Міністерство освіти і науки України
Криворізький національний університет
Факультет інформаційних технологій
Кафедра автоматизації, комп'ютерних наук і технологій

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти – бакалавр
за освітньо-професійною програмою
«Комп'ютерні науки»

зі спеціальності
122 – Комп'ютерні науки

тема роботи:

«Розробка мобільного додатку для продажу кави та чаю з використанням технології React Native»

Виконав студент гр. КН-20 _____ Кацалап О.С.

Керівник _____ Рубан С. А.

Нормоконтроль _____ Маринич І. А.

Завідувач кафедри _____ Рубан С. А.

Кривий Ріг – 2024

КРИВОРІЗЬКИЙНАЦІОНАЛЬНИЙУНІВЕРСИТЕТ

Факультет:*інформаційних технологій*

Кафедра:*автоматизації, комп'ютерних наук і технологій*

Ступінь вищої освіти: *Бакалавр*

Спеціальність:*122 – Комп'ютерні науки*

ЗАТВЕРДЖУЮ

Зав. кафедрою: *к.т.н. Рубан С.А.*

_____

« *27*»*березня* 2024 р.

ЗАВДАННЯ

на кваліфікаційну роботу бакалавра

студентові групи КН-20 Кацалап Олександри Сергіївни

1. Тема кваліфікаційної роботи: «Розробка мобільного додатку для продажу кави та чаю з використанням технології React Native»

затверджено наказом по університету № *235с* від *27.03.2024 р.*

2.Термін здачі кваліфікаційної роботи:*05.06.2024 р.*

3.Склад кваліфікаційної роботи:*Пояснювальна записка обсягом 64с., додатки, презентація у Microsoft PowerPoint (11 слайдів) в електронному та друкованому вигляді*

4. Консультанти кваліфікаційної роботи:

Розділ 1-2 доц. Рубан С. А.

_____

Нормоконтроль доц. Маринич І. А.

5. Календарний план:

№	Етапи роботи	Термін виконання
1	*Вступ*	*01.04.24*
2	*Розділ 1*	*05.04.24*
3	*Розділ 2*	*01.05.24*
4	*Висновки*	*25.05.24*
5	*Оформлення кваліфікаційної роботи*	*28.05.24*
6	*Підготовка презентації та графічного матеріалу*	*20.05.24*
7	*Підготовка доповіді до захисту*	*05.06.24*

6. Дата видачі завдання: *29.01.2024р.*

Керівник _____ /РубанС. А./

7. Запевнення: *Я, Кацалап Олександра Сергіївна, запевняю, що ця кваліфікаційна робота виконана самостійно, не містить академічного плагіату, фабрикації, фальсифікації. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.*

Із чинним Положенням про академічну доброчесність Криворізького національного університету ознайомлений.

Чітко усвідомлюю, що в разі виявлення у кваліфікаційній роботі умисних порушень робота не допускається до захисту або оцінюється незадовільно.

Студент _____ /Кацалап О.С./

АНОТАЦІЯ

Кацалап О.С. Розробка мобільного додатку для продажу кави та чаю з використанням технології React Native .

Кваліфікаційна робота на здобуття ступеня вищої освіти – бакалавр, за спеціальністю 122 Комп'ютерні науки. Криворізький національний університет, Кривий Ріг, 2024.

Робота складається зі вступу, двох розділів, висновків, переліку використаної літератури з 22 позиції та 1 додатку. Загальний обсяг роботи становить 75 сторінок, з яких основний зміст роботи викладено на 60 сторінках, включає 30 рисунків.

Було розглянуто ринок мобільних застосунків, подано статистичні дані про стан ринку та валідність впровадження мобільних застосунків для торгівлі. В ході дослідження було розглянуто актуальні мобільні операційні системи, фреймворки для розробки нативних та кросплатформенних додатків а також платформи для розробки, приведено дані щодо розповсюдженості фреймворків для розробки мобільних кросплатформенних застосунків. В результаті було створено основний функціонал для мобільних додатків з продажу та розроблено діаграму варіантів використання мобільного застосунку з продажу чаю та кави.

Використовуючи платформу для розробки Visual Studio Code та Android Studio було розроблено кросплатформенний мобільний застосунок з продажу чаю та кави який включає функціонал мобільних додатків для продажу. Додаток було розроблено на основі мови програмування React Native. До застосунку надано інструкцію з використання з реальним виглядом екранів мобільного додатку.

Ключові слова: REACT NATIVE, ANDROID, IOS, ANDROID STUDIO, VISUAL STUDIO CODE, ОПЕРАЦІЙНА СИСТЕМА.

ЗМІСТ

ВСТУП

В минулому столітті було створено та вдосконалено перший комп’ютер, пізніше – персональний комп’ютер, далі телефон і смартфон. Світ продовжує змінюватись і з ним змінюються реалії життя людей.

На сьогоднішній день постійний доступ до інтернету, наявність персональних портативних пристроїв комунікації та цифровізації дозволяють зробити більш доступними та мобільними всі сфери нашого життя, а всесвітня пандемія 2020 року лише прискорила темпи світової цифровізації.

Однією із сфер що найбільше змінилась із розвитком технологій є сфера торгівлі, а саме з’явилась електронна торгівля, яка розширила ринок торгівлі надавши можливість покупцям здійснювати покупки або отримувати послуги використовуючи інтернет та отримувати товари чи послуги які були їм недоступні або їх отримання було занадто витратне у минулому.

Основним інструментом електронної комерції є онлайн-платформи (сайти та мобільні застосунки), що забезпечують перегляд, фільтрацію, пошук та інші функції для покупців. Хоча деякі сайти й націлені на клієнтів типу B2B, здебільшого електронна комерція розширила та облегшила роздрібну торгівлю, зменшивши кількість посередників між виробником та клієнтом та здешевивши товар. Саме завдяки розвитку елекронної комерції збільшується попит на створення мобільних застосунків та сайтів з типом B2C. Такий принцип дозволив бізнесу невеликих та середніх розмірів представляти себе на ринку без посередництва великих мереж та напряму контактувати з клієнтами, що облегшило отримання зворотній зв’язок від користувачів. За даними Statista у 2024 році роздрібні продажі електронної комерції в усьому світі перевищать 6,3 трильйона доларів США, і очікується, що ця цифра досягне нових висот у найближчі роки.

Оскільки мобільні пристрої не потребують великих вкладень та більш зручні у щоденному використанні порівняно з іншими електронними пристроями, їх використання як засіб оформлення покупок переважає

порівняно з використання комп’ютерів та інших гаджетів. Станом на 2023 рік на смартфони припадало 80% відвідуваня всіх сервісів роздрібної торгівлі у світі та за їх допомогою генерується 66% кількість онлайн замовлень [1].

Подібні тенденції на ринку створюють попит на розробку та підтримку кросплатформених мобільних застосунків. В даній дипломній роботі на прикладі розробки мобільного застосунку з продажу чаю та кави ми ознайомимось з основними аспектами розробки мобільних застосунків використовуючи популярний фреймворк “React Native”.

РОЗДІЛ 1.
ДОСЛІДЖЕННЯ АКТУАЛЬНОСТІ РОЗРОБКИ МОБІЛЬНИХ ЗАСТОСУНКІВ

1.1. Актуальність мобільних застосунків

В наш час мобільні застосунки стали невід'ємною частиною повсякденності, вони надають нам доступ до отримання послуг, освіти, медицини, урізноманітнюють наше дозвілля та розширюють межі світосприйняття, мобільні застосунки змінюють спосіб яким ми взаємодіємо з технологіями та світом.

З розвитком цифровізації актуальність мобільних застосунків лише росте, вже в 2024 кількість власників телефонів перевищила 4,8 мільярда людей, а до 2027 прогнозується що їх кількість буде близько 6 мільярдів

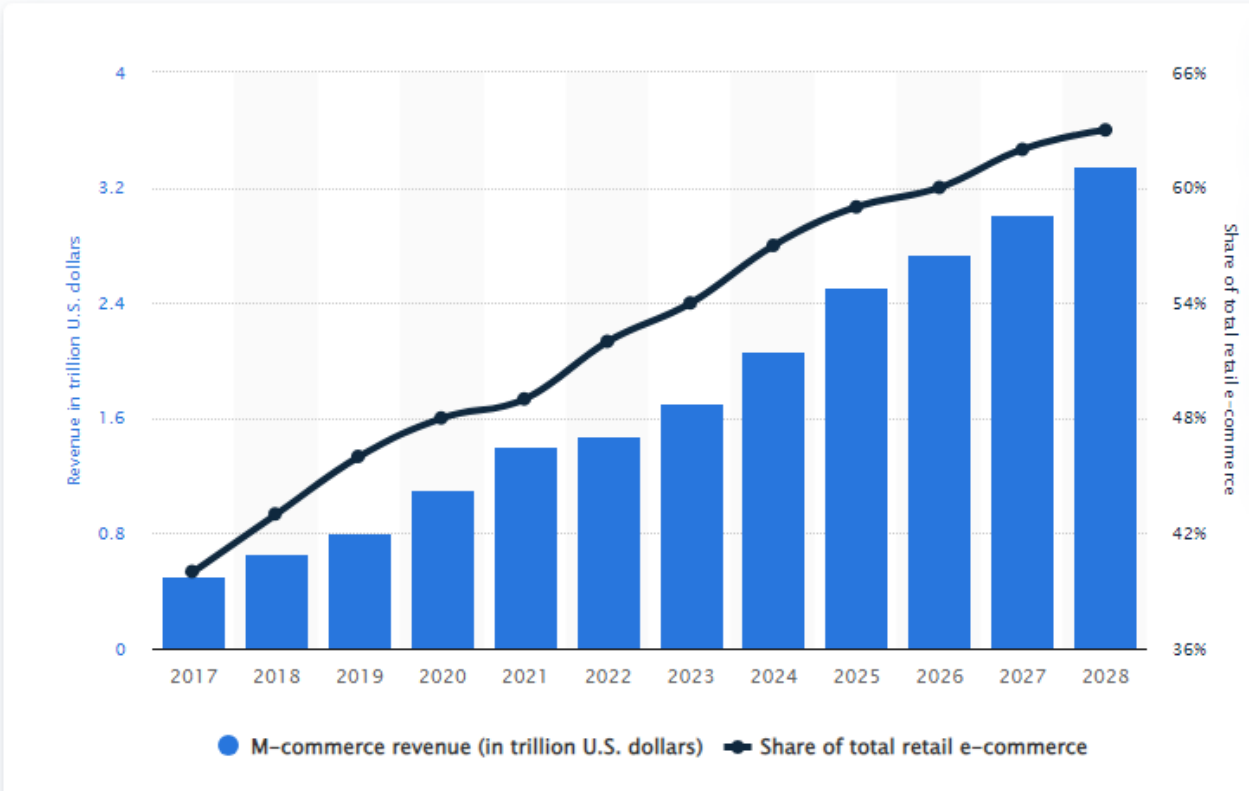

Рисунок 1.1 – Дохід від мобільної комерції та частка загальної роздрібної електронної комерції в усьому світі з 2017 по 2028 рік[4]

. В середньому в 2023 році людина в день витрачала 4,5 години в смартфоні, при цьому 3,5 години витрачаються у мобільних застосунках, а на вебсайти приходиться лише близько 50 хвилин на день [3].

Значним поштовхом до створення мобільних застосунків є попит, який створює бізнес, адже великою частиною електронної комерції є мобільна комерція. На рисунку 1.1 можна побачити, що ринок електронної комерції росте, в 2024 році ринок мобільної комерції оцінюється в 2,07 трильйонів доларів і займав 54% загальної електронної комерції, до 2027 року планується що ринок зросте до 3,35 трильйонів доларів і буде мати частку в 63 %.

Електронна комерція включає не тільки продаж фізичних товарів, є безліч бізнес моделей для отримання прибутку, найбільш популярні в наш час підписка та преміям, але зважаючи на перехід фізичних бізнесів в онлайн також є попит і на створення ПЗ для покупок. З точки зору бізнесу важливу роль відіграє окупність вкладень.

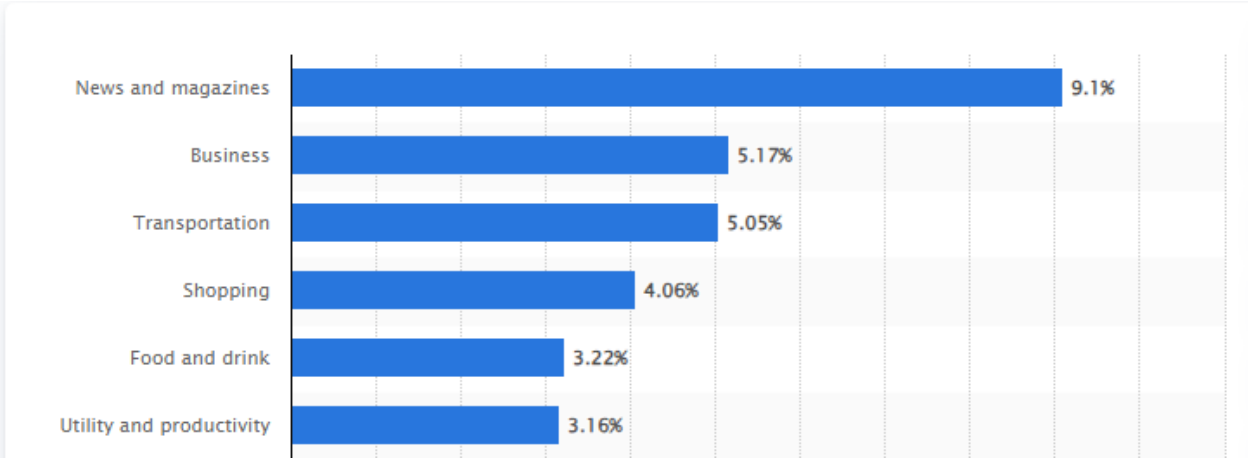

Рисунок 1.2 – Рівень утримання на 30-й день встановлення мобільних додатків у всьому світі в 3-му кварталі 2023 року за категоріями

За даними Satista в рейтингу коефіцієнту утримання користувача протягом 30 днів після установки в третьому кварталі 2023 року категорія

«покупки» займає четверте місце і має близько 4 відсотків утримання. Ще один позитивним ключем є можливість реклами в інтернеті, адже заволікти покупця набагато легше в онлайн магазин ніж в реальний, а також онлайн модель бізнесу дозволить знизити витрати на персонал, приміщення та супутні витрати.

