функції, а розробник може тільки цими функціями керувати. Також фреймворк зобов'язує використовувати спеціальну структуру коду та папок для розбиття коду по головним частинам. Це допомагає підтримувати код у одному стилі з будь-

якими розробниками. Фреймворк використовує шаблон проєктування MVC(Model-view-controller – «модель-подання-контролер»).

2.1.1 Flutter

Нещодавно випущений Google фреймворк, за який малий час завоював велику популярність, для створення кросплатформних застосунків для Android та IOS. На сьогодні багато застосунків використовують цю технологію.

Flutter – це сучасний інструмент для створення застосунків з гарним та стабільним UI/UX на базі мови програмування Dart, який створили Google для свого фреймворка. Завдяки Flutter можна створювати інтерфейс користувача будь-якої важкості, з анімаціями, з тінями, створювати елементи будь-якої форми, кольору тощо. При цьому зв'язувати інтерфейс з логікою застосунка, налаштовувати слухачів для різних кнопок, полів вводу тощо. Також головна особливість це використання двигуна для рендерінгу елементів інтерфейсу. Фреймворк використовує Skia, завдяки ньому інтерфейс можна запустити майже на будь-якому пристрої.

Продуктивність застосунків при використанні Flutter має схожі значення з розробкою завдяки рідним мовам/інструментам програмування (офіційні мови операційної системи). Дуже важливо щоб UX мав гарну продуктивність для створення гарного досвіду користування. Flutter не використовує будь-які компілятори та інтерпретатори коду. Застосунок вбудована безпосередньо в машинний код, що усуває будь-які помилки продуктивності інтерпретації. Такий підхід гарантує максимальну ідентичність рідним програмам, користувач зможе користуватись навігацією як в android або схожим зображенням анімації, захист завдяки faceId та touchId, а також робота з жестами [9]. Отже, важливо мати інтерпретатор у машинний код для створення продуктивного застосунку.

Останні покоління фреймворка Flutter містить в собі розширення підтримки Material 3 компонентів, завдяки переносу деяких рідних віджетів, створення строки меню та каскадних меню, використання нового двигуна Impeller для тесту продуктивності, розробники вважають, що цей двигун може зрівнятись з продуктивністю Skia або перевершити його, перевірка головного списку параметрів перед випуском застосунку до магазинів, лупа над текстом для відображення тексту більш детально.

Є велика кількість застосунків, що були створенні завдяки Flutter. Його можливості використовуються відомими компаніями, такими як BMW, Toyota, Google, Intelvita, Rive [10]. Проте Flutter нова технологія, багато компаній ще не зрозуміли потенціал цього інструмента. Google випускає багато оновлень, виправляє помилки, тобто підтримує свій продукт. Останньою версією фреймворку на сьогодні є Flutter 3.7 release. З моменту створення будь-який розробник зможе почати його використовувати, тому що він безкоштовний та є зрозуміла документація.

Переваги:

- використання двигуна для інтерпретації в машинний код для досягнення максимальної продуктивності;
- наявність великої кількості рідних для платформ віджетів та компонентів;
- підтримка великої компанії, тобто постійні оновлення;
- можливість створювати власні віджети з різним функціоналом, додавання анімацій, жестів у додаток та взаємодій з телефоном на низькому рівні.

Недоліки:

- трудний вхід у розробку завдяки Flutter, тому що технологія вийшла нещодавно по міркам програмування, не вистачає досвідчених фахівців;
- слабка підтримка спільноти розробників, це змушує розробників розробляти модулі, віджети самотійно з нуля;

– не зрозуміла перспектива щодо повноцінної кросплатформної розробки, бо на сьогодні Flutter використовується здебільшого для мобільних пристроїв(Android/IOS).

2.1.2 Xamarin

Програмна компанія заснована у 2011 році та придбана компанією Microsoft. Зараз це один з популярних рішень для створення кросплатформних продуктів, на сьогодні він використовується для великої кількості застосунків.

Xamarin – це сучасна платформа для створення продуктивних програм для мобільних пристроїв. Тобто програма може використовувати спільну кодову базу для обох платформ. Інструменти збудовані на основі мови C# розробкою та підтримкою, якої займається компанія Microsoft. Використання дуже потужної мови для написання застосунків робить цей варіант кросплатформної розробки гнучким, бо мова C# має багато функцій для створення кодової бази, яку можна довго підтримувати, використовується у таких технологіях як .Net, Unity, Window form. Тобто компанія може не витрачати багато грошей на різних розробників для створення серверу та застосунку для мобільних пристроїв.

Платформа має дві основні форми це: Xamarin.IOS та Xamarin.Android. З IOS, код компілюється в рідний асемблерний код ARM, він використовує селектори для надання коду Objective-C керованому коду C# та Registrars для надання керованому коду C# коду Objective-C. Для компіляції використовується компілятор Mono C#(F#). Машинний код – це код, який спочатку буде виконуватися на конкретній платформі [11]. Так як android компілюється з мови C# в мову IL, яка при запуску застосунка компілюється у машинну збірку.

Створення простих елементів інтерфейсу займає малу часу для розробників. Це дозволяє задовольнити бізнес вимоги щодо інтерфейсу

користувача, тобто зміна кольору, форми та положення елементів є в інструментах Xamarin. Є можливість завантажувати open-source бібліотеки це надає можливість використовувати такі функції, як обробка платежів, підтримка маячків та приладів, служба Push Notifications, рішення для стрімінга, хмарного сховища. Завдяки компонентам, які залежать від платформи, розробники можуть створювати інтерфейс близький к створенню застосунку завдяки рідній мові, тобто мові ОС. Цей інструмент показує продуктивність близьку к рідній розробці, завдяки таким інструментам як Xamarin Test Cloud разом з Xamarin Test Recorder дозволяє запускати автоматичні тести до запуску застосунку. Test Cloud надає 2 тисячі мобільних пристроїв «у хмарі» для тестування [12]. Також є аналітичні інструменти такі як: Hockeyapp та Xamarin.Insights. Це допомагає знаходити проблеми в інтерфейсі та логіці програми до етапу тестування на дійсних платформах.

Завдяки інструментам Xamarin створено багато застосунків для комерційних цілей, для самостійного використання, тощо. Використання цієї платформи важко відслідкувати, бо Microsoft не дає інформацію, щодо вибору інструменту як платформи для створення мобільних застосунків. Використання Xamarin було помітно в таких програмах як: Olo, UPS, FreshDirect, Alaska Airlines, тощо [13]. Як і попередній фреймворк, Xamarin є безкоштовним, тобто будь-яка компанія може використати цей потужний інструмент.

Переваги:

- легкий вступ для нових розробників, бо є гарна документація, інститут Xamarin, в якому записані відео головними розробниками цієї платформи;
- велика кількість C# розробників, це допомагає швидкому розвитку;
- готова екосистема для розробників;
- легка підтримка застосунку для обох мобільних пристроїв;

- стек технологій одного типу, завдяки цьому розробник самостійно може створити взаємопов’язану повноцінну систему.

Недоліки:

- великий розмір застосунку, тому що використання зовнішніх бібліотек та компілятора Mono змушує розробників дуже уважно слідкувати за розміром своїх програм;
- хоча продуктивність в Xamarin близька к рідним застосункам, але велика кількість елементів або багатофункціональні анімації можуть зменшити головний фактор досвіду користувача;
- маленька підтримка спільноти розробників, це змушує кожного створювати код для проєктів заново, це гірше ніж використовувати вже готові рішення та елементи;
- малий штат розробників платформи, це сприяє затримці оновлення;
- важливо знати рідні мови ОС для впровадження більш важкого функціоналу;
- незрозуміле майбутнє щодо повноцінної кросплатформної розробки, чи залишиться платформа тільки для мобільних пристроїв.

2.1.3 React Native

React Native – це фреймворк з відкритим кодом створений компанією Facebook у 2015 році. Він дозволяє розробникам створити кросплатформні застосунки будь-якої складності та для будь-яких пристроїв. Тобто підтримка планшетів, мобільних пристроїв та розумних телевізорів. Зараз ця платформа мабуть найпопулярніша серед розробників для мобільних пристроїв, любов спільноти до React Native – має велике значення для вибору цієї технології як основу для свого комерційного проєкту.

Платформа використовує бібліотеку JavaScript для створення інноваційних та провідних інтерфейсів – React. Зараз React займає перше

місце по створенню будь-якого інтерфейсу. Використання провідної платформи для створення вебсайтів допомагає розробникам створювати такі ж складні та багатофункціональні інтерфейси для мобільних пристроїв.

React пропонує такі підходи для більш легкого процесу розробки: написання UI компонентів, які можна використовувати декілька разів, JSX – це синтаксис, який допомагає створювати легкий та зрозумілий інтерфейс. Використання провідного підходу для оновлення даних в інтерфейсі, який називається віртуальне DOM-дерево, тобто цей підхід забезпечує оновлення інтерфейсу тільки, в якому змінилися дані, зі сторони користувача. Такий результат досягається завдяки: порівнянню змінених елементів завантаженого (це дерево будується при першому рендеру програми) дерева з віртуальним.

React Native пропонує використовувати схожий підхід при створенні інтерфейсу користувача що і React, але при цьому у мобільній розробці розробник повинен користуватися рідними компонентами React Native. При використанні цих елементів, розробник може використовувати написання стилів такі як у вебсайтах, тобто написання CSS параметрів. Завдяки цьому будь-які вимоги щодо UI від бізнесу можуть бути виконані. Використання CSS параметрів робить стилювання елементів цікавим та зручним, бо є такий підхід як Flex Box, він пропонує розташовувати елементи залежно від інших, тобто не потрібно прописувати зайві відступи, щоб виділити якийсь головний компонент. Таким чином розробник може редагувати форму, колір та розташування компоненту як захоче, ще можна передавати параметри для більш детального налаштування елементу, такі як вид клавіатури для поля вводу, кількість строків, область натискання. Важливою можливістю у застосунках для мобільних пристроях є контроль жестів, це полегшує користування програмою.

Розробники, зробили легкий спосіб створення рідних модулів на мові операційної системи та використовувати його в коді React Native. Дивлячись на те що, платформа дуже популярна, тобто має дуже велику спільноту. Це

сприймає створенню будь-яких бібліотек для полегшення розробки застосунків. Також важливим є те що багато досвідчених фахівців можуть допомогти з проблемами у створенні програми.

Використання моста між JavaScript кодом та UI, дозволяє слідкувати за продуктивністю програми, але останній час розробники, які порівнюють архітектуру застосунків RN з новою платформою Flutter, прийшли до того, що цей зв'язок між UI та JavaScript коду сильно впливає на результати продуктивності. Проте це не заважає розробникам використовувати складні анімації для своїх проєктів, але потрібно бути дуже обережним та слідкувати за параметрами продуктивності, для цього в React Native є багато інструментів для відлагоджування коду.

Останні оновлення React Native мають великі перспективи для створення гарної архітектури, яка повинна бути більш продуктивна, тому що розробники у Meta бажають позбавитись цього через код на мові JavaScript та UI. В останньому великому оновленні було покращено інструмент для відлагодження коду, тобто можна бути вибирати та дивитись як кожний елемент відображається самостійно в дереві.

Використання цього провідного та потужного інструмента можна знайти в таких всесвітньо відомих компаніях: Facebook, Microsoft, Shopify, Wix, Puma, Sony, Discord, Tesla, NFL, Bloomberg, Pinterest, Bolt [3]. З цього можна зрозуміти що використання цієї платформи для створювання великих та малих застосунків з складними анімаціями для авто, дзвінками, використанням мапи та звичайний комерційний інтерфейс для продажі товарів.

Переваги:

- велика спільнота розробників, завдяки цьому можна знайти допомогу з будь-якої проблемі та рішення особисто для застосунку, над яким працюють;
- гарна документація, розділені компоненти, які можна використовуватись в програмі та наявна інформація щодо їх використання;

- великий набір інструментів для створення оригінального, з гарною оптимізацією застосунку;
- постійні оновлення зі сторони розробників Meta;
- наявність можливості створювати рідні для ОС компоненти, які можна потім використати у основному коді;
- єдиний стек для створення повноцінної системи з веб-сайтом, мобільним пристроєм та серверною частиною;
- легкий вступ до розробки, так як технологія використовує такий же підхід, як в React, бо він теж має легкий вступ для нових розробників;
- модульний дизайн, завдяки цьому компоненти розбиті на самостійні файли, при цьому їх можна використати декілька разів;
- підтримка TypeScript, це мова з статичною типізацією, надає можливість підтримувати проєкт на протязі життя програми;
- Однонаправлений потік даних забезпечує стабільний та зрозумілий код, також є можливість відділення логіки від UI, тобто використання шаблону проєктування MVC.

Недоліки:

- слабка продуктивність з великою кількістю елементів та багатофункціональними анімаціями, також проблемою є знаходження контролю над пам'яттю на низькому рівні, тобто неможливо збільшити продуктивність за рахунок контролю пам'яті;
- проблема з впровадженням інноваційних та сучасних функцій операційних систем, вони потребують нової моделі використання, розробники затримують ці оновлення;
- можливість проблем та специфічному пристрої, здебільшого дуже важко знайти причину цієї проблеми;
- проблема безпечності мови програмування JavaScript, тобто є можливість заволодіти важливими даними завдяки вразливості мови;
- нестабільність роботи застосунку, можуть бути конфлікти між різними бібліотеками, для вирішення цих проблем потрібно використати

пошукові системи, спонтанне вихід с застосунку з приводу помилки, яку можна вислідити тільки завдяки рідним інструментам.

2.2 Порівняльна таблиця найвідоміших платформ для створення кросплатформних застосунків

В таблиці 2.1 можна побачити, що провідні компанії – зацікавленні в розробці програм для різних пристроїв, а компанії розробники продукту у розвитку цієї сфери.

При виборі платформи потрібно зосередити увагу на важливих на таких параметрах як:

- продуктивність;
- швидкість розробки;
- єдиний стек для одної технології;
- розмір застосунку;
- підтримка спільноти.

Дивлячись на ці параметри, було вибрано як платформу для розробки кросплатформної розробки React Native, бо має гарну документацію, є підтримка спільноти, це важливо при знаходженні помилок та впровадження складних рішень. Продуктивність у React Native має не найбільші показники, але в звичайному комерційному проєкті без складних анімація, ця технологія покаже нормально продуктивність. Також важливим є те що, використання єдиного стеку, полегшує програмісту написання повноцінної системи, тобто застосунків для обох пристроїв та серверної частини, у майбутньому Web проєкту.

Таблиця 2.1 – Порівняльна таблиця найвідоміших платформ

Назва	Вартість розробки	Мова програмування	Платформи	Найвідоміший приклад використання	Розмір застосунку
Dart/Flutter	Низька	Dart	IOS, Android та Web	Rive, Sua Music, BMW	Малий
C#/ Xamarin	Висока	C#	IOS, Android та Windows	Olo, UPS	Великий
Java Script/ React Native	Низька	JavaScript	IOS та Android	Facebook, Skype, Tesla, Discord	Середній

2.2 Основні відомості про розробку мобільного застосунку з використанням React Native

Платформа React Native дає можливість створювати застосунки для мобільних пристроїв, не вимагаючи для цього якихось особливих знань. Головним підходом створення програми є компонентно-орієнтований підхід, завдяки якому розробник створює ключові одиниці інтерфейсу та може користуватись декілька разів (наприклад, відображення кнопки, полю для вводу тексту, заголовків та колонтитулів).

Детальніше про архітектуру застосунку, використання моста робить програму менш ефективною, тому що є JavaScript Bundle, в якому компілюється код, коли застосунок розвернений для виходу вже в сучасні

магазини, JavaScript ядро запускає Bundle який запускається кожного разу, коли користувач запускає програму. Коли код вже запустився, починає працювати міст через який проходять події(тобто дії користувача у застосунку) у JavaScript потік, а потім коли JS код виконаний результат чи відгук на дії повертаються у потік інтерфейсу користувача(рис 2.1). Тобто цей крок, через який проходять події користувача або відгуки на них, заважає бути продуктивним, також з цим підходом складно створювати складні жести та анімації, бо при дії зі сторони користувача результат, який він повинен побачити буде відображатись с затримкою.

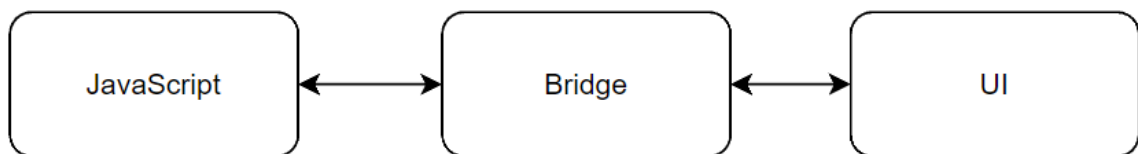


Рисунок 2.1 – Відображення роботи моста

Завдяки потужним інструментам, які були створені спеціально для React Native, можна створювати навігацію зі складною вмістимістю [14], контролювати жести користувачів [15], малювати 2D графіку у застосунку [16] та багато іншого. Також наявність JavaScript коду дозволяє використовувати його бібліотеки, тобто всі менеджери стану для відділення логіки від відображення(MVC), контролю полів вводу, бібліотеки для форматування часу тощо. Також підключення мови TypeScript до React Native є не складним процесом, тобто користувач зможе використовувати примітивні типи такі як: number, string, Boolean, тощо. Також ця мова надає можливість створювати такі структури як Enum, Type, Interface, Record, Omit, Pick та інші, які є в Type Script.

Можливість запускати локально на емуляторі дозволяє створювати різні елементи інтерфейсу користувача і розставляти компоненти у реальному часі і відразу ж тестувати отриманий результат та бачити

інтерфейс. Ще один великий плюс у розробці, що можна продивлятися всі помилки, також побачити, де ця помилка з'явилась, коли в консольній панелі розробник запускає локальний сервер, через який двигун Node JS компілює та інтерпретує Java Script код.

Кожна розробка починається з планування архітектури. За допомогою Java Script бібліотеки React, можна створювати гарний проєкт з відділеними шарами логіки, тобто завдяки використанню функціонального підходу створення інтерфейсу користувача, з'явилась можливість використовувати так звані хуки, це звичайна функція, в якій відділяється логіка окремої сутності та потім викликається ця функцію у компоненті [17].

