МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Кваліфікаційна робота магістра

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ ПРОЕКТУВАННЯ МОБІЛЬНИХ СИСТЕМ ЕЛЕКТРОННОЇ КОМЕРЦІЇ

Здобувач вищої освіти гр. ІН.м-13 Дмитрій ГИРЯВЕНКО

Науковий керівник,
кандидат фізико-математичних наук,
асистент кафедри комп'ютерних наук Ольга ШУТИЛЄВА

В.о. завідувача кафедри,
кандидат технічних наук, доцент
доцент кафедри комп'ютерних наук, Ігор ШЕЛЕХОВ

Суми 2022

Сумський державний університет

Факультет *ЕлІТ* *Кафедра* *Комп'ютерних наук*
Спеціальність *«122 – Комп'ютерні науки»*

Затверджую:

в.о. зав. кафедрою _____

_____

"_____" _____2022 р.

ЗАВДАННЯ

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТОВІ

Гирявенко Дмитрія Романовича

1. Тема проєкту (роботи) *Інформаційна технологія проєктування мобільних систем електронної комерції*

затверджую наказом по інституту від "___" _____2022 р. № _____
2. Термін здачі студентом закінченого проєкту (роботи)_____
3. Вхідні данні до проєкту (роботи) _____
_____
_____
_____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) огляд та аналіз існуючих аналогів; 2) формулювання завдань дослідження; 3) огляд технологій для реалізації завдання; 4) проєктування інформаційної системи; 6) програмна реалізація.*

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

6. Консультанти до проєкту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання _____

Керівник _____

(підпис)

Завдання прийняв до виконання _____

(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів дипломного проекту (роботи)	Термін виконання проєкту (роботи)	Примітка
1.	Аналіз поставленої проблеми предметної області, постановка задачі	*17.10.2022*	
2.	Огляд стека технологій, методів, які плануються використовуватися	*28.10.2022*	
3.	Розробка клієнтської та серверної частини мобільного додатку	*01.10.2022*	
4.	Аналіз та тестування отриманої системи	*02.12.2022*	
5.	Оформлення пояснювальної записки до кваліфікаційної магістерської роботи	*10.12.2022*	

Студент-дипломник _____

(підпис)

Керівник проєкту _____

(підпис)

РЕФЕРАТ

Записка: 88 стор., 36 рис., 1 табл., 3 додатки, 13 джерел.

Об’єкт дослідження – актуальність онлайн продажів.

Мета роботи – розробка технології проєктування мобільних систем електронної комерції, що включає проектування інформаційної системи, програмну реалізацію клієнтської та серверної частини, тестування.

Методи дослідження – метод аналітично-статистичний.

Результати – розроблено мобільний додаток, створений на базі мови програмування Java Script з використанням фреймворку React Native та інтеграціями Poster API та Liqpay API.

ІНФОРМАЦІЙНА СИСТЕМА, БАЗА ДАНИХ, REACT NATIVE, СЕРВЕР, КОРИСТУВАЦЬКИЙ ІНТЕРФЕЙС

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

JSON – JavaScript Object Notation

JS – Java Script

POS – Point of sale(точка продажу)

ЦА – Цільова аудиторія

ТЗ – Технічне завдання

MVP – Minimum Viable Product (мінімальний життєздатний продукт)

CRM – Customer Relationship Management (система для управління взаємовідносинами з клієнтами)

SQL – Structured Query Language

ВСТУП

У сучасних умовах кожного дня бізнес є в пошуках нових способів та каналів продажу своїх товарів чи послуг. З недавніх часів мобільні додатки слугують чудовим рішенням, за допомогою якого бізнес може залучати нових клієнтів та мати підтримку вже існуючих.

Однак розробка мобільного додатку достатньо складний, довготривалий та дорогий процес. Від ідеї до реалізації сягає в середньому пів року та коштувати декілька тисяч доларів. В більшості випадків для невеликого бізнеса розробка мобільного додатку є великим ризиком. Бізнес втрачає кошти, не отримуючи очікуваного результату, тому так важливо зменшити ціну розробки мобільного додатку.

На даний час майже не існує рішень на ринку України, яка дасть змогу запустити мобільний додаток за 5-7 днів за невеликі кошти. Зокрема проектів які нададуть якісний користувацький досвід.

Отже, **метою** даної роботи є розробки платформи для запуску мобільних додатків для середнього бізнесу. Це дасть змогу в значній мірі зменшити ціну запуску.

Наукова новизна одержаних результатів полягає в запровадженні нового підходу до запуску мобільних додатків для малого та середнього бізнесу.

Актуальність мобільного додатку полягає в створенні першої white-label платформи для запуску додатків для бізнесу з інтеграцією POS-системи та методів оплати.

Проект має практичну значущість так як відразу ж після запуску може використовуватися бізнесом для свої цілей – створенням нового каналу продажів за допомогою мобільного додатку за невеликою ціною.

1. ЛІТЕРАТУРНИЙ ОГЛЯД

1.1. Аналіз предметної області

Стільниковий телефон не просто апарат для зв'язку, це платіжний засіб, записник, фотоапарат, календар, flash-накопичувач, радіо, навігатор, ліхтарик та багато іншого. У цьому іншому ховаються різні мобільні програми, що тануть у собі багато можливостей для бізнесу. Не варто упускати цю користь [5].

Галузь мобільних додатків активно розвивається і продовжить це робити. За даними аналітичної компанії App Annie, в 2021 році користувач в середньому витрачав 4,8 години на різні мобільні сервіси. Це максимальний показник за період існування цього ринку.

Якщо говорити про скачування, то минулого року за хвилину у світі завантажувалося понад 435 000 додатків. А витрати на це збільшилися на 25% і досягли 135 млрд. доларів за рік. Мобільний додаток дозволяє завжди бути під рукою клієнта, буквально бути доступним 24/7.

Мобільні програми формують імідж. Важко уявити бізнес, який не дбає про свій імідж. Додаток – спосіб його сформувати чи покращити. З власним мобільним додатком бізнес показує конкурентам та клієнтам рівень компанії, а також відповідність до тенденцій сучасного бізнесу. Додаток може стати дзеркалом сайту або способом презентації компанії.

Мобільні додатки підтримують конкурентоспроможність. Вже не лише світові гіганти на кшталт Apple, McDonald's та Facebook мають власні мобільні програми. Навіть прямі конкуренти – компанії, що пропонують схожі товари та послуги – вже напевно пропонують користувачам завантажити їхню програму. Додаток гарантує зручність для користувача, а комфорт – важлива складова бажання людини щось купити в бізнесу.

Мобільні програми формують лояльність до бренду. Наявність мобільного додатку підвищує лояльність до бренду. Вона досягається зокрема за рахунок:

- швидкого доступу до програми;
- використання програми офлайн;
- персоналізації.

За допомогою програми бізнес можете допомогти людині відчути себе частиною бренду бізнесу.

Мобільні програми спрямовані на сервіс. Клієнтський сервіс – складова лояльності до бренду та інструмент збільшення продажів. Додаток повинен піклуватися про зручність користувача. Давши людині отримати потрібну інформацію, товар або послугу в один-два торкання до екрану, бізнес стає для нього зручним способом отримати бажане. Наприклад, сервіс таксі Uklon дозволяє вибрати клас машини, оптимальну вартість поїздки, вказати наявність тварин та підібрати автомобіль за іншими параметрами:

Мобільний додаток безпосередньо взаємодіє з цільовою аудиторією (ЦА). Ті, хто його завантажує, вже знайомі з брендом та зацікавлені у послугах. Бізнесу залишається донести до них інформацію про акції та спеціальні пропозиції. Зменшуючи ланцюжок дій, необхідних для здійснення покупки або замовлення послуги, знижується кількість користувачів, які встигають передумати купувати. Як наслідок, підвищується відсоток конверсії.

Мобільні програми – запорука ефективного маркетингу. Замість білбордів на автострадах можна надсилати клієнтам push-сповіщення. Замість контекстної реклами – використовуйте оголошення всередині програми та splash екрани. З цільовою аудиторією маркетингові та рекламні кампанії покажуть себе значно ефективніше. Більше не доведеться витрачати бюджет: витрачені на рекламу кошти будуть окуповуватись за рахунок взаємодії з знайомими з компанією користувачами.

Мобільний додаток розширює можливості маркетингу. Ви зможете:

- збирати дані та відправляти акції лише зацікавленим людям;
- яскраво презентувати товари;
- влаштовувати інтерактивні активності для підвищення залучення користувачів;

- персоналізувати контент.

Зробивши правильний брендинг, навіть видом програми бізнес спонукає користувача скористатися пропонованими послугами. Правильне просування програми допоможе збільшити кількість його установок.

Мобільні додатки підвищують продаж. Згідно з дослідженням, проведеним Google у 2016 році, 38% опитаних людей регулярно купують за допомогою мобільних пристроїв. В іншому дослідженні, проведеному компанією Adobe, більше половини опитаних заявили, що зручним способом здійснювати інтернет-покупки вони вважають мобільний додаток. Сьогодні мобільний додаток – один з основних інструментів продажу. Відмовляючись від нього, втрачається значна частина цільової аудиторії, яка щоразу повертатиметься за знижками, та акційними пропозиціями.

Мобільні програми, як основа для автоворонки. Додаток – спосіб створення автоворонки продажів. Автоворонка – це шлях клієнта від першого знайомства з брендом до покупки товару або послуги компанії. І чим він коротший, тим вища ймовірність того, що людина здійснить покупку.

Мобільний додаток знижує кількість покупок, що не відбулися за рахунок зручної подачі інформації. Вдало продуманий шлях користувача знижує ризик втратити клієнта на етапах вибору товару та оформлення замовлення. Програма дозволяє уникнути зайвих етапів.

Мобільні програми та зворотний зв'язок. Залишивши у додатку поле для зворотного зв'язку, а також налаштувавши аналітику, бізнес зможе отримати повний відгук від користувачів. Це важливо для вдосконалення самого додатку та вирішення проблем користувачів.

Клієнту важливо відчувати, що дбають про нього. Аналітика допоможе персоналізувати пропозицію для кожного конкретного користувача, а безпосереднє спілкування з ним краще зрозуміти його бажання.

Наприклад, Netflix збирає дані про перегляди користувача, а при реєстрації просить людину вибрати фільми, які їй подобаються. За підсумками цього сервіс будує індивідуальну систему рекомендацій. Любителі комедій бачать фільми в

цьому жанрі вгорі сторінки, поціновувачі кримінальних серіалів можуть одразу перейти до нових детективів і таке інше. Подібна персоналізація позитивно впливає на престиж бренду та лояльність користувачів.

Мобільні програми та хитрощі просування. За допомогою додатку можна перетворити звичайного клієнта на прихильника бренду. Допоможуть у цьому маленькі хитрощі – від квестів усередині програми та до невеликих акцій для постійних клієнтів. Підвищуючи залучення користувача, підвищується ймовірність того, що він здійснить покупку.

Хороший приклад такої хитрості – кейс мобільного додатка Mono Bank, в якому у квітні запустили "квест місяця". Суть квесту в тому, щоб знайти у додатку гру та набрати в ній 100 очок для отримання нагороди. Ці нагороди також вдалий приклад використання хитрощів для підвищення лояльності до бренду. Перетворюючи використання програми на гру, в якій кожен може перемогти, покращуєте ставлення клієнтів до компанії.

Створення мобільного додатка – це комплексний, скрупульозний процес, який передбачає чітке дотримання плану, дотримання термінів та досягнення взаєморозуміння між замовником та розробником [9]. Весь процес від ідеї до реалізації можна розділити кілька ключових етапів:

- **Дослідження ідеї**. На даному етапі потрібно зрозуміти та визначити ідею додатку, які функції воно має містити, які завдання виконувати і для кого.
- **Проектування.** Створення карти, що демонструє функціонал додатку. Як правило, карта містить схему екранів та переходів між ними.
- **Створення дизайну.** Розробка графічних елементів: екранів, фонів, іконок, кнопок та ін. Цей етап передбачає перевірку на зручність використання – потрібно визначити, наскільки зручно користуватиметься елементами інтерфейсу програми.
- **Складання докладного технічного завдання (ТЗ).** Технічне завдання містить докладний опис функціоналу мобільного додатка, бізнес-процеси та основні сценарії, які мають бути в ньому реалізовані. ТЗ складає замовник, або

компанія-розробник на вимогу замовника. Після підготовки ТЗ можна точно оцінити тимчасові та грошові витрати на розробку програми.

- **Створення прототипів.** Прототип потрібен, щоб показати, як працюватиме додаток, він може бути інтерактивним та статичним.
- **Розробка**. Після погодження ТЗ та прототипів із замовником настає етап активної розробки. У комплексних та складних проектах рекомендується застосовувати MVP (minimum viable product) – мінімальний реалізований функціонал продукту, який допомагає в оцінці майбутньої програми з боку замовника та в подальшому плануванні розробки.
- **Тестування.** Після завершення розробки потрібно протестувати фінальну версію на мобільних пристроях. На даному етапі виявляють і виправляють усі недоліки та недопрацювання програми, готують його до повноцінного релізу.
- **Передача клієнту/публікація в Google Play, App Store**. По суті – це фінальний етап розробки, після якого програма переходить у стадію підтримки. Після виявлення та виправлення всіх помилок та фінального узгодження із замовником, програма публікується в Google Play та App Store. Варто звернути увагу на те, що після публікації в даних сервісах, можна вносити зміни до додатку без повторної модерації, що дуже зручно для замовників.
- **Технічна підтримка та моніторинг.** За весь час користування вашим додатком виникатимуть ті чи інші технічні складності, помилки та нештатні ситуації, з якими стикатимуться користувачі. На даному етапі відбувається оперативна підтримка і (якщо потрібно) доопрацювання програми.

Із вище сказаного можна зробити висновок, що процес створення достатньо складний. Також існує ризики при створенні мобільного додатку:

- **Ціна.** Розробка мобільного додатку процес довготривалий, та вимагає підключення спеціалістів, тому ціна може досягати безмежних масштабів. Мобільний додаток для ресторану може коштувати від 2000 до 10 000$. Ціна є достатньо великою та є ймовірність неповернення інвестицій.

•**Терміни реалізації.** Середня тривалість розробки мобільного додатку – від 2 місяців до 6 місяців. Як бачимо, процес є достатньо довготривалим, особливо якщо бізнес перебуває на етапі швидкого масштабування.

•**Код низької якості.** Низька якість коду – одна з найпоширеніших проблем у розробці. Найчастіше замовник не розуміє код і не може визначити його якість. До завершення розробки може з'ясуватися, що програма має неякісно розроблений код, проблеми в роботі та відсутність грамотного тестування.

•**Неправильна оцінка.** Коли складається оцінка проекту, буває, що вона не виправдовує очікувань. Команда може вибрати тривалість ітерації проекту, стек технологій та інші фактори. Між клієнтом та командою часто виникають розбіжності, що призводить до збільшення тривалості завдання, витрат, через які у клієнта закінчуються гроші, і він не може завершити проект [7].

Для бізнесу суттєво важливо отримати рішення для своїх цілей швидко та за оптимальну ціну. Гарним рішенням, яке суттєво зменшить ризики при розробці мобільного додатку, скоротить час від ідеї до готового продукту до 5 раз та за меншу ціну є white-label (WL) підхід – модель співпраці, коли одна компанія виробляє продукт, а інша продає його під своїм брендом.

Цей термін прийшов у підприємництво з музичної індустрії: колись вінілові платівки розсилали на радіо та клуби з білою етикеткою. Пізніше продавці самі наносили дизайн, залежно від реакції слухачів на музику. Сьогодні модель WL поширилася на багато сфер бізнесу — як в офлайн, так і в онлайн, наприклад:

- товари, що мережі супермаркетів продають під власною торговою маркою;
- банківські картки, які більші банки випускають для дрібних;
- добірки купонів або «кешбек» на сайтах;
- партнерські посилання у блогах або на форумах;
- супутні онлайн-послуги на сайтах.

У всіх випадках товари та послуги WL купуються без брендингу. Бізнес самі оформлює продукт у фірмовому стилі, щоб споживачі порівнювали його з його брендом. Саме тому даний підхід почав використовуватись і для розробки

Існує безліч переваг використання white-label підходу до створення та налаштування мобільних бізнес-додатків. Ці мобільні програми вимагають менше часу на створення, оскільки вони мають коротший термін готовності до встановлення. Скорочення часу створення дозволяє компаніям запустити свій мобільний додаток у більш короткий термін, ніж додаток користувача. White-label – це більш бюджетний варіант для бізнесу, оскільки він вимагає менше інвестицій, ніж створення мобільного користувача.

Налаштувавши готовий white-label додаток, компанія може створити власний мобільний додаток за невелику частину вартості. Ці мобільні програми без коду також можуть бути масштабовані, щоб задовольнити конкретні потреби вашого бізнесу. Мобільні програми white-label є кращим рішенням для компаній з обмеженими технологічними та людськими ресурсами. Ці мобільні програми обслуговуються стороннім продавцем, який відповідає за оновлення, підтримку та модернізацію програми white-label.

Додатки white-label часто розміщуються та контролюються на серверах сторонніх розробників. Для компаній, які не є спеціалістами, це мінімізує стрес, пов'язаний з розробкою, створенням та обслуговуванням мобільного додатка користувачів.

1.2. Аналіз існуючих рішень

Існує немало рішень-конструкторів для розробки мобільних додатків та сервісів з white-label підходом до надання послуг розробки, однак рішень які дають повню інтеграцію з українськими сервісів POS-терміналів та методів оплати невелика кількість.

Після огляду та аналізу конкурентних рішень для електронної комерції були розглянуті наступні мобільні сервіси: Appie Appmakr, Mo-Apps, Mobincube

Apps, SalesBox. Всі програмні рішення були проаналізовані по запитах «white-label додаток для бізнесу», «мобільний додаток для бізнесу», «замовити мобільний додаток для бізнесу».

Appie appmakr

Один з найпопулярніших онлайн-конструкторів у світі – понад 7 млн. бізнесів скористалося його послугами. Сервіс підтримує конструктор мобільного додатку та white-label підхід. Сервіс заточений на американський ринок, тому відсутня українська мова. Що важливо, так як сервіс не розрахований на ринок України, то недоступні інтеграції з основними POS-системами та онлайн-сервісами для оплати. Ціна залежить від доступних функцій. Від 300$ до 1000$.

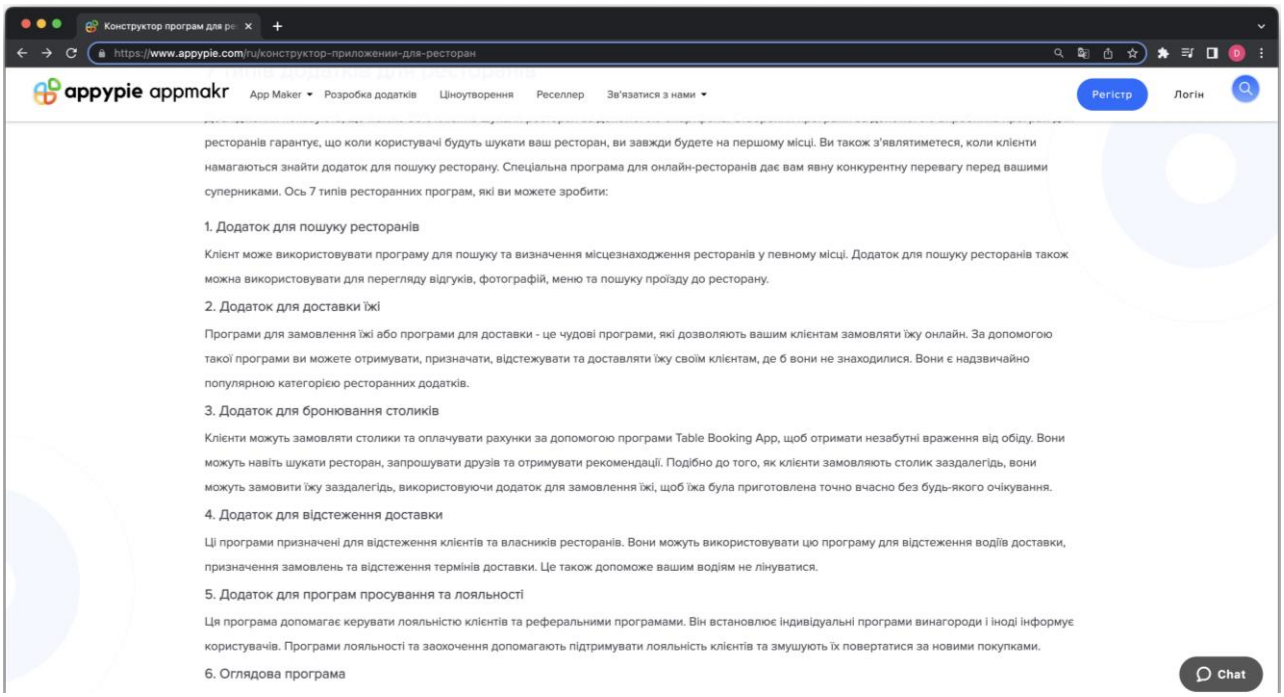

Рисунок 1.1 – Приклад Appie appmakr

Amo-Apps

Український конструктор мобільних додатків для IOS та Android. У ньому можна вибрати тему програми, налаштувати кнопки, категорії, меню і сторінки,

зібрати потрібний функціонал. Залежно від типу програми змінюється кількість доступних модулів.