Для користувачів робити покупки через мобільні додатки набагато безпечніше, оскільки використовуючи вебсайти можна легко натрапити на підробні сайти створені для викрадання особистої інформації або коштів, в порівнянні з мобільними за стосунками які неможливо підробити.

Виходячи із статистичної інформації останніх років можна з впевненістю сказати, що сфера створення, розвитку та підтримки мобільних застосунків, у тому числі для продажу, є актуальною і перспективною.

1.2. Мобільні операційні системи

Для успішної розробки мобільних додатків надзвичайно важливо враховувати операційну систему, для якої призначений додаток. Дві найбільш поширені платформи – Android та iOS – мають свої особливості, вимоги та стандарти, що значною мірою впливають на процес розробки та остаточний продукт. Додаток має бути точно узгодженим з обладнанням, на якому він буде використовуватися, що забезпечить його стабільну та ефективну роботу. Важливо також враховувати сумісність програмного забезпечення з іншими пристроями, з якими додаток може взаємодіяти. Це може включати різноманітні гаджети, сенсори та інші смарт-пристрої, що формують сучасний екосистему користувача. Розробники повинні ретельно тестувати додаток на відповідність усім необхідним стандартам та сумісність з широким спектром пристроїв, щоб забезпечити найкращий досвід для користувачів.

1.2.1. ОС Android

Android – це операційна система для мобільних пристроїв, розроблена компанією Google, яка вперше була представлена у вересні 2008 року. Остання версія цієї операційної системи – Android 14 була випущена у 2023 році. є найбільш поширеною операційною системою для мобільних пристроїв, займаючи близько 70% ринку мобільних ОС, що представлено на рисунку 1.3, і базується на ядрі Linux.

Рисунок 1.3 – Частка ринку мобільних операційних систем у світі – квітень 2024 року[7]

На сьогодні ОС Android викуплена Google які створили корсорціум Open Handset Alliance (OHA) ціллю якого є розвиток стандартів для мобільних пристроїв та просування системи Android. До складу консорціуму входять провідні компанії областей розробки програмного забезпечення, оператори мобільного зв’язку та виробники смартфонів.

Але відкритість ОС та її розповсюдженість надає користувачам більшу свободу аніж iOS, а також наражає їх на небезпеку у вигляді зловмисників та зловмисних програмних застосунків, 97% зловмисних програм націлено саме на Android.

Не зважаючи на небезпеку використання Android в порівнянні з iOS вона все ще залишається більш популярною, що мотивує розробників в першу чергу орієнтуватись на розробку застосунку для цієї ОС.

1.2.2. ОС iOS

iOS – це операційна система для мобільних пристроїв, розроблена компанією Apple, яка вперше була представлена у червні 2007 року, за рік до виходу Android. Остання версія цієї операційної системи – iOS 17.5. близько

27% мобільних пристроїв у світі працюють під управлінням iOS. Ця частка ринку є досить значною, враховуючи, що iOS використовується лише на пристроях Apple. Завдяки високій якості продуктів Apple та лояльності клієнтів, iOS залишається популярною серед користувачів по всьому світу. За даними на 2024 рік, кількість активних пристроїв Apple, що працюють на iOS, перевищує 1,5 мільярда.

Для розробки додатків для iOS необхідне середовище розробки, яке називається Xcode. Це офіційне інтегроване середовище розробки (IDE) від Apple, яке надається безкоштовно. Xcode включає всі необхідні інструменти для створення, тестування та налагодження додатків для iOS. Важливо зазначити, що Xcode працює лише на комп'ютерах Apple або пристроях під керуванням macOS, що є ще одним аспектом ексклюзивності платформи iOS та вимагає від розробників встановлювати емулятори або використовувати оригінальну продукцію Apple для розробки.

1.3. Мови програмування для розробки нативних додатків

Нативними називають застосунки розроблені під певну операційну систему, в здебільшого це iOS чи Android. Для кожної ОС є перелік мов програмування та середовищ які використовуються для розробки саме під образу платформу.

Перевагою нативних додатків є швидкість, адже вони створюються під певну ОС, а отже розробник може підлаштувати додаток конкретно від певні вимоги. Також користувачам більш інтуїтивно зрозуміло як користуватись такими додатками так як такі додатки відповідають інтерфейсу платформи та стандартам дизайну. Також нативні додатки можуть використовувати функції біометричні методи ідентифікації та інтегруватись з іншими функціями пристрою, такими як мікрофон, сповіщення, вібрація.

Але усі переваги несуть за собою і недоліки, зокрема орієнтованість додатку на певну платформу робить його недоступним для користувачів

інших платформ, для його інтеграції з ними потрібно перепрограмувати та оновити додаток, а також пройти затвердження магазином за стосунків, особливо складний та довгий процес перевірки у App Store. Сама розробка додатку доволі складна та потребує кваліфікованого спеціаліста який володіє певною нативною мовою програмування. Цей аспект збільшує вартість розробки та оновленя застосунку.

1.3.1. Додатки для iOS

Насьоогодні існує дві сови програмування за допомогою яких можна створити нативні застосунки для iOS а також iPadOS, masOS та watchOS:

Swift – це сучасна мова програмування, розроблена Apple і вперше представлена у 2014 році. Swift відома своєю безпекою, продуктивністю та простотою у використанні. Вона дозволяє розробникам писати чистий та ефективний код, а також надає сучасні можливості для розробки мобільних додатків. Ця мова програмування має відкритий репозиторій з код та сучасний синтаксис, що дозволяє швидше її опанувати. Відкритість коду дозволяє розробникам з усього світу впливати на розвиток мови програмування та акцентує зацікавленість Apple в розвитку Swift та бажання компанії зробити її основною мовою для розробки додатків на ОС iOS.

Objective-C – це об’єктно-орієнтована мова програмування що була створена в 1983 році,вона є частиною мов програмування C та використовувалася для розробки додатків для iOS до появи Swift. Objective-C має динамічне середовище розробки. Дотого ж внутрішній код ядра iOS написаний саме на C++ та C, що дозволяє Objective-C більш вдало ніж Swift інтегруватись в ОС iOS.

Objective-C все ще підтримується і використовується у багатьох існуючих проєктах, але нові розробники здебільшого переходять на Swift який активно розвивається за допомогою компанії Apple. Але відкидати вивчення Objective-C не варто, оскілки часто низько рівневі задачі або , до прикладу, управління пам’яту, набагато легше робити на Objective-C ніж на

Swift, а також більша частина кодової бази, аку потрібно підтримувати та оновлювати, написана саме на цій мові.

1.3.2. Додатки для Android

Розробка нативних мобільних застосунків для Android передбачає створення додатків, які використовують рідні API та інструменти платформи Android. Це дозволяє досягти високої продуктивності, доступу до всіх функцій пристрою та забезпечити користувачам найкращий досвід.

Основною мови програмування для Android є Java. Ця мова програмування є об’єктно-орієнтованою, яка забезпечує стабільність та надійність. Вона має обширну спільноту розробників, що сприяє швидкому вирішенню проблем та доступу до ресурсів та велику кількість бібліотек, що розширює обсяг можливих завдань. Код, написаний на Java, компілюється у байт-код, який може виконуватись на будь-якій платформі, що має Java Virtual Machine, що дозволяє запускати Java-додатки на різних пристроях.

Kotlin є сучасною мовою програмування, яка поєднує простоту та безпеку. Вона була вперше представлена в 2011 році та створена для для усунення недоліків Java, а вже в 2017 Google оголосив про підтримку Kotlin як основної мови програмування для розробки додатків для Android.

Kotlin є статично типізованою мовою програмування, що компілюється у байт-код, який може виконуватись на Java Virtual Machine. Це дозволяє Kotlin інтегруватись з існуючим Java-кодом, що робить його дуже привабливим для розробників, які бажають поступово мігрувати на нову мову без необхідності переписувати весь існуючий код. Також Kotlin має більш сучасний та компактний синтаксиз, в Kotlin код класу займе 1 рядок, у той час як у Java більше 10 рядків, оскільки Java вимагає написання великої кількості шаблонного коду.

Ще однією перевагою Kotlin є швидкість компіляції яка значно перевищує Java за рахунок лаконічності коду та використання інтелектуальної компіляції.

Найбільш популярним середовищем розробки мобільних застосунків для Android є Android Studio.

Android Studio – це середовище розробки створене Google та доступна безкоштовно до завантаженя, вона працює у режимі неперервної інтеграції коду, що забезпечує швидке виявленя помилок, а після закінчення розробки Android Studio приведе код до формату належного для завантаження в магазин за стосунків. Також Android Studio не має вимог до того, на яку операційну систему вона встановлюється: Windows, Linux чи macOS

Нативна розробка для Android забезпечує розробникам потужні інструменти та можливості для створення високоякісних мобільних додатків. Використання Java або Kotlin у поєднанні з Android Studio дозволяє створювати додатки, які повністю використовують потенціал платформи Android, забезпечуючи користувачам найкращий досвід.

1.4. Кросплатформні фреймворки

Оскільки створення нативних додатків вимагає від програміста знання мови відповідно до ОС, а також неможливість створювати одночасно програму для різних платформ, витрати на розробку ростуть, що невигідно для малого та середнього бізнесу. Для розробки додатку, який не вимагає особливого доступу до функціоналу пристрою, чутливість мобільного додатку, що важливо для мобільних ігор, та не повинен працювати в офлайні, більш підходить розробка кросплатформеного застосунку.

Кросплатформені застосунки можуть працювати на декількох ОС, це забезпечується за допомогою використання при розробці високорівневої мови програмування. В основі роботи таких за стосунків лежить браузер, що надає адаптивність застосунку до різних ОС та пристроїв та робить застосунок доступним для більшої кількості користувачів, але для забезпечення коректної роботи застосунку на різних платформах під кожну з них потрібно написати додатковий код.

Важливою частиною розробки є виконання гайдлайнів ОС, на яких планується використання застосунку, оскільки застосунок налаштований для Android буде некоректно відображатись на IOS та інших платформах.

За даними сервісу Statista а топ-5 найбільш популярних фреймворків для розробки кросплатформених мобільних застосунків входять: Flutter, React Native, Cordova, Ionic, Xamarin. За даними про популярність фреймворків які представлені на рисунку 1.4 можна побачити що фреймворк Cordova втрачає популярність, така ж тенденція зберігалась і в 2023 році, це зумовлено її обмеженістю в продуктивності та покращеню вебтехнологій та вебдодатків.

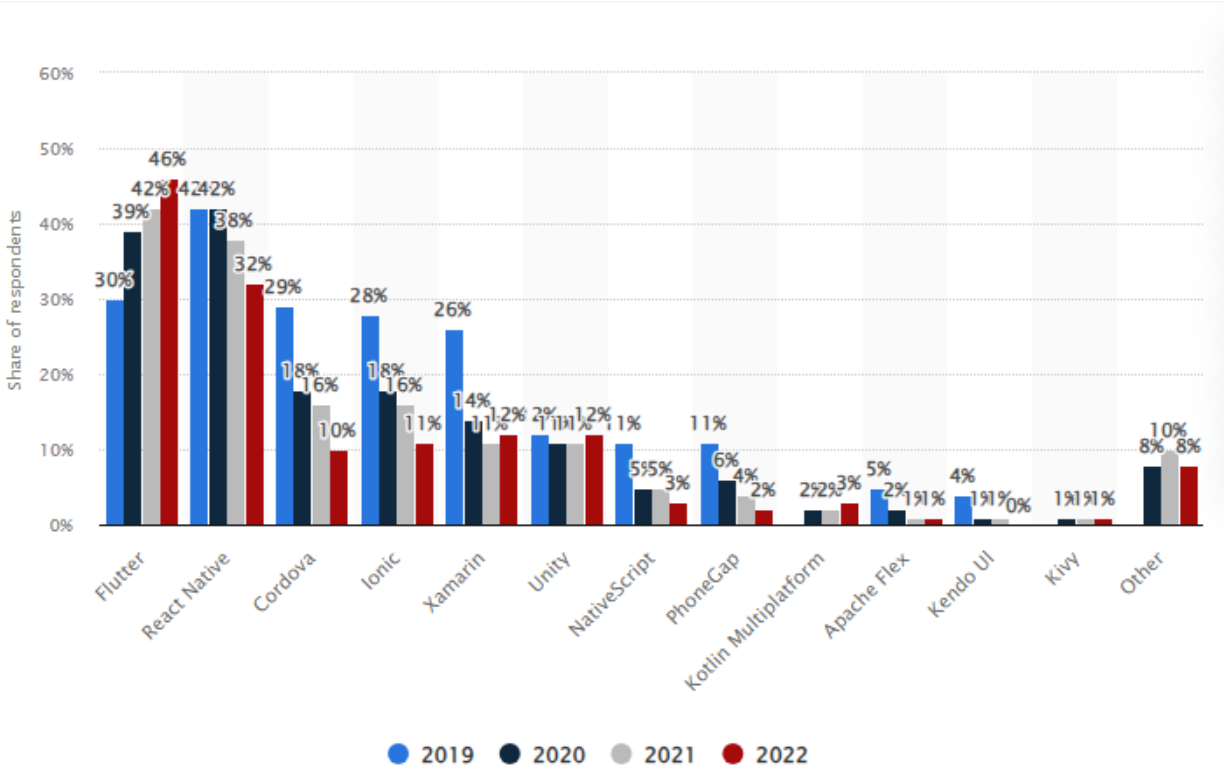

Рисунок 1.4 – Кросплатформні мобільні фреймворки, які використовували розробники програмного забезпечення по всьому світу з 2019 по 2022 рік [8]

При цьому фреймворк Xamarin стабілізувався в популярності за останні роки завдяки підтримці Microsoft та інтеграцію з C# та .NET. Незмінними лідерами залишаються Flutter та React Native.

Загалом приблизно третина розробників мобільних додатків використовують кросплатформені застосунки і враховуючи що за 2024 рік дохід від ринку розробки мобільних за стосунків може вирости приблизно на 12 відсотків сфера кросплатформеної мобільної розробки все ще є актуальною.

