Київський національний торговельно-економічний університет

Кафедра комп'ютерних наук та інформаційних систем

ВИПУСКНИЙ КВАЛІФІКАЦІЙНИЙ ПРОЕКТ

на тему:

«Проектування та створення мобільного Web-додатку для електронної торгівлі на базі ОС Android»

Студента 2 курсу, 7м групи спеціальності 122 «Комп'ютерні науки» _____ *підпис студента* Веремієнка Андрія Михайловича

Науковий керівник кандидат фізико-математичних наук, доцент _____ *підпис керівника* Самойленко Анна Тимофіївна

Гарант освітньої програми доктор фізико-математичних наук, професор _____ *підпис керівника* Пурський Олег Іванович

Київ 2020

Київський національний торговельно-економічний університет

Факультет інформаційних технологій
Кафедра комп'ютерних наук та інформаційних систем
Спеціальність 122 «Комп'ютерні науки»
Спеціалізація «Комп'ютерні науки»

Затверджую
Зав. кафедри _____Пурський О.І.
«5» грудня 2019р.

Завдання
на випускний кваліфікаційний проект студенту

Веремієнко Андрію Михайловичу
(прізвище, ім'я, по батькові)

1. Тема випускного кваліфікаційного проекту
«Проектування та створення мобільного Web-додатку для електронної торгівлі на базі ОС Android»
Затверджена наказом ректора від *«02» грудня 2019 р. № 4110*

2. Строк здачі студентом закінченого проекту *05 листопада 2020 року*

3. Цільова установка та вихідні дані до проекту
Мета роботи: проектування та розробка мобільного Web-додатку для електронної торгівлі на базі ОС Android.
Об'єкт дослідження: процес розробки універсального мобільного Web-додатку для електронної торгівлі.
Предмет дослідження: програмні засоби розробки та проектування Web-додатку.

4. Перелік графічного матеріалу_____
_____
_____
_____

5. Консультанти по роботі із зазначенням розділів, за якими здійснюється консультування:

Розділ	Консультант (прізвище, ініціали)	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Самойленко А.Т.	5.12.2019 р.	5.12.2019 р..
2	Самойленко А.Т.	5.12.2019 р.	5.12.2019 р.
3	Самойленко А.Т.	5.12.2019 р.	5.12.2019 р.

6. Зміст випускного кваліфікаційного проекту (перелік питань за кожним розділом)

ВСТУП

РОЗДІЛ 1. ТЕОРЕТИКО-МЕТОДОЛОГІЧНІ ЗАСАДИ МОБІЛЬНОЇ РОЗРОБКИ В СФЕРІ E-COMMERCE.

1.1. Поняття e-commerce. Переваги та недоліки e-commerce.

1.2. Мобільні додатки в сфері e-commerce. M-commerce.

1.3. Вибір ОС для розробки Web-додатку.

Розділ 2. РОЗРОБКА МОДЕЛЕЙ ДОДАТКУ ТА БАЗИ ДАНИХ. ПІДБІР ІНСТРУМЕНТІВ ДЛЯ РОЗРОБКИ WEB-ДОДАТКУ.

2.1. Опис концепції додатку (MVVM)

2.2. Модель бази даних та хмарного сховища.

2.3. Підбір інструментів для розробки мобільного web-додатку

РОЗДІЛ 3 ПРОЕКТУВАННЯ ТА РОЗРОБКА МОБІЛЬНОГО WEB-ДОДАТКУ ДЛЯ ЕЛЕКТРОННОЇ ТОРГІВЛІ

3.1 Проектування архітектурі та опис функціоналу внутрішніх модулів додатку

3.2 Програмна реалізація додатку та бази даних. Опис класів.