Visual Studio Code – це інтегроване середовище розробки (IDE), що можна скачати з офіційного сайту. IDE містить в собі функції зручного тестового редактору. Завдяки спільноти та розробникам програми є багато можливостей створити найзручніше середовище для себе, бо є велика кількість безкоштовних розширень. Також можливість користуватись інтерфейсом для публікації та інших дій у системі контролю версій Git. VS Code – зручне середовище розробки, найкращий вибір для розробника, який розпочинає розроблювати код за допомогою мови Java Script.

Переваги:

- офіційність, так як ця мова була створена компанією Microsoft, тому можна бути впевненим, що цей застосунок буде підтримуватись;
- безкоштовність, є дуже гарна альтернатива VS Code – Webstorm, але він не безкоштовний, та багато функціоналу можна знайти у VS Code;
- функціональність, у середовищі є безліч якісних та корисних плагінів. З їх допомогою можна розширити можливості та підключити інші мови програмування;
- зручність роботи з системою контролю версій, розробник не повинен знати всі команди для створення версій застосунку, потрібно лише користуватись зручним інтерфейсом;

- легкий старт, щоб зрозуміти весь функціонал середовища не потрібно багато часу та перегляд складних уроків;
- зручні інструменти відлагодження, для знаходження помилок у коді завдяки встановленим правилам для написання коду або інструменту налагодження.

Недоліки:

- помилки у роботі інструментах форматування, іноді потрібно перезавантажити програму, щоб вони запрацювали;
- недостатність функціоналу у самому середовищі, часто потрібно знаходити важливі саме розробнику розширення;
- відсутність запуску різних платформ або застосунків, потрібно писати команди вручну, але це доказує універсальність середовища, бо він зручний для написання будь-якими мовами.

Головний екран виконання при написанні коду в Visual Studio дуже легкий, для цього потрібно відкрити будь-яку папку завдяки середовищу, інтерфейс користувача представлено на рисунку 2.2. Можна побачити зручне дерево файлів проєкта, всі інструменти управління та робочі інструменти.

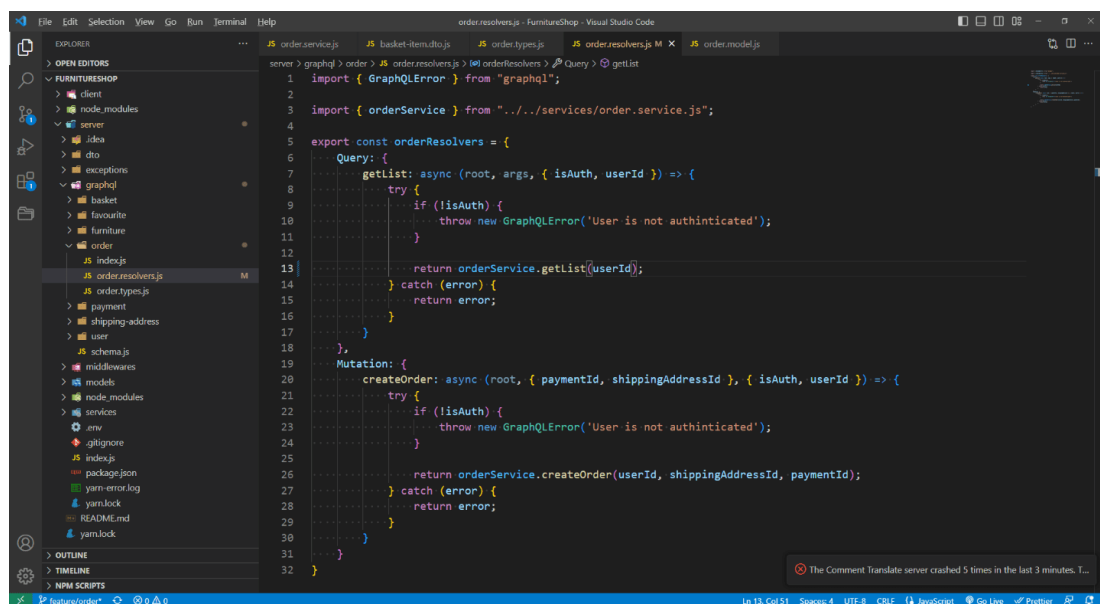


Рисунок 2.2– Головний екран в “Visual Studio Code”

2.3 Node JS

Для кожної мови потрібен компілятор, щоб запускати його на будь-якому приладі, раніше розробники могли запускати Java Script код тільки у браузері для сайтів, потім розробники вирішили створити можливість запускати його як програму на будь-якому комп'ютері. Таким рішенням було створення Node JS платформи.

Node JS – середовище виконання для мови Java Script з відкритим кодом, яке виконує код завдяки двигуну V8, на основі якого була створена платформа. Середовище було створено в 2009 році Райном Далем у центрі розробки Google, яке знаходиться у Німеччині. Платформа допомогла створити велику спільноту розробників та об'єднати їх.

V8 – це написаний на мові C++ двигун, який використовується в браузері Google Chrome, саме він запускає код Java Script на стороні клієнта. Двигун перетворює код розробника у швидкий машинний код. Завдяки дуже потужній мові C++ він робить Java Script дуже швидким, це зрівнює мову з іншими швидкими скриптовими мовами. Двигун компілює код JS у збірку під час виконання, реалізуючи Компілятор JIT(Just-In-Time) [18]. Головна особливість є те що він не виробляє проміжний код, це сильно впливає на продуктивність написаного проекту.

Завдяки Node JS створили зручний менеджер пакетів npm для встановлення сторонніх бібліотек, платформ тощо. Будь-який Node JS проєкт містить package.json файл, який містить скрипти, які використовують розробники для швидкого запуску та розгорнення застосунків, модулів та їх версії.

Роль Node JS у розробці для мобільних пристроїв укладається у тому, що спочатку потрібно запустити файли за допомогою платформи, а потім вже вони використовуються як код у Bundle у застосунку, який вже потім розробник встановлює на симулятор.

Роль середовища для створення серверу: запуск серверу локально виконується тільки завдяки V8 двигуну, це робить роботу серверу дуже швидкою. Також основні види роботи з файлами надаються у Node JS, завдяки ним розробники можуть створювати, читати, редагувати або видаляти файли на системі, яка запустила проєкт. Також завдяки вже пакету http, розробник може створювати сервер на основі протоколу.

Http – це протокол прикладного рівня, який може передавати вебсторінки, зображення, це є головною задачею протоколу, але він ще може передавати інші файли, тому завдяки ньому Node JS передає JSON або з іншим форматом файли з даними для інших застосунків. Аббревіатура HTTP розшифровується як HyperText Transfer Protocol – протокол передачі гіпертексту (тобто текстових документів, які містять посилання на інші документи) [19]. Http – це приклад звичайного протоколу з використанням REST архітектурного стилю створення сервер застосунків з такими методами як Put, Delete, Post, Get.

2.4 Express

Як для створення застосунків для мобільних пристроїв, так і для створення серверу потрібен фреймворк, який матиме багато інструментів, які можуть прискорити та допомогти створити велику систему. Виконання основного функціоналу самостійно, признак гарної платформи, користувачам потрібно тільки користуватися інструментами.

Express – це швидкий, гнучкий серверний фреймворк, який працює на Node JS, платформа має відкритий код, тому дозволяє іншим розробникам створювати модулі. Розробник платформи заявляє, що фреймворк дуже мінімалістичний, але має велику кількість пакетів, які розробник може під'єднати. Він надає розробникам зручний інтерфейс для створення обробників запитів. Крім того, Express надає багато функціоналу для обробки звичайних HTTP запитів, включаючи такі методи як GET, POST, PUT,

DELETE та інші. Також зручні інструменти для керування сесансами, файловою системою та роботи з базами даних, включаючи Mongo DB, MySQL, PostgreSQL та інші.

Архітектура платформи намагається бути дуже гнучкою та зручною, це надає можливість розробникам створювати застосунки з різним функціоналом від малої складності до великої. Основи архітектури є концепція маршрутизації – це процес направлення запитів зі сторони клієнта на обробку, після цього результат відправляється звідки був запит. Express надає інструменти для роботи з шаблонами, що дозволяє розробникам легко та швидко створювати відображення даних у форматі HTML.

На рисунку 2.3 можна побачити як запускається звичайний застосунок з використанням Express. Це вхід у програму, на якому створюється програма та підключаються базу даних, завдяки бібліотеці Mongoose, і вже потім просимо express слухати цей порт.

```

20  });
21
22  |
23  const app = express();
24
25  const start = async () => {
26    try {
27      await mongoose.connect(process.env.DB_URL, {
28        useNewUrlParser: true,
29        useUnifiedTopology: true
30      });
31      app.listen(PORT, () => console.log(`Server works on ${PORT}`));
32    } catch (error) {
33      console.log({ error });
34    }
35  }
36
37  start();

```

Рисунок 2.3 – Запуск застосунку завдяки Express

Переваги:

- простота використання, це дуже гнучкий та без зайвого функціоналу фреймворк, що робить його швидким для використання;

- гнучкість, архітектура надає повне керування всією програмою, не всі платформи надають таку можливість;
- велика спільнота розробників, завдяки цьому, користувачі платформи можуть знайти дуже багато документації, плагінів та розширень, які полегшують роботу з API.

Недоліки:

- велика свобода, платформа не має чіткої структури та архітектури, тобто розробник може створити код, який буде складно підтримувати;
- небезпечність, так як Express використовує JavaScript, це означає, що код не має сильного захисту;
- не найбільша продуктивність, означає що при великих запитів до БД, або запитів з великими об'ємами даних може бути проблеми з нею.

Отже, вибір найпопулярнішого фреймворку для створення серверу на сьогодні, ґрунтується на тому що це швидкий спосіб написати продуктивний для малої кількості даних API, при цьому використати мову що ї для написання мобільного застосунку.

2.5 Graph QL

Graph QL – мова запитів, створена компанією Meta для роботи з API, який описує як запитувати дані, і, в основному, використовується клієнтом для завантаження даних з сервера [20]. Його найголовніші особливості: можливість клієнту точно вказувати те, що йому потрібно, полегшує агрегацію даних з декількох джерел, використання типів для виділення полів даних. Він надає більш ефективний шлях одержання даних від сервера, чим звичайний стиль RESTful.

У Graph QL клієнтська сторона відсилає запит на сервер із конкретними полями та зв'язками, які потрібно отримати. Сервер, у свою чергу, повертає лише запитані дані, на відміну від REST, де клієнт отримує

всі дані. Мова запитів – сильно типізована мова, що дозволяє попередньо оцінити коректність запиту в рамках системи типів цього синтаксису, до виконання [21]. Це здійснюється завдяки написанню схем з типами запитів, вхідних та вихідних параметрів. На рисунку 2.4 можна побачити написання звичайного типу для мови запитів.

```
server > graphql > furniture > JS furniture.types.js > furnitureSchema
1  import { buildSchema } from "graphql";
2
3  export const furnitureSchema = buildSchema(`#graphql
4      type Furniture {
5          id: ID!
6          name: String!
7          cost: Int!
8          description: String!
9      }
10
11     type Query {
12         getFurnitures: [Furniture]
13     }
14 `);
```

Рисунок 2.4 –Схема Graph QL запитів

Graph QL також надає можливість використовувати більше глибокі запити даних, дозволяю клієнту запитувати багато зв'язаних об'єктів в одному запиті, що знижує кількість запитів до серверу и прискорює час від відгуку. При створенні серверу завдяки Graph QL, платформа створює єдиний ключовий шлях, або кінцеву точку, через яку клієнт та сервер спілкується. Це досягається завдяки додатковому слою між клієнтом та сервером. Клієнтська частина відсилає запит будь-яким протоколом передачі даних, до обробника Graph QL, потім вже сервер приймає цей запити і надсилає відповідь до клієнту, також через Graph QL(рис 2.5).

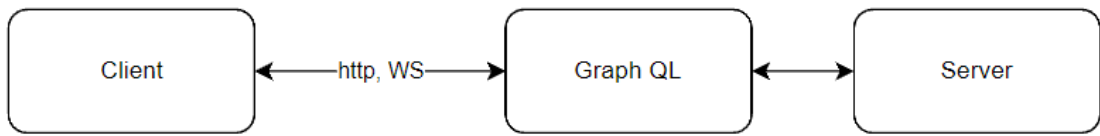


Рисунок 2.5 – Робота слою мови запитів Graph QL

Також наявний дуже зручний інструмент для тестування запитів, яка дозволяє створити документацію запитів, щоб будь-який розробник проекту зміг побачити всі запити та типи для них.

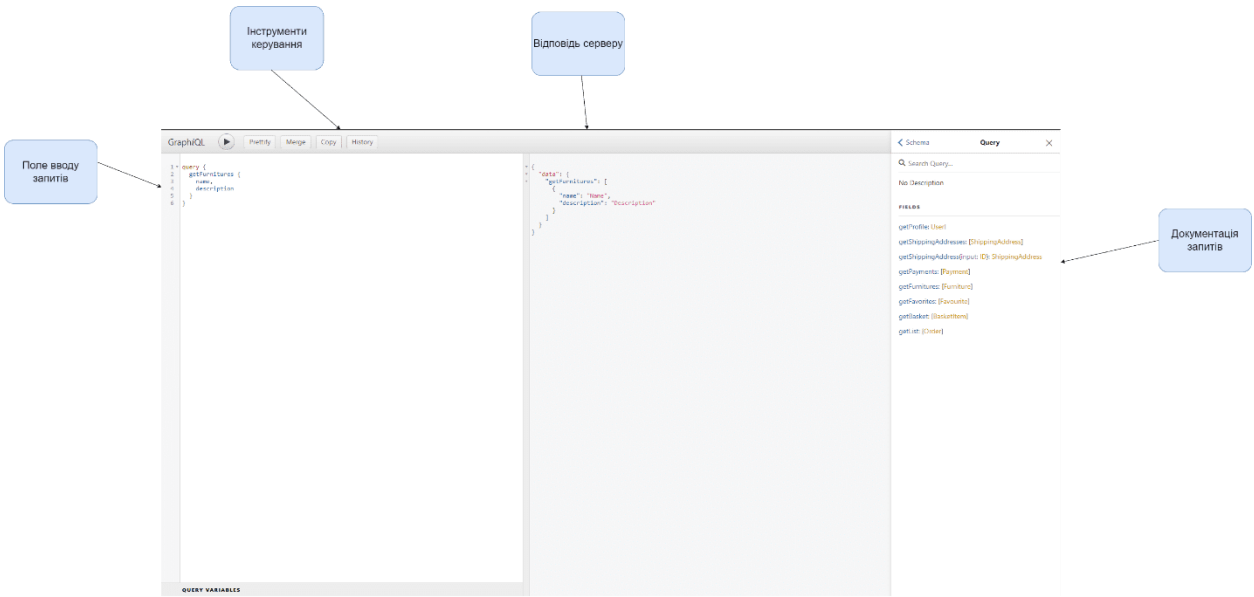


Рисунок 2.6 – Інструмент тестування запитів

Для запитів з клієнта, тобто з мобільного пристрою або вебсайтів, завдяки JavaScript коду, до серверу використовується Apollo Client бібліотека. Apollo Client – комплексна бібліотека керування станом, яка дає можливість змогу керувати як локальним, так і віддаленими даними за

допомогою Graph QL запитів [22]. Apollo Client зручний допоміжний інструмент, який зменшує кількість необхідного коду для роботи з Graph QL.

Основні функції:

- запити та мутації;
- кешування даних;
- механізм підписки та оновлення даних в реальному часі, це може бути корисним для розробки чату;
- управління станом застосунку.

2.6 MVC

MVC – це обов’язковий шаблон для організації коду чи компонентів [23], він визначає головну архітектуру проєкту. За MVC шаблоном в застосунку код повинен бути розділеним на 3 взаємопов’язані частини: модель даних, уявлення(інтерфейс користувача), модуль керування, кожна частина повинна визначити свою ділянку коду.

Модуль керування застосовується для зв’язку між двома іншими частинами. Він повинен відокремити використання даних(модель) від виду(інтерфейсу користувача). Контролер відповідає за надання даних, присутніх Моделі, компоненту Відображення та інтерпретацію відповідей користувачів[24]. Цей підхід робить код більш підтримуючим та надає можливість безпечно додавати новий функціонал застосунку.

Модель даних – це головний компонент шаблону, він інкапсулює ядро даних [25] і основну логіку роботи з ними, це дозволяє не залежати від вхідних або вихідних даних, тобто незалежність від інтерфейсу користувача.

Вигляд або інтерфейс – це будь-який спосіб надання інформації, яку модель даних обробляє і користувач повинен побачити. Також забезпечує інтерфейс взаємодії з користувачем. Вид може бути представленим у вигляді графічного інтерфейсу користувача, веб-сторінки або будь-якого іншого способу відображення.

Головна ідея шаблону полягає в тому, що кожна частина коду має свою мету, і ці цілі різні [26]. Структура файлів для використання стилю – є досить проста, потрібно лише відсортувати файли по папкам.

2.7 Mongo DB

Mongo DB – це нереляційна база даних, яка зберігає дані у вигляді документів у форматі JSON (Java Script Object Notation). Це означає, що дані можуть складатись з інших об'єктів, масивів та звичайних примітивів. Це дозволяє надавати дані у більш гнучкому форматі, ніж у реляційних базах даних. База даних також забезпечує повну підтримку індексування та реплікацію з багатими та інтуїтивно зрозумілими API [24]. Балансування навантаження, які дозволяють легко масштабувати базу даних для обробки великих обсягів даних. Більшість операцій у MongoDB виконуються за допомогою мови запитів Mongo DB Query Language (MQL).

База даних має різноманітні інструменти для адміністрування та моніторингу бази даних, такі як Mongo DB Compass, який надає зручний інтерфейс для роботи з даними. Також база даних володіє зручністю контролювати базу даних, завдяки вебсайту.

Mongo DB була розроблена з орієнтацією на потреби сучасних вебзастосунків, які потребують високої доступності та масштабованості.