Фото та відео в Mo-Apps можна завантажувати скільки завгодно. Обмежень немає. А товари заливати просто із CSV- та YML-файлів (спеціальні формати для представлення таблиць). Зміни оновлюються швидко – за кілька хвилин їх уже бачать користувачі. Присутня інтеграція з українськими сервісами оплати такими як LiqPay та Mono Bank Payments.

У Amo-Apps можна створювати програми різними мовами світу, а як виглядатиме програма після збереження, можна побачити в програмі Previewer.

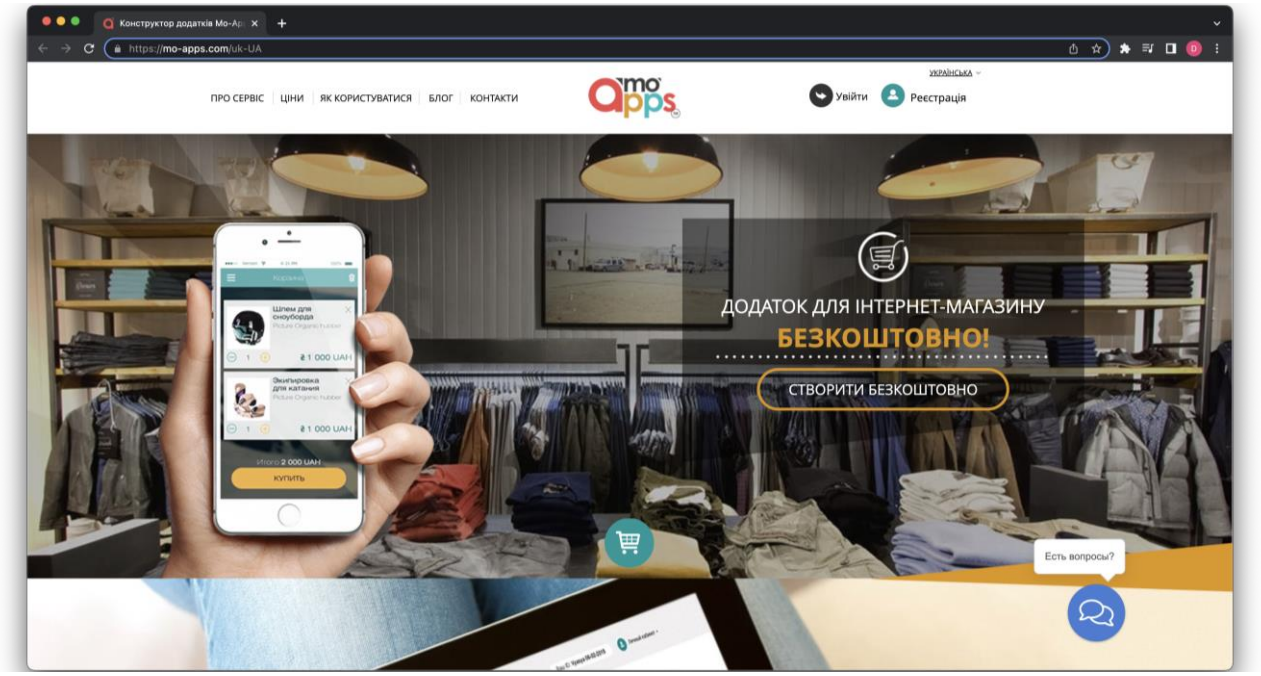

Рисунок 1.2 – Приклад Amo-Apps

Amo-Apps дає широкий спектр функціоналу:

- Можливість заздалегідь подивитися меню;
- Зробити замовлення до приходу у ресторан;
- Замовити доставку додому або в офіс;
- Участь в програмах лояльності;
- Резервування столів на бажаний час.

Mobincube-Apps

Безкоштовно на цій платформі можна створити стільки програм, скільки забажаєте. Це можуть бути сервіси для бізнесу або освіти, для розваги або допомоги по дому.

Те, що відрізняє Mobincube від інших конструкторів – гнучкість параметрів. Шаблони можна редагувати до невпізнання, створюючи свій власний стиль.

Кількість завантажень теж не обмежена. Публікувати можна як на Google Play так і App Store.

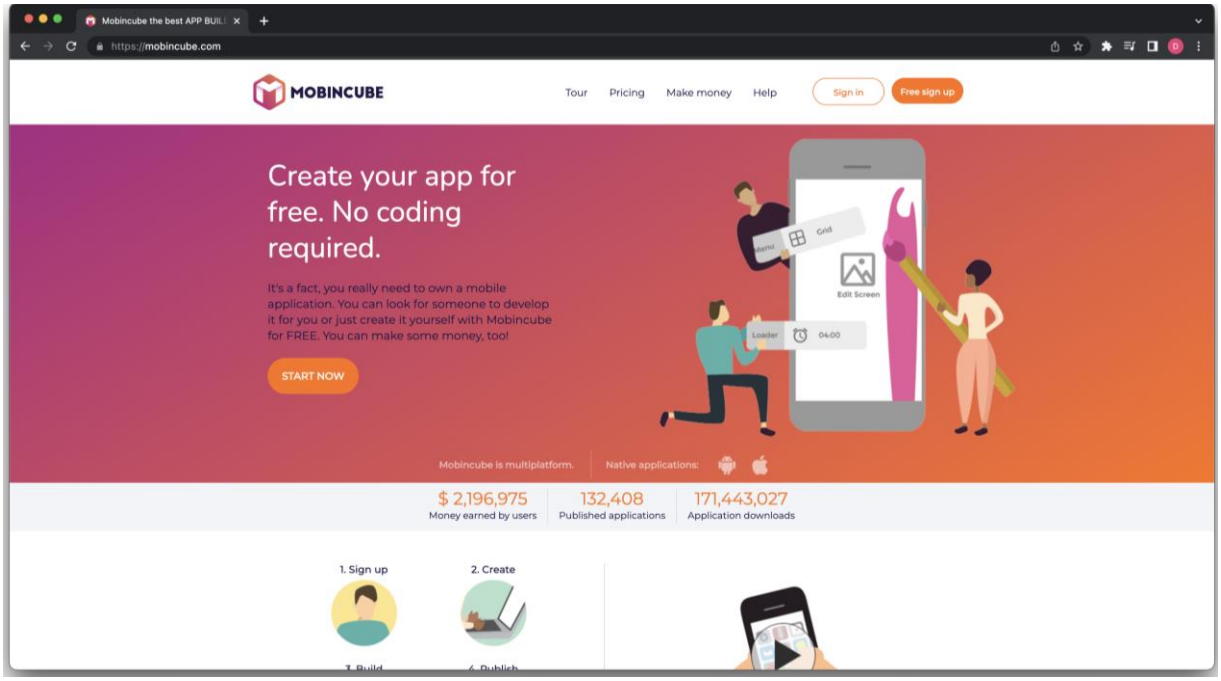

Рисунок 1.3 – Приклад Mobincube-Apps

Заплатити доведеться, коли потрібно зберегти програму на Mobincube, щоб оновлювати в майбутньому. У платних версіях функціонал розширюється. В платному функціоналу можна:

- створити онлайн-магазин;
- розмістити всередині програми оголошення;
- деталізувати дизайн;
- додати спливаючі повідомлення;
- створити необмежену кількість вкладок;

- завантажити великі бази даних;
- розмістити відео та аудіо;
- вказати координати на Google Картах;
- інтегруватись з Google Аналітикою.

SalesBox

Український конструктор мобільних додатків для IOS та Android. У ньому можна вибрати тему програми, налаштувати кнопки, категорії, меню і сторінки, зібрати потрібний функціонал. Великою перевагою SalesBox є комплексний підхід до надання послуг автоматизації бізнесу: сервіс надає власну CRM систему для інтеграцій з додатком. Інтеграції з українськими сервісами оплати такими як LiqPay та MonoBank Payments присутня, але інтеграція з зовнішніми POS-системами відсутня.

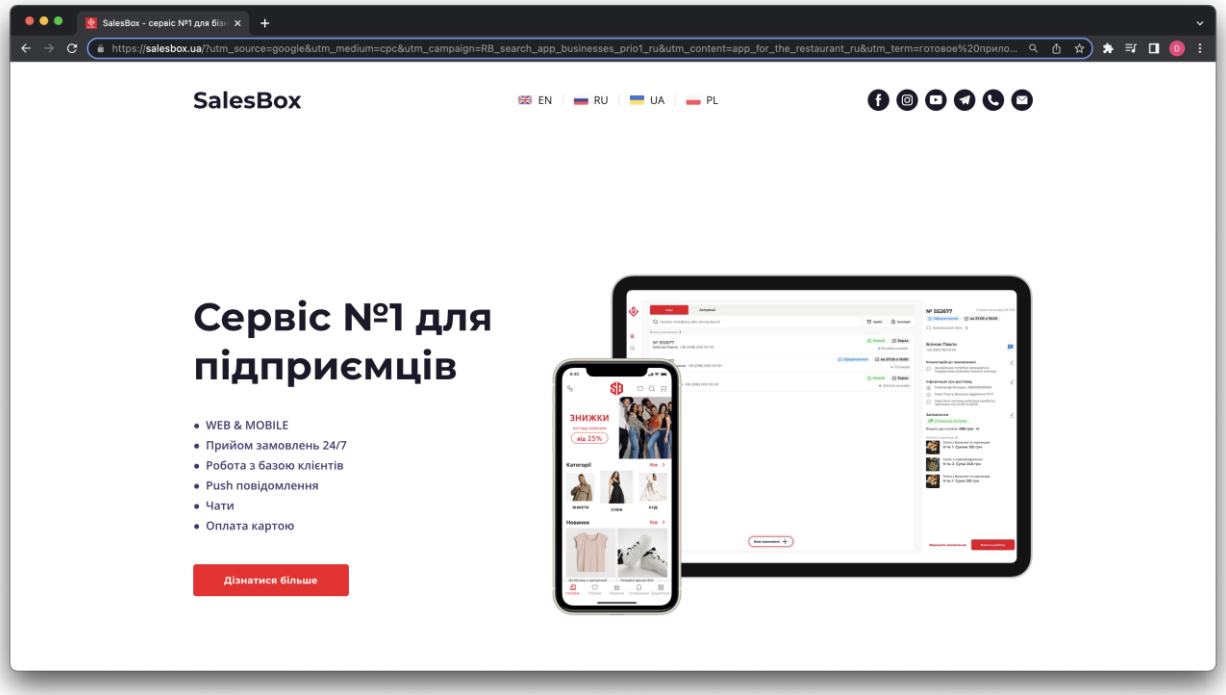

Рисунок 1.4 – Приклад SalesBox

- Перелік користувачів з автоматичним додаванням в базу даних при замовленні;
- зручна робота з замовленнями та створення нових;

- додавання адміністратора і менеджера, які працюватимуть над замовленнями;
- просте створення товарів/послуг та категорій з ними;
- персональне налаштування: знижок, акцій, кешбек.

Було проведено огляд існуючих програмних продуктів, що можуть бути використані для вирішення поставлених задач. Всі сервіси надають можливість досить гнучко налаштовувати інтерфейс мобільного додатку. За функціональністю всі варіанти дуже близькі, з невеликою різницею в підходах. Цінова політика достатньо сильно залежить від функціоналу, який надає той чи інший сервіс.

Однак, проаналізовані сервіси не дають можливості інтегруватися з сервісами оплати доступними в Україні та найпопулярніші POS-системи, що надзвичайно важливо для українського бізнесу. Так, є можливість налаштувати інтерфейс, але шаблони достатньо сильно застарілі та не можуть конкурувати з іншими додатками, які зараз є на ринку.

1.3. Постановка задачі

На основі проведеного аналізу проблеми та предметної області, проаналізувавши актуальність мобільного додатку для бізнесу в сучасних умовах конкуренції, вивчивши основні недоліки під час розробки мобільних додатків та ризики, було сформовано наступні завдання:

1. Спроектувати інформаційну системи мобільного додатку, а саме:

- Діаграму потоків даних;
- Функціональну модель;
- Спроектувати діаграму варіантів використання доданку;
- Побудувати діаграму станів акторів.

2. Змоделювати серверну частину, а саме розробити реляційну модель бази даних.

3. Проаналізувати та порівняти рішення для розробки клієнтської та серверної частин мобільного додатку, для проектування інформаційної системи та прототипування інтерфейсу;

4. Програмно реалізувати:

- Серверну частину мобільного додатку;
- Клієнтську частину мобільного додатку;
- Інтегрувати POS-систему;
- Інтегрувати систему оплати LiqPay.
- Інтегрувати серверну частину.

5. Протестувати на недоліки та недопрацювання додаток

6. Підготувати мобільний додаток до публікації в магазини Google Play та App Store для атрибуції.

2. ВИБІР МЕТОДУ РІШЕННЯ

2.1. Аналіз і порівняння рішень для розробки клієнтської частини

Розробляючи мобільний додаток, варто враховувати кілька факторів: функціональність, адаптивність, вартість, оптимізація. Кросплатформенність дозволяє розробникам писати код відразу під кілька операційних систем. Для бізнесу – це можливість заощадити кошти, швидше запуститися та охопити більший сегмент користувачів.

Кросплатформові додатки – це додатки, які працюють відразу на кількох операційних системах. Завдання програмістів полягає у написанні коду, який добре розгортається на всіх операційних системах.

Універсальний підхід до розробки дозволяє виконати дві важливі умови: економія часу та коштів. Розробники набагато швидше роблять додаток. Вірніше, сама програма розробляється стільки ж, скільки і програми для IOS або для Android. Але якщо бізнес замовляє додаток під різні системи, часу на розробку потрібно вдвічі більше.

Але є й недоліки: кросплатформові додатки не такі гнучкі, як нативні, тому що складно реалізувати всі функції, щоб вони добре працювали на різних пристроях. Магазини додатків мають свої вимоги, їх потрібно враховувати під час розробки. Це створює додатковий дискомфорт та труднощі.

Незважаючи на це, кросплатформові додатки дуже популярні та ефективні. Залежно від сфери бізнесу можна створити унікальні інструменти, з якими користувач взаємодіятиме. Швидкий запуск, широке охоплення аудиторії, порівняно невисока вартість розробки дозволяють швидко реалізувати ідеї, запустити потужні інструменти та діджіталізувати бізнес.

Кросплатформенна розробка додатків проводиться за допомогою фреймворків. Це стек технологій, дозволяють впроваджувати інструменти, створювати необхідний функціонал, вирішальний бізнес завдання. Існує досить

велика кількість фреймворків, проте було виділено найпопулярніші, які активно використовують розробники:

- Flutter
- React Native

Flutter – це кросплатформенний набір засобів розробки, який підтримує Google. З його допомогою фахівці можуть у рекордно короткі терміни створювати сучасні мобільні програми, що одночасно охоплюють операційні системи Android та IOS. При цьому продуктивність проектів, створених за допомогою цієї SDK, практично не поступається нативним.

Реліз Flutter відбувся відносно нещодавно – у 2017 році. Однак, незважаючи на свою молодість, він уже встиг за популярністю свого головного конкурента в особі React Native на таких майданчиках, як GitHub та Stack Overflow. Крім того, за допомогою Flutter були розроблені мобільні програми багатьох великих корпорацій зі світовим ім'ям: Google, BMW, Ebay, Alibaba та інших, що ще раз говорить про спроможність та актуальність технології.

Таке швидке визнання та популярність Flutter отримав не випадково. Він справді надає розробникам потужні інструменти для створення швидких, ефективних та надійних програмних продуктів. Розглянемо основні переваги SDK докладніше.

Таблиця 2.1 – Переваги та недоліки Flutter

Переваги	**Недоліки**
Висока швидкість та продуктивність	API недопрацьований, специфікація може змінюватися досить часто;
Легкість навчання, велика бібліотека документації та активна підтримка.	складна робота з віджетами, часто доводиться окремо розробляти їх за допомогою нативного підходу.
Відкритий вихідний код, що дає можливість розробникам	

використовувати його у своїх проектах абсолютно безкоштовно.	

Отже, головними перевагами Flutter є продуктивність мобільних додатків, створених за допомогою даного фреймоворку, швидкість навчання та відкритий вихідний код. Так, ця технологія має ряд недоліків, але попри це достатньо зручний та потужний інструмент.

React Native – це фреймворк для розробки мобільних додатків, який дає змогу розробляти мультиплатформенні додатки для Android та IOS за допомогою власних елементів інтерфейсу користувача [10]. Він заснований на середовищі виконання JavaScriptCore і трансформаторах Babel. Завдяки такому налаштуванню RN підтримує нові функції JavaScript (ES6+), напр. функції стрілок, async/wait тощо.

Рисунок 2.1 – Популярність RN технології в порівнянні з нативними технологіями розробки

Його перша публічна попередня версія була випущена в січні 2015 року на конференції Reactjs, а в березні 2015 року Facebook зробив React Native відкритим і доступним на GitHub. З тих пір він був широко прийнятий розробниками та організаціями через його здатність створювати рідні програми та чудові інтерфейси користувача. На графіку нижче ви можете візуалізувати

тенденцію зростання React Native. Всього за 1,5 роки після випуску він випередив розвиток Android і IOS.

Фреймворк React Native справді надає розробникам потужні інструменти для створення швидких, ефективних та надійних програмних продуктів [8]. Розглянемо основні переваги:

- **Легко працювати**. Змістовні повідомлення про помилки, економія часу та надійні інструменти роблять її кращим вибором серед інших платформ.
- **Працює скрізь.** Опанувавши React Native, можна створювати програми для IOS, Android і Windows.
- **Легке налагодження.** Зручний інструмент для налагодження Flipper використовується за умовчанням.
- **Спільнота**. Велика спільнота розробників, які роблять свій внесок день у день.
- **Можливість повторного використання коду.** Розробники можуть легко інтегрувати 90% нативного фреймворку та повторно використовувати код для будь-якої платформи. Ця функція не тільки економить час, але й допомагає скоротити витрати на створення двох програм.
- **Попередньо розроблені компоненти.** Численні бібліотеки з відкритим кодом доступні для прискорення вашої роботи.
- **Функція живого перезавантаження.** Допомагає скомпілювати та прочитати файл з того місця, де розробник вніс зміни. Потім новий файл пропонується стимулятору, який автоматично зчитує файл з самого початку.
- **Сумісний із плагінами сторонніх виробників і не потребує великого обсягу пам’яті для обробки.** Жодних спеціальних функцій веб-переглядів не потрібно, а нативні модулі пов’язані з плагіном через фреймворк. Плавна робота та швидке завантаження є його ключовими характеристиками.
- **Більш плавний і швидкий інтерфейс користувача.** Порівняно з класичними гібридними. Програми React Native використовують рідні API для відтворення свого інтерфейсу користувача.

- **Популярність.** React Native є найпопулярнішою технологією розробки кросплатформенних додатків. Це є суттєвою перевагою, так як дає більшу кількість готових плагінів та більша спільнота [11].

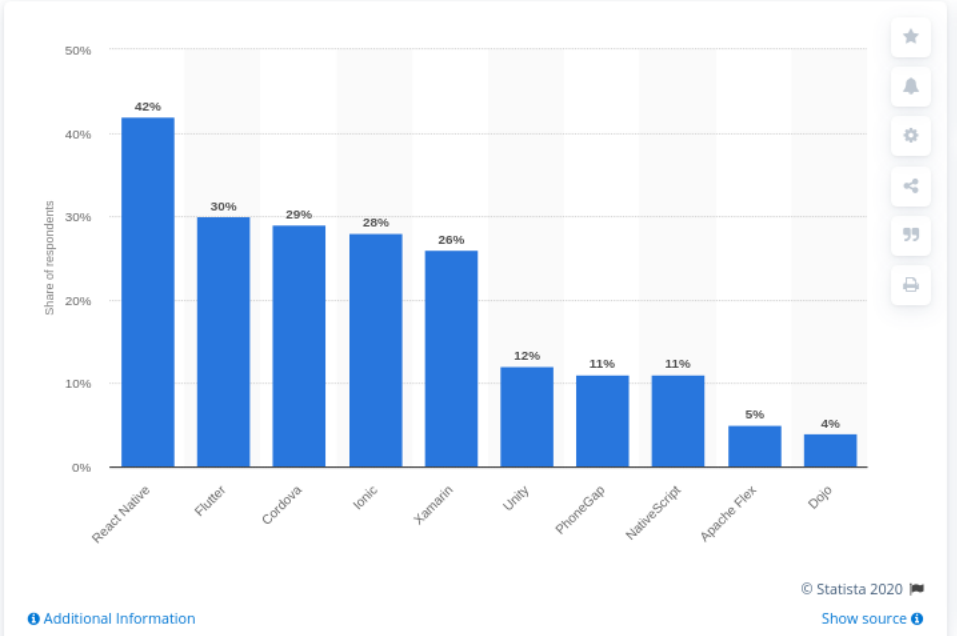

Рисунок 2.2 – Популярність технологій кросплатформенної розробки мобільних додатків

Отже, проаналізувавши всі переваги React Native, саме цю технологію було обрано для реалізації клієнтської частини.

2.2. Аналіз і порівняння рішень для розробки серверної частини

Для серверної частини було обрано технологію PostgreSQL. PostgreSQL – це об’єктно-реляційна система керування базами даних (ORDBMS), заснована на POSTGRES, версія 4.21, розроблена на факультеті комп’ютерних наук Каліфорнійського університету в Берклі. [6]

PostgreSQL започаткував багато концепцій, які стали доступними в деяких комерційних системах баз даних лише набагато пізніше. PostgreSQL підтримує як SQL для реляційних, і JSON для нереляційних запитів. Вона підтримує

розширені типи даних та покращену оптимізацію продуктивності, функції, доступні лише у дорогій комерційній базі даних, такій як Oracle та SQL Server.