3.3 Мануал по використанню мобільного Web-додатку для електронної торгівлі

ВИСНОВКИ

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

ДОДАТОК

7. Календарний план виконання проекту

№ Пор.	Назва етапів випускної кваліфікаційної роботи	Строк виконання етапів роботи	
		За планом	фактично
1	2	3	4
1	*Вибір теми випускного кваліфікаційного проекту*	*01.11.2019*	*01.11.2019*
2	*Розробка та затвердження завдання для випускного кваліфікаційного проекту*	*05.12.2019*	*05.12.2019*
3	*Вступ*	*01.06.2020*	
4	*РОЗДІЛ 1. Теоретичні аспекти оцінки конкурентоспроможності підприємства*	*25.06.2020*	
5	*РОЗДІЛ 2. Математичні моделі оцінки та управління конкурентоспроможністю*	*02.09.2020*	
6	*Підготовка статті у збірник наукових статей магістрів*	*09.09.2020*	
7	*РОЗДІЛ 3. Інформаційна технологія оцінки конкурентоспроможності підприємств електронної торгівлі*	*21.10.2020*	
8	*Висновки*	*02.11.2020*	
9	*Здача випускного кваліфікаційного проекту на кафедру науковому керівнику*	*05.11.2020*	
10	*Попередній захист випускного кваліфікаційного проекту*	*20.11.2020*	
11	*Виправлення зауважень, зовнішнє рецензування випускного кваліфікаційного проекту*	*22.11.2020*	
12	*Представлення готової зшитого випускного кваліфікаційного проекту*	*25.11.2020*	
13	*Публічний захист випускного кваліфікаційного проекту*	*Згідно роботи ЕК*	

8. Дата видачі завдання «5» грудня 2019 р.

9. Керівник випускного кваліфікаційного проекту

Самойленко А. Т.
(прізвище, ініціали, підпис)

10. Гарант освітньої програми Пурський О.І.
(прізвище, ініціали, підпис)

11. Завдання прийняв до виконання студент-дипломник

Веремієнко А. М.
(прізвище, ініціали, підпис)

12. Відгук керівника випускного кваліфікаційного проекту

_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____

Керівник випускного кваліфікаційного проекту

_____
(підпис, дата)

13. Висновок про випускний кваліфікаційний проект

Випускний кваліфікаційний проект студента _____
(прізвище, ініціали)
може бути допущений до захисту в екзаменаційній комісії.

Гарант освітньої програми_____Пурський О.І.
(підпис, прізвище, ініціали)

Завідувач кафедри_____Пурський О.І.
(підпис, прізвище, ініціали)
«_____»_____2020 р.

Анотація

У випускному кваліфікаційному проекті було зроблено проектування за сучасними технологіями мобільного Web-додатку для електронної торгівлі з метою створення універсального додатку, який можна легко впровадити в будь-який проект з продажу товарів, не витрачаючи зайві кошти та час на розробку. Теоретично обґрунтовано переваги m-commerce над іншими варіантами e-commerce. Описані основні концепції та інструменти для розробки мобільного Web-додатку для електронної торгівлі на базі ОС Android. Були представлені переваги операційної системи Android над іншими ОС на території України. Створено мобільний Web-додаток для електронної торгівлі.

Ключові слова:e-commerce, m-commerce,Android, web-додаток, MVVM, база даних.

Abstract

In the final qualification work, a mobile Web-application for e-commerce was designed using modern technologies in order to create a universal application that can be easily implemented in any project for the sale of goods without spending extra money and time on development. The advantages of m-commerce over other variants of e-commerce are theoretically substantiated. Describes the basic concepts and tools for developing a mobile Web-application for e-commerce on Android. The advantages of the Android operating system over other operating systems in Ukraine were presented. A mobile Web-application for e-commerce has been created.

Keywords: e-commerce, m-commerce, Android, web-application, MVVM, database.

ЗМІСТ

ВСТУП

Електроннаторгівля через мобільні додатки - один з напрямківонлайн-торгівлі,що стрімко розвивається. Зростання покупок, здійснених через мобільні пристрої, відзначають всі провідні світові аналітичні агентства по e-commerce.

E-commerce - це сфера економіки, коли торгові і фінансові операції проводяться в інтернеті. Тобто будь-яка транзакція, виконана з електронного пристрою, підключеного до мережі.