Для взаємодії сервера за базою даною Mongo DB використовується бібліотека Mongoose. Вона дозволяє розробникам легко працювати з базою даних. Mongoose – це ODM(Object Document Mapper – об'єктно-документний відображувач). В ньому розробнику дозволяється визначати об'єкти зі строго-типізованою схемою [27]. В типізацію входить: визначення обов'язкових полів, типи даних(String, Number, Object, Array тощо), валідацію та інші параметри. Також Mongoose пропонує багато інших корисних функцій, таких як підтримка middleware, яка дозволяє розробникам виконувати код перед або після збереження даних у базі даних та підтримка

пакета Promise, який дозволяє виконувати асинхронні операції з базою даних. Додатково, є можливість використання псевдооб'єднань, це можливість об'єднувати документи з кількох колекцій в одній базі даних, підхід називається заповненням [28].

2.8 Вибір програмної мови для розробки комерційної системи

Вибір мови для розробки комерційної системи повинен бути зроблений на основі детального аналізу вимог та потреб розробників та користувачів систем. Розглянемо найпопулярніші для створення мобільного застосунку, вебпроектів та серверу:

- Python;
- JavaScript;
- Swift;
- Kotlin;
- Dart;
- C#.

Python – це дуже зручна, скриптова мова загального призначення, яка була розроблена в 1991 році. Мова має чистий та легкий синтаксис та конструкції, що робить його дуже легким для початківців та популярним для досвідчених розробників [31].

Головна перевага Python велика кількість доступних та безкоштовних бібліотек для різних цілей розробки, таких як машинне навчання, веб-розробка, обчислення, серверна розробка тощо. Також розробник може користуватися декількома парадигмами програмування, а саме: об'єктно-орієнтоване програмування, функціональне, програмування зі змінними типами даних.

Python – гарна мова для розробки серверної частини, але для розробки мобільного застосунку, ця мова не підходить, отже, це зобов'язує розробника знати 2 мови для написання повноцінної системи.

JavaScript – інтерпретована мова програмування, яка використовується в більшості для створення логіки для інтерфейсу користувача, але є інші варіанти користування.

JavaScript підтримує об'єктно-орієнтовану, функціональну та імперативну парадигми програмування. У мови є велика спільнота, яка створює основні інструменти для створення багатьох комерційних продуктів. Ця мова стала найголовнішою для створення веб-застосунків з двома іншими технологіями HTML та CSS. JavaScript дуже вільна та легка мова для навчання, але JavaScript є небезпечною мовою, бо ця вільність може зробити проникнення у проєкт простим [18].

Використання JavaScript економить багато часу та коштів для замовників, тому що розробник може створювати систему одним стеком.

Swift – мова розроблена компанією Apple для розробки програмного забезпечення для платформ компанії. Мова має сучасний синтаксис, так як була випущена нещодавно [29].

Swift є статично типізованою мовою програмування з можливостями користуватися об'єктно-орієнтованими та функціональними парадигмами. Мова має гарні характеристики для створення сучасних та продуктивних застосунків, завдяки безпеці, швидкості, легкому синтаксису, що дозволяє швидко створювати проєкти. Тому використання для створення рідного застосунку для Apple пристроїв найкращий вибір, але неможливо створювати Android програмні засоби.

Kotlin – це мова програмування, яку створили JetBrains у 2011 та призначена для розробки програмного забезпечення для платформи Java. Копія Java має дуже схожий синтаксис [30].

Безпечність та використання статичної типізації робить мову гнучкою та допомагає уникати помилки у коді. Також розробник може піти двома шляхами програмування об'єктно-орієнтоване та функціональне. Підтримка корутинів робить мову зручною для використання асинхронних застосунків. Kotlin підтримує велику кількість платформ, він є мовою загального

використання, тобто завдяки ньому можна створити вебзастосунки, мобільні програми для обох операційних систем, серверну розробку. Але розробники вибирають мову як основну для розробки Android засобів, бо мова є офіційною для розробки для цієї платформи.

Dart – це мова, розроблена компанією Google, якою користуюся для розробки веб-застосунків, мобільних та серверних програм [10]. Dart є одночасно і компільованою, так і інтерпретованою мовою програмування. Вона забезпечує зборку сміття, що робить розробку легше та швидше.

Найпопулярнішим прикладом використання мови є кросплатформна розробка завдяки платформі Flutter. Вона дозволяє створити рідні для операційної системи інтерфейси користувача для IOS та Android з використання одного коду.

Dart має свіжий синтаксис, тобто легкий для освоєння, а також підтримує основні парадигми програмування.

C# – це ООП(об'єктно-орієнтована мова програмування), розроблена компанією Microsoft. C# має велику кількість вбудованих бібліотек, що надає багато можливостей розробки для різних платформ. Ця мова використовується для різноманітних цілей, таких як створення веб-застосунків, мобільних та серверних програм, а також різної складності ігор. Підтримка асинхронного програмування робить роботу з асинхронними операціями легкою та ефективною [32].

Детальніше про характеристики та подальший вибір програмної мови для розробки комерційної системи для мобільних пристроїв можна побачити на таблиці 2.2.

Так що є декілька варіантів створити мобільний застосунок з сервером завдяки використанню однієї мови програмування. На мою думку Java Script більш всього підходить для цієї мети, бо легкість мови дозволяє вивчити її за малий час та починати розробляти комерційний продукт. Єдиний стек задовольняє потреби бізнесу, один розробник може створити та підтримувати систему. Наявність підтримки декількох парадигм робить мову з широким

застосуванням. Завдяки великій спільноті розробників є велика кількість бібліотек та готових рішень, з якими користувач може працювати та створювати застосунки різної складності.

Таблиця 2.2 – Характеристики програмних мов

Характеристика	Python	JavaScript	Swift	Kotlin	Dart	C#
Синтаксис	Лаконічний	Гнучкий	Простий, безпечний	Простий	Схожий на Java	Схожий на C++
Мобільна розробка	-	+	+	+	+	+
Серверна розробка	+	+	+	+	+	+
Тип системи	Динамічна	Динамічна	Статична	Статична	Статична	Статична
Використання	Загальне призначення	Загальне призначення	IOS розробка	Android розробка	Flutter розробка	Загальне призначення
Популярність	Дуже популярний	Надзвичайно популярний, особливо веб	Серед Apple розробників	Серед Android розробників	Популярність зростає	Дуже популярний
Парадигми	ООП, функ.	ООП, функ.	ООП	ООП, функ.	ООП	ООП

2.9 Висновки за розділом

В ході вибору предметної області для вирішення задачі дипломної роботи був вибраний фреймворк React Native для розробки кросплатформної проєкту, а також бібліотека Express для створення серверної частини, які з легкістю допоможуть досягти мети, а саме створення комерційної системи з мобільним застосунком, завдяки своїй підтримки зі сторони спільноти та легкістю розробки. Використання нереляційної бази даних підходить, бо при аналізі моделей підходить саме такий тип, тому що проєкт не містить великих та складних взаємозв'язків між типами даних. Завдяки використанню

бібліотеки Mongoose, яка надає зручні та швидкі інструменти для використання саме Mongo DB. Також, в процесі аналізу було порівняно мови програмування та обрана Java Script, яка найбільше підходить для розробки багатофункціональної розробки. Також, було обране середовище для розробки програмного коду для розробки системи – Visual Studio Code.

3 ОСНОВНІ РІШЕННЯ З РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

В даному розділі розглянуті основні рішення з розробки класів, основні рішення щодо розробки інтерфейсу користувача, рішення щодо збереження та використання бази даних.

Застосунок поділений на дві папки з головними елементами системи, а саме серверна система та клієнтська, тобто мобільний застосунок.

3.1 Основні рішення щодо розроблення серверної частини

Кожна сутність серверного застосунку має однаковий шлях створення, тобто поділено логіку на 2 шари, це рішення робить легше підтримку та розробку кожної сутності окремо. Реалізовано, завдяки класам, які доступні в Java Script, також можна використати ООП та притримуватися SOLID принципів написання коду.

Шар контролер – це місце знаходження логіки роботи з Graph QL запитами, в цьому місці потрібно перевіряти вхідні параметри та повертати вихідні дані, які отримуються з сервісів. У контролерах прописано всі типи запитів, які пропонуються технологією Graph QL, а саме Query, Mutation та Subscription.

Query – це тип запиту, через який користувач отримує дані, головна відмінність: запит не повинен змінювати щось у базі даних або в інших сховищах. Він схожий на звичайний GET запит у REST стилі. Запит може получати параметри, які можна використати у пошуку, наприклад ідентифікатор якогось елементу.

Mutation – це тип запиту, через який користувач може змінювати значення у базі даних або в інших сховищах. Він схожий на PUT або POST запити у REST стилі. Також запит отримує параметри, їх можна прописати як input тип, якщо параметрів багато, або використати примітиви, що Graph QL надає.

Subscription – це тип запиту, який не було використано проєкті, бо він надає можливість встановити слухача на якійсь дані, тобто клієнт отримує інформацію завжди актуальну. Це тип дуже корисний для чатів, адже повідомлення повинні бути відображенні миттєво після відправки.

Запити використовуються мобільним застосунком, також їх використання необмежено тільки мобільною програмою. Тобто при розширенні застосунку, може використовуватись інші API, або веб чи інші застосунки.

Шар сервіс – це місце, де знаходиться основна робота з моделями бази даних, тобто збереження, зміна на видалення документів з БД. Цей шар містить основну бізнес логіку, якщо розробник вирішить змінити стиль будівництва API, в такому випадку йому не потрібно змінювати цей шар, бо він залежить тільки від моделей БД.

Серверна частина через допоміжну бібліотеку mongoose підключається до Mongo DB. При створенні будь-якої сутності застосунку потрібно визначити модель та тип даних, які будуть використовуватись у програмі. Завдяки інструментам, що надає mongoose визначено тип, чи обов'язковим є поле у моделі, початкове значення та деякі параметри щодо розмірів значення, а також є можливість зв'язати сутності завдяки посиланням.

Було використано шаблон проєктування DTO (Data Transfer Object), він необхідний для передачі даних між частинами системи, наприклад коли API відправляю дані до клієнту, потрібно створити об'єкт без методів, тільки з полями, які потрібні в цій сутності. Також навпаки, мені потрібно чітко визначити, що сервер отримує з клієнту, щоб зберігати тільки потрібні дані у базі даних.

Останнім з елементів серверної частини є middleware (функції проміжної обробки). Їх використання необхідне для визначення чи клієнт авторизований та надання інформації про користувача для подальшої взаємодії з даними, які зв'язані з цим профілем.

3.1.1 Сутність User

User схема створена для Graph QL запитів містить Query та Mutation, а також типи даних, які сервер повертає або приймає.

User – це основний тип для цієї сутності, він має такі поля: id, email, name, shippingAddresses, addressesCount, payments, paymentsCount. Було використано інші сутності в типі користувача, бо вся логіка з адресами та методами оплати зосереджена на екрані профілю, а завдяки Graph QL, можна брати тільки ті дані, які потрібні. Таким чином було створено окремо поля для отримання кількості адресів та методів (рис. 3.1).

```
22  type User {  
23      id: ID  
24      email: String  
25      name: String  
26      shippingAddresses: [ShippingAddress]  
27      addressesCount: Int  
28      payments: [Payment]  
29      paymentsCount: Int  
30  }
```

Рисунок 3.1 – Тип User у Graph QL схемі

Payment – це тип методу оплати, він містить такі поля: id, holderName, cardNumber, isActive, cvv. Потрібен для отримання масиву методів у User типі (рис. 3.2).

```

14     type Payment {
15         id: ID!
16         holderName: String!
17         cardNumber: String!
18         isActive: Boolean!
19         cvv: String!
20     }
21

```

Рисунок 3.2 – Тип Payment у Graph QL схемі

Тип ShippingAddress – адрес відправки, має такі поля: id, name, address, zipcode, country, isActive, city. Потрібен для отримання масиву адресів у User типі (рис. 3.3).

```

4     type ShippingAddress {
5         id: ID!
6         name: String!
7         address: String!
8         zipcode: String
9         country: String!
10        isActive: Boolean!
11        city: String!
12    }
13

```

Рисунок 3.3 – Тип ShippingAddress у Graph QL схемі

AuthResponse – відповідь, яка використовується у запитах для авторизації або реєстрації. Має такі поля: accessToken, user. Токен потрібен для подальшого використання застосунку вже як авторизованим користувачем (рис. 3.4).

```

32     type AuthResponse {
33         user: User
34         accessToken: String
35     }

```

Рисунок 3.4 – Тип AuthResponse у Graph QL схемі

Типи вхідних параметрів для авторизації та реєстрації, віділені в окремий тип Graph QL, а саме input. Як можна побачити на рисунку 3.5 для авторизації потрібні електронна адреса та пароль, а для реєстрації почта, пароль та ім'я користувача. Всі поля є обов'язковими, це можна зрозуміти завдяки “!” оператору.

```

37 //
38     input LoginInput {
39         email: String!
40         password: String!
41     }
42
43     input RegisterInput {
44         email: String!
45         password: String!
46         name: String!
47     }

```

Рисунок 3.5 – Типи вхідних параметрів у Graph QL схемі

Саме головне це приписання типів запитів, без них неможливо буде спілкуватись з сервером. User має один Query запит, а саме отримання профілю, тип який має головні значення користувача. Також є два запита типу Mutation: “createUser”, “login”. Вони потрібні для створення користувача та авторизації. Як можна побачити на рисунку 3.6 запит отримає різні вхідні параметри але повертає AuthResponse.

```
48     ... type Query {  
49         ... profile: User!  
50     }  
51  
52     ... type Mutation {  
53         ... createUser(input: RegisterInput): AuthResponse  
54         ... login(input: LoginInput): AuthResponse  
55     }  
56 `);
```

Рисунок 3.6 – Запити у GraphQL схемі

Логіку написання цих запитів було виділено в окремий об'єкт, щоб не змішувати реалізацію. В ньому описано роботу Query та Mutation запитів саме для User сутності. Як можна побачити на рисунку 3.7: було обернуто логіку у конструкцію try/catch для обробки виключень, а також одне з виключень генерується у запиті, який повертає дані користувача, а саме перевірка чи авторизован користувач, якщо ні, то відправляється помилка на клієнт, що “Користувач не авторизован”. В свою чергу, контролер тільки викликає методи сервісів, логіка роботи з даним у БД та інша бізнес логіка знаходиться у класах сервісу.

```

5 export const userResolvers = {
6   Query: {
7     profile: (root, args, { isAuthenticated, userId }) => {
8       try {
9         if (!isAuthenticated) {
10           throw new GraphQLError('User is not authenticated');
11         }
12
13         return userService.getUser(userId);
14       } catch (error) {
15         return error;
16       }
17     },
18   },
19   Mutation: {
20     createUser: (root, { input }) => {
21       try {
22         const { email, password, name } = input;
23         return userService.register(email, password, name);
24       } catch (error) {
25         return error;
26       }
27     },
28     login: (root, { input }) => {
29       try {
30         const { email, password } = input;
31         return userService.login(email, password);
32       } catch (error) {
33         return error;
34       }
35     }
36   }
37 }

```

Рисунок 3.7 – Реалізація User запитів

Основним класом є UserService, в ньому знаходиться основна бізнес логіка застосунку для сутності користувача. Він має три методи: реєстрація, авторизація та отримати профіль користувача.

В методі реєстрація виконуються основні операції, щодо цього етапу авторизації. Використовується перевірка чи є такий користувач, завдяки електронній адресі, якщо є, то створюється виключення з помилкою “Такий користувач вже існує”. Далі завдяки бібліотеці “bcrypt”, хешується пароль та створюється документ або, його називають, екземпляр моделі User завдяки колекції у форматі об’єкту, що створюється через mongoose і надає методи для роботи з Mongo DB. І вже користувач буде збережений у базу даних з хешованим паролем, електронною адресою та ім’ям. Після цього створюється токен завдяки сервісу, по якому користувач може зайти у застосунок і дані повертаються.

В методі авторизації, також є перевірка на існування користувача, якщо його не має, відправляється виключення з помилкою “Користувача не знайдено”. Завдяки встроєним методам бібліотеки для хешування, перевіряється чи збігається пароль, який надсилає клієнт та пароль збережений у базі даних. Якщо не має збіжностей, повертається виключення, що пароль не вірний. Далі створюється токен для клієнту та повертається дані користувача та токен доступу.

В методі отримання користувача, вже знаходиться документ по ідентифікатору, якщо його не має повертаю виключення з помилкою “Користувача не знайдено”. Потім завдяки сервісам для методів оплати та адресів, отримуються масиви, використовуючи конструкцію “Promise.all” у мові Java Script, вона запускає дві асинхронні функції одночасно. Таким чином повертаються дані користувача та набір методів та адресів, а також їх кількість.

3.1.2 Сутність PaymentMethod

Сутність має тільки Mutation запити та вхідні параметри у Graph QL. Схема надає можливості створювати та оновлювати методи оплати користувача.

Тип вхідних параметрів PaymentInput – використовується для обох мутацій, тобто змін у базі даних. Він має такі поля: holderName, cardNumber, cvv, isActive. Як можна побачити вхідні дані потрібні для створення чи оновлення документа метода оплати.