1.4.1. Xamarin та .NET MAUI

Xamarin – це фреймворк створений в 2011 році і викуплений Microsoft у 2016 році, який надає інтелектуальний спосіб розробки. За допомогою цього фреймворка пишуться додатки на мові C# в Visual Studio.

Xamarin дозволяє швидко створювати та випускати додатки. У травні 2024 року в останньому оновленні Xamarin було додано декілька нових функцій та покращень: Xamarin.Forms 6.0, Performance Improvements, Hot Reload and Hot Restart Enhancements, Shell Improvements, MAUI Integration.

Додатки, створені на платформі Xamarin, відповідають стандартам нативної розробки. Проте, Xamarin не рекомендується для додатків, що потребують інтенсивної графіки, через те, що кожна платформа має власні методи візуального компонування екранів.

Архітектура Xamarin включає платформу візуального дизайну для створення нативних додатків. Графічний дизайн iOS забезпечується через їх IDE, що допомагає розробникам відкривати X-Code. LINQ можна використовувати з колекціями або створювати налаштовані делегати та події, що звільняє розробників від обмежень Objective-C та Java.

Xamarin використовує архітектури MVC та MVVM, які прискорюють розробку з меншими витратами. Модель MVVM дозволяє розробляти різні процеси з єдиною кодовою базою, тоді як модель MVC допомагає розділяти логіку представлення та логіку додатка, що прискорює процес розробки.

Xamarin дає можливість створювати інтерфейси з використанням платформно-орієнтованих елементів користувацького інтерфейсу. Також можна створювати кросплатформенні додатки для iOS, Android та Windows за допомогою інструменту Xamarin.Forms, який перетворює компоненти інтерфейсу користувача на платформно-орієнтовані елементи під час виконання. Розширення Xamarin.Forms збільшує швидкість розробки додатків, що є чудовим варіантом для бізнес-орієнтованих проєктів.

Хоча додаткова абстракція може знизити продуктивність через надбудови, для покращення інтерфейсу користувача та продуктивності можна використовувати Xamarin.iOS і Xamarin.Android окремо для досягнення відмінних результатів.

Перевагою Xamarin є використання C#, який добре працює на різних платформах, таких як Android та iOS. Спільнота налічує понад 50 000 розробників і більше 3000 компаній. Розробники мають можливість повторного використання до 96% вихідного коду завдяки використанню платформи C# + .NET для створення додатків для iOS та Android.

Інтеграція з Visual Studio без додаткових витрат.

Розробка додатків за допомогою Xamarin може бути дорогою для компаній, оскільки для цього потрібна ліцензія на Visual Studio від Microsoft. Також не рекомендується для додатків з інтенсивною графікою через специфіку методів візуального проєктування екранів.

Серед найкращих додатків, розроблених на Xamarin: HCL Connections, American Cancer Society, Fox Airports, Alaska Airlines.

Нажаль з травня 2024 року компанія Microsoft опублікувала останнє оновлення мови Xamarin та оголосило про припинення підтримки мови, що означає відсутність подальших оновлень та виправлень, компанія Microsoft активно підтримує перехід розробників на .NET MAUI. .NET MAUI є еволюцією Xamarin.Forms і пропонує розширені можливості та поліпшення для розробки кросплатформених додатків.

.NET MAUI представляє собою потужний інструмент для розробників, які бажають створювати високоякісні, кросплатформні додатки з використанням сучасних технологій. Її глибока інтеграція з .NET 6, покращена продуктивність та спрощена модель розробки роблять її привабливим вибором для нових проєктів. З її допомогою розробники можуть швидко та ефективно створювати додатки, які працюють на різних платформах, забезпечуючи найкращий користувацький досвід.

1.4.2. Flutter

Flutter - це фреймворк для створення нативно скомпільованих додатків для мобільних, веб- та настільних платформ з єдиного коду представлений Google в 2018 році. Він використовує мову програмування Dart і пропонує широкий набір налаштовуваних віджетів для розробки вражаючих інтерфейсів користувача. Функція гарячого перезавантаження у Flutter дозволяє розробникам миттєво бачити зміни, значно підвищуючи ефективність процесу розробки.

Широкий каталог віджетів Flutter і налаштовувані компоненти інтерфейсу користувача дозволяють розробникам створювати візуально привабливі додатки з постійною продуктивністю на різних платформах. Хоча підтримка платформ-специфічних функцій у Flutter постійно розширюється, розробникам слід враховувати, що додатки Flutter часто мають більший розмір файлів порівняно з деякими іншими фреймворками, що може вплинути на час завантаження і використання пам'яті пристрою.

Flutter може бути вбудований в автомобілі, телевізори та розумну побутову техніку. Помітним прикладом ефективності Flutter є його використання провідним німецьким автовиробником BMW. Компанія прагнула оптимізувати розробку продуктів, орієнтованих на клієнтів. Створення окремих нативних додатків для кожної функції було б повільним і дорогим.

Прийнявши Flutter, BMW швидко отримала кілька переваг:

— Уніфікована розробка для декількох платформ з єдиного коду.

— Швидші цикли розробки завдяки гарячому перезавантаженню.

— Стабільний користувацький досвід на різних пристроях.

— Зниження витрат і часу на розробку.

Flutter дозволив BMW оптимізувати процес розробки додатків, забезпечуючи ефективні та економічно вигідні рішення.

1.4.3. React Native

React Native, створений компанією Facebook, є однією з передових платформ для розробки гібридних мобільних додатків. Ця мова програмування здобула широку популярність завдяки своїй гнучкості та здатності інтегруватися з існуючими веб-технологіями.

Однією з ключових переваг React Native є його висока гнучкість у роботі з компонентами. Розробники можуть використовувати різноманітні компоненти з масиву доступних, а також створювати власні, що дозволяє створювати додатки з унікальним дизайном та функціональністю. Крім того, React Native має широкий вибір сторонніх бібліотек і модулів, що полегшує розширення функціоналу додатка.

Ще однією важливою перевагою React Native є його здатність працювати на крос-платформеному рівні. Це означає, що розробники можуть використовувати один і той же код для створення додатків для різних платформ, включаючи iOS та Android. Це спрощує процес розробки і зменшує час, потрібний для виведення продукту на ринок.

Нарешті, React Native відрізняється відмінною швидкістю розробки. Завдяки гнучкості та зручності використання, розробники можуть швидко реагувати на зміни вимог та швидко створювати додатки з унікальними функціями та дизайном.

У порівнянні з Flutter, React Native має свої переваги та недоліки. Однією з найбільш очевидних переваг React Native є його гнучкість та можливість інтеграції з існуючими веб-технологіями, що робить його

відмінним вибором для проєктів, які потребують багатофункціональний та індивідуальний дизайн.

Однак, у порівнянні з Flutter, React Native може мати деякі обмеження у швидкості роботи та продуктивності. Крім того, відсутність повного контролю над нативним кодом може стати обмеженням у випадках, коли потрібні розширені можливості або оптимізація продуктивності.

Загалом, обираючи між React Native і Flutter, розробники повинні враховувати потреби свого проєкту та власні вподобання у розробці. Кожен з цих фреймворків має свої унікальні переваги та обмеження, які варто врахувати при прийнятті рішення. Додатки, створені за допомогою React Native Cross-Platform App Framework: Pinterest, Skype, Tesla, Wix, Uber Easts, Bloomberg.

1.5. Програмне забезпечення для створення графічного інтерфейсу користувача

Розробка графічного інтерфейсу мобільного застосунку є надзвичайно важливою, особливо у сфері роздрібної торгівлі. У сучасному світі, де мобільні пристрої стали невід'ємною частиною нашого повсякденного життя, якісний і зручний інтерфейс може значно вплинути на успіх застосунку та рівень задоволення користувачів. Більшість користувачів вважають зручний інтерфейс основним фактором при виборі мобільного додатку для покупок.

Ефективний графічний інтерфейс не тільки сприяє залученню нових клієнтів, але й допомагає утримати існуючих. Наприклад, дослідження показують, що 57% користувачів не рекомендуватимуть компанію з поганим мобільним додатком, а 40% перейдуть до конкурентів, якщо їхній досвід використання додатку буде незадовільним. До того ж інтуїтивно зрозумілий дизайн інтерфейсу знижує когнітивне навантаження, що в свою чергу збільшує кількість виконаних задач і збільшує задоволення від використання застосунку.

Окрім цього, зручний інтерфейс може значно підвищити конверсію та середній чек. Наприклад, компанія Walmart після редизайну свого мобільного додатку зафіксувала збільшення конверсії на 20%, а середній чек виріс на 10%. Ці показники демонструють, наскільки важливою є увага до деталей у процесі розробки графічного інтерфейсу.

Таким чином, розробка якісного графічного інтерфейсу для мобільного застосунку у сфері роздрібної торгівлі є критично важливою для досягнення комерційного успіху.

Є декілька найбільш популярних програмних за стосунків для створення графічного інтерфейсу, які представлені нижще.

Figma —високоефективне програмне забезпечення для дизайну інтерфейсу користувача, що сприяє продуктивній міжфункціональній співпраці команд. Завдяки Figma, дизайнери, розробники та інші учасники проєкту можуть працювати разом у реальному часі, що забезпечує швидке і узгоджене створення та вдосконалення інтерфейсу;

Invision — провідне програмне забезпечення для дизайну інтерфейсу користувача, відоме своєю обширною колекцією шаблонів для різних етапів розробки продуктів. Invision допомагає створювати прототипи, тестувати і впроваджувати нові ідеї, що дозволяє ефективно керувати процесом розробки від концепції до готового продукту;

Marvel — інтуїтивне програмне забезпечення для дизайну інтерфейсу користувача, яке пропонує зручні шаблони для створення вайрфреймів iOS методом перетягування. Marvel дозволяє швидко створювати і тестувати прототипи, що робить його ідеальним інструментом для дизайнерів, які хочуть максимально спростити процес створення інтерфейсу;

UXPin — потужне програмне забезпечення для дизайну інтерфейсу користувача, спеціалізоване на прототипуванні на основі компонентів. UXPin дозволяє створювати інтерактивні прототипи, які точно відтворюють функціональність і вигляд кінцевого продукту, забезпечуючи глибоке розуміння користувацького досвіду ще на етапі розробки;

Justinmind — розширене програмне забезпечення для дизайну інтерфейсу користувача, що забезпечує створення інтерактивних прототипів з адаптивним дизайном і підтримкою жестів. Justinmind дозволяє створювати прототипи, які точно відображають взаємодію користувача з додатком, завдяки чому дизайнери можуть оптимізувати користувацький досвід на різних пристроях і платформах.

Все більш розповсюдженим є залучення ШІ, в розробці графічних інтерфейсів також є ПЗ які використовують ШІ для облегшення роботи дизайнерів. Є повноцінні застосунку, здатні по опису чи фото ескізу розробити вигляд майбутнього сайту чи мобільного застосунку, наприклад Uizard.

Також існує ПЗ в якому ШІ працює паралельно з дизайнером і надає підказки чи поради, як наприклад у Figma використовується ШІ для покращення командної роботи та автоматизації рутинних задач у процесі дизайну. Інструмент дозволяє дизайнерам спільно працювати над проєктами в режимі реального часу, надаючи інтелектуальні рекомендації та спрощуючи управління дизайном. ШІ автоматично вирівнює та організовує елементи графічного інтерфейсу.

1.6. Онлайн конструктори мобільних застосунків

В останні роки почали набирати популярність онлайн конструктори мобільних додатків. Ці конструктори мають готові рішення для створення мобільних застосунків, вони надають можливість створювати власні додатки без знань програмування, що підходить для малому та середньому бізнесу, а також стартапам створювати власні рішення не роблячи великих грощових вкладень.

При створені додатку за допомогою конструктора користувач бачить одразу як буде виглядати той чи інший модуль, кожен модуль все має всій код і користувач лише розміщує його та змінює дизайн. В залежності від

застосунку і рівня підписки користувач може масштабувати додаток, отримувати аналітичні дані про користувачів, встановлювати оплату картою, налаштувати доставку, push-повідомленя, iBeacon та інші модулі. Конструктори мають бібліотеки вже готових рішень, а також шаблони, простіше кажучи кодування – це інтерфейс командного рядка в ОС, а конструктор мобільних застосунків – графічний інтерфейс.

Користувачі вже бачать переваги використання цих платформ, оскільки майже 60 відсотків із них зазначили, що використання конструкторів збільшує дохід і допомагає замінити застарілі системи. Інші переваги включають можливості моніторингу цих платформ, а також їх кросплатформну доступність і швидкість порівняно з традиційною розробкою [6].

1.7. Огляд структури існуючих мобільних застосунків з роздрібної торгівлі

Зважаючи на швидкість розвитку електронної комерції у всьому світі можна знайти безліч мобільних додатків для роздрібної торгівлі і виділити шаблон який притаманний усім додаткам для продажу.

Зазвичай застосунки мають такий функціонал:

— Авторизація та аудентифікація.
— Каталог товарів з можливість фільтрації та сортування товару.
— Кошик покупок.
— Оплата.
— Доставка.
— Відгуки та формування рейтингу на їх основі.
— Чат підтримки для користувачів.

Додатково в залежності від потреб користувачів застосунку в нього може бути доданий особливий функціонал, наприклад можливість онлайн примірки для додатків з продажу одягу.

Зважаючи на функціонал можна визначити які можливості будуть надаватись кожній стороні, що використовуватиме розроблений мобільний застосунок.

Адміністратор:

— Перегляд даних користувачів: може переглядати дані користувачів у системі.

— Управління товарами: може додавати, редагувати або видаляти товари.

— Редагування замовлень: у разі виникнення надзвичайних ситуацій адміністратор може змінювати або відміняти замовлення.

Незареєстрований або неавтентифікований користувач:

— переглядати каталог та користуватись функціями фільтрації і сортування.

— увійти до облікового запису або зареєструватись як новий користувач.

— додавати до кошика товари, але без можливості подальшого оформлення замовлення без входу у особистий кабінет користувача.

Зареєстрований користувач має наступні функції має всі можливості незареєстрованого а також додаткові:

— Додавання до списку вподобань та його перегляд.

— Змін даних особистого кабінету.

— Оформлення та оплата замовлення.

— Перегляд історії замовлень.