У такій динамічній і конкурентній галузі, як електронна комерція, йти в ногу з тенденціями і змінами - життєва необхідність.Аналітичне агентство eMarketer прогнозує, що кошти, витрачені на мобільну рекламу в 2019 році, складуть 232,34 мільярда доларів у всьому світі. Більш того, згідно зі статистикою, глобальний трафік мобільної передачі даних збільшиться майже в 3 рази в період з 2019 по 2021 рік. Причиною цього є факт зростання використання мобільних телефонів, особливо додатків. Експерти також відзначають, що 87 відсотків мобільних користувачів проводять найбільшу кількість часу саме в мобільних додатках.

Згідно TechCrunch, мобільні додатки мають набагато більш високий рівень залученості, ніж веб-сайти, оптимізовані для мобільних пристроїв і десктопні версії. У мобільних додатках також більш високі показники конверсії - на 100-300 відсотків. Подібне збільшення продажів викликане як збільшенням обсягу, так і частоти відвідувань. Як показує аналіз Bain & Co, користувачі мобільних додатків проходять «шлях клієнта» в 3 рази швидше і бачать в 4,2 рази більше продуктів, ніж інші користувачі з різних каналів.

За цими цифрами стоять конкретні дії і інструменти, які можна використовувати в адаптивних версіях сайтів для мобільних пристроїв. Перш за все, додатки запускаються і завантажуються швидше ніж мобільні сайти.. Згідно зі звітом App Annie, час, проведений в додатках для покупок, зросла до 18 мільярдів годин в 2018 році по всьому світу. Такі показники безпосередньо пов'язані з можливостями миттєвого зв'язку з користувачами,

що дає маркетологам безліч можливостей для їх залучення - через додаток, але також і за допомогою push-повідомлень, електронної пошти та смс-повідомлень.

Це підводить нас до найбільш важливої функції мобільних додатків, основна причина використання яких – персоналізація - від надання пропозицій і програми лояльності до геолокації і моніторингу залученості користувачів. В цілому, середа додатків - відмінна область тестування різних маркетингових підходів і тактик, які, в кінцевому підсумку, приведуть до більш високих показників конверсії. Крім того, є ще одна річ, яку варто знати. Аналіз Compuware показує, що 85 відсотків споживачів воліють вбудовані додатки, а не мобільні сайти.

В той же самий час – перевага мобільних додатків частково нівелюється затратами на розробку та підтримку додатків, а саме затрати фінансів та часу. Не кожна компанія зможе виділити достатню кількість коштів, адже є ризик, що компанія (особливо якщо це стартап) взагалі через певний час припинить існування, й витрачати зайві кошти на розробку не має сенсу. Саме тому постає необхідність в легкому додатку, який не потрібно буде розробляти, а тільки підтримувати й стилізувати. Тобто замість команди розробників буде потрібен тільки один розробник рівня Junior.В сучасному світі стартапів, **актуальність** та пріорітет цієї задачі є на одному з найвищих рівнів.

Мета і завдання дослідження : метою даного дослідження є проектування та створення мобільного Web-додатку для електронної торгівлі на базі ОС Android.Для досягнення поставленої мети необхідно було вирішити наступні **завдання:**

- Проектування архітектури мобільного додатку
- Створення хмарної бази даних для зберігання товарів
- Створення локальної бази даних для зберігання даних користувача
- Розробка зрозумілого графічного інтерфейсу для користувача.
- Розробка програмних модулів для функціонування додатку

- Реалізація зворотної сумісності, з мобільними пристроями попередніх версій операційної системи Android, розпочинаючи з Android 5.0 (Lollipop);
- Створення додатку з можливістю його модернізації та підтримки.

Об’єкт дослідження: процес розробки універсального мобільного Web-додатку для електронної торгівлі.

Предмет дослідження: програмні засоби розробки та проектування Web-додатку.

Практичне значення. Отриманий додаток можна буде поширювати та використовувати стартапам на початку їх розвитку – не витрачаючи зайві кошти та час на розробку. Додаток буде поширюватися безкоштовно.