Mutation, які визначенні у схемі методів: створення та оновлення метода оплати. Можна побачити на рисунку 3.8, що запит створення отримає лише вхідні параметри PaymentInput типу, а запит оновлення отримає ще ідентифікатор, щоб знайти екземпляр. Результат повернення у запитів, це ідентифікатор створеного або оновленого метода оплати.

```

3 export const paymentSchema = buildSchema(`#graphql
4   ...input PaymentInput {
5     ...holderName: String
6     ...cardNumber: String
7     ...cvv: String
8     ...isActive: Boolean
9   }
10
11   ...type Mutation {
12     ...updatePaymentMethod(id: ID, input: PaymentInput): ID
13     ...createPaymentMethod(input: PaymentInput): ID
14   }
15 `);

```

Рисунок 3.8 – Схема для Payment моделі

Реалізація цих методів знаходиться в окремому об'єкті, а саме paymentResolver (рис. 3.9). В ньому описано роботу Mutation запитів саме для Payment сутності. На рисунку 3.9 можна побачити, що використано try/catch конструкцію для того, щоб піймати виключення. Так як це вже робота з даними, які належать користувачу, їх потрібно забезпечити безпекою, тому було вирішено перевірити чи є користувач, якщо немає, створюється виключення. В іншому випадку, викликається необхідний метод у сервісі медів оплати з різними вхідними параметрами.

```

1 import { GraphQLError } from "graphql";
2
3 import { paymentService } from "../../services/payment.service.js";
4
5 export const paymentResolver = {
6   ...Mutation: {
7     ...createPaymentMethod: async (root, { input }, { userId, isAuth }) => {
8       ...try {
9         ...if (!isAuth) {
10           ...throw new GraphQLError('User is not authenticated');
11         }
12         ...return paymentService.createPaymentMethod(userId, input);
13       } catch (error) {
14         ...return error;
15       }
16     },
17     ...updatePaymentMethod: async (root, { input, id }, { isAuth }) => {
18       ...try {
19         ...if (!isAuth) {
20           ...throw new GraphQLError('User is not authenticated');
21         }
22         ...return paymentService.updatePaymentMethod(id, input);
23       } catch (error) {
24         ...return error;
25       }
26     }
27   }
28 }
29

```

Рисунок 3.9 – Реалізація Payment запитів

Методи сервісу, де міститься бізнес логіка роботи з методами оплати: отримання колекції, створення методу та оновлення.

В методі отримання списку перевіряється, чи існує користувач з таким ідентифікатором, при негативному результаті повертається виключення з помилкою. Інакше робиться пошук по документах у базі даних, як параметр пошуку використовує ідентифікатор користувача. Потім перетворюється кожний елемент списку у DTO об'єкт та повертається на рівень, де метод викликали.

Метод створення отримує два параметра: ідентифікатор користувача та об'єкт, в якому міститься дані методу оплати, які потрібно зберегти у базі даних. В ньому завдяки класу бібліотеки mongoose та методу “create”, створюється новий екземпляр моделі Payment.

Метод оновлення знаходить та оновлює метод оплати, також метод моделі “findByIdAndUpdate” повертає вже оновлений документ бази даних.

3.1.3 Сутність ShippingAddress

Сутність ShippingAddress має схожі на методи оплати типи для GraphQL, бо має логіку створення та оновлення моделей адресів.

Вхідний об'єкт, який має такі параметри name, address, zipcode, country, city, id, isActive. Цей тип використовується для обох мутація.

Схема має два запита типу Mutation для створення чи оновлення адресів. В обох випадках вони приймають вхідний об'єкт ShippingAddressInput та повертають ідентифікатор адресу. Це можна побачити на рисунку 3.10.

```

1  import { buildSchema } from "graphql";
2
3  export const shippingAddressSchema = buildSchema(`#graphql
4  ...input ShippingAddressInput {
5  ...     name: String
6  ...     address: String
7  ...     zipcode: String
8  ...     country: String
9  ...     city: String
10 ...     id: ID
11 ...     isActive: Boolean
12 ... }
13
14
15 ...type Mutation {
16 ...     createShippingAddress(input: ShippingAddressInput): ID
17 ...     updateShippingAddress(input: ShippingAddressInput): ID
18 ... }
19 `);

```

Рисунок 3.10 – Схема типів для Shipping Address моделі

Реалізація цих методів знаходиться окремо у об'єкті shippingAddressResolver (рис. 3.11), в ньому було описано роботу Mutation запитів для ShippingAddress сутності. Для обробки помилок було використано конструкцію try/catch. Необхідність використання – захистити дані користувача. Таким чином перевіряється чи існує користувач. Потім повертаю значення з метода сервісу, де знаходиться основна бізнес логіка цієї сутності.

```

1  import { GraphQLError } from "graphql";
2
3  import { shippingAddressService } from "../../services/shipping-address.service.js";
4
5  export const shippingAddressResolver = {
6    Mutation: {
7      createShippingAddress: (root, { input }, { userId, isAuth }) => {
8        try {
9          if (!isAuth) {
10             throw new GraphQLError('User is not authenticated');
11          }
12          return shippingAddressService.createAddress(userId, input)
13        } catch (error) {
14          return error;
15        }
16      },
17      updateShippingAddress: (root, { input }, { userId, isAuth }) => {
18        try {
19          if (!isAuth) {
20             throw new GraphQLError('User is not authenticated');
21          }
22          return shippingAddressService.updateAddress(userId, input)
23        } catch (error) {
24          return error;
25        }
26      }
27    }
28  }
29
30 }

```

Рисунок 3.11 – Реалізація Shipping Address запитів

Методи сервісу, які працюють з моделлю адреси доставки використовуючи mongoose: отримання колекції, створення нового адреси та оновлення.

Метод отримання колекції: знаходить всі документи у базі даних, які відносяться до користувача, який робить запит та повертає масив перероблених даних завдяки DTO шаблону.

Метод створення: завдяки вхідним параметрам, а саме ідентифікатор користувача та дані адреси, створює екземпляр моделі адреси доставки, використовуючи метод моделі “create”.

Метод оновлення: має вхідний об’єкт, який може містити оновленні дані адреси, далі завдяки методу “findByIdAndUpdate” у моделі адреси, знаходиться адрес, який клієнт змінив та оновлюю його у базі даних. Також це запит у моделі повертає вже оновлений об’єкт адреси.

3.1.4 Сутність Furniture

Сутність Furniture, основна модель даних у системі, так як це продукт, який продається. Користувач не може змінювати меблі, в майбутньому можна додати admin panel, де адміністратор зможе додавати меблі через вебзастосунок.

Схема має тільки запити типу Query та тип самої модель. Тип меблі містить такі дані: id, name, cost, description, image. Цей тип повертається в обох випадках виклику запитів Query від Graph QL.

Сутність має такі запити для отримання інформації: отримання колекції та одного елемента за ідентифікатором екземпляру. Це можна побачити на рисунку 3.12.

```

3  export const furnitureSchema = buildSchema(`#graphql
4  ... type Furniture {
5  ...     id: ID!
6  ...     name: String!
7  ...     cost: Int!
8  ...     description: String!
9  ...     image: String!
10 ... }
11
12 ... type Query {
13 ...     furnitures: [Furniture]
14 ...     furniture(id: ID!): Furniture
15 ... }
16 `);

```

Рисунок 3.12 – Типи Furniture у Graph QL схемі

Реалізація цих методів знаходиться у об'єкті з назвою `furnitureResolvers` (рис. 3.13). В ньому було описано роботу Query запитів, також методи мають перевірку на існування користувача з помилкою, якщо його не знайдено та повернення роботи сервісу, що можна побачити на рисунку(рис. 3.13).

Сервіс роботи з моделлю меблі у базі даних має тільки два легких метода: отримання одного екземпляру по ідентифікатору та колекції елементів.

Метод отримання колекції знаходить всі екземпляри модель `FurnitureModel`, потім використовуючи шаблон DTO форматує масив елементів та повертає на рівень, де викликали метод.

Метод отримання по ідентифікатору – знаходить елемент по вхідному параметру, завдяки методу моделі “`findById`” та використовуючи клас шаблону DTO повертає один екземпляр.

```

5  export const furnitureResolvers = {
6    Query: {
7      furnitures: async (root, args, { isAuth }) => {
8        try {
9          if (!isAuth) {
10             throw new GraphQLError('User is not authenticated');
11          }
12          return furnitureService.getList();
13        } catch (error) {
14          return error;
15        }
16      },
17      furniture: async (root, { id }, { isAuth }) => {
18        try {
19          if (!isAuth) {
20             throw new GraphQLError('User is not authenticated');
21          }
22          return furnitureService.getFurniture(id);
23        } catch (error) {
24          return error;
25        }
26      }
27    }
28  }

```

Рисунок 3.13 – Реалізація Furniture запитів

3.1.5 Сутність Basket

Сутність Basket надає можливість використовувати CRUD операції, завдяки типам запитів Graph QL.

Схема має один тип даних BasketItem, який запити повертають, цей тип має такі параметри: id, productName, quantity, productCost, productDescription.

Єдиним Query запитом для цієї моделі даних є отримання колекції продуктів у кошику з їх кількості, тобто скільки користувач хоче замовити.

Можливість змінювати, видаляти меблі з корзини визначенні як Mutation запити, бо вони оновлюють моделі у базі даних. Типи мутацій для сутності Basket: додати до кошику, збільшити та зменшити кількість продуктів одного типу, а також видалити з кошику. Як можна побачити на рисунку 3.14, більшість мутацій отримують ідентифікатор елементу кошику, щоб змінити саме цей документ у базі даних.

```

3  export const basketSchema = buildSchema(`#graphql
4      type BasketItem {
5          id: ID!
6          productName: String!
7          quantity: Int!
8          productCost: Int!
9          productDescription: String!
10     }
11
12     type Query {
13         basket: [BasketItem]
14     }
15
16     type Mutation {
17         addToBasket(furnitureId: ID!, quantity: Int): BasketItem
18         incrementBasketItem(basketItemId: ID): ID
19         decrementBasketItem(basketItemId: ID): ID
20         deleteBasketItem(basketItemId: ID): ID
21     }
22 `);

```

Рисунок 3.14 – Типи Basket у Graph QL схемі

Реалізація цих запитів міститься у об'єкті `basketResolvers`, в ньому використовується конструкцію `try/catch` для обробки помилок, також наявна перевірка, що користувач існує. Якщо користувач існує, викликається необхідний метод з сервісу для кошику, де зосереджена логіка роботи з моделлю кошика.

В свою чергу сервіс кошику має методи для отримання, видалення, створення та оновлення елементів кошику, для зручності, було виділено екземпляр кошику в окрему модель та зв'язав її з моделлю кошика. Тобто при виклику методів кошику, потрібно звертатись до методів сервісу екземпляра кошика, який працює з документами бази даних.

Сервіс елементу кошика має головні методи роботи з моделлю: створення, збільшення та зменшення кількості на один, отримання колекції та видалення. При створенні екземпляру, метод отримує такі параметри: ідентифікатор кошику, щоб зв'язати кошик з елементом, ідентифікатор продукту та кількість, яка по замовчуванню дорівнює одному. Використовуючи метод “`findOneAndUpdate`” моделі з бібліотеки `mongoose`, знаходиться елемент та збільшується його кількість, а якщо такого елемента

немає, то він створюється, це можна контролювати завдяки прапору, який потрібно вмістити в об'єкт параметрів запиту до бази даних. Інші методи працюють у звичайному стилі, знаходять елемент по ідентифікатору та оновлюють його чи видаляють з кошику.

3.1.6 Сутність Order

Це остання сутність, що надає можливість контролювати замовлення та слідкувати за ними. Завдяки запитам Graph QL користувач може створити та переглянути статус всіх замовлень, у майбутньому можна буде створити застосунок для кур'єра чи адміністратора, щоб він міг змінювати статус продуктів.

Graph QL схема має тип замовлення, який містить такі параметри: `id`, `status`, `createdAt`, `quantity`, `totalAmount`. Цей тип використовується, як значення, яке повертають запити `Query` або `Mutation`. Можна побачити на рисунку 3.15, що запит, який повертає колекцію елементів замовлення, використовується як `Query`. Мутацією замовлення є створення нового екземпляру моделі, як вхідні параметри воно приймає ідентифікатори методу оплати та адресу користувача.

```

3  export const orderSchema = buildSchema(`#graphql
4    type Order {
5      id: ID!
6      status: String!
7      createdAt: String!
8      quantity: Int!
9      totalAmount: Int!
10   }
11
12   type Query {
13     orders: [Order!]
14   }
15
16   type Mutation {
17     createOrder(paymentId: ID!, shippingAddressId: ID!): Order!
18   }
19 `);

```

Рисунок 3.15 – Типи Order у Graph QL схемі

Реалізація Graph QL запитів знаходиться у окремому об'єкті `orderResolvers`. Використовуючи конструкцію `try/catch`, щоб відловити помилки. Перевіряється чи користувач авторизований, при наявності профілю у базі даних викликається методи сервісу, які працюють з замовленнями. В цьому методі зосереджена бізнес логіка, а саме: створення нового екземпляру моделі `OrderModel` та методу `create`, завдяки іншим сервісам, підраховується кількість продуктів у кошику та загальну суму замовлення, з відсотком за доставку, який становить 5% від суми. Іншим методом, знаходиться всі замовлення, які користувач має, з допомогою його ідентифікатору.

3.2 Основні рішення щодо розробки інтерфейсу

Для розробки повноцінної комерційної системи з кросплатформним застосунком потрібна робота у команді, тому що створити самотійно проєкт дуже складно та це процес займе багато часу. Мають бути UI/UX дизайнери, тестувальники, додаткові розробники для застосунків та швидкого розгортання програм у магазині. Тому було обрано за основу шаблон (дизайн) інтерфейсу, автором якого є [@brunodesign](#). За вимогами ліцензії потрібно вказати автора. Також як посилання, надається ліцензія та сам дизайн [32] - [33].

3.2.1 Основні рішення щодо використання додаткових бібліотек

Java Script – мова, яка має велику підтримку з спільноти, тому створювати щось з нуля не завжди вигідне та швидке рішення для розробки будь-якого Java Script проєкту. В цьому випадку всі бібліотеки, які було використано знаходяться у файлі `package.json` (рис. 3.16). Найголовніші з них:

- react-navigation – дозволяє створити гарну та зрозумілу навігацію у застосунку, завдяки бібліотеці розробник створює сторінки у застосунку, прийнято називати скріном;
- Apollo Client – зручний інструмент для роботи з Graph QL запитами, а також підключення до серверу та контроль кешу. Надає інструменти для створення Query, Mutation, Subscription;
- react-hook-form – надає зручний набір інструментів для контролю поле вводу тексту чи форми. Наявність валідації допомагає контролювати поля без зайвої логіки;
- react-native-keyboard-aware-scroll-view – бібліотека, яка створює відображення, яке можна листати залежно від клавіатури смартфона, це потрібно для фокусу на поле вводу, якщо воно знаходиться під клавіатурою;
- react-native-async-storage – бібліотека надає можливість зберігати дані у телефоні як кеш, тому що при повному закритті застосунку всі дані зникають. При видаленні програми ці дані теж зникнуть;
- typescript – зручний інструмент чи повноцінна мова, яка створена на основі JavaScript, але надає можливість визначити типи, створювати інтерфейси та переліки. Ці інструменти необхідні при створенні великих проєктів, бо підтримка без типізації буде дуже важкою.

```

12  "dependencies": {
13    "@apollo/client": "3.6.6",
14    "@react-native-async-storage/async-storage": "1.17.6",
15    "@react-native-community/checkbox": "^0.5.15",
16    "@react-navigation/bottom-tabs": "6.3.1",
17    "@react-navigation/native": "6.0.10",
18    "@react-navigation/native-stack": "6.6.2",
19    "eslint-plugin-import": "^2.26.0",
20    "graphql": "16.5.0",
21    "react": "17.0.2",
22    "react-hook-form": "7.31.3",
23    "react-native": "0.68.2",
24    "react-native-keyboard-aware-scroll-view": "^0.9.5",
25    "react-native-safe-area-context": "^4.3.1",
26    "react-native-screens": "^3.13.1"
27  },
28  "devDependencies": {

```

Рисунок 3.16 – Файл в якому міститься перелік бібліотек

3.2.2 Основні рішення з впорядкування елементів інтерфейсу

Компоненти, які повторюються на різних екранах, було виділено в окремий файл, щоб можна було використовувати декілька разів, React дозволяє розробляти з компонентним підходом. Наприклад:

- кнопка, на кожному екрані вона має однакові стилі (рис. 3.17), але різний текст, тому було зроблено змінним параметром текст і використано кнопку декілька разів;

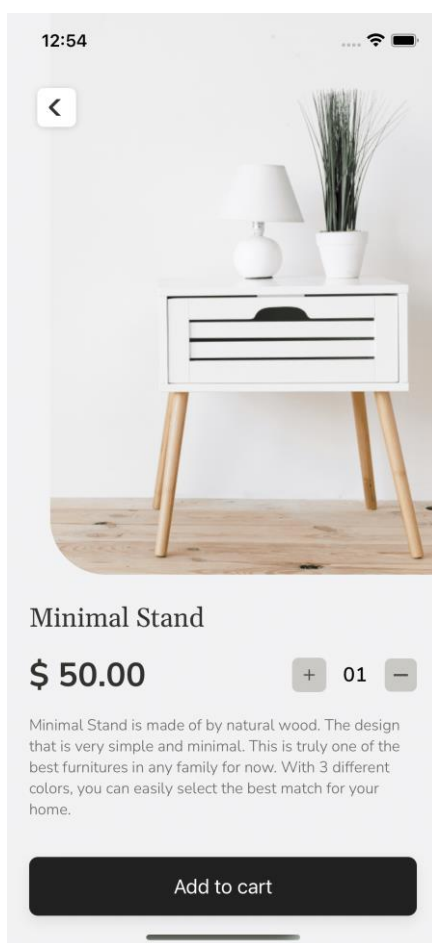


Рисунок 3.17 – Відображення одного товару

- блок з тінями (рис. 3.18);

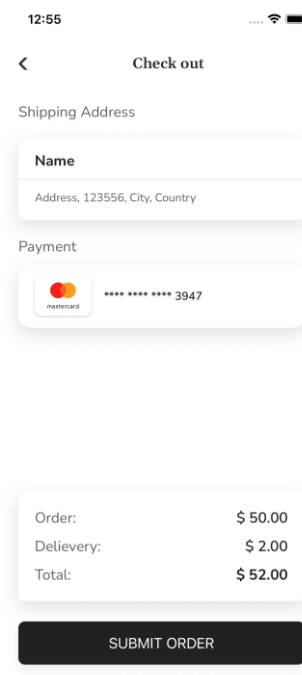


Рисунок 3.18 – Створення замовлення

— поле вводу, яке отримує параметри контролю та значення тексту (рис. 3.19), також можливість виділяти червоним кольором та надавати інформацію про помилку, якщо не правильно введено текст користувачем (рис. 3.20).