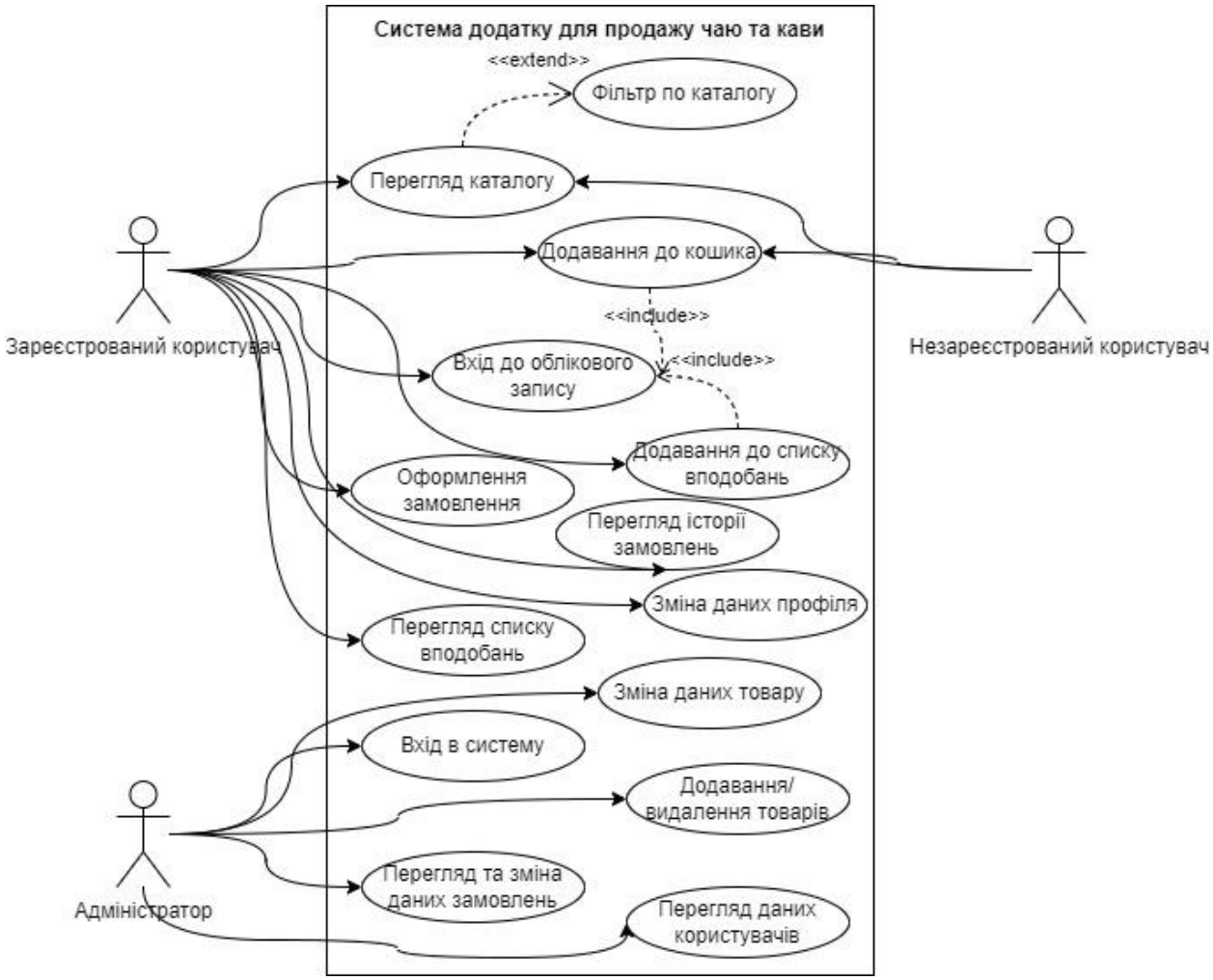

Рисунок 1.5 – Діаграма варіанті використання застосунку з продажу чаю та кави

Загалом, ураховуючи подібність всіх за стосунків з продажу, можна сказати що ця діаграма з мінімальними корективами підійшла б для будь якого іншого мобільного додатку для продажу товарів.

Висновки до розділу:

У першому розділи випускної роботи було виконано дослідження щодо ринку мобільних за стосунків та мов програмування з програмним забезмеченням на яких вони створюються. В тому числі були розглянуті переваги та недоліки розробки нативних застосунків для операційних систем Android та IOS, та розробки кросплатформенної розробки. Також були

розглянуті ПЗ та фреймворки для розробки як нитивних, так і кросплатформених додатків. Зокрема детально було розглянуто Flutter, React Native, Xamarin та .NET MAUI, які відповідно використовують ПЗ розробки Visual Studio Code та Visual Studio. Додатково було розглянуто програмні застосунки для створення графічного інтерфейсу користувача та як альтернативу програмуванню сучасні конструктори для створення мобільних за стосунків що не потребують особливих навичок.

Розроблено функціональні вимоги до мобільного застосунку та створено діаграму варіантів використання додатку.

РОЗДІЛ 2.
РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ ПРОДАЖУ ЧАЮ ТА КАВИ

З урахуванням функціоналу застосунку та комерційної складової розробки застосунків для роздрібної торгівлі, а також популярності фреймворків, було прийнято рішення розробляти кросплатформений застосунок на базі фреймворку React Native використовуючи платформи для розробки Visual Studio Code та Android Studio.

2.1. Розробка дизайну мобільного застосунку

В першу чергу потрібно створити графічний прототип того, як будуть виглядати екрани нашого застосунку для розуміння скільки екранів, класів, іконок та які саме потрібно завантажити. Дизайн буде здебільшого схематичний і виконаний в Figma.

Спочатку розробимо основні екрани. Аналізуючи функціонал для мобільного додатку, описаний раніше, та вигляд інших мобільних за стосунків для продажу, таких як Rozetka. Prom, Varus та інші застосунки, можна виділити такі основні екрани: головна сторінка, сторінка оформлення покупки, сторінка вподобань, сторінка історії покупок. На всіх екранах буде відображатись верхня та нижня навігація. Переходи між екранами будуть реалізовані у нижній навігації оскільки це найпопулярніший та найзручніший вид навігації для користувачів на постійній основі користування.

Головна сторінка сторінка передбачає наявність такого функціоналу:

— Рядок пошуку.
— Стрічка вибору товару.
— Продуктові картки товарів.

Рисунок 2.1 – Прототип головної сторінки

Продуктові картки також повинні мати кнопку додавання до списку вподобань, для зручності користувачів, відображати ціну, рейтинг товару, назву.

Кошик покупок, вподобання та історія замовлень матимуть приблизно однаковий вигляд і будуть представляти собою список, відмінність полягатиме у деталях одиниць списку, до прикладу кошик покупок має включати можливість змінювати кількість одиниць обраного товару та відображати загальну суму покупки, а список буде складатись з обраних продуктів, в той час як в історії покупок буде відображатись дата та загальна сума кожної покупки і кожна одиниця списку буде покупкою.

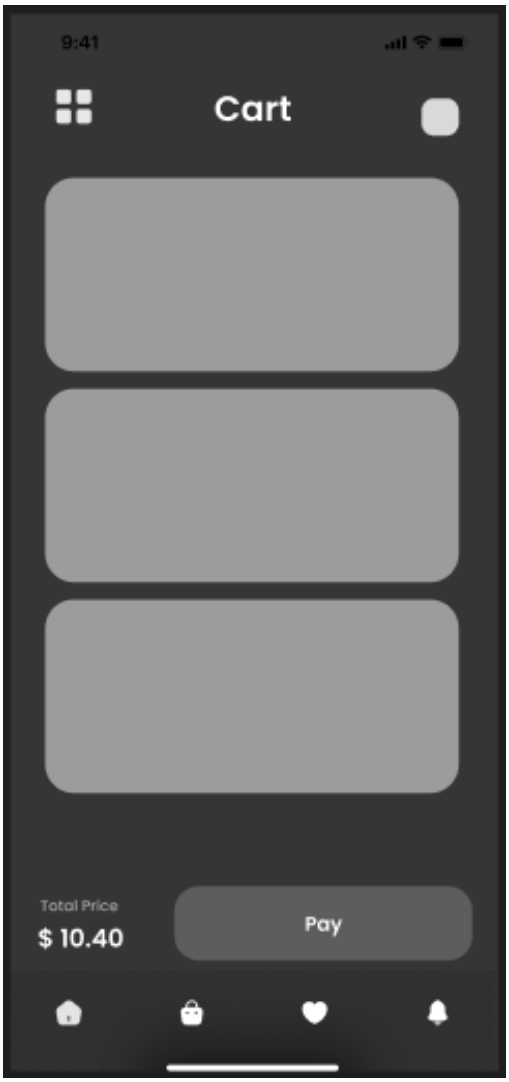

Рисунок 2.2 – Прототип сторінки кошик покупок

Також потрібно створити екран картки товару, в якому буде детальна інформація про обрану позицію, зображення товару та кнопка додавання до кошика та можливість додавати до вподобання, а також вибір користувачем розміру товару. Буде додано 3 розміри товару і ціна буде змінюватись в залежності від розміру, також кожен розмір обраний в кошику буде окремо виділений.

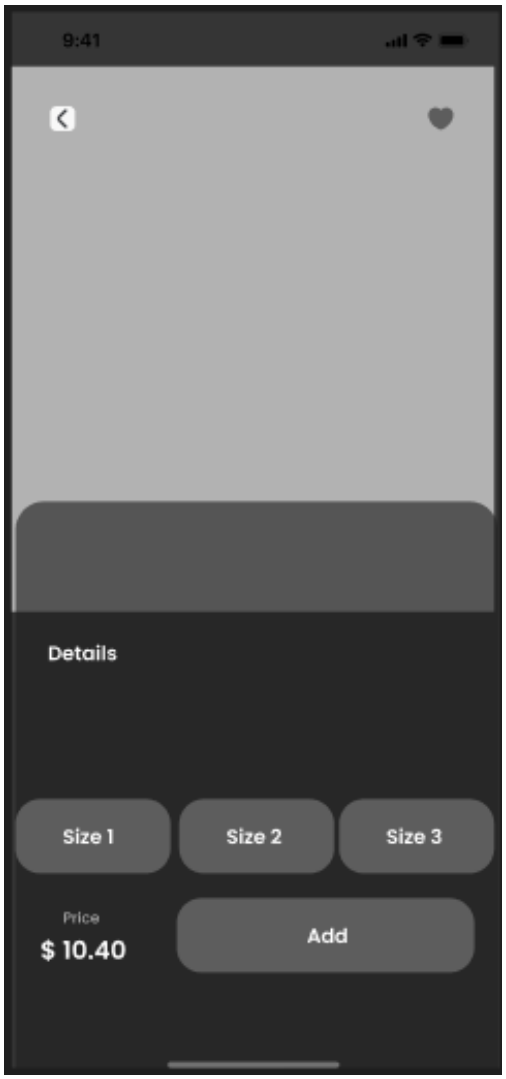

Рисунок 2.3 – Прототип сторінки картка товару

Після натискання кнопки кнопки оплати у кошику, користувач буде потрапляти на сторінку оплати, де потрібно буде або ввести реквізити картки, або вибрати зручний спосіб оплати: Google Pay або Apple Pay.

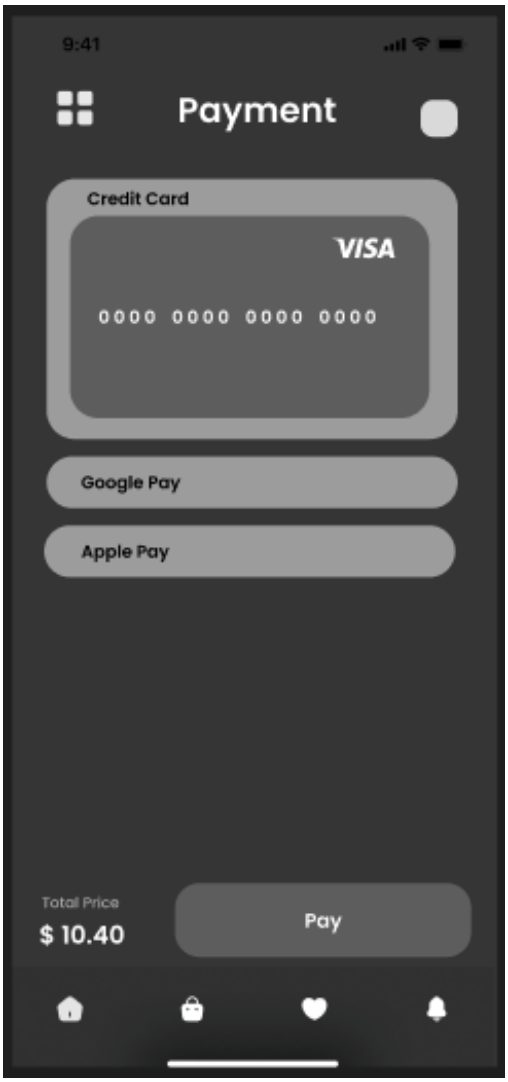

Рисунок 2.4 – Прототип сторінки оплати

Сторінка вподобань представляє собою стрічку збережених продуктів, кожен лемент стрічки буде зменшеною копією продуктової карти товару та при натисканні буде переміщати на саму картку товару. На сторінці вподобань будуть відображатись картинка товару, назва, рейтинг та частина опису продукту.

Рисунок 2.5 – Прототип сторінки вподобань

Загалом вже зрозуміло які іконки потрібно завантажити та які частини розробки додатку можна буде використовувати повторно, це облегшить розробку застосунку та оптимізує витрати часу.

2.2. Розробка мобільного застосунку

Оскільки для розробки буде використано React Native, спочатку потрібно встановити платформу, на базі якої буде проходити програмування. В даній роботі ми використовуємо Visual Studio Code та Android Studio, насамперед інсталюємо Node Version Manager та Java Development Kit остільки React Native створено на основі мов Java. Далі встановлюємо та

запускаємо Visual Studio Code, в якому створюємо проєкт для React Native, та Android Studio , в якому завантажуємо та створюємо віртуальний девайс.

Після створення проєкту структуру файлової системи, перед початком розробки зручніше знайти та росприділіти потрібні матеріали наперед знаючи функціональні вимоги до додатку та приблизний вигляд графічного інтерфейса.