Наукова новизна полягає в порівнянні кросплатформених та нативних рішень та створення таблиці для вибору конкретного рішення відповідно до завдань.

РОЗДІЛ 1. ТЕОРЕТИКО-МЕТОДОЛОГІЧНІ ЗАСАДИ МОБІЛЬНОЇ РОЗРОБКИ В СФЕРІ E-COMMERCE.

1.1. Поняття e-commerce. Переваги та недоліки e-commerce.

E-commerce або електронна комерція - це підприємницька діяльність, яка, так чи інакше, пов'язана з поширенням, рекламуванням, просуванням, продажем послуг або товарів через Інтернет. Тобто, будь-які дії з комерційним ухилом в глобальній мережі підпадають під визначення онлайн-комерція[1].

Електронна комерція об'єднує такі глобальні категорії як - онлайн-продаж, інтернет-банкінг, бронювання квитків і готелів, транзакції в платіжних системах, онлайн-маркетинг і реклама.

З технічної точки зору електронна комерція в Інтернеті стоїть на трьох основних елементах - сервер, база даних і система доставки товару або послуг покупцеві. Критично важлива перша складова - якісний і швидкий сервер. Доставка електронних товарів або послуг не вимагає складної логістики.

Сфера E-commerce підрозділяється на види залежно від цільової аудиторії, з якою працює компанія, найбільш популярні на даний час є такі сфери :

B2B (Business-to-Business). Ніша «Бізнес для бізнесу» має на увазі комерційні відносини між юридичними особами, економічними суб'єктами ринку. Тобто компанії, виробники взаємодіють між собою - укладають угоди, партнерські контракти на поставку, продаж, покупку товарів або послуг. Для налагодження контактів, пошуку партнерів і переговорів в B2B використовуються спеціалізовані інтернет-майданчики, інтерактивні бази даних[2].

B2C (Business-to-Consumer). Сфера «Бізнес для споживача» передбачає торгівлю товарами і послугами між юридичними і фізичними особами. Це свого роду роздрібні продажі, але тільки за допомогою онлайн-майданчиків - магазини, сервіси, банки та інше. Перевага клієнтів в більшому асортименті вибору, зручність замовлення та доставки товарів додому або в офіс. Електронна комерція дозволяє підприємцю знизити витрати на утримання торгових і складських площ[2].

C2C (Consumer-to-Consumer). Електронна комерція в ніші «Споживач для споживача» має на увазі здійснення угод між фізичними особами. Успіх таких інтернет-майданчиків як Пром, Olx, Ebay і інших заснований на комерційних відносинах користувачів через електронну систему оголошень[2].

Переваги:

- Зниження затрат. Електронна торгівля спрощує бізнес-процеси в багатьох галузях підприємництва. Наприклад, щоб відкрити онлайн-магазин не потрібно орендувати фізичну площу, наймати штат продавців і співробітників доставки. Всі дії можна автоматизувати. В результаті транзакційні витрати нижче, що позначається на вартості продукції або послуг.
- Розширення цільової аудиторії. Через інтернет можна продавати по всьому світу без особливих витрат. Звичайно, якщо бізнес пов'язаний з фізичними товарами, то доведеться вирішити проблеми з доставкою. Просте рішення - це співпраця з транспортними компаніями. Але якщо продаються електронні товари, послуги - електронні книги, програми, сервіси та інше, то ринок необмежений.
- Менше посередників. Електронна комерція дозволяє працювати безпосередньо з виробником, виключаючи ланцюжок посередників. Так створюється прямий канал між продавцем і покупцем, що позначається на вартості товарів і якості обслуговування.

- Можливість зі 100% точністю аналізувати продажу, просування, розвиток бізнесу в мережі. Системи аналітики, колл-трекінга дозволяють стежити за ситуацією і своєчасно вживати заходів.