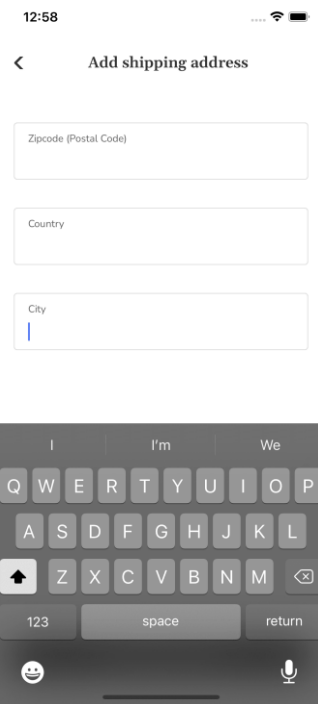


Рисунок 3.19 – Додавання адресу доставки

12:58

< Add payment method

**** * X

Card Holder Name Expire Date
XXXXXX XX/XX

Card Holder Name
Name

Card Number
Ex: 1234 1234 1234 1234

CVV Ex: 123

Expiration Date
05/23

ADD NEW CARD

Рисунок 3.20 – Додавання методу оплати

React Native надає рідні компоненти для обох платформ, тому при створенні будь-якого компоненту в застосунку, було використано базові компоненти з фреймворку:

- View – звичайне відображення, блок, в якому можна розмістити інші компоненти;
- Image – зображення, яке має як параметр посилання на рисунок;
- Text – відображення тексту, який неможливо змінити користувачу (рис. 3.21);

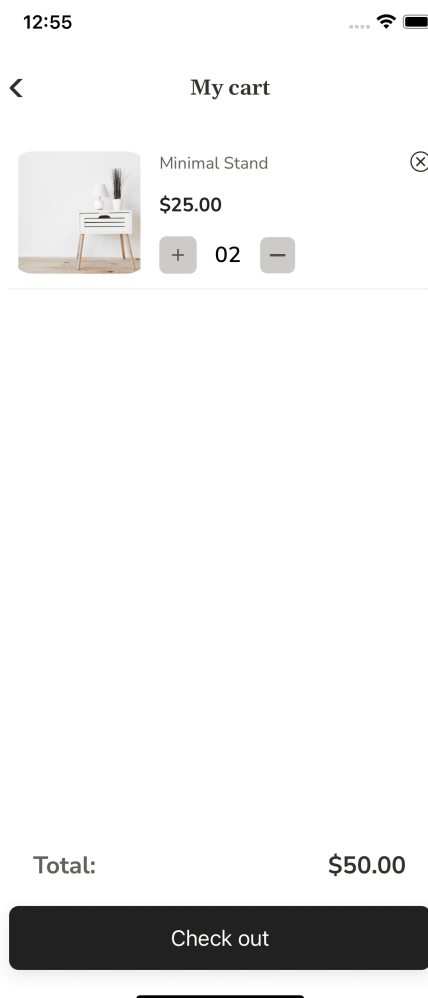


Рисунок 3.21 – Відображення корзини користувача

- `TextInput` – поле вводу тексту;
- `TouchableOpacity` – блок, область з анімацією натискання, компонент (рис. 3.22), у якого найголовніший параметр – що роботи при натисканні;

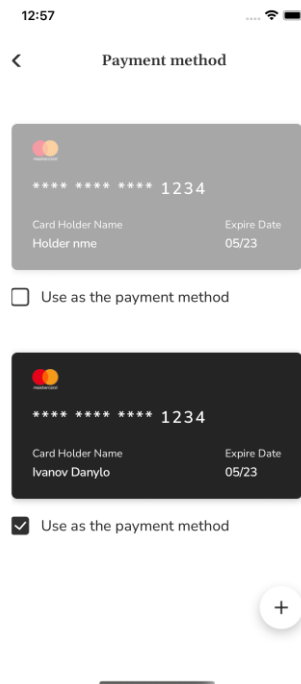


Рисунок 3.22 – Відображення методів оплати користувача

– FlatList – відображення списку елементів, який може прокручуватись (рис. 3.23), але він більш продуктивний чим наступний компонент, продуктивність досягається завдяки відображенню тільки елементів, які бачить користувач, тому цей компонент використовують для створення колекції.

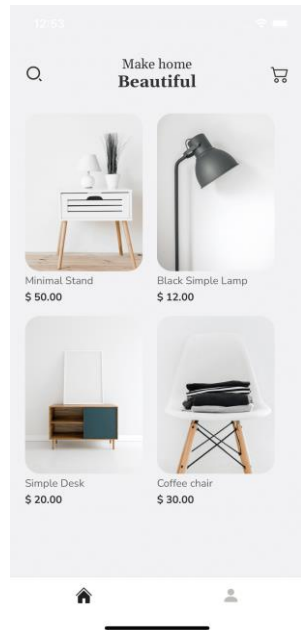


Рисунок 3.23 – Головне меню

– ScrollView – блок, який може буде прокручуватись для відображення великої кількості інформації.

3.2.3 Основні рішення щодо логіки у компонентах інтерфейсу

Розробка та підтримка великих систем потребує використання простої та зрозумілої архітектури, тому було використано MVC підхід та відокремив бізнес логіку від відображення.

Було виділено цілу папку для написання Graph QL запитів для кожної сутності окремо. Так як технологія Graph QL дозволяє отримувати тільки ті дані, які потрібні скрині, було створено декілька запитів з різними даними (рис. 3.24). Для отримання інформації чи реакція на дії користувача у відображенні потрібно лише використати так звані хуки useQuery, useMutation, useLazyQuery, які надає apollo/client.

```
1  import { gql } from '@apollo/client';
2
3  export const GET_FURNITURES = gql`
4    query {
5      furnitures {
6        id
7        name
8        image
9        cost
10     }
11   }
12 `;
13
14 export const GET_FURNITURE = gql`
15   query GetFurniture($id: ID!) {
16     furniture(id: $id) {
17       name
18       id
19       image
20       description
21       cost
22     }
23   }
24 `;
```

Рисунок 3.24 – Файл, в якому міститься запити Graph QL

`useQuery` – приймає схему запиту та конфігурації як параметри, а повертає об'єкт, який містить відповідь серверу з тими даними, які прописані у схемі. Інші поля об'єкту – це дані або інструменти для взаємодії з запитом, наприклад як `error`, `loading`, `refetch`, функція повторно отримання робить запит ще раз, тому що `useQuery` відпрацює тільки коли компонент з'явився у DOM дереві.

`useMutation` – також приймає схему запиту, але повертає масив, перший елемент якого функція, яка робить запит, а другий – це об'єкт даних або інструментів для взаємодії з запитом.

`useLazyQuery` – це ідентичний запит `useQuery`, але він повертає функцію, яка викликає запит до серверу, щоб можна запросити дані на натискання кнопки чи іншим способом.

Також React дозволяє розробляти хуки самостійно, тому якщо у компонентах, де є велика кількість важкої логіки, було відокремлена у хук. Наприклад для контролю полів вводу для реєстрації чи авторизації потрібно було створити багато типів, зазначити поля за замовчування та прописати логіку для подання форми. Вирішив не завантажувати компонент інтерфейсу зайвою інформацією, тому мені потрібно було тільки викликати хук та зробити деструктуризацію об'єкту щоб отримати потрібні мені дані та функції (рис. 3.25).

```
const { control, onSubmit }: UseSignUpReturn = useSignUp();
```

Рисунок 3.25 – Отримання даних та функції контролю

3.3 Основні рішення щодо розробки БД

Використовував зручний інструмент для створення БД `mongoose` в оточенні Node JS. Він допомагає зручну прописувати типи даних та зв'язки між ними. Як можна побачити на рисунку 3.26, було створено схему, де зазначаю типи даних, значення за замовчуванням, необхідність поля та

перелік значень. Завдяки параметру `ref` створюється зв'язок між цією моделлю та іншою. Схема замовлення має зв'язок з моделлю користувача, методом оплати та адресом. Таким чином, можна отримати дані цих моделей у схемі замовлення, якщо потрібно. Останньою строчкою було створено модель, називається `Order` та присвоюється їй схема замовлення.

```

1  import mongoose from 'mongoose';
2
3  const orderSchema = mongoose.Schema({
4    ...userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
5    ...paymentId: { type: mongoose.Schema.Types.ObjectId, ref: 'Payment', required: true },
6    ...shippingAddressId: { type: mongoose.Schema.Types.ObjectId, ref: 'Shipping-Address', required: true },
7    ...totalAmount: { type: Number, required: true },
8    ...quantity: { type: Number, required: true },
9    ...status: { type: String, enum: ["Pending", "Delievered"], default: 'Pending' }
10  }, {
11    ...timestamps: true
12  });
13
14  export const OrderModel = mongoose.model('Order', orderSchema);

```

Рисунок 3.26 – Створення моделі БД

Більшість колекцій залежить від колекції `User`, бо вона є найголовнішою. Рішення від'єднати декілька моделей від схеми користувач, тобто ці моделі мають окремі таблиці, потрібно для полегшення логіки та відокремлення непотрібної інформації. Наприклад в моделі замовлення, розробнику потрібна тільки інформація про метод оплати, але не потрібна інша інформація про користувача, тому було використано лише ідентифікатор профілю, щоб знайти методи, якими володіє користувач. На рисунку 3.27 продемонстрована візуалізація розробленої БД.

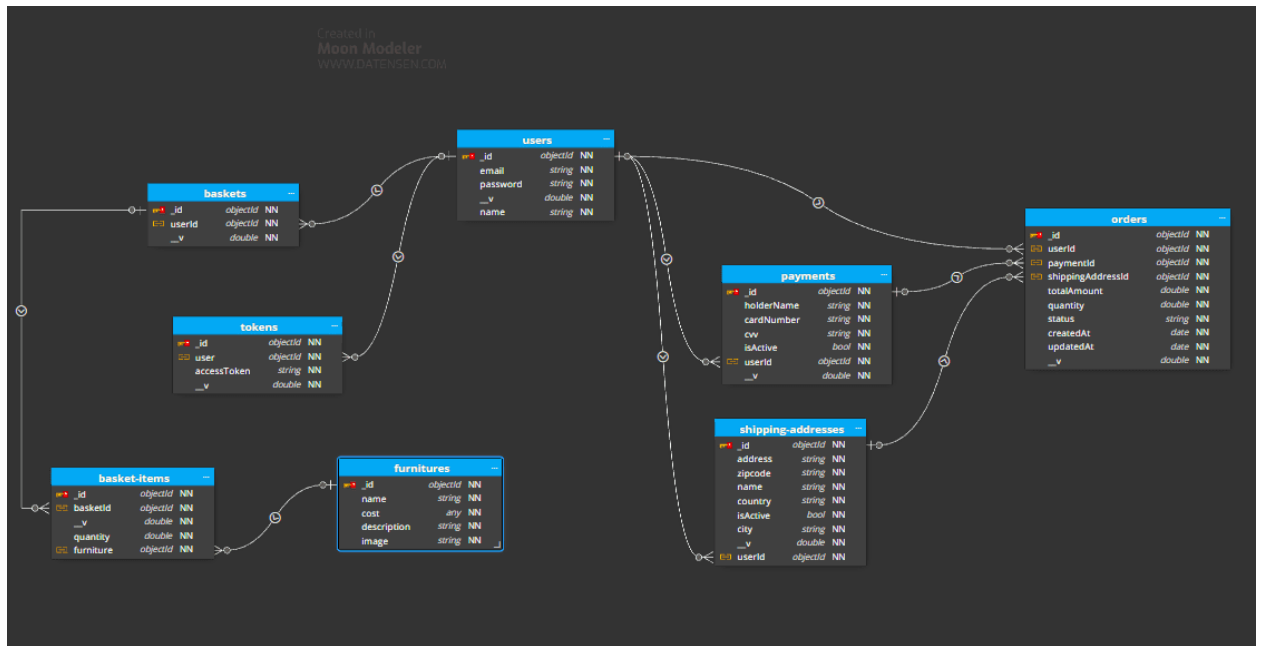


Рисунок 3.27 – Розроблена БД

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Призначення та умови застосування програми

Система призначена для покупки або перегляду широкого асортименту меблевих гарнітур для дому. При цьому користувач може вибрати колір та модель товару. Завдяки використанню сучасної кросплатформної технології React Native, користувач може робити покупки з свого телефону, для цього не потрібна присутність у реальному магазині, а також інші корисні функції:

- надання інформацію про каталог з різноманітними видами меблів, тобто користувач може переглядати опис, фотографії з товарами, колір та розміри.
- можливість створити профіль користувача та використовувати його на декількох телефонах;
- можливість покупати товари онлайн;
- можливість замовити доставки до адреси;
- управління адресами доставки;
- управління методами оплати;
- оплачувати замовлення у застосунку;
- легкий спосіб стежити за замовленнями через застосунок.

4.1.1 Умови застосування

Умови застосування системою при наявності Android: версія ОС 4.0 та більше, підключення до інтернету вільних 77 МБ пам'яті,

Умови застосування системою при наявності iOS платформи: версія ОС 11.0 та більше, підключення до інтернету, вільних 70 МБ пам'яті на пристрої.

4.2 Експлуатація програмного забезпечення

4.2.1 Розгортання проєкту

Розгортання проєкту у магазин застосунків потребує коштів для Play Market(Android) 25\$ та App Store(iOS) 99\$, тому було вирішено не додавати програмне забезпечення до платформ для завантаження, але в документації RN є поетапна інструкція, що потрібно зробити щоб розгорнути застосунок до магазинів [34] – [35].

Розгортання сервер застосунку не вимагає багато коштів або дій, потрібно тільки запустити сервер завдяки командам:

- `npm install` – потрібна для завантаження додаткових пакетів або бібліотек;
- `node index.js` – ця команда запускає головний файл серверу, це вхідна точка в програму.

4.2.2 Звертання до програми

Щоб користуватися застосунком треба завантажити apk файл для Android пристроїв та відкрити його або ipk для iPhone, цей спосіб вважають не безпечний для вашого телефона, бо магазини надають гарантії та перевіряють застосунки на шкідливе ПО перед додаванням. Також є можливість тестувати програмне забезпечення через режим розробки.

Після того як застосунок встановився, його потрібно запустити натискаючи на іконку (рис. 4.1). Після запуску програму зробить перевірку підключення до інтернету та запит до серверу, який поверне інформацію про користувача, якщо він авторизований у застосунку.

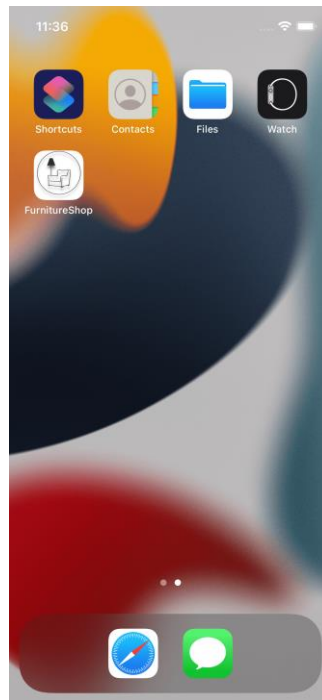


Рисунок 4.1 – Робочий стіл телефона з іконкою застосунка

Якщо користувач неавторизований, то він бачить гарну вступну частину застосунка при вході, з маленьким описом програми (рис. 4.2). Щоб продовжити потрібно натиснути на кнопку “Get Started”.

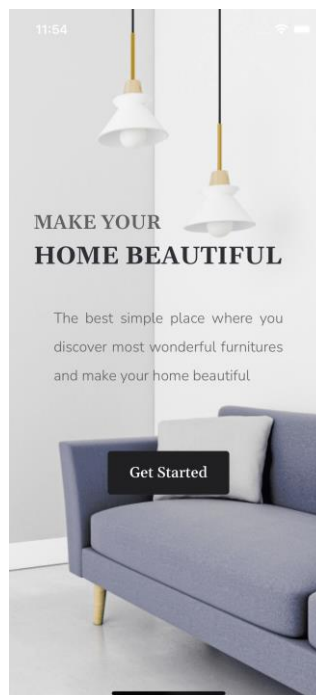


Рисунок 4.2 – Вступний скрін у застосунку

Після натискання користувач перейде до скріна, де він зможе авторизуватись або зареєструватись (рис. 4.3). Після успішної реєстрації програма перейде до головного меню з пошуком меблів та їх списком.

12:04

WELCOME

Name

Name

Email

Myemail@gmail.com

Password

.....

Confirm Password

.....|

SIGN UP

Already have account? SIGN IN

Рисунок 4.3 – Реєстрація профіля

4.3.3 Вхідні та вихідні дані

Вхідними даними до програмного забезпечення мобільного застосунку з покупки меблевих гарнітур є каталог меблів з характеристиками, цінами картинками, дії користувач, які можуть робити запити до серверу (натискання кнопок, введенні дані).

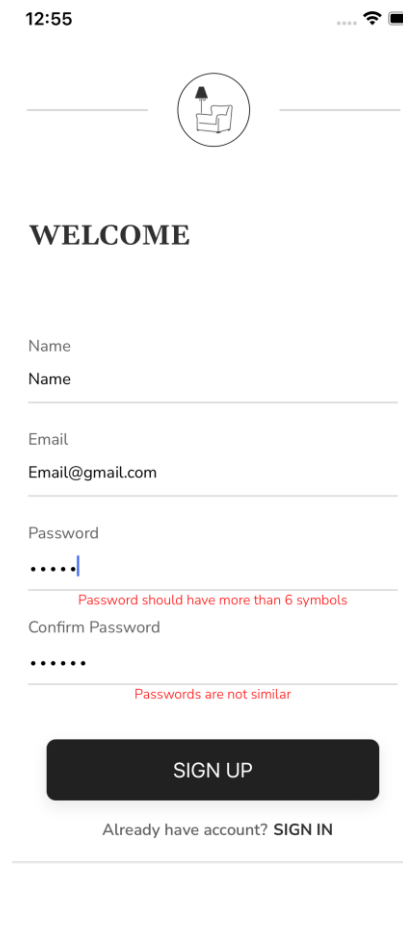
Вихідними даними до програмного забезпечення магазину меблевих гарнітур є зміна розробленого інтерфейсу на дії користувача, повідомлення про успішне замовлення продукту, додавання методу оплати або адресу доставки.