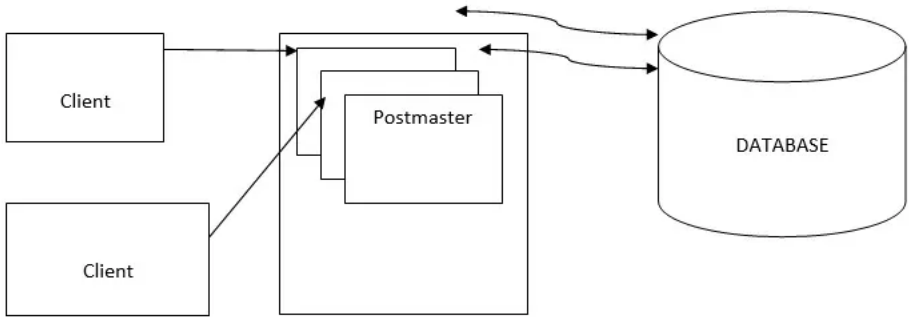

Рисунок 3 – Архітектура PostgreSQL

PostgreSQL має кілька основних переваг, які роблять його дуже привабливим для користувачів: від спільноти розробників з відкритим вихідним кодом до його надійності. Завдяки ліцензії з відкритим вихідним кодом, вихідний код PostgreSQL доступний безкоштовно. Завдяки цьому можна можете використовувати, змінювати та застосовувати його так, як це необхідно. Вам В додаток, для встановлення та використання не потрібно багато інструкцій, тому що він простий у використанні. Базу даних легко підтримувати та адмініструвати як для комплексного, так і для корпоративного використання. Ось деякі з основних переваг, які пропонує база даних:

- PostgreSQL може запускати динамічні веб-сайти та веб-додатки як опцію стека LAMP
- Запис з випередженням запису в PostgreSQL робить її базою даних з високою відмовостійкістю
- Вихідний код PostgreSQL вільно доступний за ліцензією з відкритим вихідним кодом. Це дає вам свободу використовувати, змінювати та впроваджувати його відповідно до потреб вашого бізнесу.
- PostgreSQL підтримує географічні об'єкти, тому ви можете використовувати його для сервісів на основі позиціонування та географічних інформаційних систем.

- PostgreSQL підтримує географічні об'єкти, тому його можна використовувати як сховища геопросторових даних для служб на основі визначення місцезнаходження та географічних інформаційних систем.
- Щоб вивчати PostgreSQL, вам не потрібно багато тренувань, тому що його легко використовувати
- Адміністрування з мінімальними експлуатаційними витратами як для вбудованого, так і для корпоративного використання.

Так звичайно, я і будь-яка система, PostgreSQL має недоліків:

- Якщо ми бачимо архітектуру PostgreSQL (мова структурованих запитів). на наведеній вище діаграмі створює окремий сервіс для кожного клієнта. Який перетворюється на велике використання пам'яті.
- Якщо ми зробимо порівняння, PostgreSQL не дуже хороший у плані продуктивності.
- Він не такий популярний, як інші системи управління базами даних.
- Це також брак кваліфікованих спеціалістів.
- Що стосується швидкості, PostgreSQL не заслуговує на увагу в порівнянні з іншими інструментами.
- Робити реплікацію складніше.

Для авторизації та push-повідомлень було обрано технологію Firebase від Google. Firebase — це платформа розробки програм Backend-as-a-Service (BaaS), яка надає розміщені серверні служби, такі як база даних у реальному часі, хмарне сховище, автентифікація, звіти про помилки в роботі мобільного додатку, аналітика тощо. Він створений на основі інфраструктури Google і автоматично масштабується.

Firebase зберігає текстові дані в форматі JSON і надає методи для їх читання, оновлення та видалення.

В додаток, Firebase надає зручний інструмент для авторизації юзерів мобільного додатку, зберігати сесії авторизованих юзерів, зберігати медіа файли. Firebase розповсюджуються по моделі Freemium - є безкоштовним інструментом до певних лімітів та обмежений в функціоналі для безкоштовного плану.

Функціонал, такий як: авторизація та зберігання тексту є в безкоштовному тарифі. Для цілей додатку в рамках магістерської роботи безкоштовний тариф задовольняє всі потреби.

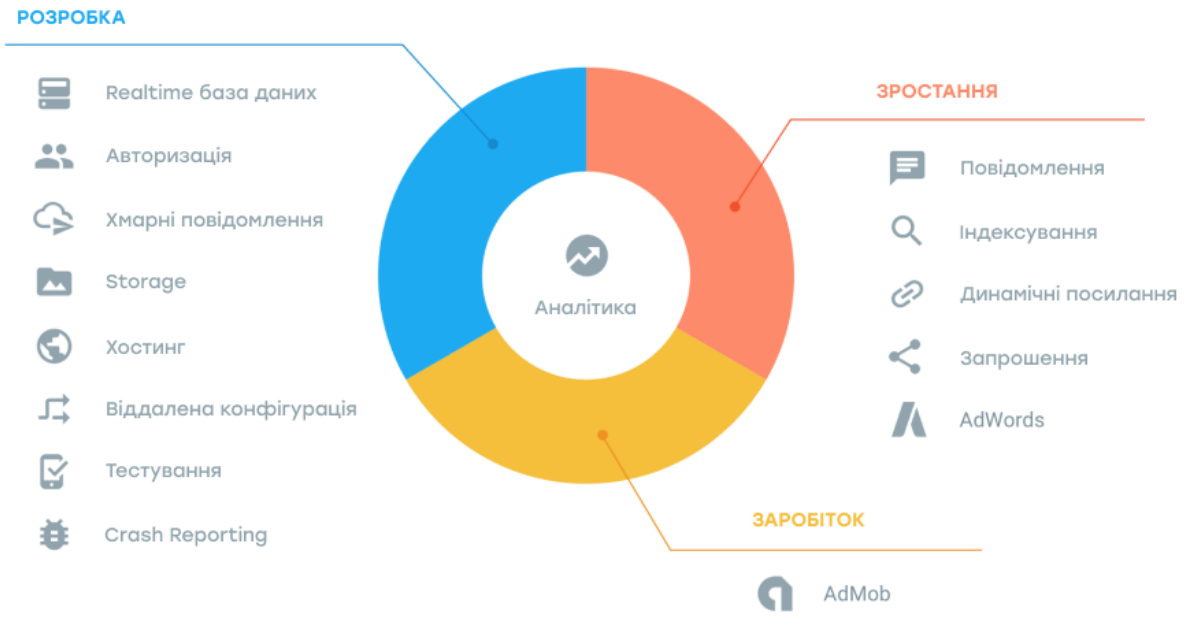

Рисунок 4 – Функціональні можливості Firebase

Firebase має ряд переваг, які повпливали на вибір технології:

- **Синхронізація в реальному часі для даних JSON**. База даних Firebase Realtime є хмарною базою даних NoSQL, яка дозволяє зберігати і синхронізувати дані між користувачами в режимі реального часу.
- **Спільно працювати з пристроями.** Синхронізація в реальному часі дозволяє користувачам отримувати доступ до своїх даних з будь-якого пристрою: веб-або мобільного, і це допомагає користувачам співпрацювати один з одним.
- **Створення без серверних програм.** База даних Realtime поставляється з мобільними та веб-SDK, тому можна створювати програми без необхідності серверів.
- **Оптимізовано для автономного використання.** Коли користувачі переходять в автономний режим, SDK бази даних Realtime використовує

локальний кеш на пристрої для обслуговування та збереження змін. Коли пристрій підключається до мережі, локальні дані автоматично синхронізуються.

- **Сильна безпека користувача.** База даних Realtime інтегрується з Firebase Authentication для забезпечення простої та інтуїтивної автентифікації для розробників.

2.3. Вибір архітектури додатку

Flux – це архітектурне рішення, яке Facebook використовує для роботи з React. Це не фреймворк чи бібліотека. Це просто новий вид архітектури, який доповнює React та концепцію односпрямованого потоку даних.

Він каже, що Facebook забезпечує репозиторій, який включає бібліотеку Диспетчер (Dispatcher). Диспетчер сортує глобальні pub/sub обробники, які транслюють навантаження зареєстрованим обробникам (callbacks).

Типова архітектура Flux буде важелем бібліотеки Диспетчера (Dispatcher library), поряд з модулем NodeJS's під назвою EventEmitter для того, щоб встановити систему подій, яка допоможе керувати станом програми [13]. Flux можливо краще пояснюється його індивідуальними компонентами:

- **Actions** – Допоміжні методи, які полегшують передачу даних Диспетчер (Dispatcher)
- **Dispatcher** – Отримує дії та транслює навантаження у зареєстровані обробники (callbacks)
- **Stores** – Сховища для логіки та стану програми, які мають обробники зареєстровані в Диспетчері
- **Controller Views** – React компоненти, які захоплюють стан із Сховищ та передають їх вниз через властивості дочірніх компонентів.

Давайте подивимося як цей процес виглядає графічно:

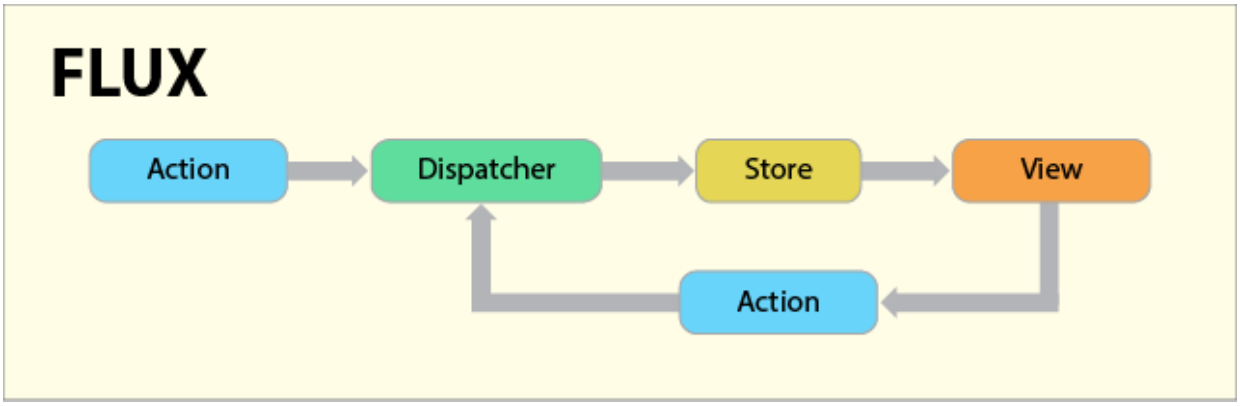

Рисунок 5 – Схема роботи Flux архітектури

2.4. Аналіз і порівняння інструментів розробки

Основним середовищем розробки програмного додатку було обрано Visual Code IDE.

Visual Studio Code – один із найпопулярніших редакторів вихідного коду, які використовуються програмістами. Він швидкий, легкий і, до того ж, потужний! Microsoft розробила VSC як крос-платформний редактор коду для написання веб- та хмарних додатків. Вперше про це було оголошено 29 квітня 2015 компанією Microsoft на конференції Build 2015, яка проходила в Сан-Франциско. Через кілька місяців, 18 листопада 2015 року, VSC було випущено під ліцензією MIT, а вихідний код був доступний на GitHub. 14 квітня 2016 року VSC було випущено в Інтернеті.

Він легкий, швидкий, з відкритим кодом і крос-платформний, а також інші цікаві функції дають йому додаткову перевагу перед будь-яким іншим редактором. Отже, ось 10 причин, з яких, на мою думку, він набув такої популярності серед спільноти розробників.

- Крос-платформний
- Підтримує безліч мов програмування, зокрема: Python, JavaScript, HTML, CSS, TypeScript, C++, Java, PHP, Go, C#, PHP, SQL, Ruby, Objective-C та багато іншого.
- Надає документацію про мову програмування

- Налагодження, яка допомагає прискорити цикл редагування, компіляції та налагодження будь-якого програміста.
- Вбудована інтеграція Git
- IntelliSense, функція, яка використовується для інтелектуального завершення коду, інформації про параметри, допомоги контенту, швидкої інформації та натяків на код.
- Кастомізація

Для налаштування параметрів додатку для операційної системи IOS було обрано XCode IDE.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ ДОДАТКУ

3.1.Побудова функціональної моделі

Розробка комплексної інформаційної системи неможлива без її ретельного проектування. Спочатку все була створена модель Structured Analysis and Design Technique (SADT – методологія структурного аналізу та проектування) – це методологія, яка розроблена спеціально для того, щоб полегшити опис та розуміння штучних систем, що потрапляють у розряд середньої складності. SADT була створена та випробувана на практиці в період з 1969 по 1973 р. Ця методологія виникла під сильним впливом PLEX, концепції клітинної моделі людина-орієнтованих функцій Хорі, загальної теорії систем технології програмування та навіть кібернетики. З 1973 р. сфера її використання істотно розширюється для вирішення завдань, пов'язаних з великими системами, такими як проектування телефонних комунікацій реального часу, автоматизація виробництва (CAM), створення програмного забезпечення для командних та керуючих систем, підтримка боєздатності. Вона успішно застосовувалася для описи великої кількості складних штучних систем із широкого спектру областей (банківська справа, очищення нафти, планування промислового виробництва, системи наведення ракет, організація матеріально-технічного постачання, методологія планування, технологія програмування). Причина такого успіху полягає в тому, що SADT є повною методологією для створення опису систем, що базується на концепціях системного моделювання. Функціональна модель може мати в собі декілька рівнів деталізації. [1]

Функціональний блок зображується як прямокутник і репрезентує будь-яку конкретну функцію в рамках системи. Кожна з чотирьох сторін функціонального блоку має своє певне значення:

- Права сторона - «Вихід» (Output);
- ліва сторона - «Вхід» (Input);
- нижня сторона - «Механізм» (Mechanism);

- верхня сторона - «Управління» (Control);

На основі аналізу теоретичних відомостей про SADT було реалізовано діаграму роботи інформаційної системи «Інформаційна технологія проектування мобільних систем електронної комерції» (рис.6).

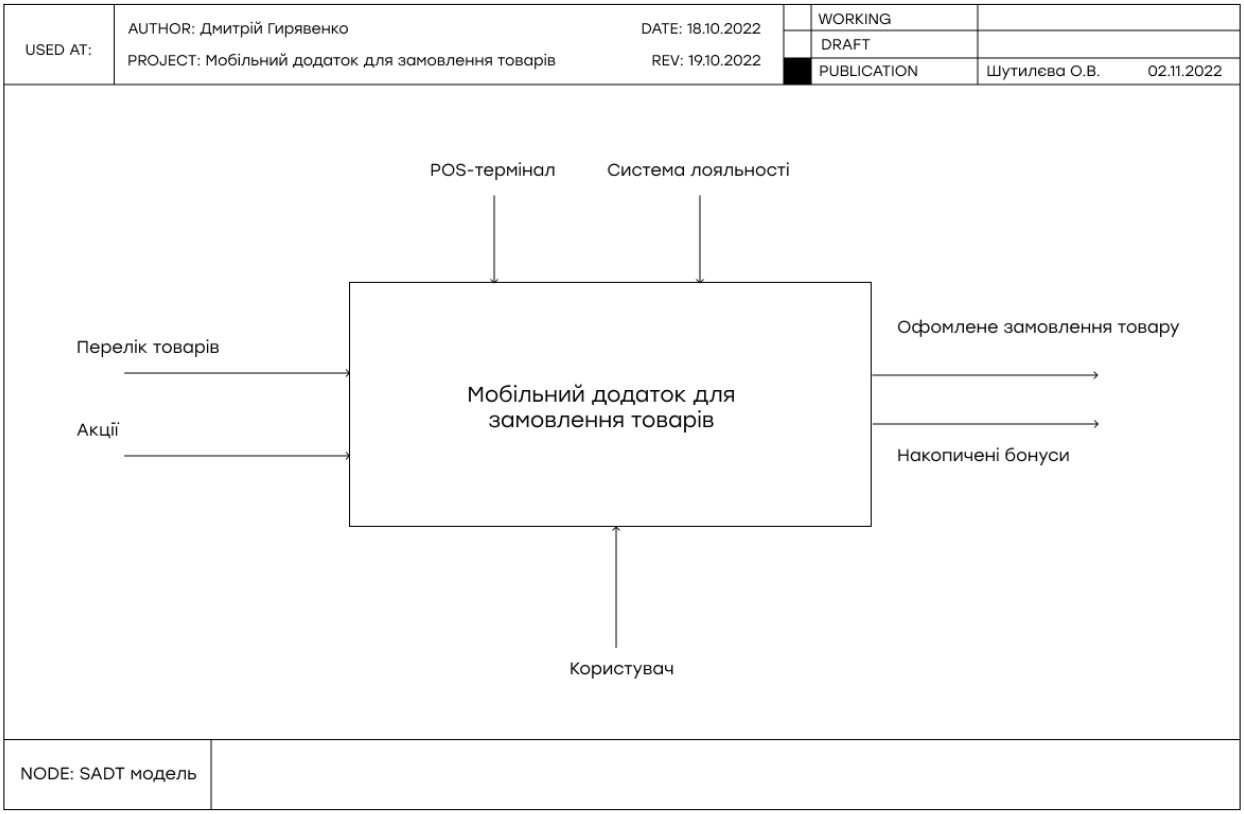

Рисунок 6 – SADT модель

3.2.Побудова діаграми потоків даних(DFD діаграма)

Діаграми потоків даних (Data Flow Diagrams – DFD) є ієрархією функціональних процесів, пов'язаних потоками даних. Мета - продемонструвати, як кожен процес перетворює свої вхідні дані на вихідні, а також виявити відносини між цими процесами. Основними компонентами діаграм потоків даних є:

- зовнішні сутності;
- системи/підсистеми;
- процеси;

- база даних;
- потоки даних.

На основі аналізу теоретичних відомостей про DFD діаграми було реалізовано діаграму роботи інформаційної системи «Інформаційна технологія проектування мобільних систем електронної комерції» (рис.7).

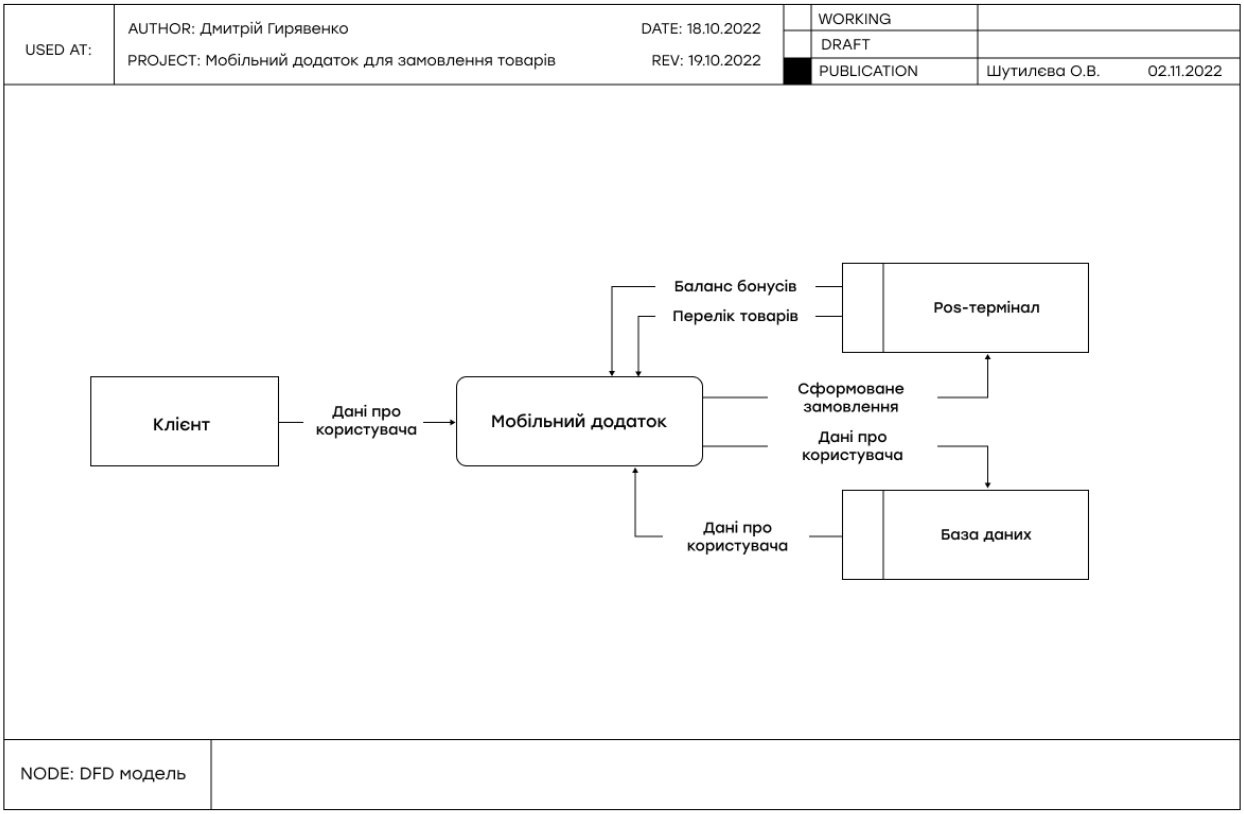

Рисунок 7 – DFD діаграма

3.3.Варіанти використання додатку

Діаграма варіантів прецедентів – це тип поведінкової діаграми UML, часто використовується для аналізу різних систем. Вони дозволяють візуалізувати різні типи ролей у системі та те, як ці ролі взаємодіють із системою. Цей посібник з діаграми варіантів використання охоплює наступні теми та допоможе вам краще створювати сценарії використання.