В папці “src” створюємо папку “assets”, яка буде включати графічні матеріали для додатку. В цій папці створимо ще три папки:

— “fonts” – зберігає шрифт, обраний для використання в застосунку.

— “app_images” – зберігає додаткові картинки та іконки.

— “coffee_assets” – зберігає папки з назвами товарів які включають фото товарів, що будуть відображатись в застосунку в двох варіантах – квадратний, який буде відображатись на головній сторінці, і прямокутний для картки продукту.

Рисунок 2.6 – Матеріали для продуктових карток

В папці “src” створюємо папку “data”, в ній ми зберігаємо дані про кожен товар, ці дані будуть відображатись у картці товару.

```
const CoffeeData = [
  {
    id: 'C1',
    name: 'Americano',
    description: `The Americano is another popular type of coffee drink, and it's very easy to make! It's just espresso with hot water
    roasted: 'Medium Roasted',
    imagelink_square: require('../assets/coffee_assets/americano/square/americano_pic_1_square.png'),
    imagelink_portrait: require('../assets/coffee_assets/americano/portrait/americano_pic_1_portrait.png'),
    ingredients: 'Milk',
    special_ingredient: 'With Steamed Milk',
    prices: [
      {size: 'S', price: '1.38', currency: '$'},
      {size: 'M', price: '3.15', currency: '$'},
      {size: 'L', price: '4.29', currency: '$'},
    ],
    average_rating: 4.7,
    ratings_count: '6,879',
    favourite: false,
    type: 'Coffee',
    index: 0,
  },
```

Рисунок 2.7 – Опис одного з товарів

В папці “data” створюємо новий файл із назвою “CoffeeData.ts” Створюємо масив, де кожен елемент це один товар. Цей масив може бути імпортований в інші частини проєкту. Масив містить такі властивості:

— id – ідентифікатор елементу.

— name – назва товару.

— description – опис товару.

— roasted – ступінь прожарки.

— imagelink_square – посилання на зображення товару в квадратному форматі.

— imagelink_portrait – посилання на зображення товару в прямокутному форматі

— ingredients – інгрідієнти в напої.

— special_ingredient – спеціальні інгрідієнти в напої.

— prices – ціна товару. Оскільки ціна залежить від розміру товару, цю властивість реалізовано в якості массива, що в свою чергу містить такі властивості: size – розмір, price – ціна та currency – валюта.

— average_rating – рейтинг товару.

— ratings_count – кількість оцінок товару.

— favourite – чи додано товар у список вподобань.

— type – тип товару.

— index – індекс елементу в масиві, оскільки перший елемент в масиві є нульовим, в данному елементі стоїть 0.

В папці “src” створюємо папку “theme ” в якій буде знаходитись файл “theme.ts”. В цьому файлі ми пишемо код для і опису інтерфейсів та констант для налаштування різних аспектів стилю в додатку, а саме: типи для відступів, типи для кольорів, типи для сімейств шрифтів, типи для розмірів шрифтів, типи для радіусів кутів. Ці значення можна легко змінити при розробці дизайн у всьому додатку змінивши дані лише в цьому файлі. Частина коду, а саме налаштування кольорів представлено на рисунку 2.8.

```
export const COLORS: Color = {
  primaryRedHex: '#DC3535',
  primaryOrangeHex: '#D17842',
  primaryBlackHex: '#0C0F14',
  primaryDarkGreyHex: '#141921',
  secondaryDarkGreyHex: '#21262E',
  primaryGreyHex: '#252A32',
  secondaryGreyHex: '#252A32',
  primaryLightGreyHex: '#52555A',
  secondaryLightGreyHex: '#AEAEAE',
  primaryWhiteHex: '#FFFFFF',
  primaryBlackRGBA: 'rgba(12,15,20,0.5)',
  secondaryBlackRGBA: 'rgba(0,0,0,0.7)',
};
```

Рисунок 2.8 – Налаштування стилю кольорів

У папці “src” створюємо три папки: “ store ”,“ components ”,“ screens ” та “ navigators ”. Для подальшої роботи потрібно встановити пакет для використання іконо в пакеті, для цього відкриваємо новий термінал та надсилаємо команду “npm i react-native-vector-icons”.

Створюємо файл “CustomIcon” в папці “ components ” та додаємо код.

```
1  import {createIconSetFromIcoMoon} from 'react-native-vector-icons';
2  import icoMoonConfig from '../../selection.json';
3  export default createIconSetFromIcoMoon(icoMoonConfig);
```

Рисунок 2.9 – Налаштування іконок

Цей код створює та експортує кастомний набір іконок для використання в додатку на базі React Native, використовуючи react-native-vector-icons і конфігураційний файл IcoMoon. Це дозволяє легко інтегрувати та використовувати кастомні іконки в різних компонентах додатку.

2.2.1. Створення навігаторів стека та вкладок

Створюємо файл “ TabNavigator” в папці “ navigators ” та використовуємо команду “ rnfes ” для створення шаблону коду. Так само створюємо шаблон в файлі “ App ”, цей файл створюється автоматично при створенні проєкту React Native.

В файлі “ App ” напишемо код представлений на рисунку 2.10, що створить простий додаток для навігації між екранами.

Імпортуємо необхідні бібліотеки та компоненти для роботи додатка:

— “React” та хук “useEffect” з “react”.

— Компоненти навігації з “@react-navigation/native” та “@react-navigation/native-stack”.

— Кастомні компоненти “TabNavigator”, “DetailsScreen”, “PaymentScreen”.

— “SplashScreen” з “react-native-splash-screen” для керування splash screen – початковий екран з логотипом, який з’являється перед завантаженням головної сторінки.

```
1  import React, {useEffect} from 'react';
2  import {NavigationContainer} from '@react-navigation/native';
3  import {createNativeStackNavigator} from '@react-navigation/native-stack';
4  import TabNavigator from './src/navigators/TabNavigator';
5  import DetailsScreen from './src/screens/DetailsScreen';
6  import PaymentScreen from './src/screens/PaymentScreen';
7  import SplashScreen from 'react-native-splash-screen';
8
9  const Stack = createNativeStackNavigator();
10
11 const App = () => {
12   useEffect(() => {
13     SplashScreen.hide();
14   }, []);
15   return (
16     <NavigationContainer>
17       <Stack.Navigator screenOptions={{headerShown: false}}>
18         <Stack.Screen
19           name="Tab"
20           component={TabNavigator}
21           options={{animation: 'slide_from_bottom'}}></Stack.Screen>
22         <Stack.Screen
23           name="Details"
24           component={DetailsScreen}
25           options={{animation: 'slide_from_bottom'}}></Stack.Screen>
26         <Stack.Screen
27           name="Payment"
28           component={PaymentScreen}
29           options={{animation: 'slide_from_bottom'}}></Stack.Screen>
30       </Stack.Navigator>
31     </NavigationContainer>
32   );
33 };
34
35 export default App;
36
```

Рисунок 2.10 – Код файлу “App”

В 9 рядку ми створюємо стек навігації. В рядках 12, 13, 14 ми використовуємо хук useEffect для приховання splash screen, коли компонент завантажується. Починаючи з 15 рядка ми повертаємо NavigationContainer, який містить Stack.Navigator з трьома екранами (Tab, Details, Payment).

Переходимо до налаштування нижньої навігації. в файлі “TabNavigator”. Цей код створюватиме вкладковий навігатор для мобільного додатку за допомогою бібліотеки react-navigation. Він міститиме чотири основні екрани: Home, Cart, Favorites та Order History. Вкладки мають іконки,

які змінюють свій колір при фокусуванні, і застосовують розмитий фон для панелі вкладок. Спочатку імпортуємо необхідні компоненти та бібліотеки. В рядку 12 створюємо вкладковий навігатор, код видно на рисунку 2.11.

```
12  const Tab = createBottomTabNavigator();
13
14  const TabNavigator = () => {
15    return (
16      <Tab.Navigator
17        screenOptions={{
18          tabBarHideOnKeyboard: true,
19          headerShown: false,
20          tabBarShowLabel: false,
21          tabBarStyle: styles.tabBarStyle,
22          tabBarBackground: () => (
23            <BlurView
24              overlayColor=""
25              blurAmount={15}
26              style={styles.BlurViewStyles}
27            />
28          ),
29        }}>
30        <Tab.Screen
31          name="Home"
32          component={HomeScreen}
33          options={{
34            tabBarIcon: ({focused, color, size}) => (
35              <CustomIcon
36                name="home"
37                size={25}
38                color={
39                  focused ? COLORS.primaryOrangeHex : COLORS.primaryLightGreyHex
40                }
41              />
42            ),
43          }}></Tab.Screen>
```

Рисунок 2.11 – Код файлу “ TabNavigator ”

В рядках 14–29 встановлюються загальні налаштування для навігатора: приховування вкладки при появі клавіатури, приховування заголовка, відключення міток для вкладок, стилізація вкладок та додавання фону з розмиттям.

Рядки 30–89 додають екран “HomeScreen”, “CartScreen”. “FavoritesScreen” та “OrderHistoryScreen” до навігатора. Встановлюємо іконки для вкладок, які змінює колір в залежності від стану фокусу. На

рисунку 2.11 з 30 по 43 ряжок описано навігацію для екрану “HomeScreen”, але інші 3 вкладки описані так само.

```
90   const styles = StyleSheet.create({
91     tabBarStyle: {
92       height: 80,
93       position: 'absolute',
94       backgroundColor: COLORS.primaryBlackRGBA,
95       borderTopWidth: 0,
96       elevation: 0,
97       borderTopColor: 'transparent',
98     },
99     BlurViewStyles: {
100      position: 'absolute',
101      top: 0,
102      bottom: 0,
103      left: 0,
104      right: 0,
105    },
106  });
```

Рисунок 2.12 – Код файлу “ TabNavigator ” – стилі навігатора

Рисунок 2.13 – Вигляд додатку після створення навігації

В рядках 90–106 визаначаємо стилі для навігатора та вкладок, а також розмиття фону навігатора, код стилів відображено на рисунку 2.12.

2.2.2. Створюємо Zustand Store

Zustand — це легка і проста у використанні бібліотека для управління станом у React. Використання zustand у цьому проєкті дозволяє легко керувати складними станами додатку та забезпечує гнучке збереження даних. Це особливо важливо для мобільних додатків, де користувацький досвід може значно покращитись за рахунок швидкого і надійного управління станом. Завдяки використанню immer, ми можемо працювати зі складними структурами даних, забезпечуючи незмінність стану, що робить код більш безпечним та зрозумілим.

Додатково, інтеграція zustand/middleware з @react-native-async-storage/async-storage дозволяє зберігати стан додатку навіть після перезапуску, що є важливою функцією для мобільних додатків, які потребують надійного збереження користувацьких даних. Це забезпечує безперервний користувацький досвід та знижує ризик втрати даних.

В проєкті створюємо папку “store” із файлом “store.ts”. Після чого додаємо необхідні бібліотеки, а саме: zustand, immer, zustand/middleware, @react-native-async-storage/async-storage.

Також імпортуємо данні з раніше вже створених CoffeeData та BeansData.

В рядках 8 – 16 створюємо zustand-сховище з використанням функції persist для зберігання стану. Встановлюються початкові значення для списків кави, бобів, ціни кошика, списку улюблених, кошика та історії замовлень.

```
8   export const useStore = create(
9     persist(
10      (set, get) => ({
11        CoffeeList: CoffeeData,
12        BeanList: BeansData,
13        CartPrice: 0,
14        FavoritesList: [],
15        CartList: [],
16        OrderHistoryList: [],
17        addToCart: (cartItem: any) =>
18          set(
19            produce(state => {
20              let found = false;
21              for (let i = 0; i < state.CartList.length; i++) {
22                if (state.CartList[i].id == cartItem.id) {
23                  found = true;
24                  let size = false;
25                  for (let j = 0; j < state.CartList[i].prices.length; j++) {
26                    if (
27                      state.CartList[i].prices[j].size == cartItem.prices[0].size
28                    ) {
29                      size = true;
30                      state.CartList[i].prices[j].quantity++;
31                      break;
32                    }
33                  }
34                  if (size == false) {
35                    state.CartList[i].prices.push(cartItem.prices[0]);
36                  }
37                  state.CartList[i].prices.sort((a: any, b: any) => {
38                    if (a.size > b.size) {
39                      return -1;
40                    }
41                    if (a.size < b.size) {
42                      return 1;
43                    }
44                    return 0;
45                  });
46                  break;
47                }
48              }
49              if (found == false) {
50                state.CartList.push(cartItem);
51              }
52            }),
53          )
```

Рисунок 2.14 – Код сховища Zustand – функція додавання елементів до кошика

Після цього до 53 рядка код, що оголошує функцію addToCart, яка додає елемент до кошика. Якщо елемент вже існує, збільшується його

кількість. Якщо елемент існує, але з іншого розміру, додається новий розмір до списку. Якщо елемент не знайдено, додається новий елемент до кошика.

```
54    calculateCartPrice: () =>
55      set(
56        produce(state => {
57          let totalprice = 0;
58          for (let i = 0; i < state.CartList.length; i++) {
59            let tempprice = 0;
60            for (let j = 0; j < state.CartList[i].prices.length; j++) {
61              tempprice =
62                tempprice +
63                parseFloat(state.CartList[i].prices[j].price) *
64                  state.CartList[i].prices[j].quantity;
65            }
66            state.CartList[i].ItemPrice = tempprice.toFixed(2).toString();
67            totalprice = totalprice + tempprice;
68          }
69          state.CartPrice = totalprice.toFixed(2).toString();
70        }),
71      ),
72    addToFavoriteList: (type: string, id: string) =>
73      set(
74        produce(state => {
75          if (type == 'Coffee') {
76            for (let i = 0; i < state.CoffeeList.length; i++) {
77              if (state.CoffeeList[i].id == id) {
78                if (state.CoffeeList[i].favourite == false) {
79                  state.CoffeeList[i].favourite = true;
80                  state.FavoritesList.unshift(state.CoffeeList[i]);
81                } else {
82                  state.CoffeeList[i].favourite = false;
83                }
84                break;
85              }
86            }
87          } else if (type == 'Bean') {
88            for (let i = 0; i < state.BeanList.length; i++) {
89              if (state.BeanList[i].id == id) {
90                if (state.BeanList[i].favourite == false) {
91                  state.BeanList[i].favourite = true;
92                  state.FavoritesList.unshift(state.BeanList[i]);
93                } else {
94                  state.BeanList[i].favourite = false;
95                }
96                break;
97              }
```

Рисунок 2.15 – Функції ціни кошика та додавання в список вподобань

В рядках 54 – 71 оголошується функція calculateCartPrice, яка обчислює загальну ціну кошика. Вона проходиться по всіх елементах у кошику, обчислює ціну для кожного з них і зберігає сумарну ціну в стані. Після чого до 101 рядка оголошується функція addToFavoriteList, яка додає елемент до

списку улюблених. Вона оновлює статус улюбленого елемента у списку кави або бобів, а також додає його до списку улюблених.