4.3.4 Повідомлення

Користувач може отримати повідомлення різними способами, завдяки рідному компоненту Alert у React Native, створенню модального вікна з інформацією та виділенням якогось поля для введення.

Було використано видалення з окресленням помилки, яку користувач зробив при введенні, як можна побачити на рис. 4.4, після спроби зареєструватись під полями паролем та його підтвердження виникли помилки.

Також іншим способом було використано рідний елемент повідомлення для обох платформ (рис. 4.5), ці помилки вже приходять з серверу завдяки конструкції try/catch. В мобільному застосунку вже було вирішено як обробляти такі помилки. Також є можливість на використати такі повідомлення на успішні операції, наприклад для створення методу оплати.



The screenshot shows a mobile app interface for user registration. At the top, the status bar displays the time 12:55, signal strength, Wi-Fi, and battery icons. Below the status bar is a header with a circular icon containing a house and a chair. The main heading is "WELCOME". The registration form includes fields for Name, Email, Password, and Confirm Password. The Name field is empty. The Email field contains "Email@gmail.com". The Password field contains "....." and has a red error message below it: "Password should have more than 6 symbols". The Confirm Password field contains "....." and has a red error message below it: "Passwords are not similar". At the bottom of the form is a dark blue button labeled "SIGN UP". Below the button is a link: "Already have account? SIGN IN".

Рисунок 4.4 – Реєстрація профіля з помилками

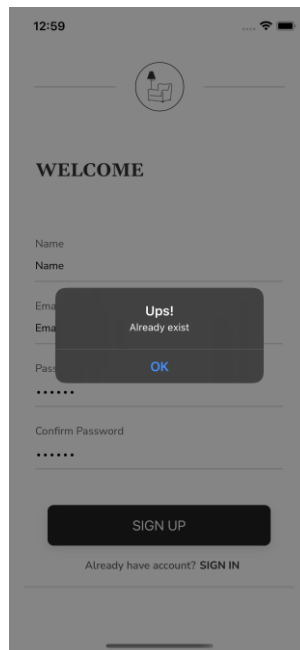


Рисунок 4.5 – Реєстрація профіля з повідомленням про помилку

4.3 Тестування системи

Тестування мобільного застосунку для Android та iOS операційних систем потребує перевірки основних функцій одночасно на обох платформах, бо поведінка UI може змінюватись на цих ОС. В ході тестування було перевірено такий функціонал на реальних телефонах з використанням різних ОС:

- реєстрація та авторизація профіля;
- відображення каталогу меблевих гарнітур;
- функції пошуку меблів;
- додавання, відображення та змінення методів оплати;
- додавання, відображення та змінення адресів доставки;
- додавання, видалення та змінення товарів у корзині;
- вихід з профілю користувача;
- створення замовлення;
- перегляд статусу та інформацію замовлення.

Тестування запитів до серверу перевірено завдяки вбудованому інструменту (рис. 4.6), який надає можливості документації та тестування

усіх Query та Mutation запитів, які були створені у схемі серверного застосунку. При використанні платформи для тестування було вибрано різні параметри та отримані дані.

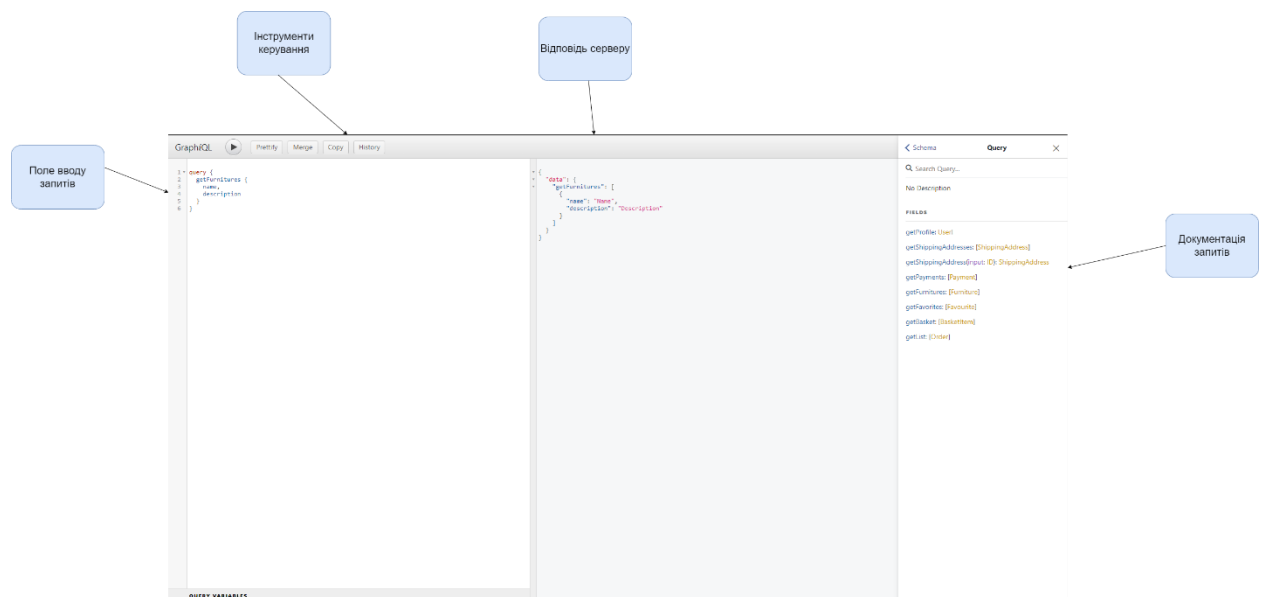


Рисунок 4.6 – Інструмент для тестування Graph QL запитів

4.4 Висновки з розділу

Було описано призначення та основні функції програмного забезпечення для покупки меблевих гарнітур. При тестуванні застосунку було зрозуміло, що потрібно якось оптимізувати процес, бо перевірка застосунку на обох платформах займає багато часу. В такому випадку при збільшенні функція та команди, було проаналізовано що, потрібно розробляти автоматичне тестування для цього проєкту. Тестування головних функцій було пройдено успішно, тому програма може розгортатись до мобільних платформ для завантаження.

У ході тестування API було використаний корисний інструмент з документацією та викликом запитів. Він надається у Graph QL бібліотеці, також його можна вимкнути, щоб користувачі не могли робити запити без мобільного застосунку.

ВИСНОВКИ

В ході виконання дипломної кваліфікаційної роботи бакалавра було проаналізовано та досліджено процес розробки кросплатформного застосунка для покупки меблевих гарнітур.

Проаналізовано процес створення мобільних застосунків для Android та iOS операційних систем та проблеми, з якими можна зустрітись при розробці: довге тестування обох пристроїв та пошук рішень окремо для обох платформ. Також розглянуто інші варіанти створення кросплатформного застосунку. Після аналізу інших фреймворків можна зробити висновок, що кожний варіант використовується у сучасності, але має свої мінуси та плюси, тому перед розробкою кожного застосунку потрібно зробити важливий вибір, який вимагає чіткого вимог щодо функцій та платформи для використання.

Для створення мобільного застосунку було обрано фреймворк React Native та мову програмування Java Script, бо вони легкі для старту та зручні для розробки таких маленьких систем. Також значним критерієм вибору було використання однієї мови для мобільного та серверного застосунку.

При розробці було використано сучасні підходи розробки програмного забезпечення щодо проектування систем, які можна з легкістю підтримувати. Розроблено адаптивний до платформи інтерфейс користувача, з використанням самостійно створених компонентів з рідних елементів технології React Native. Розроблені функція навігації, яка дозволяє переходити між скрінами з анімацією, та запити до серверу з використанням Graph Query Language. Тобто користувач може отримати тільки ті дані, які необхідні на цьому елементу навігації.

У ході розробки серверної частини були розроблені схеми для не реляційної бази даних Mongo DB та Query, Mutation запитів, використовуючи Graph QL.

Виконано тестування системи використовуючи реальні пристрої та не знайдено помилок у логіки та відображенні інтерфейсу в програмі. Завдяки

інструментам Graph QL, при розробці створена документація запитів до серверу, також вони були протестовані на цій платформі. Таким чином програма готова до розповсюдження завдяки платформам для завантаження.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Розробка мобільного додатку [Електрон. ресурс]. – Режим доступу: <https://web-systems.solutions/blog/rozrobka-mobilnogo-dodatku/>
2. Розробка мобільного застосунку від А до Я [Електрон. ресурс]. – Режим доступу: <https://dan-it.com.ua/uk/blog/rozrobka-mobilnih-dodatkiv-vid-a-do-ja-povnij-gajd/>
3. A curated list of open source React Native apps, who is using React Native? [Electronic resource]. – Access mode: <https://reactnative.dev/showcase/>
4. Puma. Про цей додаток [Електрон. ресурс]. – Режим доступу: <https://play.google.com/store/apps/details?id=com.puma.ecom.app&hl=uk>
5. Bolt Food. Про цей додаток [Електрон. ресурс]. – Режим доступу: <https://play.google.com/store/apps/details?id=com.bolt.deliveryclient&hl=uk>
6. IKEA [Electronic resource]. – Access Mode: <https://play.google.com/store/apps/details?id=com.ingka.ikea.app&hl=en&gl=US>
7. Фреймворки у веб-розробці – що це, які існують і для чого потрібні [Електрон. ресурс]. – Режим доступу: <https://highload.today/uk/frejmworki-u-veb-rozrobtsi-shho-tse-yaki-isnuyut-i-dlya-chogo-potribni/>
8. Фреймворк [Електрон. ресурс]. – Режим доступу: <https://tonyline.com.ua/glossary/framework/>
9. Розробка мобільних додатків. Technologies Flutter. [Електрон. ресурс]. – Режим доступу: <https://brander.ua/technologies/flutter>
10. Flutter apps in production [Electronic resource]. – Access mode: <https://flutter.dev/showcase>
11. iOS App Architecture [Electronic resource]. – Access mode: <https://learn.microsoft.com/uk-ua/xamarin/ios/internals/architecture>
12. Розробка мобільних додатків на Xamarin: гармонія усіх платформ [Електрон. ресурс]. – Режим доступу: <https://internetdevels.ua/blog/xamarin-mobile-app-development>

13. Top 6 Popular Apps Built with Xamarin [Electronic resource]. – Access mode: <https://infostride.com/apps-built-with-xamarin/>
14. React Navigation. Routing and navigation for Expo and React Native apps [Electronic resource]. – Access mode: <https://reactnavigation.org/>
15. React Native Gesture Handler. Declarative API exposing platform native touch and gesture system to React Native [Electronic resource]. – Access mode: <https://docs.swmansion.com/react-native-gesture-handler/>
16. React Native Skia. High Performance 2D Graphics [Electronic resource]. - Access mode: <https://shopify.github.io/react-native-skia/>
17. Built-in React Hooks [Electronic resource]. – Access mode: <https://react.dev/reference/react>
18. Двигун V8 – як він працює? [Електрон. ресурс]. – Режим доступу: <https://www.ux-republic.com/uk/%D0%B4%D0%B2%D0%B8%D0%B3%D1%83%D0%BD-v8%2C-%D1%8F%D0%BA-%D0%B2%D1%96%D0%BD-%D0%BF%D1%80%D0%B0%D1%86%D1%8E%D1%94/>
19. Різниця між HTTP та HTTPS [Електрон. ресурс]. – Режим доступу: <https://hostiq.ua/wiki/ukr/http-https/>
20. Що ж це таке GraphQL [Електрон. ресурс]. – Режим доступу: <https://cryptoaxel.medium.com/%D1%89%D0%BE-%D0%B6-%D1%86%D0%B5-%D1%82%D0%B0%D0%BA%D0%B5-graphql-5ea333add9e9>
21. GraphQL для початківців [Електрон. ресурс]. – Режим доступу: <https://devzone.org.ua/post/graphql-dlya-pochatkivtsiv>
22. Introduction to Apollo Client [Electronic resource]. – Access mode: <https://www.apollographql.com/docs/react/>
23. Розробка вебсайтів. MVC [Електрон. ресурс]. – Режим доступу: <https://uk.education-wiki.com/3886982-what-is-mvc>
24. Що таке MVC? [Електрон. ресурс]. – Режим доступу: <https://uk.education-wiki.com/3886982-what-is-mvc>

25. Програма Model-View-Controller MVC – що це, особливості і опис [Електрон ресурс]. – Режим доступу: <https://hi-news.pp.ua/kompyuteri/14628-programa-model-view-controller-mvc-scho-ce-osoblivost-opis.html>

26. What is MongoDB? [Electronic resource]. – Access mode: https://aws.amazon.com/documentdb/what-is-mongodb/?nc1=h_ls

27. Вступ до Mongoose для Mongo DB та Node.js [Електрон. ресурс]. – Режим доступу: <https://code.tutsplus.com/uk/articles/an-introduction-to-mongoose-for-mongodb-and-nodejs--cms-29527>

28. MongoDB & Mongoose: Compatibility and Comparison [Electronic resource]. – Access mode: <https://www.mongodb.com/developer/languages/javascript/mongoose-versus-nodejs-driver/>

29. Про Swift [Електрон. ресурс]. – Режим доступу: https://book.swift.org.ua/book/0_welcome_to_swift/about_swift/

30. Kotlin зараз є найкращою мовою для розробників Android [Електрон. ресурс]. – Режим доступу: <https://www.linuxadictos.com/uk/kotlin-%D0%B7%D0%B0%D1%80%D0%B0%D0%B7-%D1%94-%D0%BD%D0%B0%D0%B9%D0%BA%D1%80%D0%B0%D1%89%D0%BE%D1%8E-%D0%BC%D0%BE%D0%B2%D0%BE%D1%8E-%D0%B4%D0%BB%D1%8F-%D1%80%D0%BE%D0%B7%D1%80%D0%BE%D0%B1%D0%BD%D0%B8%D0%BA%D1%96%D0%B2-Android.html>

31. Що таке мова програмування Python [Електрон. ресурс]. – Режим доступу: <https://freehost.com.ua/ukr/faq/wiki/chto-takoe-jazik-programmirovaniya-python/>

32. A tour of the C# language [Electronic resource]. – Access mode: <https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/>

33. Furniture Shopping – Minimal UI Kit [Electronic resource]. – Access mode: <https://www.figma.com/community/file/1005321051941014567>

34. Зазначення Авторства 4.0 Міжнародна (CC BY 4.0) [Електрон. ресурс]. – Режим доступу: <https://creativecommons.org/licenses/by/4.0/deed.uk>
35. Publishing to Google Play Store [Electronic resource]. – Access mode: <https://reactnative.dev/docs/signed-apk-android>
36. Publishing to Apple App Store [Electronic resource]. – Access mode: <https://reactnative.dev/docs/publishing-to-app-store>

Додаток А
Текст програми

A.1.1 basket-item.service

```
import { BasketItemDto } from '../dto/basket-item.dto.js';
import { BasketItemModel } from '../models/basket-item.model.js'

class BasketItemService {
  createItem = async (basketId, furnitureId, quantity = 1) => {
    const basketItem = await BasketItemModel.findOneAndUpdate(
      { basketId, furnitureId },
      { $inc: { quantity } },
      { new: true, upsert: true }
    );

    return BasketItemDto.parse(basketItem);
  };

  incrementQuantity = async (id) => {
    await BasketItemModel.findByIdAndUpdate(id, { $inc: { quantity: 1 }
  });
    return id;
  }

  decrementQuantity = async (id) => {
    const basketItem = await BasketItemModel.findById(id);
    basketItem.quantity -= 1;

    if (basketItem.quantity <= 0) {
      return this.deleteItem(basketItem._id);
    }

    await basketItem.save();
    return id;
  }

  getList = async (basketId) => {
    const list = await BasketItemModel.find({ basketId
  }).populate('furniture').exec();
    return list.map(BasketItemDto.parse);
  };

  deleteItem = async (id) => {
    await BasketItemModel.findByIdAndDelete(id);
    return id;
  };
}

export const basketItemService = new BasketItemService();
```

A.1.2 basket.service

```
import { GraphQLError } from "graphql";

import { BasketModel } from "../models/basket.model.js";
import { basketItemService } from "../basket-item.service.js";

class BasketService {
  getBasketList = async (userId) => {
    if (!userId) {
      throw new GraphQLError('Does not exist');
    }
  }
}
```

```

    }
    const basket = await BasketModel.findOne({ userId });

    return basketItemService.getList(basket._id);
  }

  addToBasket = async (userId, furnitureId, quantity) => {
    if (!userId) {
      throw new GraphQLError('Does not exist');
    }

    const basket = await BasketModel.findOneAndUpdate({ userId }, {
      userId }, { new: true, upsert: true, });
    await basketItemService.createItem(basket._id, furnitureId,
      quantity);

    return basket._id;
  }

  incrementItem = (basketItemId) =>
basketItemService.incrementQuantity(basketItemId);

  decrementItem = (basketItemId) =>
basketItemService.decrementQuantity(basketItemId);

  deleteItem = (basketItemId) =>
basketItemService.deleteItem(basketItemId);
}

export const basketService = new BasketService();

```

A.1.3 furniture.service

```

import { FurnitureDto } from '../dto/furniture.dto.js';
import { FurnitureModel } from '../models/furniture.model.js';

class FurnitureService {
  getList = async () => {
    const products = await FurnitureModel.find();

    return products.map(FurnitureDto.parse);
  }

  getFurniture = async (id) => {
    const product = await FurnitureModel.findById(id);

    return FurnitureDto.parse(product);
  };
}

export const furnitureService = new FurnitureService();

```

A.1.4 order.service

```

import { basketService } from './basket.service.js';
import { OrderModel } from '../models/order.model.js';
import { OrderDto } from '../dto/order.dto.js';

class OrderService {
  static delieveryTax = 0.05;