Суть даної діаграми: система представляється у вигляді акторів, що за допомогою варіантів використання взаємодіють з системою [2]. Діаграма варіантів складаються з чотирьох сутностей:

- Актор
- Випадок використання
- Система
- Пакет

Актор. Актор – це будь-яка сутність, яка виконує роль в одній даній системі. Це може бути людина, організація або зовнішня система.

Випадок використання. Випадок використання є функцією чи дію всередині системи.

Система. Система використовується для визначення сфери застосування та намальована у вигляді прямокутника. Це необов'язковий елемент, але корисний під час візуалізації великих систем.

Пакет. Пакети використовуються для угруповання випадків використання.

Діаграма варіантів використання мобільного додатка представлена на рис. 7 з додатком взаємодіє два актори – зареєстрований і незареєстрований користувачі.

Зареєстрований користувач може авторизуватися в додатку за допомогою номеру мобільного телефону. Без авторизації подальше використання додатку можливе, але з обмеженим переліком функцій. Зареєстрований користувач має можливість переглянути меню, переглянути інформацію про товар, додати товар в корзину та до секції «Вподобані товари». Зареєстрований користувач має можливість обрати варіант доставки замовлення, час та адресу. Зареєстрований користувач має можливість оформити замовлення. Зареєстрований користувач може переглянути історію замовлень. Зареєстрований користувач може керувати персональними даними.

Незареєстрований користувач може зареєструватися в додатку, переглянути меню та детальну інформацію про товар.

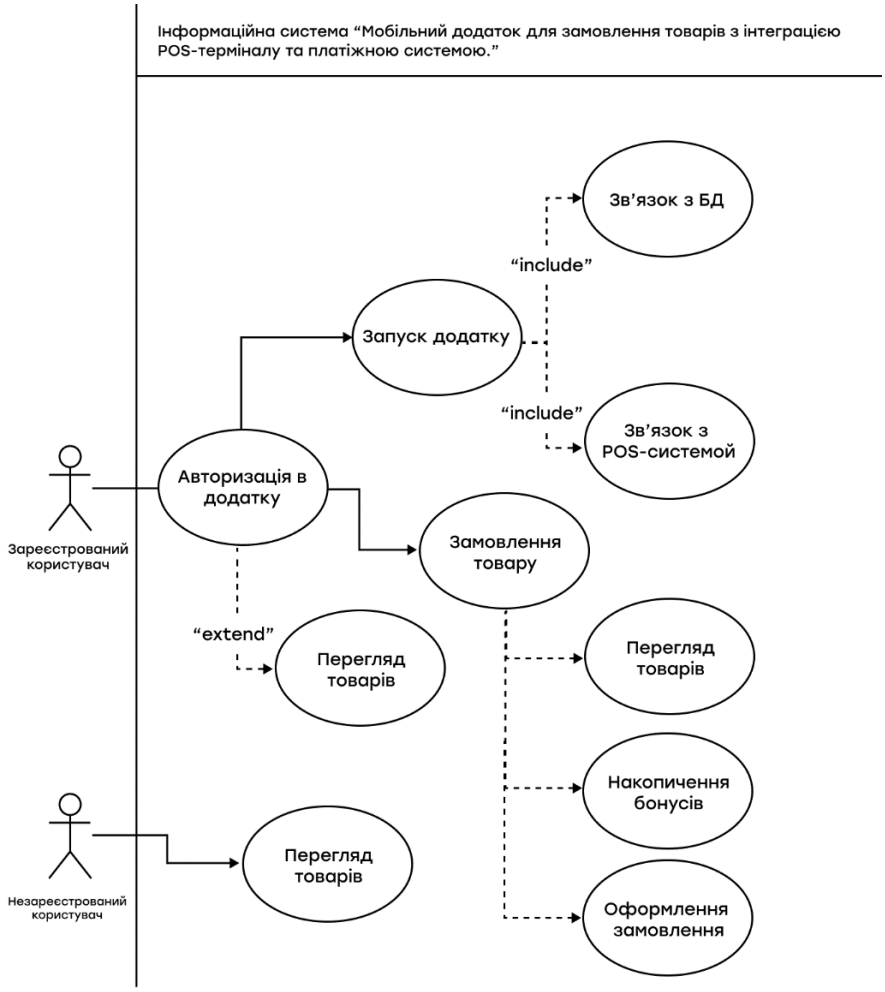

Рисунок 8 – Варіанти використання додатку

3.4.Побудова діаграми станів

Діаграми станів є добре відомим засобом опису поведінки систем. Вони визначають всі можливі стани, у яких може бути конкретний об'єкт, і навіть процес зміни станів об'єкта внаслідок впливу деяких подій.

Початковий стан виділяється чорною крапкою: воно відповідає стану об'єкта в момент його створення. Кінцевий стан позначається чорною точкою.

Діаграма станів для варіантів використання, пов'язаних з актором «Незареєстрований користувач», показана на рисунку 9.

Рисунок 9 – Варіанти використання додатку для «Незареєстрований» ролі

3.5.Прототипування

Прототипування – це процес створення схематичного зображення окремої сторінки або мобільного додатку в цілому. На відміну від технічних завдань, він дає можливість випробувати розробку [3]. Процес створення прототипу зазвичай складається з таких кроків:

- Визначення початкових вимог;
- розробка першого варіанта прототипу, який містить тільки інтерфейс користувача;
- вивчення прототипу замовником та кінцевими користувачами, отримання зворотного зв'язку про необхідні зміни та доповнення;
- переробка та покращення прототипу: з урахуванням отриманих зауважень та пропозицій змінюються як специфікації так і прототип, після цього кроки 3 та 4 можуть повторюватися.

На прототипі схематично зображуються основні елементи продукту і їх відгук на дії користувача. У процесі створення прототипу з'являються сотні, якщо не тисячі ідей [3]. Прототип сторінки сайту поняття досить широке, так як це може бути просто ескіз від руки на папері, схема у форматі картинки або детальний інтерактивний прототип. Класифікуємо їх за основними характеристиками. По глибині опрацювання деталей: з низькою деталізацією; із високою деталізацією. Наскільки можна взаємодіяти з прототипом: графічний (статичний) – як графічного зображення; інтерактивний – з елементами взаємодії, наприклад, з посиланнями, робочим «слайдером», формами, що спливають, або іншими активними елементами (рис. 10.1).

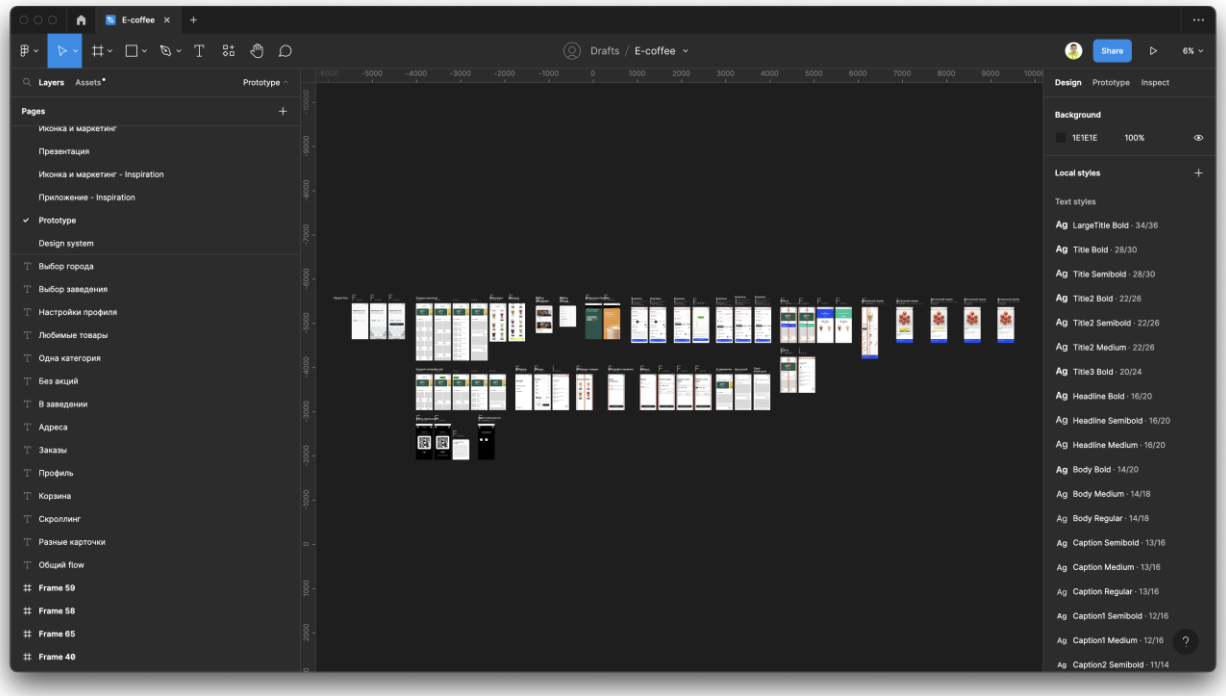

Рисунок 10.1 – Прототип додатку

Наступним кроком було реалізувати інтерактивний прототип на основі вже існуючого.

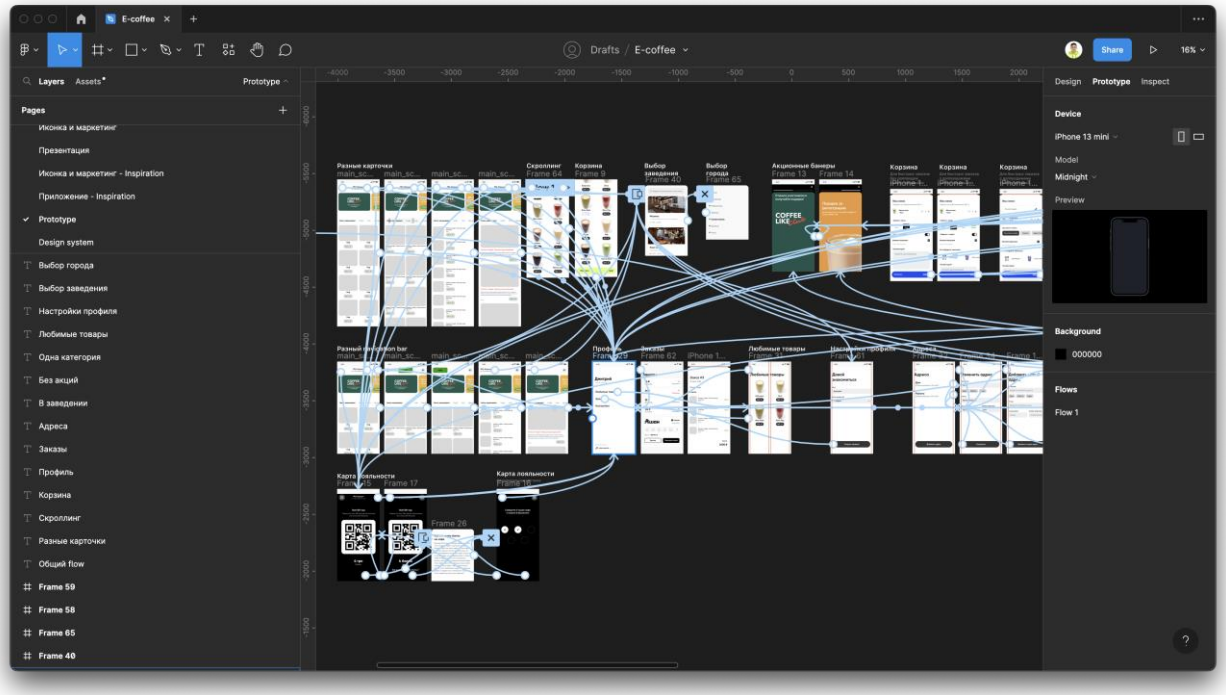

Рисунок 10.2 – Інтерактивний прототип

Перевагами такого прототипу є зручність пояснення навігації та основних функцій додатку для розробників та замовників.

3.6.Розробка інтерфейсу додатку

Розробка інтерфейсу розпочалась з дослідження сучасних тенденцій в дизайні інтерфейсів. Для аналізу було оброблено ряд інтернет-ресурсів:

- www.dribbble.com
- www.behance.net
- www.uplabs.com

Було обрано декілька стильових рішень, на які можна опиратися під час розробки користувацького інтерфейсу.

Основним інструментом для створення прототипів було обрано Figma. Figma – це веб-додаток для редагування графіки та дизайну інтерфейсу користувача. Воно підходить для виконання всіх видів робіт з графічного дизайну, починаючи зі створення каркасів веб-сайтів, розробки інтерфейсів мобільних додатків, створення прототипів, створення постів у соціальних мережах та іншого. Сервіс доступний за передплатою, передбачено безкоштовний тарифний план для одного користувача. Існують офлайн-версії для Windows, macOS. Реалізовано інтеграцію з корпоративним інструментом Slack та інструментом прототипування Framer. Використовується для створення спрощених прототипів інтерфейсів, так і для детального опрацювання дизайну інтерфейсів мобільних додатків, веб-сайтів, корпоративних порталів [12].

Figma відрізняється від інших інструментів редагування графіки. Головним чином тому, що він працює безпосередньо у браузері. Це означає, що можна отримати доступ до своїх проектів і починати їх розробку з будь-якого комп'ютера або платформи без необхідності купувати кілька ліцензій або інсталювати програмне забезпечення.

Ще одна причина, з якої дизайнери люблять цю програму, полягає в тому, що Figma пропонує щедрий безкоштовний план, в якому можна створювати та зберігати 3 активні проекти одночасно.

Загалом Figma є потужним інструментом з переліком переваг:

- Сервіс безкоштовний (до двох користувачів та трьох проектів на обліковий запис);
- Працює як на Mac, так і на Windows;
- Командна робота у режимі реального часу
- Імпорт файлів Sketch;
- Інтегрована передача проектів розробникам;
- Бібліотека дизайн-систем;
- Комплексне якісне прототипування.

До недоліків можна віднести неможливість працювати з сервісом без інтернет з’єднання, проте можна зберегти файл локально, попрацювати з ним і додати його пізніше.

Отже, наступним кроком є розробка дизайн системи. До неї увійшли наступні елементи інтерфейсу:

- Кольорова палітра;
- Шрифтова система;
- Поле вводу
- Navigation bar;
- Система кнопок.

Перевагою даного підходу є оптимізація часу у внесенні змін до готового дизайн продукту після публікації додатку в магазин. Створивши дизайн елемент один раз, його можна перевикористовувати в багатьох інших місцях. Також змінивши зовнішній вигляд один раз, заміниться всюди де використовується автоматично. Також для дизайн-система достатньо корисна для структуризації та підтримки проекту [6].

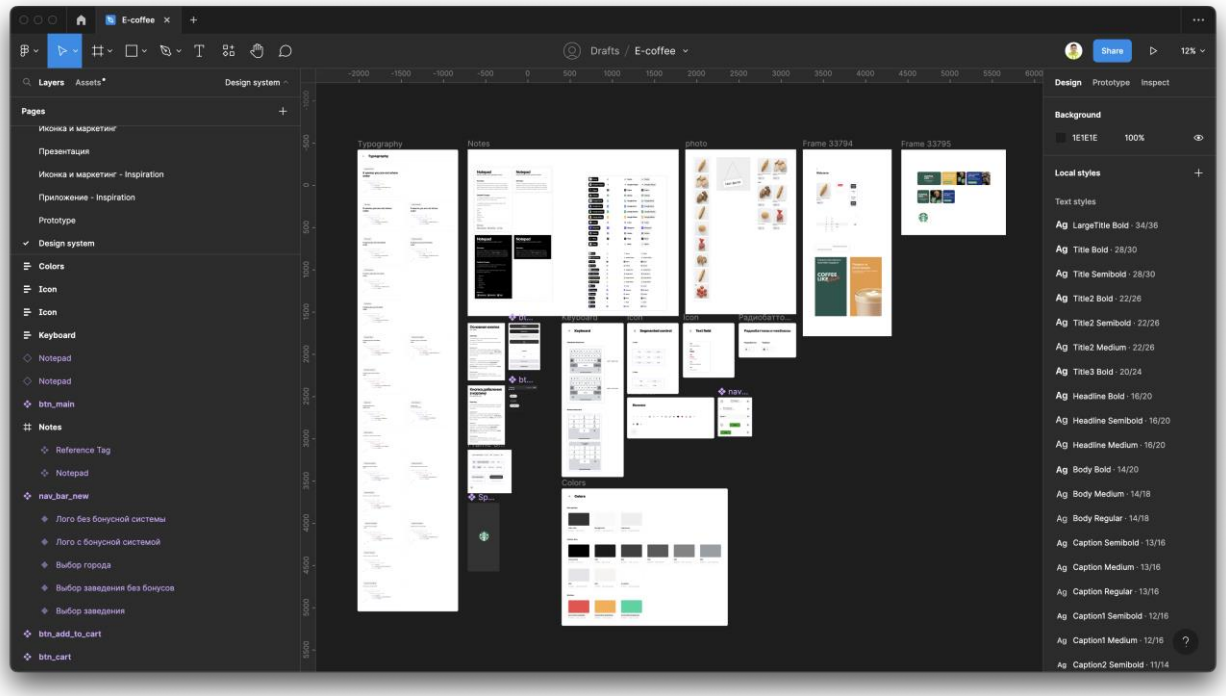

Рисунок 10.3 – Дизайн-система додатку

Наступним кроком є розробка інтерфейсу мобільного додатку для користувача. Дизайн користувацького інтерфейсу було розпочато з створення дизайну авторизації. Було реалізовано splash екран, введення номеру телефона користувача, форма заповнення особистих даних- ім’я та дата народження.

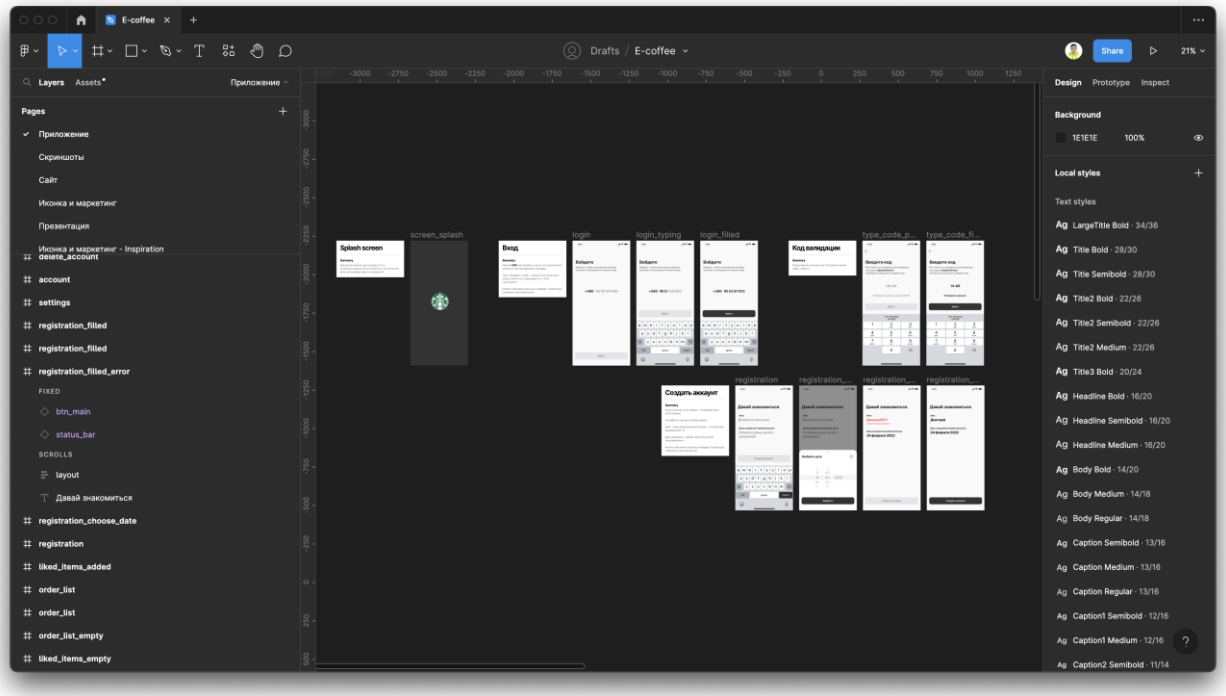

Рисунок 10.4 – Дизайн екранів авторизації

Наступним кроком є реалізація основного потоку роботи додатку.