Недоліки :

- Залежність від інформаційно-комунікаційних технологій. Не у всіх регіонах є вільний доступ до інтернету на високій швидкості, цей фактор сильно гальмує розвиток електронного бізнесу.
- Особливості законодавства, податки. Відсутність правового регулювання онлайн-комерції часто служить перешкодою при укладанні тих чи інших угод.
- Безпека інформації. Онлайн-торгівля і бізнес в мережі вимагає високої гарантії конфіденційності даних користувачів, покупців, учасників комерційної діяльності. Активно впроваджується сертифікація, авторизація, капча і інші варіанти боротьби з шахрайством.
- Авторське право. Захист прав власності - це давно не нова проблема для мережі Інтернет. Піратські копії програмного забезпечення, «злиті» у вільний доступ майстер-класи, книги і інша продукція інтелектуальної праці - все це стає проблемою для електронної комерції у всьому світі

1.2. Мобільні додатки в сфері e-commerce. M-commerce.

З розвитком гаджет-індустрії поступово зростає частка покупок, зроблених за допомогою мобільних пристроїв, що виділяє мобільну комерцію в окремий сегмент E-commerce.

За даними, отриманими зі звіту про стан мобільної комерції Criteo, в 2015 році ринок M-commerce становив близько третини світового E-commerce (34%). Серед лідерів в плані мобільних покупок - Японія, Південна Корея і США. У першому кварталі 2015 року Японія і Південна Корея

вперше показали найбільшу частку (понад 50%) транзакцій e-commerce через мобільні пристрої.

У сегменті нефізичних товарів і послуг мобільної комерції виділяються наступні категорії:

- Послуги стільникового зв'язку;
- Авіа і залізничні квитки, а також квитки на розважальні заходи;
- Телекомунікаційні послуги (крім стільникового зв'язку), включаючи послуги інтернет-провайдерів і операторів фіксованого зв'язку, платного телебачення, VOIP;
- Оплата рахунків і штрафів, включаючи оплату послуг ЖКГ, штрафів ДАІ, придбання страхових продуктів;
- Цифровий мобільний контент, що включає в себе мобільний аудіо-контент, мобільний відео-контент, мобільні ігри та програми, інший мобільний контент
- Готельні послуги;

У сегменті покупки і оплати фізичних товарів виділяються такі категорії, як:

- Мережевий маркетинг (MLM);
- Каталожна торгівля;
- Інтернет магазини.

Якщо порівнювати мобільну і електронну комерцію, то на даний момент в онлайн-продажах десктоп продовжує тримати пальму першості і являє собою основний генератор кліків і конверсій. Але мобільні пристрої його стрімко наздоганяють. Девайси надають аналітикам дуже важливу інформацію про час доби та місцезнаходження покупця в момент здійснення покупки, а також те, на якій стадії "подорожі" він зараз перебуває. На даний момент часу всього ¼ частина онлайн-покупок припадає на мобільні телефони. З персонального комп'ютера покупку зробити дійсно зручніше: є можливість розглянути товар, почитати відгуки, розмір екрану більше ніж на телефоні або планшеті, що дуже важливо при заповненні різних текстових полів. Але

якщо, для порівняння, користувачеві необхідно викликати таксі, то він явно зробить вибір на користь смартфона або планшета, тому що це набагато швидше і зручніше, покупка не коштує великих грошей, і, до того ж, не потрібно робити багато дій.

Власники онлайн-бізнесу, враховуючи мінливу ситуацію на ринку, давно зрозуміли, що необхідно звернути свою пильну увагу на мобільний аудиторію. Мобільні сайти продовжують втрачати популярність серед користувачів, упор на них буде доцільний тільки в двох випадках: по-перше, якщо користувач десктопа кардинально відрізняється від користувача мобайла в конкретному сегменті, по-друге, якщо з якихось причин створення додатка на даний момент часу не є можливим.

Багато компаній, що працюють в сегменті онлайн-продажів, пробують і перший, і другий варіант з метою виявлення реакції споживача на новий формат, щоб відпрацювати всі технологічні моменти. Витрати на створення мобільної програми набагато вище, ніж на адаптацію сайту під мобільні пристрої.

Плюс до всього, не вся цільова аудиторія дізнається про запуск мобільного додатку. Відповідно це чергові витрати, тільки вже на рекламу і просування мобільного додатка.