```

```

    createOrder = async (userId, shippingAddressId, paymentId) => {
      const basketList = await basketService.getBasketList(userId);
      const totalAmount = basketList.reduce((prev, current) => prev +
        (current.quantity * current.productCost), 0) * OrderService.delieveryTax;
      const quantity = basketList.reduce((prev, current) => prev +
        current.quantity, 0);

      const order = await OrderModel.create({
        totalAmount,
        quantity,
        userId,
        shippingAddressId,
        paymentId
      });

      return OrderDto.parse(order);
    }

    getList = async (userId) => {
      const orderList = await OrderModel.find({ userId });

      return orderList.map(OrderDto.parse);
    }
  }

export const orderService = new OrderService();

```

A.1.5 payment.service

```

import { GraphQLError } from 'graphql';

import { PaymentDto } from '../dto/payment.dto.js';
import { PaymentModel } from '../models/payment.model.js';

class PaymentService {
  getList = async (userId) => {
    if (!userId) {
      throw new GraphQLError('Does not exist');
    }
    const paymentList = await PaymentModel.find({ userId });
    return paymentList.map(PaymentDto.parse);
  }

  createPaymentMethod = async (userId, input) => {
    if (!userId) {
      throw new GraphQLError('Does not exist');
    }

    const payment = await PaymentModel.create({ userId, ...input });
    return payment._id;
  }

  updatePaymentMethod = async (id, input) => {
    const payment = await PaymentModel.findByIdAndUpdate(id, input, {
      new: true });
    return payment._id;
  }
}

export const paymentService = new PaymentService();

```

A.1.6 shipping-address.service

```
import { GraphQLError } from 'graphql';

import { ShippingAddressModel } from '../models/shipping-address.model.js';
import { ShippingAddressDto } from '../dto/shipping-address.dto.js';

class ShippingAddressService {
  getList = async (userId) => {
    if (!userId) {
      throw new GraphQLError('Does not exist');
    }
    const addresses = await ShippingAddressModel.find({ userId });
    return addresses.map(ShippingAddressDto.parse);
  }

  createAddress = async (userId, input) => {
    if (!userId) {
      throw new GraphQLError('Does not exist');
    }

    const address = await ShippingAddressModel.create({ ...input, userId
  });
  return address._id;
}

  updateAddress = async(userId, input) => {
    if (!userId) {
      throw new GraphQLError('Does not exist');
    }

    const address = await
ShippingAddressModel.findByIdAndUpdate(input.id, input, { new: true });
    return address._id;
  }
}

export const shippingAddressService = new ShippingAddressService();
```

A.1.7 user.service

```
import bcrypt from 'bcrypt';
import { GraphQLError } from 'graphql';

import { UserModel } from '../models/user.model.js';
import { tokenService } from '../token.service.js';
import { UserDto } from '../dto/user.dto.js';
import { shippingAddressService } from "../shipping-address.service.js";
import { paymentService } from "../payment.service.js";

//TODO: create error catcher
class UserService {
  register = async (email, password, name) => {
    const candidate = await UserModel.findOne({ email });

    if (candidate) {
      throw new GraphQLError('Already exist');
    }
  }
}
```

```

        const hashPassword = await bcrypt.hash(password, 3);
        const user = await UserModel.create({ email, password: hashPassword,
name });

        const userDto = new UserDto(user);
        const accessToken = await tokenService.generateToken({ ...userDto });

        return { accessToken, user: userDto }
    }

    login = async (email, password) => {
        const user = await UserModel.findOne({ email });

        if (!user) {
            throw new GraphQLError('Does not exist');
        }

        const isPasswordEquals = await bcrypt.compare(password,
user.password);

        if (!isPasswordEquals) {
            throw new GraphQLError('Password is not right');
        }

        const userDto = new UserDto(user);
        const accessToken = await tokenService.generateToken({ ...userDto });

        return { accessToken, user: userDto }
    }

    getUser = async (id) => {
        const user = await UserModel.findById(id);

        if (!user) {
            throw new GraphQLError('Does not exist');
        }

        const [shippingAddresses, payments] = await
Promise.all([shippingAddressService.getList(id),
paymentService.getList(id)]);
        return {
            email: user.email,
            name: user.name,
            id,
            shippingAddresses,
            addressesCount: shippingAddresses.length,
            payments,
            paymentsCount: payments.length
        };
    }
}

export const userService = new UserService();

```

A.1.8 basket.resolvers

```

import { GraphQLError } from "graphql";

import { basketService } from "../../services/basket.service.js";

export const basketResolvers = {

```

```

Query: {
  getBasket: async (root, args, { isAuth, userId }) => {
    try {
      if (!isAuth) {
        throw new GraphQLError('User is not authenticated');
      }

      const basket = await basketService.getBasketList(userId);
      console.log({ basket });
      return basket;
    } catch (error) {
      return error;
    }
  },
  Mutation: {
    addToBasket: async (root, { furnitureId, quantity }, { isAuth, userId
  }) => {
      try {
        if (!isAuth) {
          throw new GraphQLError('User is not authenticated');
        }

        return basketService.addToBasket(userId, furnitureId,
quantity);
      } catch (error) {
        return error;
      }
    },
    incrementBasketItem: async (root, { basketItemId }, { isAuth }) => {
      try {
        if (!isAuth) {
          throw new GraphQLError('User is not authenticated');
        }

        return basketService.incrementItem(basketItemId);
      } catch (error) {
        return error;
      }
    },
    decrementBasketItem: async (root, { basketItemId }, { isAuth }) => {
      try {
        if (!isAuth) {
          throw new GraphQLError('User is not authenticated');
        }

        return basketService.decrementItem(basketItemId);
      } catch (error) {
        return error;
      }
    },
    deleteBasketItem: async (root, { basketItemId }, { isAuth }) => {
      try {
        if (!isAuth) {
          throw new GraphQLError('User is not authenticated');
        }

        return basketService.deleteItem(basketItemId);
      } catch (error) {
        return error;
      }
    }
  }
}

```

```
    }
  }
}
```

A.1.9 basket.types

```
import { buildSchema } from "graphql";

export const basketSchema = buildSchema(`#graphql
  type BasketItem {
    id:ID!
    productName: String!
    quantity: Int!
    productCost: Int!
    productDescription: String!
  }

  type Query {
    getBasket: [BasketItem]
  }

  type Mutation {
    addToBasket(furnitureId: ID!, quantity: Int): BasketItem
    incrementBasketItem(basketItemId: ID): ID
    decrementBasketItem(basketItemId: ID): ID
    deleteBasketItem(basketItemId: ID): ID
  }
`);
```

A.1.10 user.resolvers

```
import { GraphQLError } from "graphql";

import { userService } from "../../services/user.service.js";

export const userResolvers = {
  Query: {
    profile: async (root, args, { isAuth, userId }) => {
      try {
        if (!isAuth) {
          throw new GraphQLError('User is not authenticated');
        }

        return userService.getUser(userId);
      } catch (error) {
        console.log({error});
        return error;
      }
    },
  },
  Mutation: {
    createUser: async (root, { input }) => {
      try {
        const { email, password, name } = input;
        return await userService.register(email, password, name);
      } catch (error) {
        return error;
      }
    },
    login: async (root, { input }) => {
```

```

        try {
            const { email, password } = input;
            return await userService.login(email, password);
        } catch (error) {
            return error;
        }
    }
}

```

A.1.11 user.types

```

import { buildSchema } from "graphql";

export const userSchema = buildSchema(`#graphql
    type ShippingAddress {
        id: ID!
        name: String!
        address: String!
        zipcode: String
        country: String!
        isActive: Boolean!
        city: String!
    }

    type Payment {
        id: ID!
        holderName: String!
        cardNumber: String!
        isActive: Boolean!
        cvv: String!
    }

    type User {
        id: ID
        email: String
        name: String
        shippingAddresses: [ShippingAddress]
        addressesCount: Int
        payments: [Payment]
        paymentsCount: Int
    }

    type AuthResponse {
        user: User
        accessToken: String
    }

    input AuthInput {
        email: String!
        password: String!
    }

    input RegisterInput {
        email: String!
        password: String!
        name: String!
    }

    type Query {
        profile: User!
    }
`);

```

```

    }

    type Mutation {
      createUser(input: RegisterInput): AuthResponse
      login(input: AuthInput): AuthResponse
    }
  `);

```

A.1.12 basket.resolvers

```

import { GraphQLError } from "graphql";

import { basketService } from "../../services/basket.service.js";

export const basketResolvers = {
  Query: {
    getBasket: async (root, args, { isAuth, userId }) => {
      try {
        if (!isAuth) {
          throw new GraphQLError('User is not authenticated');
        }

        const basket = await basketService.getBasketList(userId);
        console.log({ basket });
        return basket;
      } catch (error) {
        return error;
      }
    },
  },
  Mutation: {
    addToBasket: async (root, { furnitureId, quantity }, { isAuth, userId
  }) => {
      try {
        if (!isAuth) {
          throw new GraphQLError('User is not authenticated');
        }

        return basketService.addToBasket(userId, furnitureId,
quantity);
      } catch (error) {
        return error;
      }
    },
    incrementBasketItem: async (root, { basketItemId }, { isAuth }) => {
      try {
        if (!isAuth) {
          throw new GraphQLError('User is not authenticated');
        }

        return basketService.incrementItem(basketItemId);
      } catch (error) {
        return error;
      }
    },
    decrementBasketItem: async (root, { basketItemId }, { isAuth }) => {
      try {
        if (!isAuth) {
          throw new GraphQLError('User is not authenticated');
        }
      }
    }
  }
};

```

```

        return basketService.decrementItem(basketItemId);
      } catch (error) {
        return error;
      }
    },
    deleteBasketItem: async (root, { basketItemId }, { isAuth }) => {
      try {
        if (!isAuth) {
          throw new GraphQLError('User is not authenticated');
        }

        return basketService.deleteItem(basketItemId);
      } catch (error) {
        return error;
      }
    }
  }
}

```

A.1.13 basket.types

```

import { buildSchema } from "graphql";

export const basketSchema = buildSchema(`#graphql
  type BasketItem {
    id:ID!
    productName: String!
    quantity: Int!
    productCost: Int!
    productDescription: String!
  }

  type Query {
    getBasket: [BasketItem]
  }

  type Mutation {
    addToBasket(furnitureId: ID!, quantity: Int): BasketItem
    incrementBasketItem(basketItemId: ID): ID
    decrementBasketItem(basketItemId: ID): ID
    deleteBasketItem(basketItemId: ID): ID
  }
`);

```

A.1.14 order.model

```

import mongoose from 'mongoose';

const orderSchema = mongoose.Schema({
  userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required:
true },
  paymentId: { type: mongoose.Schema.Types.ObjectId, ref: 'Payment',
required: true },
  shippingAddress: { type: mongoose.Schema.Types.ObjectId, ref: 'Shipping-
Address', required: true },
  totalAmount: { type: Number, required: true },
  quantity: { type: Number, required: true },
  status: { type: String, enum: ["Pending", "Delievered"], default:
'Pending' }
});

```

```

    }, {
      timestamps: true
    });

export const OrderModel = mongoose.model('Order', orderSchema);

```

A.1.15 payment.model

```

import mongoose from 'mongoose';

const Payment = new mongoose.Schema({
  holderName: { type: String, required: true, },
  cardNumber: { type: String, required: true, min: 16, max: 16, trim: true
},
  cvv: { type: String, required: true, min: 3, max: 3 },
  isActive: { type: Boolean, required: true, default: false },
  userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User' }
});

export const PaymentModel = mongoose.model('Payment', Payment);

```

A.1.16 App

```

import React from 'react';
import { NavigationContainer } from '@react-navigation/native';
import { ApolloProvider } from '@apollo/client';
import { AppNavigator } from '@navigation/AppNavigator';
import { client } from '@graphql/client';

import { UserProvider } from 'providers/UserProvider';

export const App = () => (
  <ApolloProvider client={client}>
    <NavigationContainer>
      <UserProvider>
        <AppNavigator />
      </UserProvider>
    </NavigationContainer>
  </ApolloProvider>
);

```

A.1.17 AppNavigator

```

import React from 'react';
import { createNativeStackNavigator } from '@react-navigation/native-stack';

import { Onboarding } from 'features/auth/onboarding';
import { AuthScreen } from '@features/auth/auth_screen';
import { AddressScreen } from 'features/shipping_address/address_list';
import { CreateAddressScreen } from '@features/shipping_address/creating_address';
import { PaymentMethodScreen } from 'features/payment/method_list';
import { useUser } from '@hooks/useUser';
import { FullPageLoader } from '@components/full_page_loader';
import { TabNavigator } from './MainNavigator';
import { AppStackParamList, Routes } from './routes';
import { Header } from 'components/header';
import { CreatePaymentScreen } from 'features/payment/creating_payment';
import { ProductScreen } from 'features/product/product';

```

```

import { CartScreen } from 'features/order/cart';
import { routeNames } from 'constants/route_names';
import { CheckoutScreen } from 'features/order/checkout';
import { SuccessfulOrderScreen } from 'features/order/successful_order';

const Stack = createNativeStackNavigator<AppStackParamList>();

const routesWithHeader = [
  Routes.ShippingAddress,
  Routes.AddShippingAddress,
  Routes.PaymentMethod,
  Routes.AddPaymentMethod,
  Routes.Cart,
  Routes.Checkout,
];

export const AppNavigator = () => {
  const { isSignedIn, loading } = useUser();

  if (loading) {
    return <FullPageLoader />;
  }

  return (
    <Stack.Navigator
      screenOptions={{
        header: ({ route }) => {
          const isRouteInList = routesWithHeader.includes(route.name);
          return isRouteInList ? <Header name={routeNames[route.name]}
isBasketVisible={isRouteInList} isBackButtonVisible={isRouteInList} /> :
null;
        },
      }}
      initialRouteName={isSignedIn ? Routes.TabNavigator : Routes.Onboarding}
    >
      <Stack.Screen component={Onboarding} name={Routes.Onboarding} />
      <Stack.Screen component={AuthScreen} name={Routes.Auth} />
      <Stack.Screen component={TabNavigator} name={Routes.TabNavigator} />
      <Stack.Screen component={AddressScreen} name={Routes.ShippingAddress}
/>
      <Stack.Screen component={CreateAddressScreen}
name={Routes.AddShippingAddress} />
      <Stack.Screen component={PaymentMethodScreen}
name={Routes.PaymentMethod} />
      <Stack.Screen component={CreatePaymentScreen}
name={Routes.AddPaymentMethod} />
      <Stack.Screen component={ProductScreen} name={Routes.Product} />
      <Stack.Screen component={CartScreen} name={Routes.Cart} />
      <Stack.Screen component={CheckoutScreen} name={Routes.Checkout} />
      <Stack.Screen component={SuccessfulOrderScreen}
name={Routes.SuccessfulOrder} />
    </Stack.Navigator>
  );
};

```

A.1.18 MainNavigator

```

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

```

```

import { HomeScreen } from 'features/product/home';
import { ProfileScreen } from 'features/profile';
import { MainTabParamList, Routes } from './routes';
import { TabIcon } from 'components/tab_icon';
import { Colors } from 'constants/colors';

const BottomTab = createBottomTabNavigator<MainTabParamList>();

export const TabNavigator = () => (
  <BottomTab.Navigator
    screenOptions={({ route }) => ({
      tabBarShowLabel: false,
      tabBarActiveTintColor: Colors.SecondaryBlackBold,
      tabBarInactiveTintColor: Colors.SecondaryGreyLight,
      tabBarIcon: ({ color }) => <TabIcon name={route.name} color={color} />,
      headerShown: false,
    })
  >
    <BottomTab.Screen name={Routes.Home} component={HomeScreen} />
    <BottomTab.Screen name={Routes.Profile} component={ProfileScreen} />
  </BottomTab.Navigator>
);

```

A.1.19 TokenRepository

```

import AsyncStorage from '@react-native-async-storage/async-storage';
class TokenRepository {
  private static itemName: string = 'TOKEN';
  private token: string | null = '';

  public saveToken = async (token: string) => {
    this.token = token;
    await AsyncStorage.setItem(TokenRepository.itemName, token);
  };

  public loadToken = async () => {
    if (!this.token) {
      const token = await AsyncStorage.getItem(TokenRepository.itemName);
      this.token = token || null;
    }

    return this.token;
  };

  public removeToken = async () => {
    this.token = '';
    await AsyncStorage.setItem(TokenRepository.itemName, '');
  };
}

export const tokenRepository = new TokenRepository();

```

A.1.20 UserProvider

```

import React, { createContext, ReactElement, useCallback, useMemo } from
'react';
import { useQuery } from '@apollo/client';
import { NavigationProp, useNavigation } from '@react-navigation/native';

```

```

import { User } from 'models/user/user';
import { CHECK_AUTHORIZED_PROFILE } from 'graphql/user/queries';
import { tokenRepository } from 'repositories/TokenRepository';
import { AppStackParamList, Routes } from 'navigation/routes';

export interface UserStore {
  isSignedIn: boolean;
  loading: boolean;
  logout: () => Promise<void>;
}

interface Props {
  children: ReactElement;
}

export const UserContext = createContext<UserStore>({} as UserStore);

export const UserProvider: React.FC<Props> = (props) => {
  const { loading, data } = useQuery<{ profile: User }>(CHECK_AUTHORIZED_PROFILE);
  const navigation = useNavigation<NavigationProp<AppStackParamList>>();

  const logout = useCallback(async () => {
    await tokenRepository.removeToken();
    navigation.reset({ index: 0, routes: [{ name: Routes.Auth }] });
  }, [navigation]);

  const value = useMemo(() => ({ isSignedIn: !!data?.profile, loading, logout }), [data, loading, logout]);

  return <UserContext.Provider value={value} {...props} />;
};

```

A.1.21 Button

```

import React from 'react';
import { ActivityIndicator, StyleProp, Text, TouchableOpacity, TouchableOpacityProps, ViewStyle } from 'react-native';

import { Fonts } from 'constants/fonts';
import { Colors } from 'constants/colors';

import { styles } from './style';

export enum RoundedType {
  Little = 'Little',
  Medium = 'Medium',
}

export const borderRadiusMap = {
  [RoundedType.Little]: 4,
  [RoundedType.Medium]: 8,
};

interface Props extends TouchableOpacityProps {
  roundedType?: RoundedType;
  text: string;
  fontFamily?: Fonts;
  isLoading?: boolean;
}