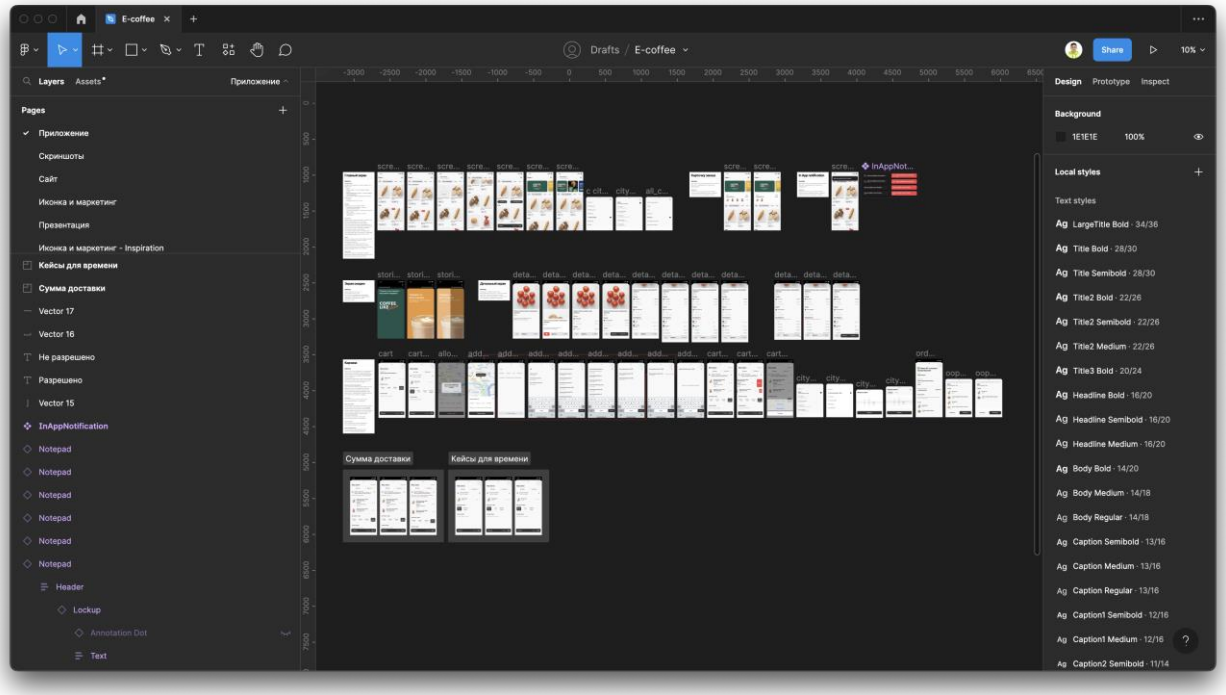

Рисунок 10.5 – Дизайн основного потоку екранів

Останнім кроком є реалізація налаштувань в додатку.

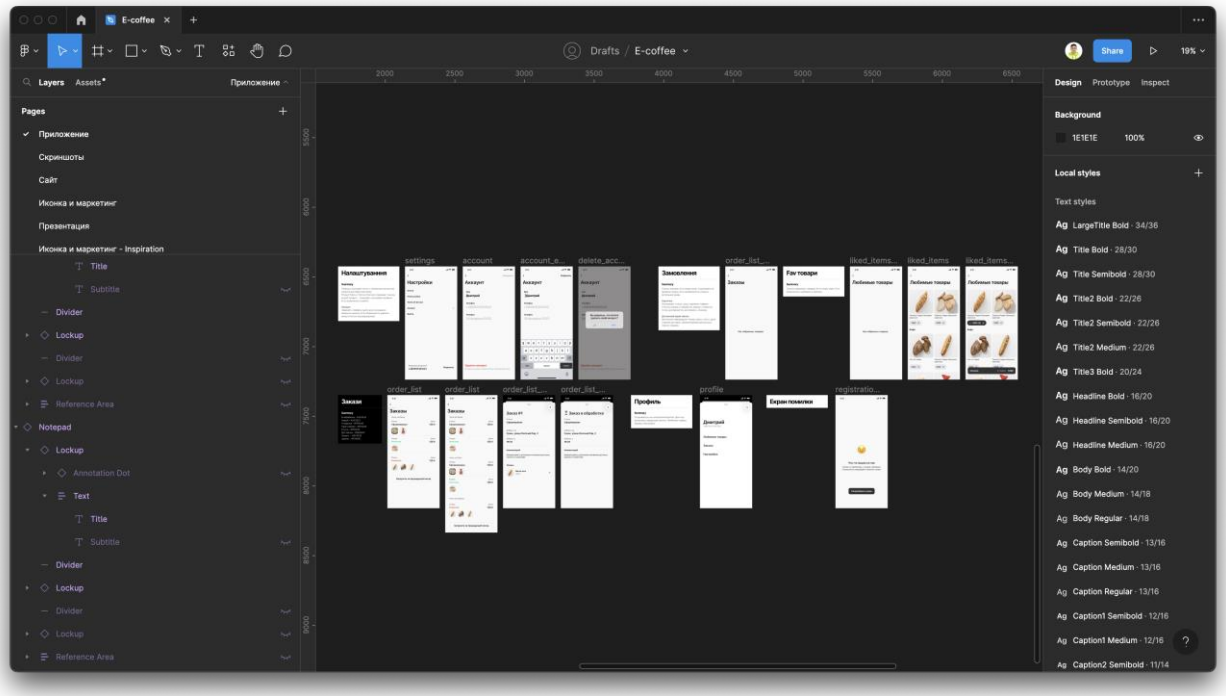

Рисунок 10.6 – Дизайн екранів налаштування

Загалом було реалізовано дизайн таких частин додатку: головне меню, карточка товару, акційна карточка товару, акційні банери та детальний екран акції, екран з детальною інформацією про товар, дизайн модифікаторів, логіка корзини, форми вибору часу замовлення, карточка замовлення на головній сторінці, вибір адреси замовлення на карті, система внутрішніх повідомлень, діалог вибору категорії, налаштування, перелік вподобаних товарів, історія замовлень.

Також було реалізовано систему для підтвердження дій користувача за допомогою «alerts view».

3.7.Розробка логотипу додатку

Для розробки логотипу додатку було використано інструмент Figma. Першим кроком було обрано концепцію та стиль логотипу. Далі було намальовано кілька десятків варіантів на чернетці. Наступним кроком було

перенести варіанти логотипу в сервіс для деталізації Figma. На основі кольорової до логотипу було додано гаму. Фінальний варіант відображений на р 11.

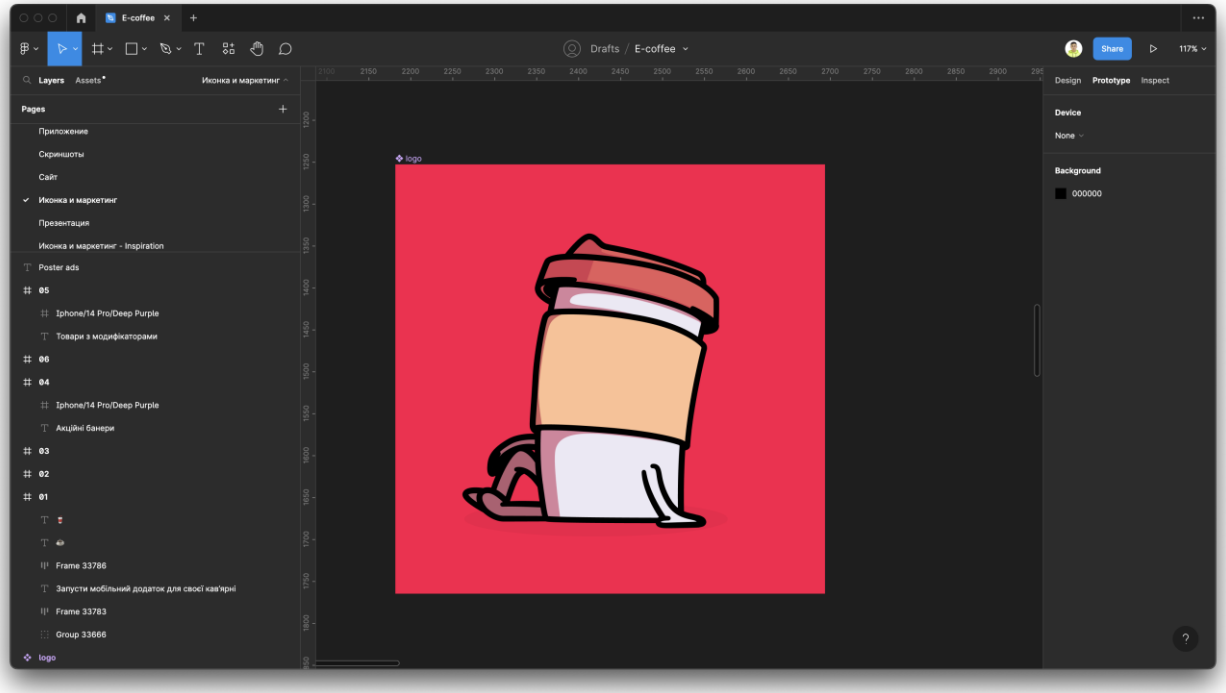

Рисунок 11 – Логотип додатку

3.8. Проектування архітектури додатку та моделювання моделі БД

Успішна розробка додатку неможлива без ретельного проектування архітектури проекту та моделювання бази даних. Під час проектування бази даних було виділено сутності та їх параметри. Загалом всі сутності було поділено на 4 групи: Customer – користувач, Menu – сутність меню, Organization – налаштування закладу, App setting - налаштування конфігурації додатку, Order - замовлення.

Група Customer відповідає за дані про користувача – customers, adress- адреса користувача(потрібно для доставки) та sms_code – смс-код для верифікації користувача в додатку.

Група Menu відповідає за дані про список товарів меню- product, всі продукти групуються на категорії – group. Товари мають налаштування

модифікаторів. Також присутня можливість сховати окремий продукт – hide_product та модифікатори продукту - hide_modifier.

Група Organization відповідає за дані закладу - organization, доступні методи оплати замовлення закладу та налаштування годин доставки – delivery_hours

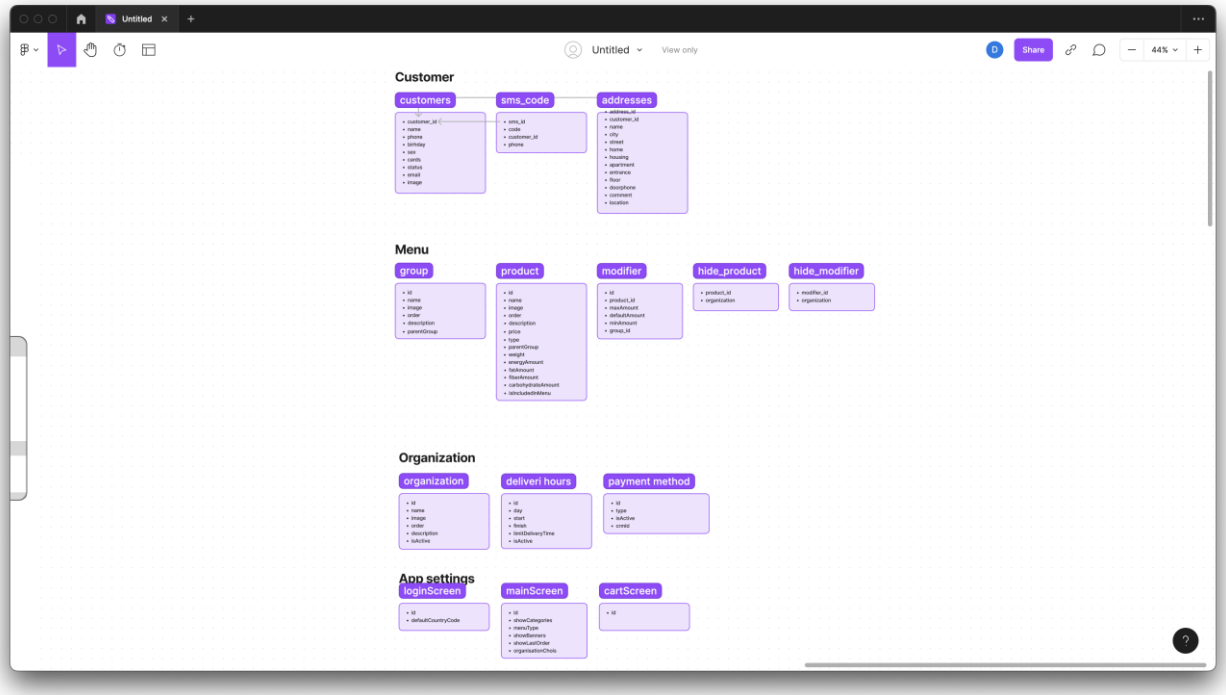

Рисунок 12 – Дизайн екранів налаштування

Група App settings відповідає за налаштування конфігурацій додатку, зокрема налаштування авторизації - loginScreen, головного екрану – mainScreen та корзини замовлення – cartScreen.

Група Order відповідає за налаштування замовлення, зокрема список товарів в замовленні – ordersItems, order – інформацію про замовлення та адресу orderAdress.

Таким чином було спроектовано зв’язки між сутностями.

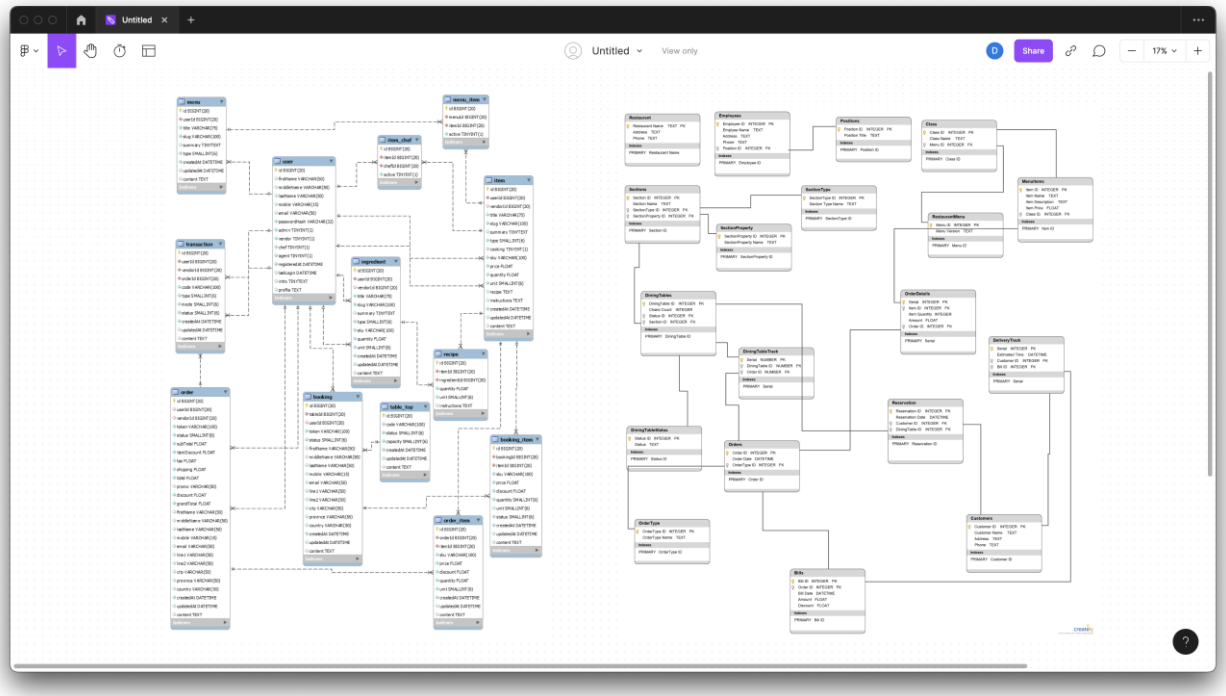

Рисунок 13 – Дизайн екранів налаштування

3.9.Проектування архітектури додатку та моделювання Firebase

Для авторизації користувачів в додатку та налаштування push-повідомлень було обрано технологію Firebase. Firebase – ієрархічна база даних тому визначити ERD діаграму (як це роблять для реляційних баз даних) неможливо, тому найкращим варіантом є показати структуру JSON дерева(ДОДАТОК):

Загалом було спроектовано 5 сутностей:

- users – список користувачів;
- order – замовлення
- sale – банери для знижок;
- notification – push-повідомлення;
- item – сутність товару.

3.10. Програмна реалізація клієнтської частини

Першим етапом в створенні клієнтської частини є розгортання проекту в інтегрованому середовищі розробки. Далі було реалізовано підключення та налаштування Flux технології [4].

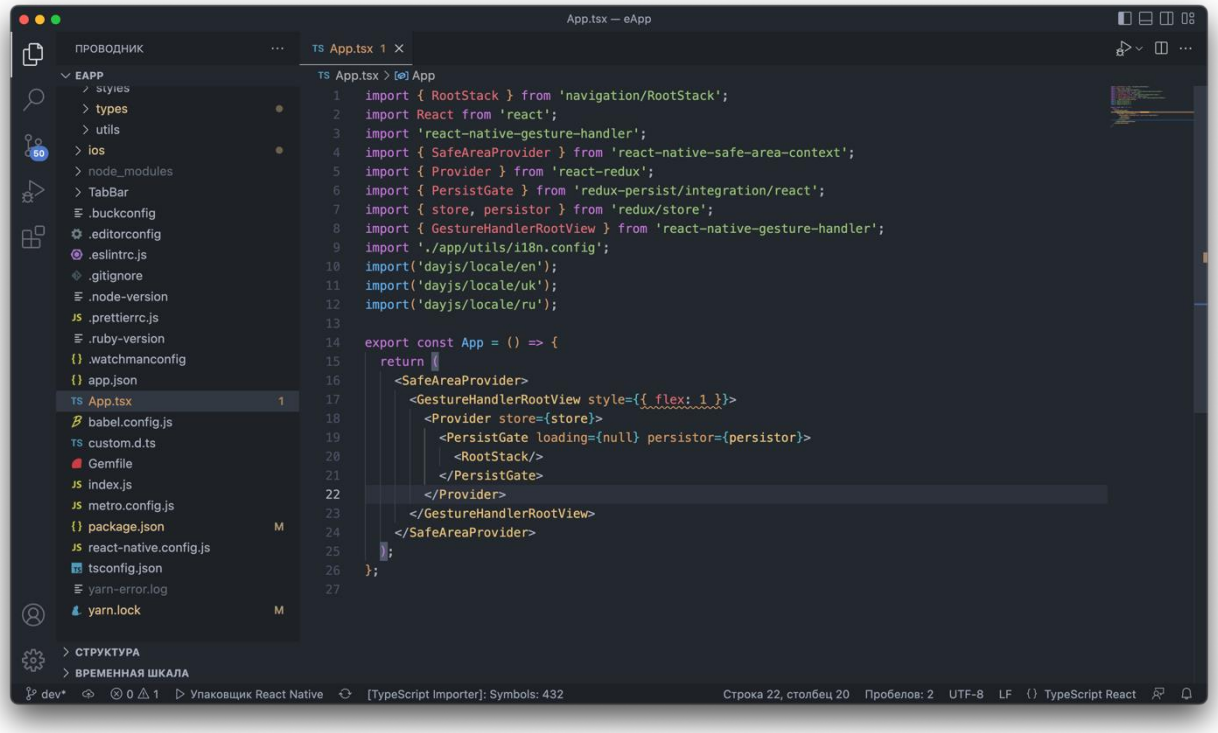

Рисунок 14.1 – Підключення та налаштування Flux

Наступним кроком було виконано налаштування навігації в межах додатку. Це один з найважливіших кроків в реалізації клієнтської частини. Перш за все було налаштовано логіку роботи нижнього бару навігації, та переходи між екранами в яких нижній бар відсутній.

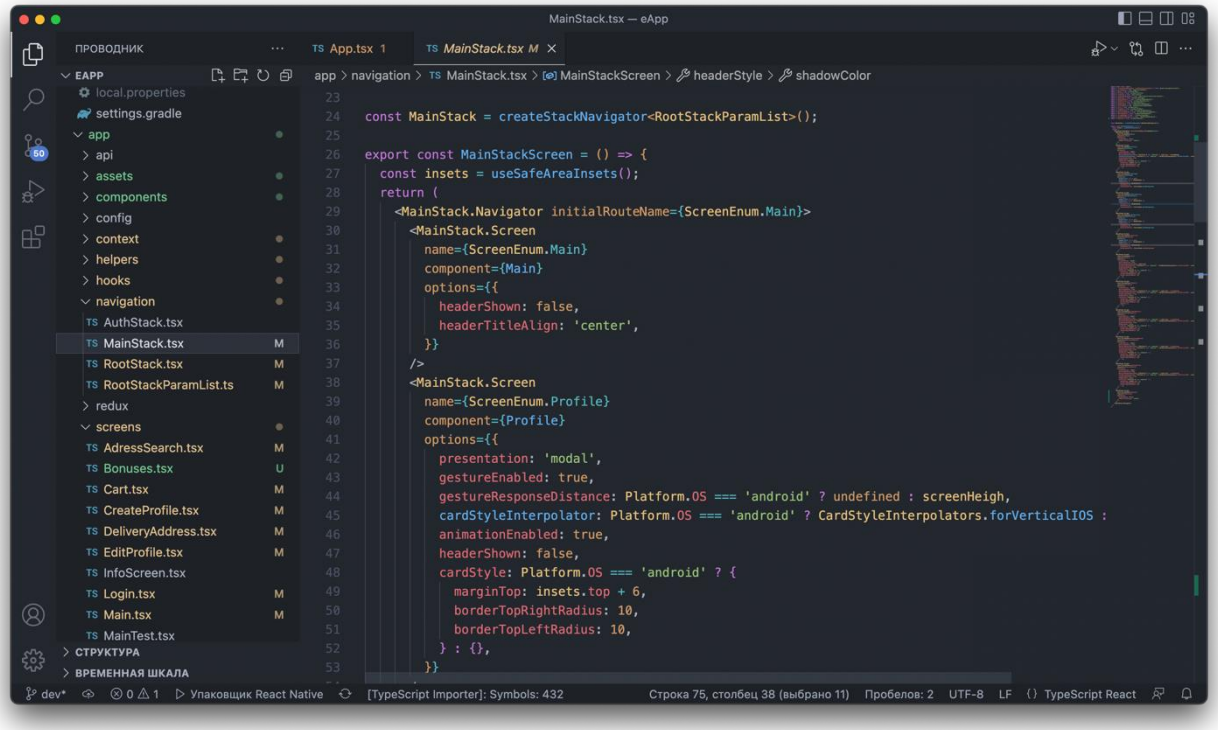

Рисунок 14.2 – Налаштування навігації

Мобільний додаток підтримує англійську та українські мови, тому важливо налаштувати локалізацію мобільного додатку. Для зручності в підтримці всі заголовки було винесено в окремі JSON файли для кожної мови. Це дозволяю досить зручно передавати ці файли на переклад як на етапі розробки мобільного додатку так і під час оновлення чи внесення нового функціоналу.