За такою ж схемою як і попередні функції оголошуються функція deleteFromFavoriteList, яка видаляє елемент зі списку улюблених. Вона оновлює статус улюбленого елемента у списку кави або бобів, і видаляє його зі списку улюблених. Далі створюється функція incrementCartItemQuantity, яка збільшує кількість певного елемента в кошику за вказаним розміром, функція decrementCartItemQuantity, яка зменшує кількість певного елемента в кошику за вказаним розміром, якщо кількість стає нульовою, елемент видаляється з кошика, та функція addToOrderHistoryListFromCart, яка додає поточний кошик до історії замовлень і очищає кошик.

Загалом цей код створює стан управління для мобільного додатку, який включає списки кави та бобів, функції для роботи з кошиком, улюбленими елементами та історією замовлень. Стан зберігається локально за допомогою AsyncStorage, щоб дані не втрачалися при закритті додатку.

2.2.3. Екран оплати

Однією з найбільш цікавих частин додатку є оплата покупки. Цей код реалізує екран оплати для мобільного додатка. Екран оплати дозволяє користувачам вибирати з декількох методів оплати, таких як кредитна картка, Google Pay, Apple Pay, а також здійснювати оплату з вибраним методом. Оскільки код занадто масштабний, він представлений в додатку А.

Основні можливості цього коду включають:

— Динамічний вибір методу оплати – користувач може вибирати між кількома методами оплати, кожен з яких має свій унікальний вигляд і іконку.

— Анімація успішної оплати – після натискання кнопки оплати, відображається анімація, яка сигналізує про успішне завершення операції.

— Взаємодія з глобальним станом додатка – використовується хук `useStore` для доступу до методів, які обчислюють загальну суму кошика та додають замовлення до історії покупок.

Цей код унікальний тим, що він не лише забезпечує функціонал оплати, але й робить це у витончений та естетично привабливий спосіб, забезпечуючи гарний користувацький досвід. Для більшого розуміння коду потрібно розглянути його діаграму класів, яка представлена на рисунку 2.16.

Далі ми опишемо UML діаграму класів, створену для компонента екрана оплати в додатку. Ми розглянемо основні компоненти, їх властивості, методи та взаємозв'язки між ними.

Компоненти та пакети представлені нижче.

React Components:

— PaymentScreen:

— головний компонент екрана оплати;

— використовує хуки «usestore» та «usestate» для управління станом;

— локальні змінні стану: «paymentmode» (режим оплати) та «showanimation» (показ анімації);

— метод «buttonpresshandler» обробляє натискання кнопки для здійснення оплати. Після натискання кнопки відображається анімація успішної оплати, додається замовлення до історії та розраховується загальна сума кошика;

— використовує наступні компоненти:

— paymentmethod – для відображення методів оплати.

— paymentfooter – для відображення нижньої частини екрана з кнопкою оплати.

— gradientbgicon – для відображення іконки з градієнтним фоном.

— customicon – для відображення кастомних іконок.

— popupanimation – для відображення анімації успішної оплати.

— PaymentMethod :

— компонент для відображення методів оплати;

— властивості: «name» (назва методу), «icon» (іконка методу), «isicon» (булеве значення, яке вказує, чи є це іконка);

— залежить від стандартних компонентів «view» та «text»;

— PaymentFooter :

— Компонент, який відображає нижню частину екрана оплати;

— Властивості: «buttonTitle» (назва кнопки), «price» (ціна), «buttonPressHandler» (обробник натискання кнопки);

— Залежить від компонентів «View» та «Text»;

— GradientBGIcon :

— Іконка з градієнтним фоном;

— Властивості: «name», «color», «size»;

— Залежить від компонентів «View» та «Text»;

— CustomIcon :

— Компонент для відображення кастомної іконки;

— Властивості: «name», «size», «color»;

— Залежить від компонентів «View» та «Text»;

— PopUpAnimation:

— Компонент для відображення анімації успішної оплати;

— Властивості: «style», «source»;

— Залежить від компонентів «View» та «Text».

Native Components :

— View , Text , StatusBar , ScrollView , TouchableOpacity , StyleSheet , LinearGradient : стандартні компоненти React Native, які використовуються в інших компонентах для побудови інтерфейсу користувача та стилізації.

Theme :

— BORDERRADIUS, COLORS, FONTFAMILY, FONTSIZE, SPACING: константи для стилізації, які використовуються в компоненті `PaymentScreen` для надання єдиного стилю додатку.

Data :

— PaymentList : список методів оплати, що включає властивості `name` (назва методу), `icon` (іконка методу), `isIcon` (булеве значення, яке вказує, чи є це іконка).

Store :

— useStore: хук для управління станом, який містить методи `calculateCartPrice` та `addToOrderHistoryListFromCart`. Ці методи використовуються в компоненті `PaymentScreen` для обчислення загальної суми кошика та додавання замовлення до історії.

Взаємодія компонентів:

— PaymentScreen використовує useStore для доступу до стану та методів управління, таких як `calculateCartPrice` та `addToOrderHistoryListFromCart`.

— PaymentScreen відображає компоненти PaymentMethod, PaymentFooter, GradientBGIcon, CustomIcon та PopUpAnimation для побудови інтерфейсу користувача.

— PaymentFooter використовує передані з PaymentScreen функції для обробки натискань кнопок.

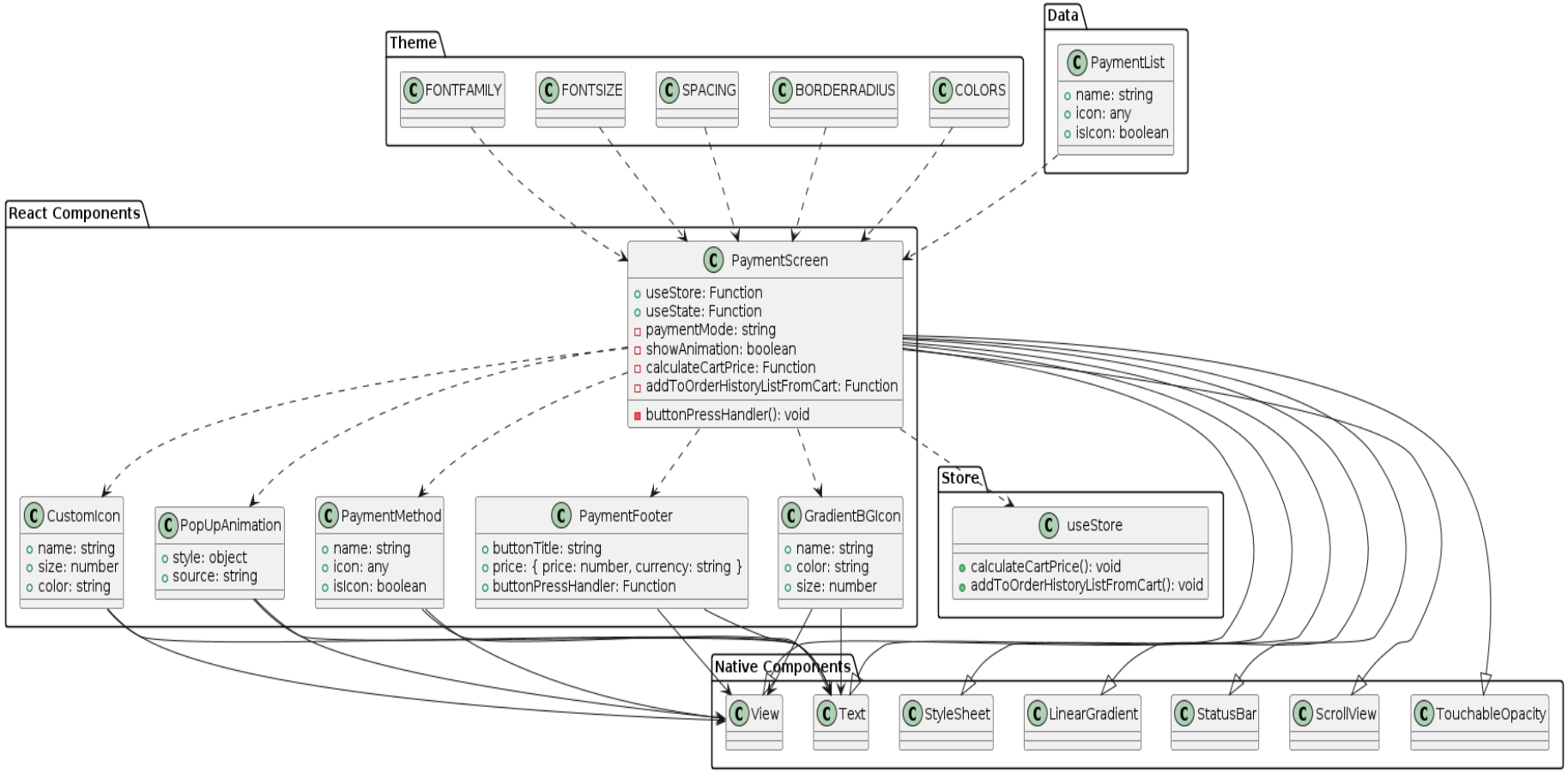

Рисунок 2.16 – Діаграма класів екрану оплати замовлення

Ця діаграма надає візуальне представлення структури компонентів, їх властивостей, методів та взаємозв'язків у React Native додатку для екрана оплати. UML діаграма допомагає зрозуміти, як компоненти взаємодіють між собою та як вони інтегровані в єдину систему.

Подальша розробка додатку складається з створення 4 екранів: HomeScreen, OrderHistoryScreen, OrderHistoryScreen та FavoritesScreen, а також екранів оплати та картки повару. Код HomeScreen застосунку представлено наведено в додатку А.

2.3. Інструкція користувача мобільного застосунку з продажу чаю та кави

Застосунок розроблено для операційних систем Android та iOS, для встановлення застосунку необхідно мати доступ в інтернет. Після встановленя додатку потрібно знайти іконку на мобільному телефоні та натиснути на неї.

Рисунок 2.17 – Іконка застосунку

Після входу в застосунок користувач опиняється на головній сторінці де можна побачити гамбургер меню, нижню навігацію, іконку користувача в правому верхньому куту, фільтри, стрічки перегляду та поле пошуку.

Рисунок 2.18 – Головний екран застосунку

Після натискання на товар користувач може побачити детальну інформацію про обрану позицію, а саме: назву товару, категорію, країну походження, ступінь обсмажування, рейтинг, детальна інформація та розміри товару. В нижній частині екрану зникла нижня навігація, але з’явилась ціна позиції та кнопка додавання до кошика “Add to cart”. В верхній частині екрану зліва розташована кнопка повернення назад, а справа кнопка вподобання після натискання на яку товар буде додану до розділу вподобань.

Рисунок 2.19 – Картка товару

На головному екрані додатку в нижній частині екрану ми бачимо панель навігації, де по порядку зліва на право розташовані кнопки: головний екран, кошик покупок, вподобання, історія замовлень. При переході на будь-який з цих екранів ми побачимо лише порожні вікна, оскільки ще нічого не було додано чи отримано в застосунку.

Рисунок 2.20 – Кошик покупок, вподобання, історія замовлень – порожні

Для того щоб заповнити кошик потрібно повернутись на головний екран і додати товар до кошика. Після додавання товарів в кошику з’явиться перелік товарі, можна побачити ціну обраних товару та змінити кількість одиниць кожної позиції, екран зображено на рисунок 2.21. Важливо, що при додаванні різних розмірів товару вони також окремо позначаються в замовлені і можна на кожен розмір регулювати кількість окремо. В нижній частині екрану можна побачити загальну суму замовлення та кнопку оплатити – “Pay”, після натискання на кнопку оплатити покупець переходить у вікно вибору способу оплати.

Рисунок 2.21 – Кошик покупок, заповнений

На даному екрані оплати, що зображений на рисунку 2.22, зображено три способи оплати: credit card, Google Pay та Apple Pay. В нижній частині екрану бачимо загальну суму замовлення та кнопку “Pay with Credit Card”, де Credit Card – це обраний спосіб оплати і надпис кнопки змінюється в залежності від вибору.

Рисунок 2.22 – Оплата замовлення

Після того як оплата була завершена користувач опиняється у вкладці історії замовлень “Order History”, де тепер відображається оформлена покупка. Окрім останнього замовлення на цій сторінці також будуть відображатись усі попередні замовлення користувача.