```

```

export const Button: React.FC<Props> = ({ roundedType, text, fontFamily,
style, isLoading, ...props }: Props) => {
  const borderRadius = roundedType ? borderRadiusMap[roundedType] :
borderRadiusMap[RoundedType.Little];
  return (
    <TouchableOpacity style={[styles.container, { borderRadius }, style]}
{...props}>
      {isLoading ? <ActivityIndicator size="small" color={Colors.Primary} />
: <Text style={[styles.text, { fontFamily }]}>{text}</Text>}
    </TouchableOpacity>
  );
};

```

A.1.22 button.styles

```

import { StyleSheet } from 'react-native';

import { Colors } from 'constants/colors';
import { Fonts } from 'constants/fonts';

export const styles = StyleSheet.create({
  container: {
    justifyContent: 'center',
    alignItems: 'center',
    paddingVertical: 15,
    width: '100%',
    backgroundColor: Colors.SecondaryBlackBold,
    shadowColor: 'rgba(48, 48, 48, 0.3)',
    shadowOffset: {
      width: 0,
      height: 5,
    },
    shadowOpacity: 0.34,
    shadowRadius: 6.27,

    elevation: 10,
  },
  text: {
    fontSize: 18,
    fontFamily: Fonts.NunitoRegular,
    color: Colors.Primary,
    textAlign: 'center',
    lineHeight: 23,
  },
});

```

A.1.23 CheckboxController

```

import React from 'react';
import { View, Text } from 'react-native';
import CheckBox from '@react-native-community/checkbox';

import { Colors } from 'constants/colors';

import { styles } from './styles';

interface Props {
  isActive: boolean;
  title: string;
}

```

```

    onChange: (value: boolean) => void;
  }

export const CheckboxController: React.FC<Props> = ({ isActive, title,
onChange }) => (
  <View style={styles.controlContainer}>
    <CheckBox
      onChange={onChange}
      value={isActive}
      style={styles.checkboxContainer}
      onCheckColor={Colors.Primary}
      onTintColor={Colors.SecondaryBlack}
      onFillColor={Colors.SecondaryBlack}
      boxType="square"
      tintColor={Colors.SecondaryBlack}
      onAnimationType="one-stroke"
      animationDuration={0.1}
      tintColors={{ true: Colors.SecondaryBlack, false: Colors.SecondaryBlack
    }}
  />
    <Text style={styles.controlTitle}>Use as the {title}</Text>
  </View>
);

```

A.1.24 SignIn

```

import React from 'react';
import { Controller, ControllerFieldState, ControllerRenderProps } from
'react-hook-form';
import { View } from 'react-native';

import { FormValues, FormTypes, useSignIn, UseSignInReturn } from
'./useSignIn';
import { Button, RoundedType } from '@components/button';
import { TransparentButton } from '@components/transparent_button';
import { AuthInput } from 'features/auth/components/input';

import { style } from './style';

interface Props {
  goToSignUp: () => void;
}

interface ControllerProps<T extends FormValues> {
  field: ControllerRenderProps<FormTypes, T>;
  fieldState: ControllerFieldState;
}

export const SignIn: React.FC<Props> = ({ goToSignUp }) => {
  const { control, onSubmit }: UseSignInReturn = useSignIn();

  return (
    <View style={style.container}>
      <Controller
        control={control}
        name={FormValues.Email}
        rules={{ required: 'Email is required' }}
        render={({ field, fieldState: { error } }):
ControllerProps<FormValues.Email>) => (
        <AuthInput value={field.value} onChangeText={field.onChange}
name="Email" keyboardType="email-address" error={error?.message} />

```

```

    })
  />
  <Controller
    control={control}
    name={FormValues.Password}
    rules={{
      required: 'Password is required',
      minLength: { value: 6, message: 'Password should have more than 6
symbols' },
      maxLength: { value: 16, message: 'Password should have less than 16
sybmols' },
    }}
    render={({ field, fieldState: { error } }):
ControllerProps<FormValues.Password>) => (
      <AuthInput value={field.value} onChangeText={field.onChange}
secureTextEntry name="Password" error={error?.message} />
    )}
  />
  <View style={style.button}>
    <Button text="SIGN IN" roundedType={RoundedType.Medium}
onPress={onSubmit} />
    <TransparentButton text="SIGN UP" onPress={goToSignUp} />
  </View>
</View>
);
};

```

A.1.25 useSignIn

```

import { useCallback, useEffect, useMemo } from 'react';
import { Alert } from 'react-native';
import { useMutation } from '@apollo/client';
import { Control, useForm, useFormReturn } from 'react-hook-form';
import { NavigationProp, useNavigation } from '@react-navigation/native';

import { LoginPayload } from '@models/user/login_payload';
import { AuthResponse } from '@models/user/auth_response';
import { tokenRepository } from 'repositories/TokenRepository';
import { MainTabParamList, Routes } from '@navigation/routes';
import { LOGIN } from 'graphql/user/mutations';

export enum FormValues {
  Email = 'email',
  Password = 'password',
}

export interface FormTypes {
  [FormValues.Email]: string;
  [FormValues.Password]: string;
}

export interface UseSignInReturn {
  control: Control<FormTypes>;
  onSubmit: () => void;
}

export const useSignIn: () => UseSignInReturn = () => {
  const [login, { data, error }] = useMutation<{ login: AuthResponse }, {
input: LoginPayload}>(LOGIN);
  const { navigate }: NavigationProp<MainTabParamList> = useNavigation();

```

```

    const { control, handleSubmit }: UseFormReturn<FormTypes> =
useForm<FormTypes>({
  defaultValues: {
    email: '',
    password: '',
  },
});

useEffect(() => {
  const userData = data?.login;
  const errorMessage = error?.message;

  if (errorMessage) {
    Alert.alert('Error', errorMessage);
    return;
  }

  if (userData) {
    tokenRepository.saveToken(userData.accessToken);
    navigate(Routes.TabNavigator);
  }
}, [data, error, navigate]);

const onLogin = useCallback((input: LoginPayload) => login({ variables: {
input } })), [login]);

const onSubmit = useCallback(() => handleSubmit(onLogin)(), [onLogin,
handleSubmit]);

return useMemo(
  () => ({
    onSubmit,
    control,
  }),
  [onSubmit, control]
);
};

```

Додаток В
Слайди презентації

Національний університет "Запорізька Політехніка"
Кафедра програмних засобів

Дипломна кваліфікаційна робота
бакалавра на тему:
РОЗРОБКА КРОСПЛАТФОРМНОГО ЗАСТОСУНКУ ДЛЯ
ПОКУПКИ МЕБЛИВИХ ГАРНІТУР ДЛЯ ДОМУ

Виконав	
Ст. гр. КНТ-119	Данило ІВАНОВ
Керівник	
К.т.н., доцент	Тетяна КАПЛІЄНКО

Рисунок В.1 – Слайд №1

Об'єкт, предмет, мета дослідження

Об'єкт дослідження - кросплатформні програмні засоби для замовлення та покупки товарів.

Предмет дослідження - мобільні програми для замовлення та покупки меблів.

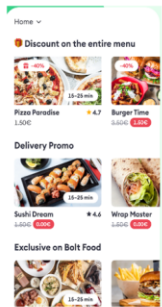
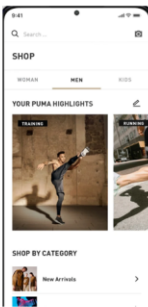
Мета роботи є створення комерційного застосунку з використанням кросплатформної розробки для продажі меблів, який буде інтуїтивно зрозумілим та корисним.

2

Рисунок В.2 – Слайд №2

Приклади мобільних застосунків

Характеристика	Puma SE	Bolt Food	IKEA
Кросплатформність	+	+	-
Продуктивність	Середня	Середня	Висока
Цікаві функції	+	-	+
Розмір	Середній	Середній	Середній
Перегляд товарів	Зручний	Зручний	Не зручний
Пошук товарів	+	-	+
Категорії	+	+	+
Легкий UI	+	-	+



4

Рисунок В.3 – Слайд №3

Порівняння платформ для розробки кросплатформних застосунків

Характеристика	Flutter	Xamarin	React Native
Вартість розробки	Низька	Висока	Низька
Мова програмування	Dart	C#	JavaScript
Платформи	IOS, Android та Web	IOS, Android та Windows	IOS та Android
Найвідоміший приклад використання	Rive, Sua Music, BMW	Olo, UPS	Facebook, Skype, Tesla, Discord
Розмір застосунку	Малий	Великий	Середній
Продуктивність	Висока	Середня	Середня

5

Рисунок В.4 – Слайд №4

Порівняння мов програмування

Характеристика	Python	JavaScript	Swift	Kotlin	Dart	C#
Синтаксис	Лаконічний	Гнучкий	Простий, безпечний	Простий	Схожий на Java	Схожий на C++
Мобільна розробка	-	+	+	+	+	+
Серверна розробка	+	+	+	+	+	+
Тип системи	Динамічна	Динамічна	Статична	Статична	Статична	Статична
Використання	Загальне призначення	Загальне призначення	IOS розробка	Android розробка	Flutter розробка	Загальне призначення
Популярність	Дуже популярний	Надзвичайно популярний, особливо веб	Серед Apple розробників	Серед Android розробників	Популярність зростає	Дуже популярний
Парадигми	ООП, функ.	ООП, функ.	ООП	ООП, функ.	ООП	ООП

6

Рисунок В.5 – Слайд №5

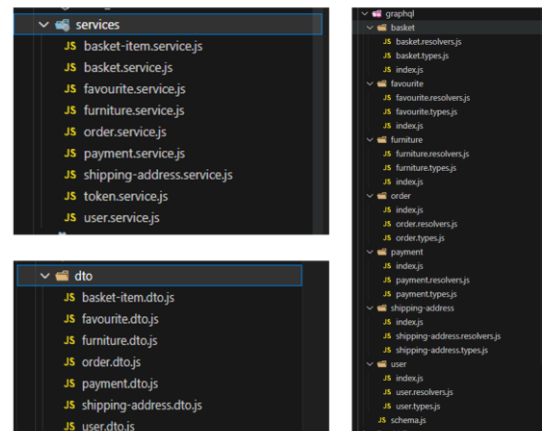
Розроблений серверний застосунок

Використаний стек технологій:

- Node JS;
- Express;
- GraphQL;
- Mongoose.

Допоміжні бібліотеки:

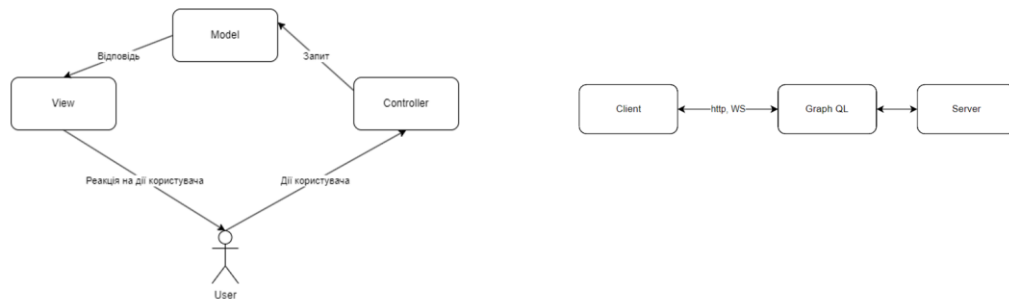
- jsonwebtoken;
- express-graphql;
- graphql-tools.



7

Рисунок В.6 – Слайд №6

MVC – Model, View, Controller



9

Рисунок В.9 – Слайд №9

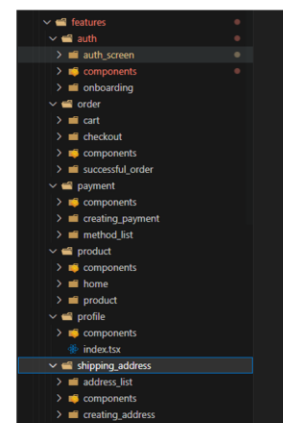
Основні розроблені компоненти мобільного застосунку

Сторінки було розділено на 6 папок за функціями:

- Auth;
- Order;
- Payment;
- Product;
- Profile;
- ShippingAddress.

Основні компоненти, що повторюються на сторінках:

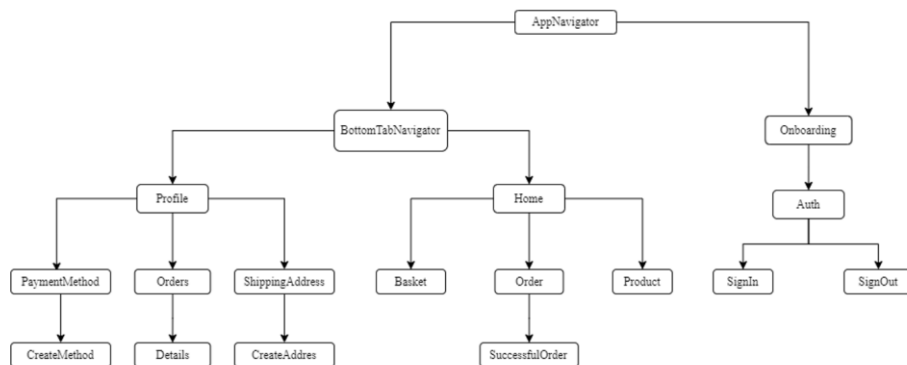
- Button;
- FloatingActionButton;
- Input;
- CheckBox;
- Loader.



10

Рисунок В.10 – Слайд №10

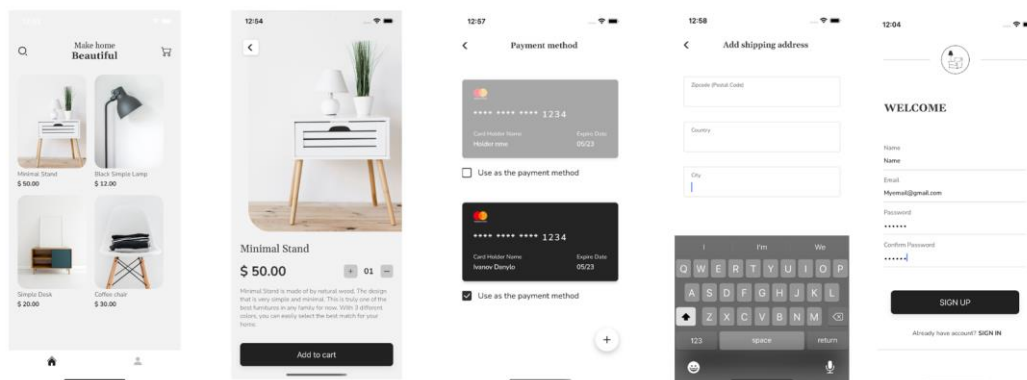
Розроблена навігація між сторінками застосунку



11

Рисунок В.11 – Слайд №11

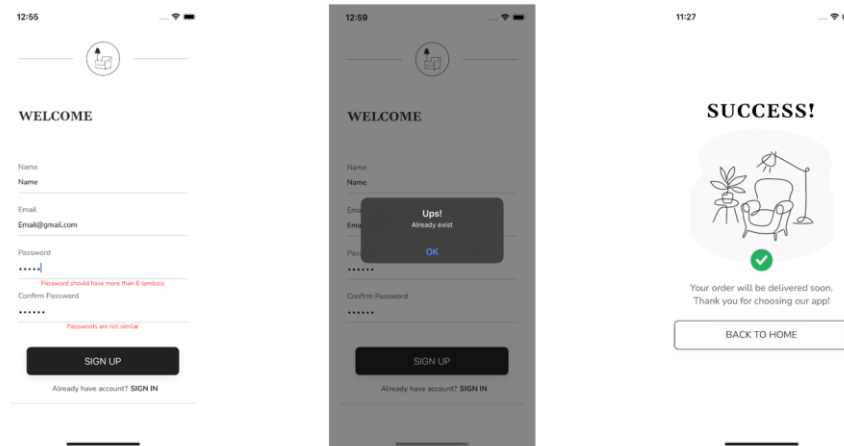
Основні сторінки програми



12

Рисунок В.12 – Слайд №12

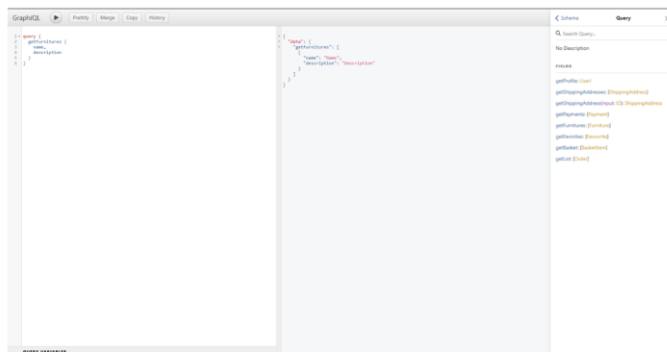
Основні повідомлення у програмі



13

Рисунок В.13 – Слайд №13

Тестування



Пристрої тестування
мобільного застосунку:

- Xiaomi Redmi 5 Plus
- Nexus 5X
- iPhone XR
- iPhone 11
- iPhone 13

Інструмент тестування та документація запитів

14

Рисунок В.14 – Слайд №14

Висновки

- проаналізовано процес створення мобільних застосунків для Android та iOS операційних систем та проблеми, з якими можна зустрітись при розробці. Також розглянуто інші варіанти створення кросплатформного застосунку;
- при розробці було використано сучасні підходи розробки програмного забезпечення щодо конструювання систем, які можна з легкістю підтримувати. Набуто навички створення адаптивного інтерфейсу користувача для різних платформ. Розроблені функція навігації, яка дозволяє переходити між екранами з анімацією;
- Розроблено запити до серверу з використанням Graph Query Language та управління станом застосунку завдяки Apollo Client;
- у ході розробки серверної частини були розроблені схеми для не реляційної бази даних Mongo DB та Query, Mutation запитів, використовуючи Graph QL.

15



Рисунок В.15 – Слайд №15