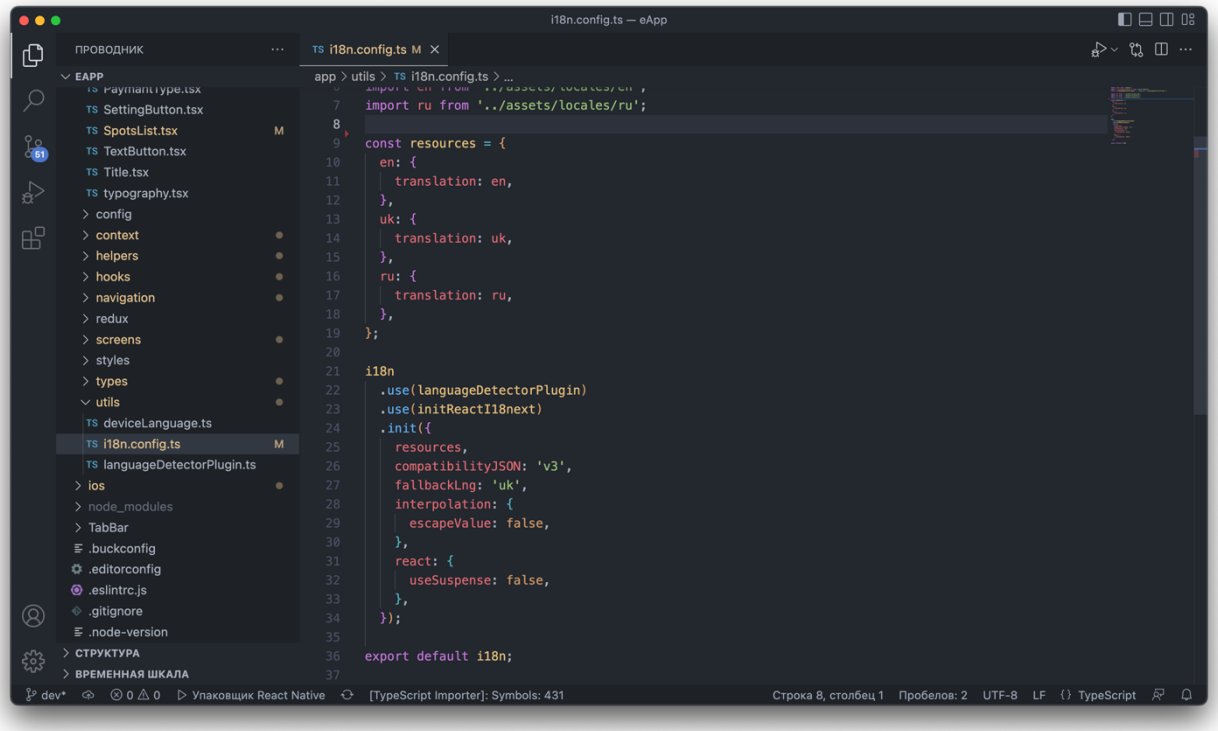

Рисунок 14.3 – Налаштування локалізації

Додаток будуть використовувати різні клієнти під власними брендами, важливо налаштувати тему мобільного додатку. Даний підхід дасть змогу в конфігурації додатку швидко налаштовувати кольори бренду.

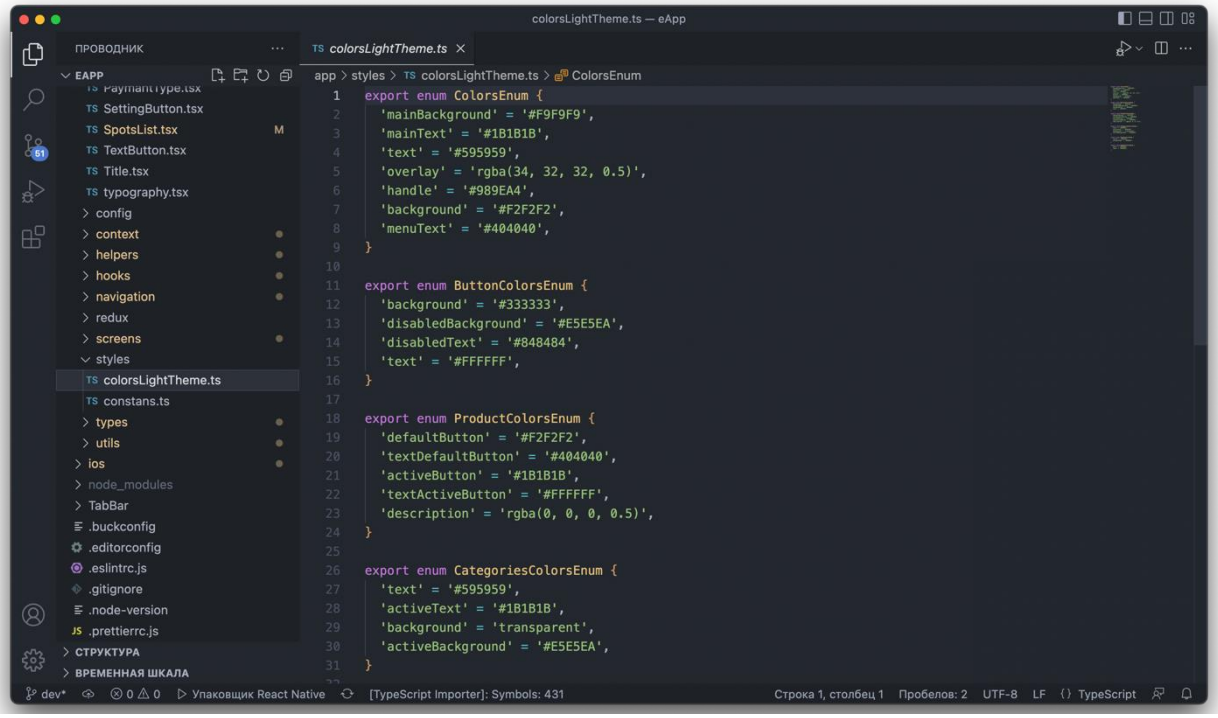

Рисунок 14.4 – Налаштування кольорової теми

Далі було виведено перелік категорій та продуктів на головному екрані.

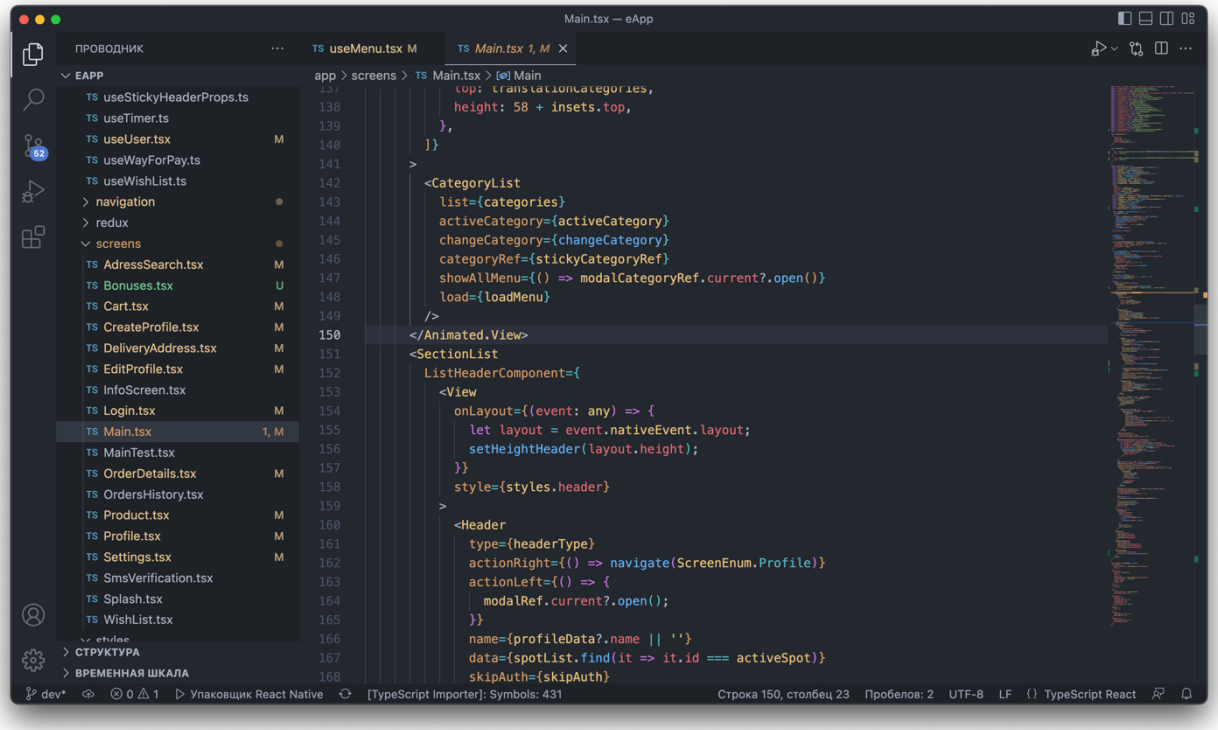

Рисунок 14.5 – Перелік категорій та продуктів

Кожен бізнес має власні умови: різні робочі години, доступність доставки тощо. Всі параметри було винесено в один файл для зручності підтримки.

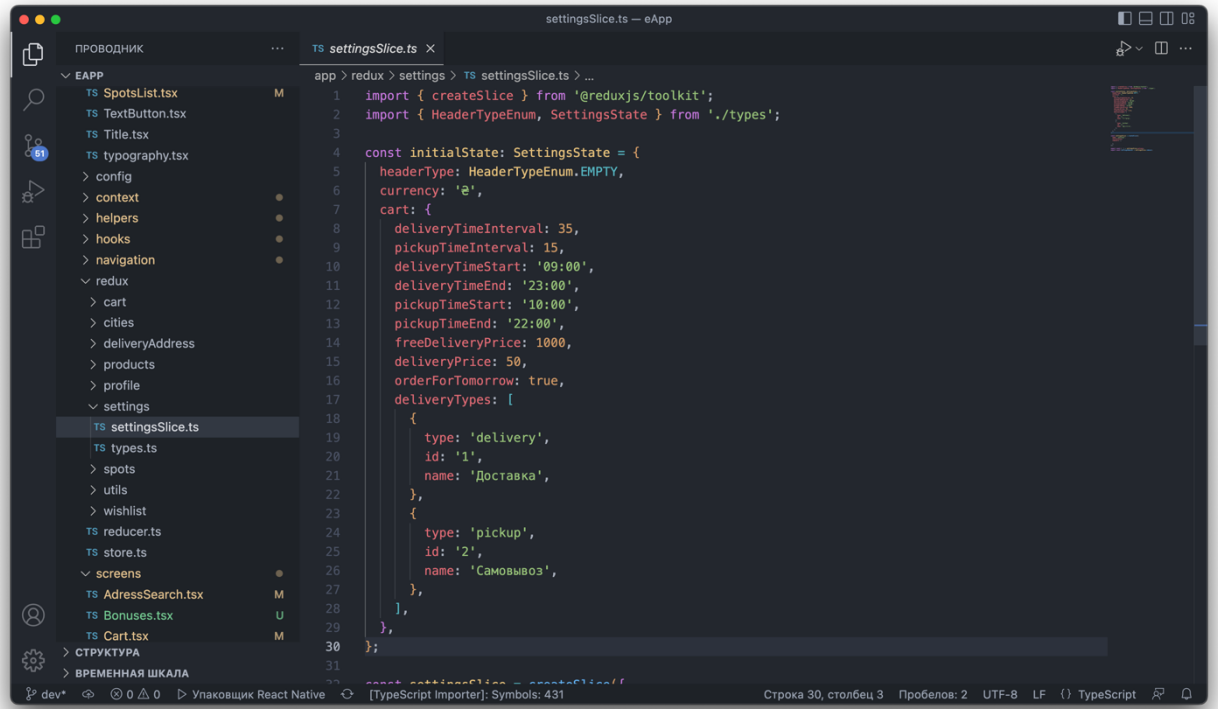

Рисунок 14.6 – Конфігурації додатку

Далі було інтегровано POS-системи для отримання переліку користувачів.

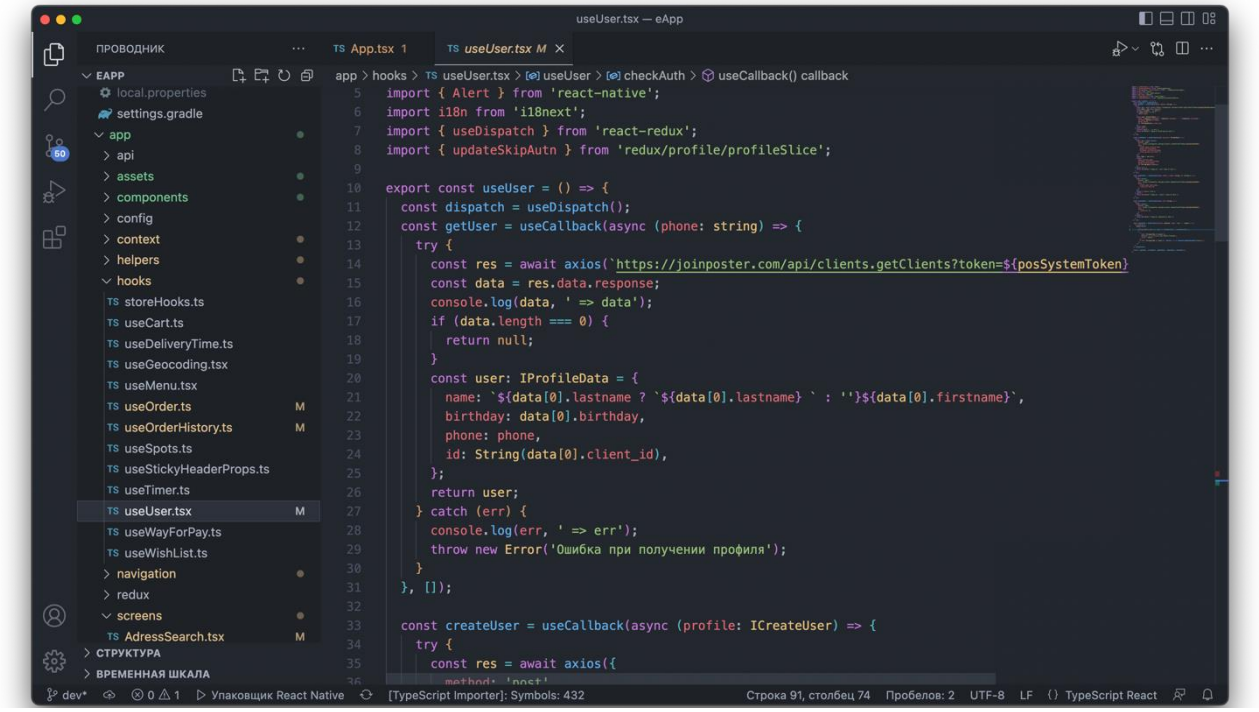

Рисунок 14.7 – Інтеграція з POS-системою

Далі було отримано перелік категорій, товарів, модифікаторів для оформлення та направлення готово замовлення користувачем на POS-термінал.

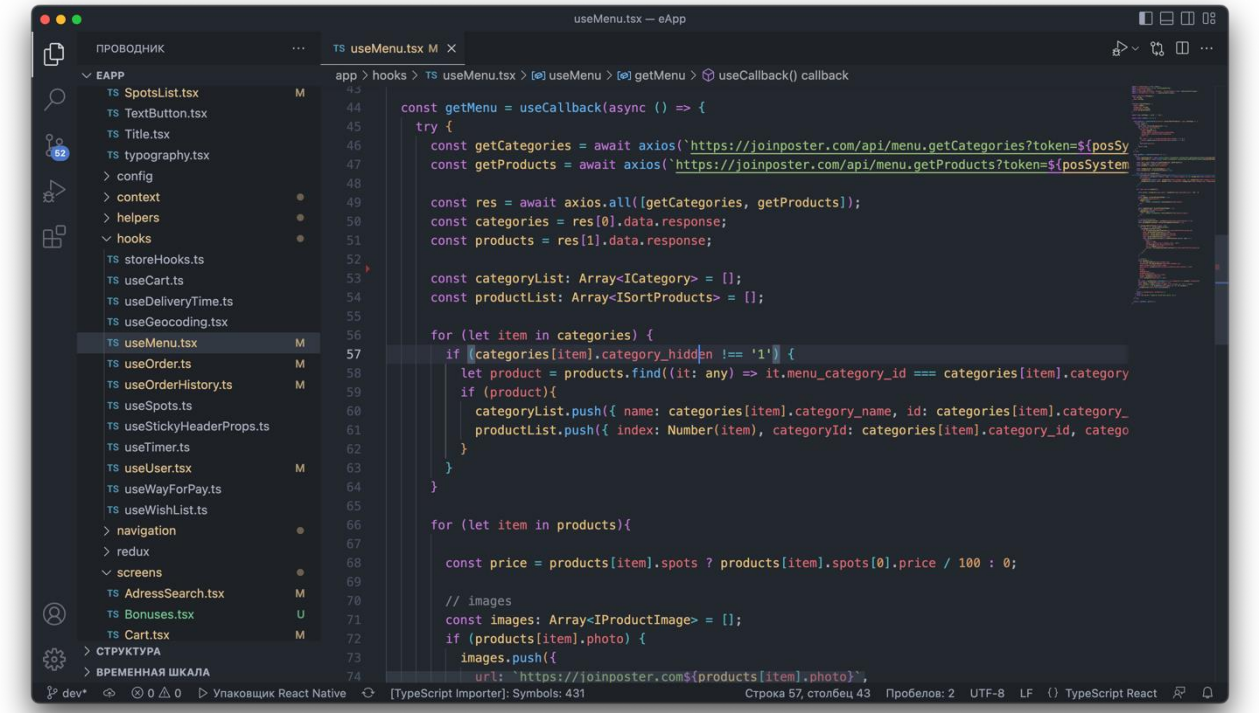

Рисунок 14.8 – Інтеграція з POS-системою для оформлення замовлення

Наступним кроком було реалізовано СМС верифікацію для авторизації користувачів.

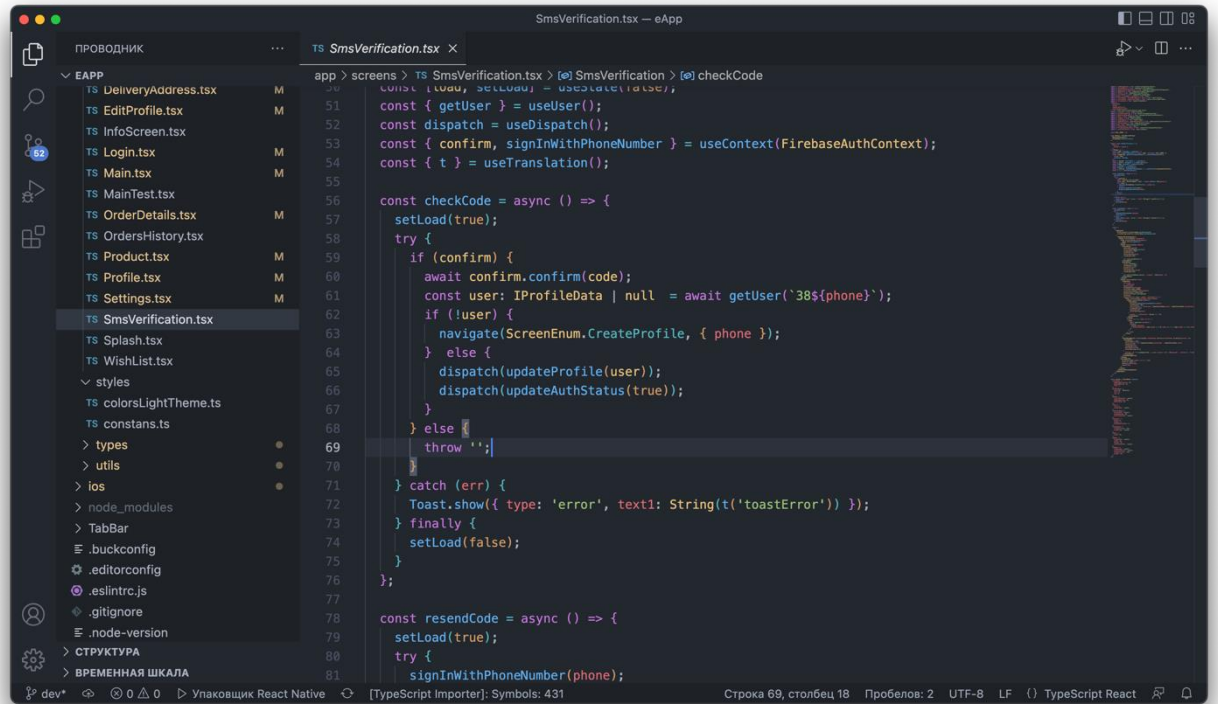

Рисунок 14.9 – Налаштування смс верифікації

Далі було реалізовано функцію вибору адреси доставки.