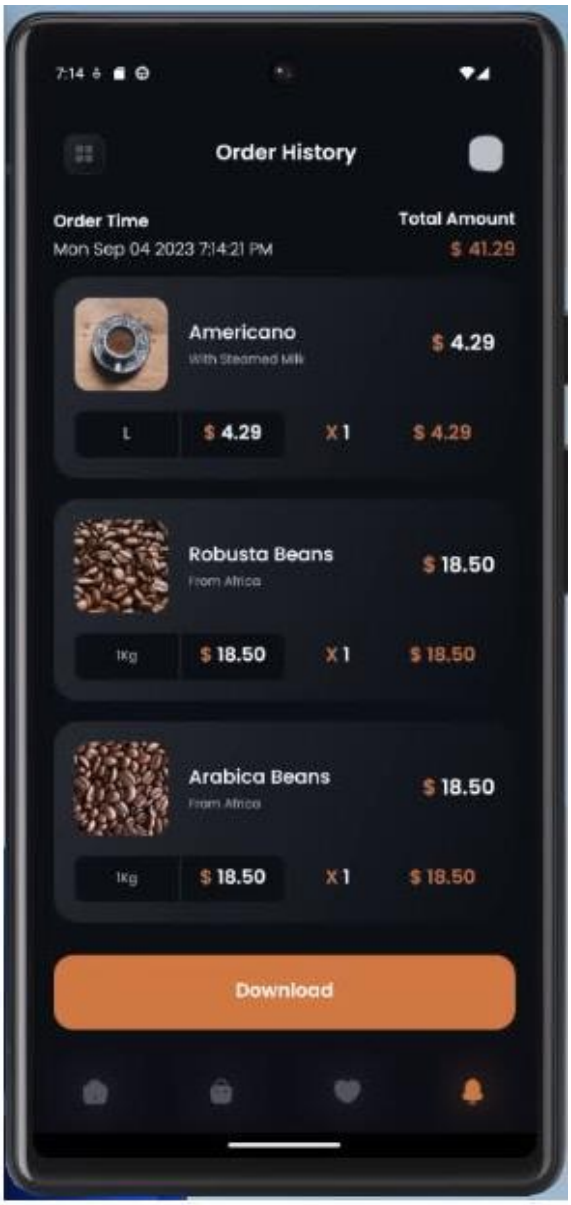

Рисунок 2.23 – Історія покупок

Під верхньою панеллю навігації знаходиться інформація про замовлення, зверху справа виділа оранжевим ціна, зліва – дата замовлення, після цього розташований перелік товарів та деталі про замовлення, знизу знаходиться кнопка завантаження квитанції про покупку “Download”.

Повертаємось до головного екрану та додаємо декілька товарів до обраного. Переходимо на екран вподобань “Favourites”, тепер на сторінці відображаються обрані товари.

Рисунок 2.24 – Екран вподобань

Загалом всю навігацію по застосунку можна описано в рисунку 25.

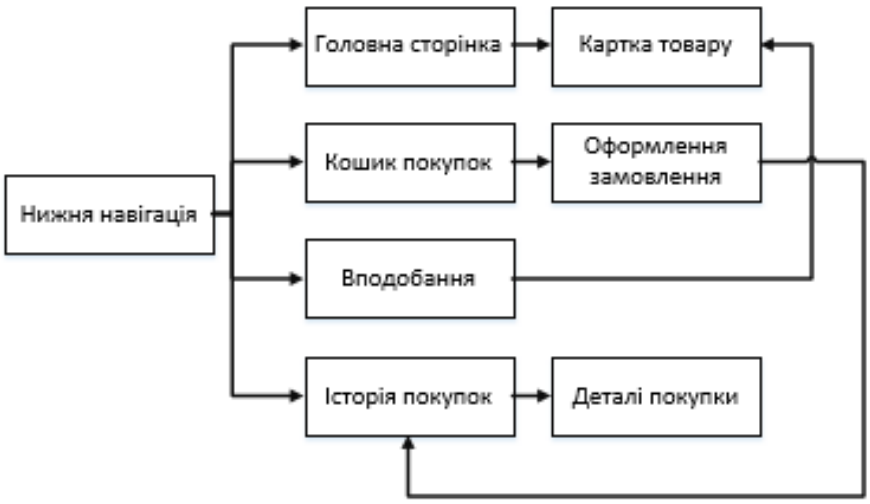

Рисунок 2.25 – Мапа навігації по застосунку

Після завершення роботи із за стосунком і його закриття всі данні, історія замовлень ,кошик покупок та вподобання зберігаються та будуть відображатись при наступному запуску додатку.

Висновки до розділу:

У цьому розділі було детально розглянуто процес розробки дизайну та мобільного застосунку для продажу чаю та кави. Початковим етапом було проведення дослідження, виконане в 1 розділі, та визначення основних функціональних вимог, що дозволило сформувати чітке уявлення про майбутній продукт . На основі цих даних був створений прототип інтерфейсу користувача, який враховував усі сучасні тенденції дизайну та забезпечував зручність і простоту використання.

Після затвердження прототипу розпочався етап розробки самого застосунку. Для реалізації було обрано кросплатформену технологію React Native, яка дозволила одночасно створювати додаток для iOS та Android, що значно знизило час і вартість розробки. Використання бібліотек та інструментів, таких як Redux для керування станом та React Navigation для організації навігації, забезпечило високу продуктивність і масштабованість застосунку.

Було розроблено функціонал для перегляду та вибору продуктів, додавання їх до кошика, обробки замовлень, а також перегляду історії покупок і збереження улюблених товарів. Особлива увага приділялася безпеці даних користувачів та інтеграції з платіжними системами для забезпечення зручного та безпечного процесу оплати.

Наступним кроком у цьому проекті було створення інструкції користувача, яка детально описує всі можливості застосунку та допомагає користувачам швидко освоїтися з ним. Також для зручності розуміння

навігації по застосунку була створена мапа навігації, що зображена на рисунку 2.25. Кінцевим результатом даного розділу є розроблений мобільний застосунок для продажу.

ВИСНОВКИ

У першому розділі випускною роботи було виконано аналіз розвитку ринку електронної комерції та актуальність цифровізації роздрібної торгівлі, були надані статистичні дані. Розгорнуто розглянуто дві мобільні операційні системи – Android та IOS, а також мови програмування та програмні застосунки для розробки нативних мобільних за стосунків. Надана статистика щодо популярності фреймворків для розробки кросплатформених мобільних застосунків, розглянуто фреймворки Flutter, React Native, Xamarin та .NET MAUI. Розглянуто застосунки для створення графічного інтерфейсу користувача, а саме : Figma, Invision, Marvel, UXPin та Justinmind. Було зроблено короткий огляд на конструктори для створення за стосунків, які дозволяють користувачам без досвіду програмування створювати власні додатки. У заключення першого розділу було розглянуто основний функціонал мобільних за стосунків для роздрібної торгівлі та створено діаграму варіантів використання.

У другому розділі було розроблено прототип графічного інтерфейсу для мобільного застосунку на основі онлайн-сервісу Figma. Використання React Native дозволило розробити кросплатформенний додаток.

Застосунок включає функції перегляду та вибору продуктів, додавання до кошика, обробки замовлень та збереження улюблених товарів. Завдяки сучасним технологіям забезпечено зручність, безпеку та високу продуктивність застосунку. Останнім кроком випускної роботи було створення інструкції користувача застосунку та мапи навігації по застосунку.

Розробка мобільного застосунку для продажу чаю та кави, яку ми виконали, демонструє як технічні, так і дизайнерські аспекти створення сучасного та ефективного інструменту для електронної комерції. У процесі розробки були враховані основні потреби користувачів, що дозволило створити інтуїтивно зрозумілий і зручний інтерфейс.

Загалом, цей проект показує, що за допомогою сучасних технологій і кращих практик розробки можна створити мобільний застосунок, який не тільки відповідає високим стандартам якості, але й задовольняє потреби користувачів у зручності та функціональності. Цей досвід буде цінним для майбутніх проектів у сфері електронної комерції та розробки мобільних додатків.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Koen van Gelder. E-commerce worldwide - statistics & facts.URL : https://www.statista.com/topics/871/online-shopping/#topicOverview.
2. Laura Ceci. Number of mobile app downloads worldwide from 2016 to 2023.URL : https://www.statista.com/statistics/271644/worldwide-free-and-paid-mobile-app-store-downloads/.
3. Smartphone Usage Statistics. URL : https://backlinko.com/smartphone-usage-statistics.
4. Mobile commerce revenue and share of total retail e-commerce worldwide from 2017 to 2028. URL : https://www.statista.com/statistics/1449284/retail-mobile-commerce-revenue-worldwide/
5. Laura Ceci. Retention rate on day 30 of mobile app installs worldwide in 3rd quarter 2023, by category. URL : https://www.statista.com/statistics/259329/ios-and-android-app-user-retention-rate/
6. Lionel Sujay Vailshery. Low-code and no-code platforms - statistics & facts. URL : https://www.statista.com/topics/8461/low-code-and-no-code-platforms/#topicOverview.
7. Mobile Operating System Market Share Worldwide - April 2024. URL : https://gs.statcounter.com/os-market-share/mobile/worldwide.
8. Lionel Sujay Vailshery. Cross-platform mobile frameworks used by developers worldwide 2019–2022. URL : https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/.
9. M. Hassenzahl, User Experience (UX): Towards an Experiential Perspective on Product Quality, in Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, 2018, pp. 1–16

10. Чиннатамбі В. Р. Практичне керівництво React Native: навч. посіб. Київ:ООО “Ексмо”, 2019. 368 с.
11. Хедрік Б. К. Вивчення React Native: навч. посіб. Дніпро: “ОРілі”, 2018. 251 с.
12. Мардан А. К. React швидко. Мобільні додатки на React Native, JSX, Redux: навч. посіб. Львів: “ОРілі”, 2019. 560 с.
13. Янг А. А., Кантелон М. Р. Node.js в дії: навч. посіб. Київ: ООО “Маннінг”, 2017. 378 с.
14. React Native документація. URL : https://reactnative.dev/ (дата звернення: 24.05.2024).
15. React.js документація. URL: https://uk.reactjs.org/ (дата звернення: 12.05.2024).
16. Янг А. А., Кантелон М. Р. Node.js в дії: навч. посіб. Київ: ООО “Маннінг”, 2017. 378 с.
17. Ronald N. Kostoff. The Handbook of Research Impact Assessment. Edition 7. Summer 1997.
18. Моркун Н. В., Маринич І. А. Методичні вказівки до виконання кваліфікаційної роботи бакалавру для студентів спеціальності 151 “Автоматизація та комп’ютерно-інтегровані технології”. Кривий Ріг : Видавничий центр КНУ, 2019. 50 с.
19. ДСТУ 3008:2015. Звіти у сфері науки і техніки. Структура і правила оформлення. Київ, ДП «УкрННЦ», 2015. 26с. (Інформація та документація).
20. ДСТУ 8302:2015. Бібліографічне посилання. Загальні вимоги та правила складання Київ, ДП «УкрННЦ», 2016. 16 с. (Інформація та документація).
21. ДСТУ 3582:2013. Бібліографічний опис. Скорочення слів і словосполучень в українській мові. Загальні вимоги та правила. Київ, ДП «УкрННЦ», 2013. 23 с. (Інформація та документація)

22.. ДСТУ 3651.0-97 Метрологія. Одиниці фізичних величин. Основні одиниці фізичних величин Міжнародної системи одиниць. Основні положення, назви та позначення Київ, Держстандарт України, 1998. 27 с. (Інформація та документація).