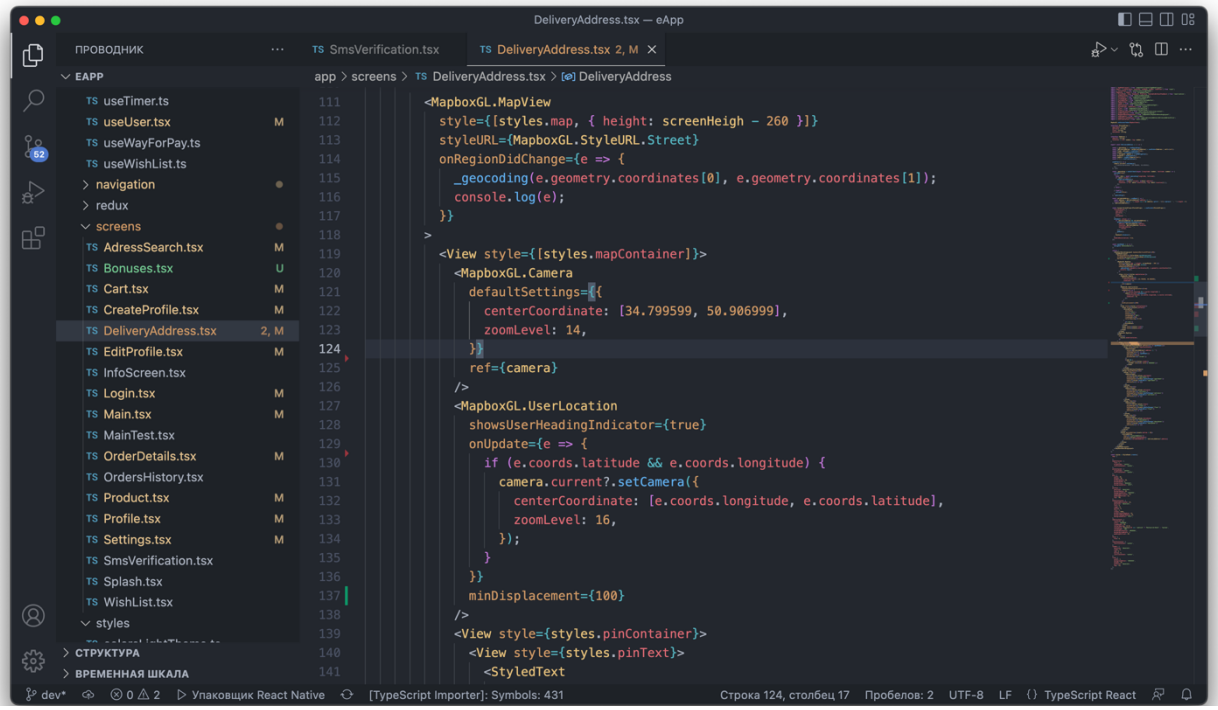

Рисунок 1.10 – Налаштування адреси доставки

Наступним кроком було можливість відправлення готового замовлення в POS систему.

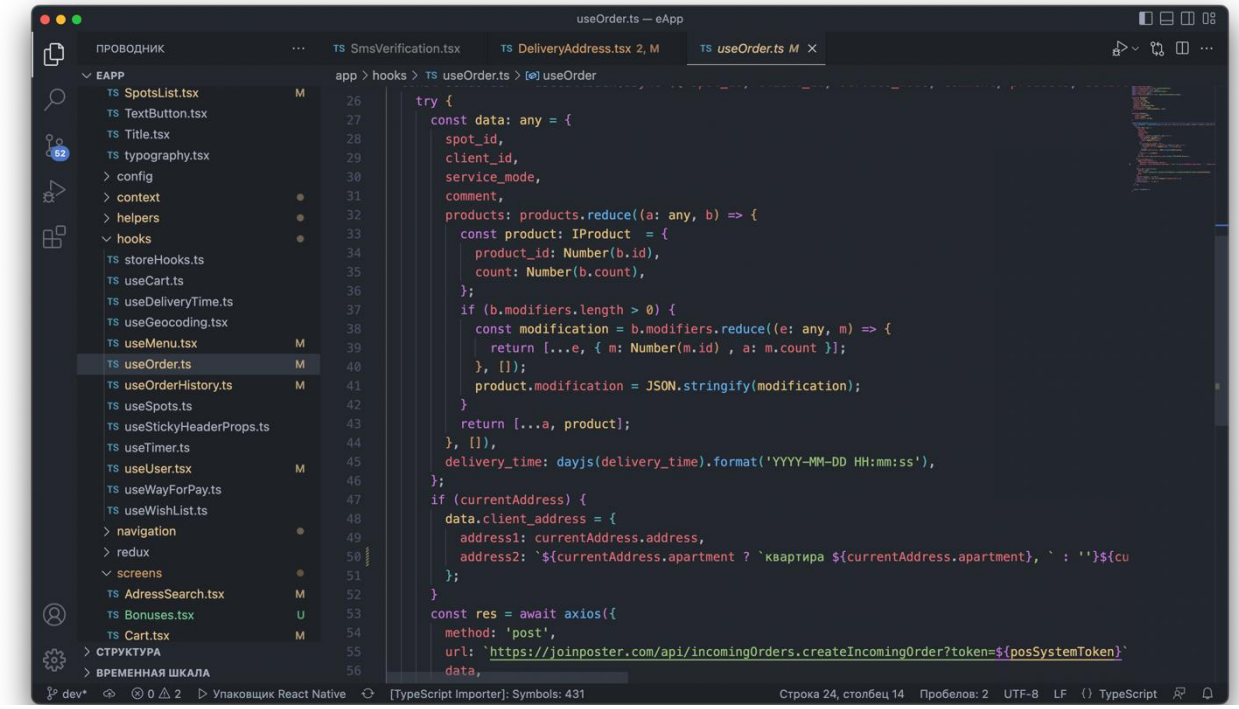

Рисунок 14.11 – Відправлення замовлення в POS-систему

Останнім пунктом в реалізації клієнтської частини додатку є налаштовування параметрів для ОС IOS.

3.11. Програмна реалізація серверної частини продукту

Перш за все було створено структуру JSON дерева в Firebase. Також за допомогою Firebase було реалізовано авторизацію в додаток, реєстрацію користувачів та систему push-повідомлень.

Наступним кроком є інтеграція бази даних Firebase з клієнтською частиною додатку за допомогою XCode IDE.

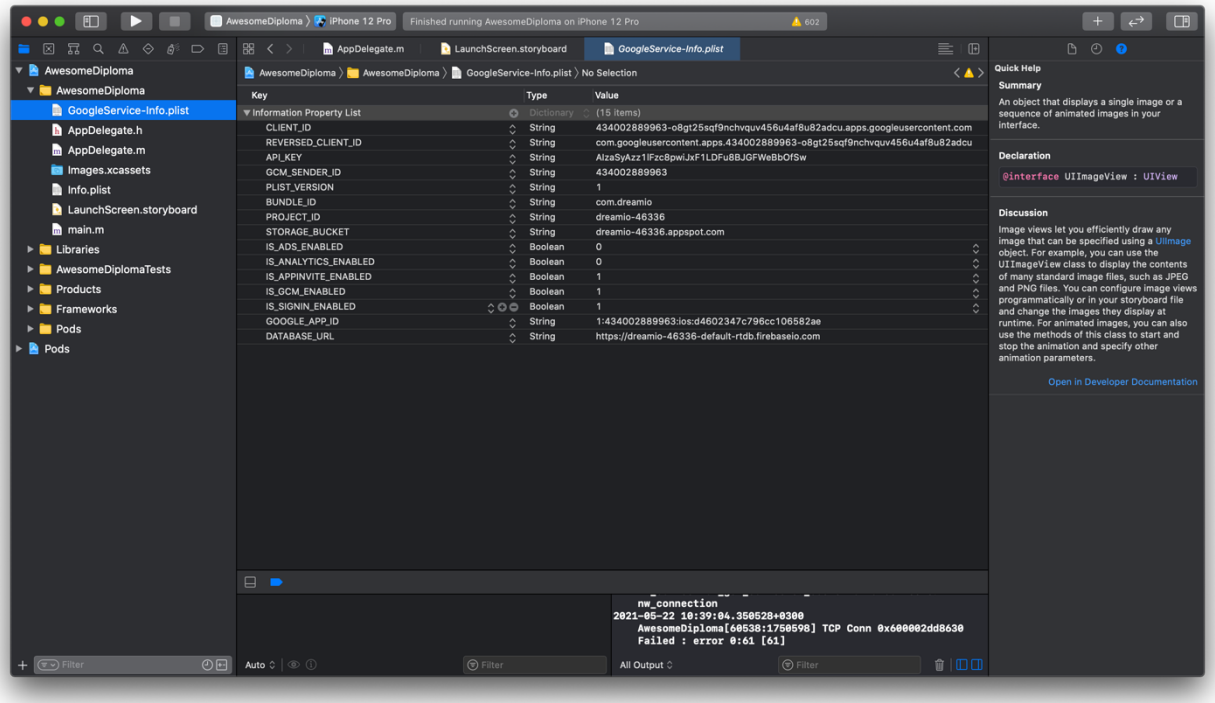

Рисунок 15 – Інтеграція бази даних з клієнтською частиною

На поточному етапі життєвого циклу мобільного доданку неможливо точно. визначити цільову. аудиторію, тому до рішення буде інтегровано сервіси з аналітичними системами, де можна аналізувати метрики для визначення поведінки користувача, та ще більш точно описувати їх портрети.

В якості рішення, було використано сервіс Amplitude.

Amplitude – платформа аналітики, це проміжний рівень між вихідними даними і дешбордами. Amplitude пропонує збільшити масштаб окремих точок для перегляду даних щодо поведінки користувачів та дій.

Потоки показують, які дії частіше роблять користувачі після передплати, і до покупки. Amplitude дозволяє вимірювати коефіцієнти конверсії за допомогою вирв для ідентифікації точок, в яких користувачі кидають реєстрацію, оновлення або покупку. Можна виміряти утримання користувача та частоту взаємодії з програмою. Користувачів можна поділити на сегменти та групи для порівняння за різними показниками. Основні характеристики Amplitude:

- Відстеження подій;
- утримання;

- перегляд у режимі реального часу;
- Формули;
- аналіз прибутків;
- гнучка сегментація;
- поведінкова аналітика;
- слідопит;
- розширений аналіз утримання;
- експорт даних;
- доступ до API;
- оповіщення електронною поштою;
- база знань;
- А/Б-тестування.

3.12. Підготовка до публікації

Для публікації мобільного додатку в магазини додатків App Store та Google Play потрібно підготувати матеріали для зовнішнього виду сторінки, зокрема:

- Логотип додатку;
- заголовок;
- короткий опис мобільного додатку;
- «скріншоти».

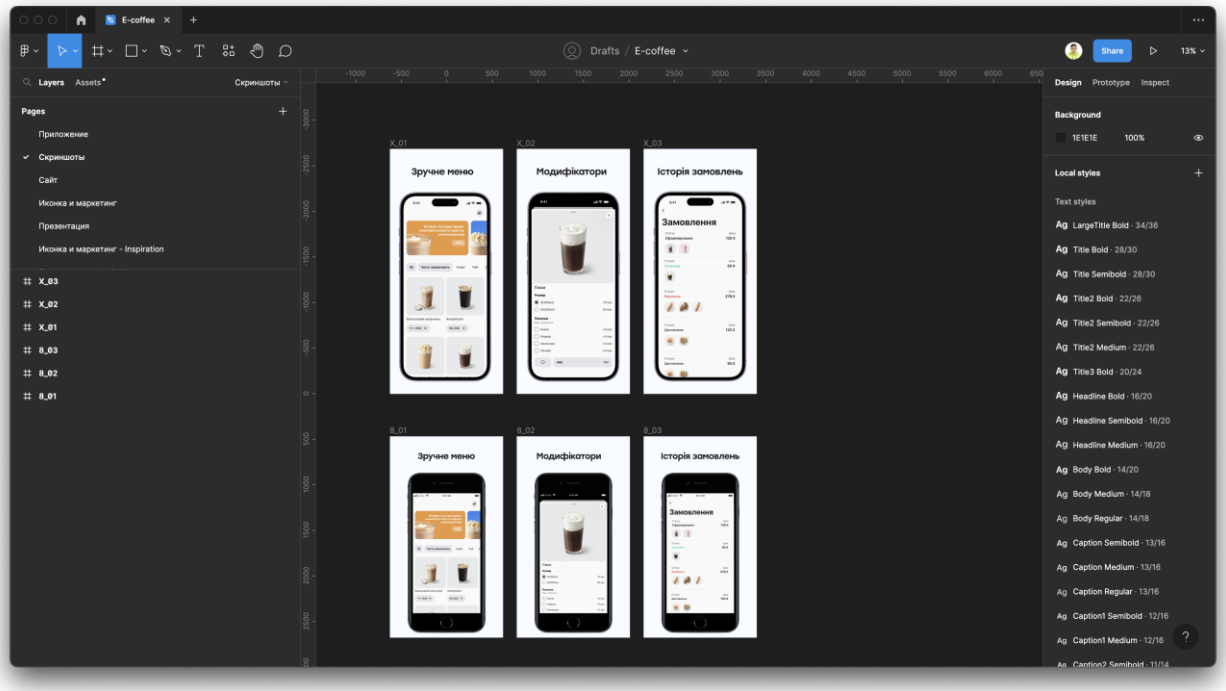

Рисунок 16 – Підготовка інтеграція бази даних з клієнтською частиною

3.13. Тестування додатку

Для стабільної роботи мобільного додатку необхідно протестувати на весь функціонал мобільного додатку – знайти та ліквідувати всі логічні та технічні помилки, помилки інтерфейсу. Для управляння помилками було використано сервіс Jira.

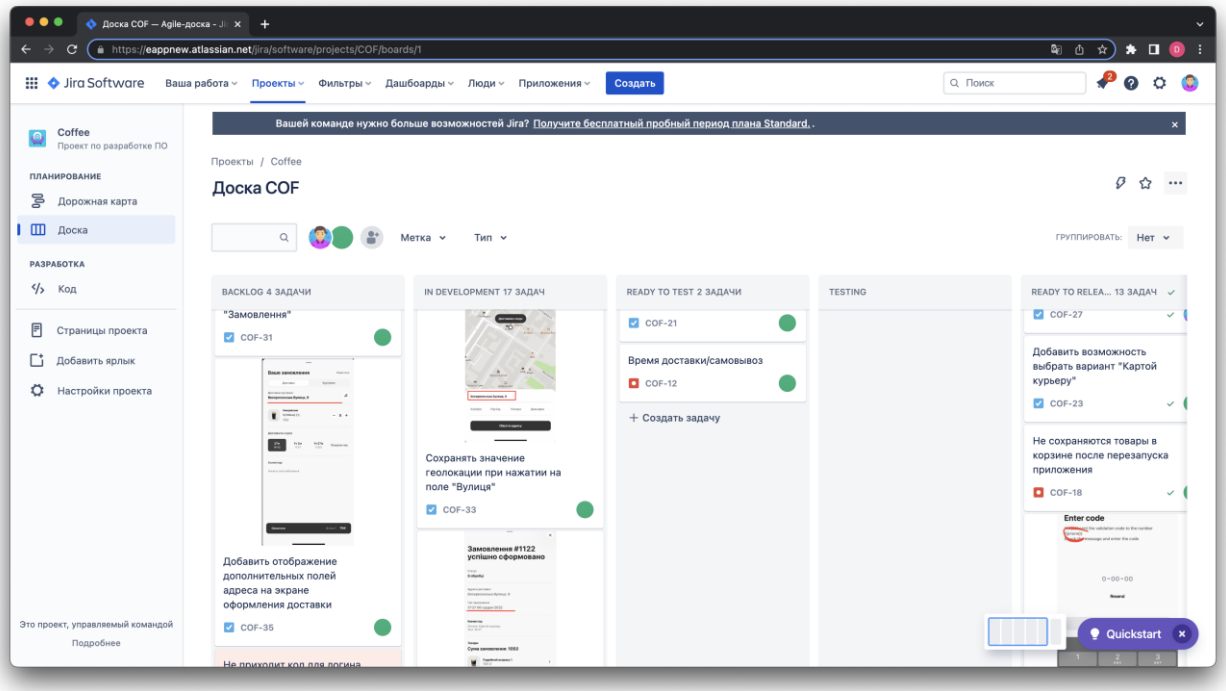

Рисунок 17 – Тестування мобільного додатку

Загалом, під час тестування мобільного додатку помилок з рівнем «critical» було виявлено: 32 логічні помилки, 52 технічні помилки та 15 помилок графічно інтерфейсу. не було знайдено.

Всі помилки були вирішені та виправленні. При публікації мобільного додатку в магазини додатків App Store та Google Play помилок з рівнем «critical» не було виявлено.

ВИСНОВКИ

Під час роботи над магістерською роботою перш за все було проведено аналіз проблеми: визначено предметну область, проаналізовано існуючі рішення та знайдено точки диференціації для проекту магістерської роботи. На основні аналізу предметної області було поставлено задачі на проект.

Отже, в результаті роботи було спроектовано white-label підхід до розробки мобільного додатку, що задовольняє всім вимогам сучасного конкурентного бізнесу.

На базі проведеного аналізу предметної області було спроектовано ІС мобільного додатку: спроектовано діаграму прецедентів, функціональну модель SADT, спроектовано діаграму потоків даних DFD, діаграму використання інформаційної системи та діаграму стану акторів. Була спроектована ERD діаграму серверної частини додатку.

Було досліджено методи і засоби програмної реалізації мобільного додатку. Для програмної реалізації клієнтської частини мобільного додатку було обрано кросплатформенну технологію «React Native» з основною мовою програмування Java Script. Даний підхід має ряд переваг: швидкість розробки завдяки єдиній кодовій базі для операційних систем «Android» та «IOS». Розроблений мобільний додаток було випробувано та протестовано. Він задовольняє всі вимогами поставлені під час проектування. Під час тестування мобільного додатку помилок з рівнем «critical» не було знайдено.

Загалом, в випускній роботі була повністю вирішена поставлена задача, розглянуто всі супутні завдання та вирішено проблеми, які виникали в ході виконання. Також було отримано нові знання з розробки програмних продуктів для мобільних систем.

СПИСОК ЛІТЕРАТУРИ

[1] Марка Девід, МакГоуен Клемент Методологія структурного аналізу та проектування SADT. 231 с.

[2] Terry Quatrani Visual modeling in IBM Rational Rose. Addison Wesley. 1998. 222 p.

[3] TMcElroy Kathryn Prototyping for Designers. O'Reilly Media Inc. 2017. 326 p.

[4] B. Eisenman Learning React Native. O'Reilly Media, Inc. 2017. 278 p.

[5] Д. УКРАЇНИ, Система розроблення та поставлення продукції на виробництво. Основні терміни та визначення. ДСТУ 3278-95, Київ. 1996.

[6] Russ Unger Carolyn Chandler A Project Guide to UX Design: For user experience designers in the field or in the making. New Riders. 2009. 267 p.

[7] Steve Diller, Nathan Shedroff, Darrel Rhea Diller Making meaning : how successful businesses deliver meaningful customer experiences, Berkeley, California New Riders. 2006. 160 p.

[8] Alexandr Stavonin. Developer Survey Results. 2019. URL: https://insights.stackoverflow.com/survey/

[9] Matthew Carrington Mobile App Development Process: Ultimate Guide To Build an App. 2022. URL: https://www.velvetech.com/blog/mobile-app-development-process/

[10] Bonnie Eisenman Learning React Native: Building Native Mobile Apps with JavaScript 2nd Edition. O'Reilly Media, Inc. 2017. 242 p

[11] Statista Cross-platform mobile frameworks used by developers worldwide 2019 and 2020 published by Shanhong Liu, Jul 2, 2020 React Native is the most popular cross-platform mobile framwork used by global developers, according to a 2020 developer survey. 2020. URL: https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours.

[12] Uoung Tools. 2020. Design Tools Survey,» 2020. URL: https://uxtools.co/survey-2020

[13] Marina Kovalyova Flux: React.js Application Architecture - A Comprehensive Study. 2018. URL: https://medium.com/@marina.kovalyova/flux-the-react-js-application-architecture-773f515d068d.

ДОДАТОК А

Лістинг файлу cartSlice.ts

```
import { FirebaseAuthTypes } from '@react-native-firebase/auth';
import React, { useState, createContext } from 'react';
import auth from '@react-native-firebase/auth';

interface FirebaseAuthContextModel {
  confirm: FirebaseAuthTypes.ConfirmationResult | null;
  signInWithPhoneNumber: (phone: string) => void;
  loading: boolean;
}

export const FirebaseAuthContext = createContext<FirebaseAuthContextModel>({
  confirm: null,
  signInWithPhoneNumber: () => {},
  loading: false,
});

interface Props {
  children: React.ReactNode;
}

export const FirebaseAuthProvider = ({ children }: Props) => {

  const [confirm, setConfirm] = useState<FirebaseAuthTypes.ConfirmationResult |
null>(null);
  const [loading, setLoading] = useState(false);

  const signInWithPhoneNumber = async (phone: string) => {
    setLoading(true);
    try {
      auth().settings.appVerificationDisabledForTesting = true;
      const confirmation = await auth().signInWithPhoneNumber(phone);
      setConfirm(confirmation);
    } catch {
      throw '';
    } finally {
      setLoading(false);
    }
  };


  return (
```

```
    <FirebaseAuthContext.Provider
      value={{
        confirm,
        signInWithPhoneNumber,
        loading,
      }}
    >
      {children}
    </FirebaseAuthContext.Provider>
  );
};
```

ДОДАТОК Б

Лістинг файлу cartSlice.ts

```
import { createSlice, PayloadAction } from '@reduxjs/toolkit';
import { CartState, ICartItem, IUpdateItemCount } from './types';

const initialState: CartState = {
  items: [],
};

const cartSlice = createSlice({
  name: 'cart',
  initialState,
  reducers: {
    addItem: (
      state: CartState,
      { payload }: PayloadAction<ICartItem>,
    ) => {
      state.items = [...state.items, payload];},
    updateCount: (
      state: CartState,
      { payload }: PayloadAction<IUpdateItemCount>,
    ) => {
      let index = payload.index || state.items.findIndex(it => it.id === payload.id);
      let items = [...state.items];
      items[index].count = payload.type === 'minus' ? items[index].count - 1 < 0 ? 0 :
items[index].count - 1 : items[index].count + 1;
      state.items = items;},
    deleteItem: (
      state: CartState,
      { payload }: PayloadAction<{id: string; index?: number;}>,
    ) => {
      let index = payload.index || state.items.findIndex(it => it.id === payload.id);
      let items = [...state.items];
      items.splice(index, 1);
      state.items = items;    },
    clearCart: (
      state: CartState,
    ) => {      state.items = [];},
  },
});

export const { addItem, updateCount, deleteItem, clearCart } = cartSlice.actions;
export const cartReducer = cartSlice.reducer;
```

Лістинг файлу navigationMainStack.ts

```
import React from 'react';
import { createStackNavigator, CardStyleInterpolators } from '@react-
navigation/stack';
import { ScreenEnum } from 'types/ScreenEnum';
import { Main } from 'screens/Main';
import { Profile } from 'screens/Profile';
import { Platform } from 'react-native';
import { useSafeAreaInsets } from 'react-native-safe-area-context';
import { Settings } from 'screens/Settings';
import { OrdersHistory } from 'screens/OrdersHistory';
import { EditProfile } from 'screens/EditProfile';
import { WishList } from 'screens/WishList';
import { HeaderBack } from 'components/HeaderBack';
import { ColorsEnum } from 'styles/colorsLightTheme';
import { Cart } from 'screens/Cart';
import { Product } from 'screens/Product';
import { OrderDetails } from 'screens/OrderDetails';
import { DeliveryAddress } from 'screens/DeliveryAddress';
import { AdressSearch } from 'screens/AdressSearch';
import { screenHeigh } from '../styles/constans';
import { RootStackParamList } from './RootStackParamList';


const MainStack = createStackNavigator<RootStackParamList>();

export const MainStackScreen = () => {
  const insets = useSafeAreaInsets();
  return (
```

```
    <MainStack.Navigator initialRouteName={ScreenEnum.Main}>
      <MainStack.Screen
        name={ScreenEnum.Main}
        component={Main}
        options={{
          headerShown: false,
          headerTitleAlign: 'center',
        }}
      />
      <MainStack.Screen
        name={ScreenEnum.Profile}
        component={Profile}
        options={{
          presentation: 'modal',
          gestureEnabled: true,
          gestureResponseDistance: Platform.OS === 'android' ? undefined :
screenHeigh,
          cardStyleInterpolator: Platform.OS === 'android' ?
CardStyleInterpolators.forVerticalIOS : undefined,
          animationEnabled: true,
          headerShown: false,
          cardStyle: Platform.OS === 'android' ? {
            marginTop: insets.top + 6,
            borderTopRightRadius: 10,
            borderTopLeftRadius: 10,
          } : {},
        }}
      />
      <MainStack.Screen
        name={ScreenEnum.Settings}
```

```
  component={Settings}
  options={{
    headerTitle: () => null,
    headerLeft: () => <HeaderBack />,
    headerStyle: {
      shadowColor: 'transparent',
      elevation: 0,
      backgroundColor: ColorsEnum.mainBackground,
    },
  }}
/>
<MainStack.Screen
  name={ScreenEnum.OrdersHistory}
  component={OrdersHistory}
  options={{
    headerTitle: () => null,
    headerLeft: () => <HeaderBack />,
    headerStyle: {
      shadowColor: 'transparent',
      elevation: 0,
      backgroundColor: ColorsEnum.mainBackground,
    },
  }}
/>
<MainStack.Screen
  name={ScreenEnum.EditProfile}
  component={EditProfile}
  options={{
    headerTitle: () => null,
    headerLeft: () => <HeaderBack />,
```

```
          headerStyle: {
            shadowColor: 'transparent',
            elevation: 0,
            backgroundColor: ColorsEnum.mainBackground,
          },
        }}
      />
      <MainStack.Screen
        name={ScreenEnum.WishList}
        component={WishList}
        options={{
          headerTitle: () => null,
          headerLeft: () => <HeaderBack />,
          headerStyle: {
            shadowColor: 'transparent',
            elevation: 0,
            backgroundColor: ColorsEnum.mainBackground,
          },
        }}
      />
      <MainStack.Screen
        name={ScreenEnum.Cart}
        component={Cart}
        options={{
          presentation: 'modal',
          gestureEnabled: true,
          gestureResponseDistance: undefined,
          cardStyleInterpolator: Platform.OS === 'android' ?
CardStyleInterpolators.forVerticalIOS : undefined,
          animationEnabled: true,
```

```
        headerShown: false,
        cardStyle: Platform.OS === 'android' ? {
          marginTop: insets.top + 6,
          borderTopRightRadius: 10,
          borderTopLeftRadius: 10,
        } : {},
      }}
    />
    <MainStack.Screen
      name={ScreenEnum.Product}
      component={Product}
      options={{
        presentation: 'modal',
        gestureEnabled: true,
        gestureResponseDistance: Platform.OS === 'android' ? undefined :
screenHeigh,
        cardStyleInterpolator: Platform.OS === 'android' ?
CardStyleInterpolators.forVerticalIOS : undefined,
        animationEnabled: true,
        headerShown: false,
        cardStyle: Platform.OS === 'android' ? {
          marginTop: insets.top + 6,
          borderTopRightRadius: 10,
          borderTopLeftRadius: 10,
          opacity: 1,
        } : {},
      }}
    />
    <MainStack.Screen
      name={ScreenEnum.OrderDetails}
```

```
        component={OrderDetails}
        options={{
          presentation: 'modal',
          gestureEnabled: true,
          gestureResponseDistance: Platform.OS === 'android' ? undefined :
screenHeigh,
          cardStyleInterpolator: Platform.OS === 'android' ?
CardStyleInterpolators.forVerticalIOS : undefined,
          animationEnabled: true,
          headerShown: false,
          cardStyle: Platform.OS === 'android' ? {
            marginTop: insets.top + 6,
            borderTopRightRadius: 10,
            borderTopLeftRadius: 10,
          } : {},
        }}
      />
      <MainStack.Screen
        name={ScreenEnum.DeliveryAddress}
        component={DeliveryAddress}
        options={{
          presentation: 'modal',
          gestureEnabled: true,
          gestureResponseDistance: Platform.OS === 'android' ? undefined :
screenHeigh,
          cardStyleInterpolator: Platform.OS === 'android' ?
CardStyleInterpolators.forVerticalIOS : undefined,
          animationEnabled: true,
          headerShown: false,
          cardStyle: Platform.OS === 'android' ? {
```

```
            marginTop: insets.top + 6,
            borderTopRightRadius: 10,
            borderTopLeftRadius: 10,
          } : {},
        }}
      />
      <MainStack.Screen
        name={ScreenEnum.AdressSearch}
        component={AdressSearch}
        options={{
          presentation: 'modal',
          gestureEnabled: true,
          gestureResponseDistance: Platform.OS === 'android' ? undefined :
screenHeigh,
          cardStyleInterpolator: Platform.OS === 'android' ?
CardStyleInterpolators.forVerticalIOS : undefined,
          animationEnabled: true,
          headerShown: false,
          cardStyle: Platform.OS === 'android' ? {
            marginTop: insets.top + 6,
            borderTopRightRadius: 10,
            borderTopLeftRadius: 10,
          } : {},
        }}
      />
    </MainStack.Navigator>
```

Лістинг файлу mainScreenDesign.tsx

```
import React, { useCallback, useState } from 'react';
import { CustomButton } from 'components/CustomButton';
import { AddressInput } from 'components/Form/AddressInput';
import { KeyboardAvoidingLayout } from 'components/KeyboardAvoidingLayout';
import { AppModalLayout } from 'components/Layout/AppModalLayout';
import { Loader } from 'components/Loader';
import { SView } from 'components/Styled/SView';
import { IAddress, useGeocoding } from 'hooks/useGeocoding';
import {  FlatList, Keyboard, Platform, StyleSheet, TouchableOpacity, View } from
'react-native';
import { ColorsEnum } from 'styles/colorsLightTheme';
import { AddressItem } from 'components/Cart/AddressItem';
import IconCross from 'assets/icons/cross.svg';
import { FormikProps, useFormik } from 'formik';
import { SFlex } from 'components/Styled/SFlex';
import { useDispatch } from 'react-redux';
import { updateCurrentAddress } from 'redux/deliveryAddress/deliveryAddressSlice';
import { useNavigation } from '@react-navigation/native';
import { ScreenEnum } from 'types/ScreenEnum';
import { useAppSelector } from 'hooks/storeHooks';
import { useTranslation } from 'react-i18next';

interface IFormikProps {
  apartment: string;
  entrance: string;
  floor: string;
  doorphone: string;
}
```

```
export const AdressSearch = () => {

  const {
    deliveryAddress: {
      currentAddress,
    },
  } = useAppSelector(state => state);
  const [addressVal, setAddressVal] = useState(currentAddress?.address || '');
  const [load, setLoad] = useState(false);
  const [addressList, setAddressList] = useState<Array<IAddress>>([]);
  const { searchAddress } = useGeocoding();
  const [showAdditionalData, setShowAdditionalData] = useState(true);
  const dispatch = useDispatch();
  const { navigate } = useNavigation();
  const { t } = useTranslation();

  const _searchAddress = useCallback(async (val: string) => {
    setAddressVal(val);
    if (val.length > 3) {
      setShowAdditionalData(false);
      setLoad(true);
      try {
        const addressList = await searchAddress(`${val}`);
        setAddressList(addressList || []);
      } catch {

      } finally {
        setLoad(false);
      }
```

```
    } else if (val.length === 0) {
      setAddressList([]);
      setShowAdditionalData(false);
    }
  }, [searchAddress]);

  const validateAddress = useCallback((address: string) => {
    return address.split(',').length === 2 && address.split(',')[1].replace(' ', '').length >
0;
  }, []);

  const getAddress = useCallback(async (data: IAddress) => {
    const validate = validateAddress(data.address);
    if (validate) {
      setAddressVal(data.address);
      setShowAdditionalData(true);
      setAddressList([]);
    } else {
      setAddressVal(`${data.address}, `);
    }
  }, [validateAddress]);

  const refreshSearch = () => {
    setShowAdditionalData(true);
    setAddressList([]);
    setAddressVal('');
  };


  const formik:FormikProps<IFormikProps>  = useFormik<IFormikProps>({
```

```
  initialValues: {
    apartment: '',
    entrance: '',
    floor: '',
    doorphone: '',
  },
  onSubmit: values => {
    dispatch(updateCurrentAddress({
      address: addressVal,
      location: { lat: 0, lng: 0 },
      secondaryAdress: '',
      ...values,
    }));
    Keyboard.dismiss();
    navigate(ScreenEnum.Cart);},
  enableReinitialize: true,});

return (
  <AppModalLayout
    backgroundColor={ColorsEnum.mainBackground}
    screenBackgroundColor={ColorsEnum.mainBackground}
  >
    <KeyboardAvoidingLayout keyboardVerticalOffset={50}>
      <View
        style={styles.modalContainer}
      >
        <View style={{ flex: 1 }}>
          <View style={styles.inputContainer}>
            <AddressInput
              value={addressVal}
```

```
            onChangeText={e => _searchAddress(e)}
            onPressIn={() => {}}
            placeholder={t('street')}
          />
          {load && (
            <View style={styles.inputRightBox}>
              <Loader size={22} color={'#848484'}/>
            </View>
          )}
          {!load && addressVal.length > 0 && (
            <TouchableOpacity onPress={refreshSearch}
style={styles.inputRightBox}>
              <IconCross />
            </TouchableOpacity>
          )}
        </View>
        {showAdditionalData && (
          <SFlex>
            <SView flex={1}>
              <AddressInput
                value={formik.values.apartment}
                placeholder={t('apartment')}
                onChangeText={formik.handleChange('apartment')}
                onBlur={formik.handleBlur('apartment')}
                onPressIn={() => {}}
              />
            </SView>
            <SView flex={1}>
              <AddressInput
                value={formik.values.entrance}
```

```
        placeholder={t('entrance')}
        onChangeText={formik.handleChange('entrance')}
        onBlur={formik.handleBlur('entrance')}
        onPressIn={() => {}}
      />
    </SView>
    <SView flex={1}>
      <AddressInput
        value={formik.values.floor}
        placeholder={t('floor')}
        onChangeText={formik.handleChange('floor')}
        onBlur={formik.handleBlur('floor')}
        onPressIn={() => {}}
      />
    </SView>
    <SView flex={1}>
      <AddressInput
        value={formik.values.doorphone}
        placeholder={t('doorphone')}
        onChangeText={formik.handleChange('doorphone')}
        onBlur={formik.handleBlur('doorphone')}
        onPressIn={() => {}}
      />
    </SView>
  </SFlex>
)}
<SView marginBottom={24}/>
<FlatList
  showsVerticalScrollIndicator={false}
  data={addressList}
```

```
            renderItem={({ item }) => (
              <AddressItem
                onPress={() => getAddress(item)}
                address={item.address}
                secondaryAdress={item.secondaryAdress}
              />
            )}
            ItemSeparatorComponent={() => <SView marginTop={24}/>}
            keyboardShouldPersistTaps={'handled'}
          />
        </View>
        <SView>
          <CustomButton
            text={t('confirmAddress')}
            onPress={formik.handleSubmit}
            disabled={!validateAddress(addressVal)}
          />
        </SView>
      </View>
    </KeyboardAvoidingLayout>
  </AppModalLayout>
);};

const styles = StyleSheet.create({
  modalContainer: {
    paddingHorizontal: 24,
    flex: 1,
    paddingTop: 50,
    paddingBottom: 20,},
  addressInput: {
```

```
    fontSize: 14,
    color: '#1B1B1B',
    lineHeight: 20,
    letterSpacing: -0.32,
    fontFamily: Platform.OS === 'android' ? 'Montserrat-Bold' : 'System',
    fontWeight: '700',
    borderBottomColor: '#E5E5EA',
    borderBottomWidth: 1,
    paddingVertical: 20,},
  form: {
    flex: 0,
  },
  inputContainer: {
    justifyContent: 'center',},
  inputRightBox: {
    position: 'absolute',
    right: 0,
    top: 0,
    bottom: 0,
    justifyContent: 'center',},
  line: {
    width: 2,
    backgroundColor: '#333333',
    height: 20,
    position: 'absolute',
    top: -12,
  },
});
```

Лістинг файлу cartScreenDesign.tsx

```
export enum ColorsEnum {
  'mainBackground' = '#F9F9F9',
  'mainText' = '#1B1B1B',
  'text' = '#595959',
  'overlay' = 'rgba(34, 32, 32, 0.5)',
  'handle' = '#989EA4',
  'background' = '#F2F2F2',
  'menuText' = '#404040',
}

export enum ButtonColorsEnum {
  'background' = '#333333',
  'disabledBackground' = '#E5E5EA',
  'disabledText' = '#848484',
  'text' = '#FFFFFF',
}

export enum ProductColorsEnum {
  'defaultButton' = '#F2F2F2',
  'textDefaultButton' = '#404040',
  'activeButton' = '#1B1B1B',
  'textActiveButton' = '#FFFFFF',
  'description' = 'rgba(0, 0, 0, 0.5)',
}

export enum CategoriesColorsEnum {
  'text' = '#595959',
  'activeText' = '#1B1B1B',
```

```
  'background' = 'transparent',
  'activeBackground' = '#E5E5EA',
}

export enum InputColorsEnum {
  'value' = '#1B1B1B',
  'placeholder' = '#989EA4',
}

export enum HeaderColorsEnum {
  'main' = '#404040',
  'text' = '#595959',
}
```

Лістинг файлу cartScreenDesign.tsx

```
import { SFlex } from 'components/Styled/SFlex';
import React from 'react';
import { View, StyleSheet, TouchableOpacity } from 'react-native';
import { StyledText } from '../typography';
import IconCheck from 'assets/icons/check.svg';
import { CountBtn } from 'components/Cart/CountBtn';
import { SView } from 'components/Styled/SView';
import { IModification } from 'redux/products/types';

interface IProps {
  item: IModification;
  active: boolean;
  maxCount: number;
  minCount: number;
  onPress: () => void;
  updateCount: (id: string, type: 'plus' | 'minus') => void;
  count: number;
  disabled: boolean;
}

export const GroupModifier = ({ item: { name, price, id }, active, count, onPress,
updateCount, disabled }: IProps) => {
  return (
    <SFlex style={styles.item}>
      <TouchableOpacity disabled={!active && disabled} onPress={onPress} style={{
flex: 1 }}>
        <SView flex={1}>
          <SFlex flex={1} alignItems={'flex-start'}>
```

```
          <View style={[styles.checkbox, {
            backgroundColor: active && disabled ? '#404040' : disabled ? '#F9F9F9' :
active ? '#404040' : '#F9F9F9',
            borderColor: active && disabled ? '#404040' : disabled ? '#E5E5EA' : active
? '#404040' : '#989EA4',
          }]}
          >
            {active &&  (
              <IconCheck/>
            )}
          </View>
          <SView flex={1} marginRight={12}>
            <StyledText
              extraFonts
              fontSize={14}
              lineHeight={20}
              fontWeight={'700'}
              letterSpacing={-0.32}
              color={'#595959'}
              ellipsizeMode={'tail'}
              numberOfLines={2}
            >
              {name}
            </StyledText>
          </SView>
        </SFlex>
        {active && (
          <StyledText
            extraFonts
            fontSize={12}
```

```
        lineHeight={16}
        fontWeight={'600'}
        letterSpacing={-0.2}
        color={'#404040'}
        marginTop={4}
      >
      +{price * count} грн
      </StyledText>
    )}
  </SView>
</TouchableOpacity>
{active ? (
  <CountBtn
    count={count}
    pressOnMinus={() => updateCount(id, 'minus')}
    pressOnPlus={() => updateCount(id, 'plus')}
    disabled={disabled}
  />
) : (
  <StyledText
    extraFonts
    fontSize={14}
    lineHeight={20}
    fontWeight={'700'}
    letterSpacing={-0.32}
    color={'#595959'}
  >
  +{price} грн
  </StyledText>
)}
```

```
    </SFlex>
  );
};

const styles = StyleSheet.create({
  checkbox: {
    width: 20,
    height: 20,
    borderRadius: 4,
    borderWidth: 2,
    borderColor: '#989EA4',
    alignItems: 'center',
    justifyContent: 'center',
    marginRight: 10,
  },
  activeCheckbox: {
    backgroundColor: '#404040',
  },
  item: {
    paddingHorizontal: 16,
    justifyContent: 'space-between',
    alignItems: 'center',
    flexDirection: 'row',
  },
});
```

ДОДАТОК В

Firebase структура

Firebase-root

| |

--- *users*

| | |

| --- *uid*

| | |

| --- *id*

| | |

| --- *name*

| --- *day of birthday*

| | |

| --- *month of birthday*

|

| --- *notification*

| | |

| --- *uid1*

| | |

| --- *img*

| | |

| --- *title*

| | |

| --- *subtitle*

| | |

| --- *text*

| | |

| --- *url*

|

| --- *item*

| | |

| --- *uid1*

| | |

| --- *img*

```
|              |
|              --- title
|              |
|              --- description
|              |
|              --- sale
|              |
|              --- url
|
|    --- sale
|          |
|          --- uid1
|              |
|              --- img
|              |
|              --- url_item
|              |
|              --- title
|              |
|              --- small_image
|              |
|              --- sale_log
|
|    --- order
|          |
|          --- uid1
|              |
|              --- time
|              |
|              --- adress
|              |
|              --- comment
|              |
|              --- bonuses
|              |
```

```
|
|           --- items
|                |
|                --- item_id
|                |
|                --- amount
|                |
|                --- comment
|                |
--- way
|   |
|   --- uid1
|       |
|       --- uid2
|               |
|               --- id
|               |
|               --- name
```