ДОДАТОК А

Код екрану HomeScreen:

```
import React, {useRef, useState} from 'react';
import {
  ScrollView,
  StatusBar,
  StyleSheet,
  Text,
  TextInput,
  TouchableOpacity,
  View,
  ToastAndroid,
} from 'react-native';
import {useStore} from '../store/store';
import {useBottomTabBarHeight} from '@react-navigation/bottom-tabs';
import {
  BORDERRADIUS,
  COLORS,
  FONTFAMILY,
  FONTSIZE,
  SPACING,
} from '../theme/theme';
import HeaderBar from '../components/HeaderBar';
import CustomIcon from '../components/CustomIcon';
import {FlatList} from 'react-native';
import CoffeeCard from '../components/CoffeeCard';
import {Dimensions} from 'react-native';

const getCategoriesFromData = (data: any) => {
  let temp: any = {};
  for (let i = 0; i < data.length; i++) {
    if (temp[data[i].name] == undefined) {
      temp[data[i].name] = 1;
    } else {
      temp[data[i].name]++;
    }
  }
  let categories = Object.keys(temp);
  categories.unshift('All');
  return categories;
};

const getCoffeeList = (category: string, data: any) => {
  if (category == 'All') {
    return data;
  } else {
    let coffeelist = data.filter((item: any) => item.name == category);
```

```
    return coffeelist;
  }
};

const HomeScreen = ({navigation}: any) => {
  const CoffeeList = useStore((state: any) => state.CoffeeList);
  const BeanList = useStore((state: any) => state.BeanList);
  const addToCart = useStore((state: any) => state.addToCart);
  const calculateCartPrice = useStore((state: any) => state.calculateCartPrice);

  const [categories, setCategories] = useState(
    getCategoriesFromData(CoffeeList),
  );
  const [searchText, setSearchText] = useState('');
  const [categoryIndex, setCategoryIndex] = useState({
    index: 0,
    category: categories[0],
  });
  const [sortedCoffee, setSortedCoffee] = useState(
    getCoffeeList(categoryIndex.category, CoffeeList),
  );

  const ListRef: any = useRef<FlatList>();
  const tabBarHeight = useBottomTabBarHeight();

  const searchCoffee = (search: string) => {
    if (search != '') {
      ListRef?.current?.scrollToOffset({
        animated: true,
        offset: 0,
      });
      setCategoryIndex({index: 0, category: categories[0]});
      setSortedCoffee([
        ...CoffeeList.filter((item: any) =>
          item.name.toLowerCase().includes(search.toLowerCase()),
        ),
      ]);
    }
  };

  const resetSearchCoffee = () => {
    ListRef?.current?.scrollToOffset({
      animated: true,
      offset: 0,
    });
    setCategoryIndex({index: 0, category: categories[0]});
    setSortedCoffee([...CoffeeList]);
    setSearchText('');
  };
```

```
const CoffeCardAddToCart = ({
  id,
  index,
  name,
  roasted,
  imagelink_square,
  special_ingredient,
  type,
  prices,
}: any) => {
  addToCart({
    id,
    index,
    name,
    roasted,
    imagelink_square,
    special_ingredient,
    type,
    prices,
  });
  calculateCartPrice();
  ToastAndroid.showWithGravity(
    `${name} is Added to Cart`,
    ToastAndroid.SHORT,
    ToastAndroid.CENTER,
  );
};

return (
  <View style={styles.ScreenContainer}>
    <StatusBar backgroundColor={COLORS.primaryBlackHex} />
    <ScrollView
      showsVerticalScrollIndicator={false}
      contentContainerStyle={styles.ScrollViewFlex}>
      {/* App Header */}
      <HeaderBar />

      <Text style={styles.ScreenTitle}>
        Find the best{'\n'}coffee for you
      </Text>

      {/* Search Input */}

      <View style={styles.InputContainerComponent}>
        <TouchableOpacity
          onPress={() => {
            searchCoffee(searchText);
          }}>
          <CustomIcon
            style={styles.InputIcon}
```

```
        name="search"
        size={FONTSIZE.size_18}
        color={
          searchText.length > 0
            ? COLORS.primaryOrangeHex
            : COLORS.primaryLightGreyHex
        }
      />
    </TouchableOpacity>
    <TextInput
      placeholder="Find Your Coffee..."
      value={searchText}
      onChangeText={text => {
        setSearchText(text);
        searchCoffee(text);
      }}
      placeholderTextColor={COLORS.primaryLightGreyHex}
      style={styles.TextInputContainer}
    />
    {searchText.length > 0 ? (
      <TouchableOpacity
        onPress={() => {
          resetSearchCoffee();
        }}>
        <CustomIcon
          style={styles.InputIcon}
          name="close"
          size={FONTSIZE.size_16}
          color={COLORS.primaryLightGreyHex}
        />
      </TouchableOpacity>
    ) : (
      <></>
    )}
  </View>

  {/* Category Scroller */}

  <ScrollView
    horizontal
    showsHorizontalScrollIndicator={false}
    contentContainerStyle={styles.CategoryScrollViewStyle}>
    {categories.map((data, index) => (
      <View
        key={index.toString()}
        style={styles.CategoryScrollViewContainer}>
        <TouchableOpacity
          style={styles.CategoryScrollViewItem}
          onPress={() => {
            ListRef?.current?.scrollToOffset({
```

```
            animated: true,
            offset: 0,
          });
          setCategoryIndex({index: index, category: categories[index]});
          setSortedCoffee([
            ...getCoffeeList(categories[index], CoffeeList),
          ]);
        }}>
        <Text
          style={[
            styles.CategoryText,
            categoryIndex.index == index
              ? {color: COLORS.primaryOrangeHex}
              : {},
          ]}>
          {data}
        </Text>
        {categoryIndex.index == index ? (
          <View style={styles.ActiveCategory} />
        ) : (
          <></>
        )}
      </TouchableOpacity>
    </View>
  ))}
</ScrollView>

{/* Coffee Flatlist */}

<FlatList
  ref={ListRef}
  horizontal
  ListEmptyComponent={
    <View style={styles.EmptyListContainer}>
      <Text style={styles.CategoryText}>No Coffee Available</Text>
    </View>
  }
  showsHorizontalScrollIndicator={false}
  data={sortedCoffee}
  contentContainerStyle={styles.FlatListContainer}
  keyExtractor={item => item.id}
  renderItem={({item}) => {
    return (
      <TouchableOpacity
        onPress={() => {
          navigation.push('Details', {
            index: item.index,
            id: item.id,
            type: item.type,
          });
```

```
          }}>
          <CoffeeCard
            id={item.id}
            index={item.index}
            type={item.type}
            roasted={item.roasted}
            imagelink_square={item.imagelink_square}
            name={item.name}
            special_ingredient={item.special_ingredient}
            average_rating={item.average_rating}
            price={item.prices[2]}
            buttonPressHandler={CoffeCardAddToCart}
          />
        </TouchableOpacity>
      );
    }}
  />

  <Text style={styles.CoffeeBeansTitle}>Coffee Beans</Text>

  {/* Beans Flatlist */}

  <FlatList
    horizontal
    showsHorizontalScrollIndicator={false}
    data={BeanList}
    contentContainerStyle={[
      styles.FlatListContainer,
      {marginBottom: tabBarHeight},
    ]}
    keyExtractor={item => item.id}
    renderItem={({item}) => {
      return (
        <TouchableOpacity
          onPress={() => {
            navigation.push('Details', {
              index: item.index,
              id: item.id,
              type: item.type,
            });
          }}>
          <CoffeeCard
            id={item.id}
            index={item.index}
            type={item.type}
            roasted={item.roasted}
            imagelink_square={item.imagelink_square}
            name={item.name}
            special_ingredient={item.special_ingredient}
            average_rating={item.average_rating}
```

```
                price={item.prices[2]}
                buttonPressHandler={CoffeCardAddToCart}
              />
            </TouchableOpacity>
          );
        }}
      />
    </ScrollView>
  </View>
);
};

const styles = StyleSheet.create({
  ScreenContainer: {
    flex: 1,
    backgroundColor: COLORS.primaryBlackHex,
  },
  ScrollViewFlex: {
    flexGrow: 1,
  },
  ScreenTitle: {
    fontSize: FONTSIZE.size_28,
    fontFamily: FONTFAMILY.poppins_semibold,
    color: COLORS.primaryWhiteHex,
    paddingLeft: SPACING.space_30,
  },
  InputContainerComponent: {
    flexDirection: 'row',
    margin: SPACING.space_30,
    borderRadius: BORDERRADIUS.radius_20,
    backgroundColor: COLORS.primaryDarkGreyHex,
    alignItems: 'center',
  },
  InputIcon: {
    marginHorizontal: SPACING.space_20,
  },
  TextInputContainer: {
    flex: 1,
    height: SPACING.space_20 * 3,
    fontFamily: FONTFAMILY.poppins_medium,
    fontSize: FONTSIZE.size_14,
    color: COLORS.primaryWhiteHex,
  },
  CategoryScrollViewStyle: {
    paddingHorizontal: SPACING.space_20,
    marginBottom: SPACING.space_20,
  },
  CategoryScrollViewContainer: {
    paddingHorizontal: SPACING.space_15,
  },
```

```
  CategoryScrollViewItem: {
    alignItems: 'center',
  },
  CategoryText: {
    fontFamily: FONTFAMILY.poppins_semibold,
    fontSize: FONTSIZE.size_16,
    color: COLORS.primaryLightGreyHex,
    marginBottom: SPACING.space_4,
  },
  ActiveCategory: {
    height: SPACING.space_10,
    width: SPACING.space_10,
    borderRadius: BORDERRADIUS.radius_10,
    backgroundColor: COLORS.primaryOrangeHex,
  },
  FlatListContainer: {
    gap: SPACING.space_20,
    paddingVertical: SPACING.space_20,
    paddingHorizontal: SPACING.space_30,
  },
  EmptyListContainer: {
    width: Dimensions.get('window').width - SPACING.space_30 * 2,
    alignItems: 'center',
    justifyContent: 'center',
    paddingVertical: SPACING.space_36 * 3.6,
  },
  CoffeeBeansTitle: {
    fontSize: FONTSIZE.size_18,
    marginLeft: SPACING.space_30,
    marginTop: SPACING.space_20,
    fontFamily: FONTFAMILY.poppins_medium,
    color: COLORS.secondaryLightGreyHex,
  },
});

export default HomeScreen;
```

Код екрану PaymentScreen:

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

```
  BORDERRADIUS,
  COLORS,
  FONTFAMILY,
  FONTSIZE,
  SPACING,
} from '../theme/theme';
import GradientBGIcon from '../components/GradientBGIcon';
import PaymentMethod from '../components/PaymentMethod';
import PaymentFooter from '../components/PaymentFooter';
import LinearGradient from 'react-native-linear-gradient';
import CustomIcon from '../components/CustomIcon';
import {useStore} from '../store/store';
import PopUpAnimation from '../components/PopUpAnimation';

const PaymentList = [
  {
    name: 'Wallet',
    icon: 'icon',
    isIcon: true,
  },
  {
    name: 'Google Pay',
    icon: require('../assets/app_images/gpay.png'),
    isIcon: false,
  },
  {
    name: 'Apple Pay',
    icon: require('../assets/app_images/applepay.png'),
    isIcon: false,
  },
  {
    name: 'Amazon Pay',
    icon: require('../assets/app_images/amazonpay.png'),
    isIcon: false,
  },
];

const PaymentScreen = ({navigation, route}: any) => {
  const calculateCartPrice = useStore((state: any) => state.calculateCartPrice);
  const addToOrderHistoryListFromCart = useStore(
    (state: any) => state.addToOrderHistoryListFromCart,
  );

  const [paymentMode, setPaymentMode] = useState('Credit Card');
  const [showAnimation, setShowAnimation] = useState(false);

  const buttonPressHandler = () => {
    setShowAnimation(true);
    addToOrderHistoryListFromCart();
    calculateCartPrice();
```

```
  setTimeout(() => {
    setShowAnimation(false);
    navigation.navigate('History');
  }, 2000);
};

return (
  <View style={styles.ScreenContainer}>
    <StatusBar backgroundColor={COLORS.primaryBlackHex} />

    {showAnimation ? (
      <PopUpAnimation
        style={styles.LottieAnimation}
        source={require('../lottie/successful.json')}
      />
    ) : (
      <></>
    )}

    <ScrollView
      showsVerticalScrollIndicator={false}
      contentContainerStyle={styles.ScrollViewFlex}>
      <View style={styles.HeaderContainer}>
        <TouchableOpacity
          onPress={() => {
            navigation.pop();
          }}>
          <GradientBGIcon
            name="left"
            color={COLORS.primaryLightGreyHex}
            size={FONTSIZE.size_16}
          />
        </TouchableOpacity>
        <Text style={styles.HeaderText}>Payments</Text>
        <View style={styles.EmptyView} />
      </View>

      <View style={styles.PaymentOptionsContainer}>
        <TouchableOpacity
          onPress={() => {
            setPaymentMode('Credit Card');
          }}>
          <View
            style={[
              styles.CreditCardContainer,
              {
                borderColor:
                  paymentMode == 'Credit Card'
                    ? COLORS.primaryOrangeHex
                    : COLORS.primaryGreyHex,
```

```
        },
      ]}>
      <Text style={styles.CreditCardTitle}>Credit Card</Text>
      <View style={styles.CreditCardBG}>
        <LinearGradient
          start={{x: 0, y: 0}}
          end={{x: 1, y: 1}}
          style={styles.LinearGradientStyle}
          colors={[COLORS.primaryGreyHex, COLORS.primaryBlackHex]}>
          <View style={styles.CreditCardRow}>
            <CustomIcon
              name="chip"
              size={FONTSIZE.size_20 * 2}
              color={COLORS.primaryOrangeHex}
            />
            <CustomIcon
              name="visa"
              size={FONTSIZE.size_30 * 2}
              color={COLORS.primaryWhiteHex}
            />
          </View>
          <View style={styles.CreditCardNumberContainer}>
            <Text style={styles.CreditCardNumber}>3879</Text>
            <Text style={styles.CreditCardNumber}>8923</Text>
            <Text style={styles.CreditCardNumber}>6745</Text>
            <Text style={styles.CreditCardNumber}>4638</Text>
          </View>
          <View style={styles.CreditCardRow}>
            <View style={styles.CreditCardNameContainer}>
              <Text style={styles.CreditCardNameSubitle}>
                Card Holder Name
              </Text>
              <Text style={styles.CreditCardNameTitle}>
                Robert Evans
              </Text>
            </View>
            <View style={styles.CreditCardDateContainer}>
              <Text style={styles.CreditCardNameSubitle}>
                Expiry Date
              </Text>
              <Text style={styles.CreditCardNameTitle}>02/30</Text>
            </View>
          </View>
        </LinearGradient>
      </View>
    </View>
  </TouchableOpacity>
  {PaymentList.map((data: any) => (
    <TouchableOpacity
      key={data.name}
```

```
              onPress={() => {
                setPaymentMode(data.name);
              }}>
              <PaymentMethod
                paymentMode={paymentMode}
                name={data.name}
                icon={data.icon}
                isIcon={data.isIcon}
              />
            </TouchableOpacity>
          ))}
        </View>
      </ScrollView>

      <PaymentFooter
        buttonTitle={`Pay with ${paymentMode}`}
        price={{price: route.params.amount, currency: '$'}}
        buttonPressHandler={buttonPressHandler}
      />
    </View>
  );
};

const styles = StyleSheet.create({
  ScreenContainer: {
    flex: 1,
    backgroundColor: COLORS.primaryBlackHex,
  },
  LottieAnimation: {
    flex: 1,
  },
  ScrollViewFlex: {
    flexGrow: 1,
  },
  HeaderContainer: {
    paddingHorizontal: SPACING.space_24,
    paddingVertical: SPACING.space_15,
    flexDirection: 'row',
    alignItems: 'center',
    justifyContent: 'space-between',
  },
  HeaderText: {
    fontFamily: FONTFAMILY.poppins_semibold,
    fontSize: FONTSIZE.size_20,
    color: COLORS.primaryWhiteHex,
  },
  EmptyView: {
    height: SPACING.space_36,
    width: SPACING.space_36,
  },
```

```
PaymentOptionsContainer: {
  padding: SPACING.space_15,
  gap: SPACING.space_15,
},
CreditCardContainer: {
  padding: SPACING.space_10,
  gap: SPACING.space_10,
  borderRadius: BORDERRADIUS.radius_15 * 2,
  borderWidth: 3,
},
CreditCardTitle: {
  fontFamily: FONTFAMILY.poppins_semibold,
  fontSize: FONTSIZE.size_14,
  color: COLORS.primaryWhiteHex,
  marginLeft: SPACING.space_10,
},
CreditCardBG: {
  backgroundColor: COLORS.primaryGreyHex,
  borderRadius: BORDERRADIUS.radius_25,
},
LinearGradientStyle: {
  borderRadius: BORDERRADIUS.radius_25,
  gap: SPACING.space_36,
  paddingHorizontal: SPACING.space_15,
  paddingVertical: SPACING.space_10,
},
CreditCardRow: {
  flexDirection: 'row',
  justifyContent: 'space-between',
  alignItems: 'center',
},
CreditCardNumberContainer: {
  flexDirection: 'row',
  gap: SPACING.space_10,
  alignItems: 'center',
},
CreditCardNumber: {
  fontFamily: FONTFAMILY.poppins_semibold,
  fontSize: FONTSIZE.size_18,
  color: COLORS.primaryWhiteHex,
  letterSpacing: SPACING.space_4 + SPACING.space_2,
},
CreditCardNameSubitle: {
  fontFamily: FONTFAMILY.poppins_regular,
  fontSize: FONTSIZE.size_12,
  color: COLORS.secondaryLightGreyHex,
},
CreditCardNameTitle: {
  fontFamily: FONTFAMILY.poppins_medium,
  fontSize: FONTSIZE.size_18,
```

```
    color: COLORS.primaryWhiteHex,
  },
  CreditCardNameContainer: {
    alignItems: 'flex-start',
  },
  CreditCardDateContainer: {
    alignItems: 'flex-end',
  },
});

export default PaymentScreen;
```