M-commercесильно вплинула на ринок e-commerce, а саме:

- Надали зручний купівельний досвід. Згідно з дослідженням Walker, до кінця 2020 року купівельний досвід стане важливіше ціни і буде грати головну роль у виборі бренду. І коли мова заходить про мобільних додатках, можна сказати, що саме вони надають той самий зручний досвід[3].
- Сфокусували клієнта на конкретній меті. На відміну від сайтів, що містять велику кількість інформації про компанії, мобільні додатки орієнтовані на конкретну мету. Користувач знаходить зручним отримати доступ саме до того, що він шукає. Таким чином, замість того

щоб переглядати веб-сайт, клієнт вважає за краще використовувати рішення в один клік[3].

- Налаштовані саме під смартфони це абсолютно очевидний факт. Саме через залежності від смартфонів мобільні додатки процвітають. Вони побудовані таким чином, що здатні використовувати більшість функцій смартфонів: камеру, Bluetooth, GPS. Ці функції полегшують онлайн-платежі, а також пошук потрібної інформації та отримання персоналізованих повідомлень[3].
- Працюють і в автономному режимі (коли немає швидкого інтернету). На відміну від веб-сайтів, деякі мобільні додатки піклуються про це і включають в себе деякі автономні функції. Додатки також дозволяють управляти тим, скільки інтернет-трафіку використовується. Саме тому є Facebook Lite для додатку Facebook або Skype Lite - для Skype[3].
- Пропонують зручну навігацію. Не доводиться сумніватися в тому, що смартфони стають розумнішими з кожним днем, і не можна залишати поза увагою зручність, яку вони пропонують. Смартфон пропонує практично все, що і ноутбук. Тому частіше клієнт вважатиме за краще плавно рухатися по мобільному додатку, а не веб-сайті, які іноді навіть не адаптовані під мобільні екрани[3].
- Швидкодія. Якщо є два варіанти - замовити товар з програми бренду або замовити те ж саме з веб-сайту –буде обране перше. Чому клієнт повинен пройти через клопоти у вигляді введення URL-адреси в браузері, чекаючи завантаження сайту, знайти елемент, який він шукає, почекати ще трохи, коли кожна сторінка завантажиться, а потім виконати свою мету, якщо він може зробити те ж саме за кілька кліків. Мобільні додатки швидкі порівняно з веб-сайтами, і це покращує користувальницький досвід клієнтів - а саме його вони і шукають[3].
- Допомагають знизити відсоток відмови від кошика. Через довгий і складний процес оформлення замовлення, близько 70% клієнтів відмовляються від свого кошика під час оформлення замовлення.

Мобільні додатки запобігають цьому завдяки можливості оформлення замовлення в один клік. Також вони можуть зберігати облікові дані різних способів оплати, що робить оформлення замовлення ще більш плавним[3].

Мобільні додатки пропонують набагато кращу взаємодію з клієнтами в порівнянні з веб-сайтами. Оскільки залучення клієнтів стає надзвичайно важливим, мобільні додатки розглядаються як самий унікальний спосіб взаємодії. Вони роблять це, використовуючи функції смартфона.

GPS визначає місцеположення клієнта, і мобільні додатки можуть потім відображати наявність певного продукту для доставки в цій конкретній галузі та кількість днів, необхідних для доставки продукту. GPS також дозволяє мобільним додаткам автоматично передавати адресу.Крім того, функція GPS дозволяє клієнтам відстежувати місце розташування своїх об'єктів в режимі онлайн і знати, де перебуває продукт.

Мобільні додатки навіть не вимагають від користувача вводити назву продукту, для цього досить використовувати функцію аудіо-пошуку. З поширенням голосового типу пошуку, більшість користувачів вже вважають цю функцію обов'язковою.

Мобільні додатки пропонують користувачеві високо персоналізований досвід. Саме завдяки інтерактивному інтерфейсу клієнти витрачають більше часу на додатки, що, в свою чергу, збільшує шанси на продажу. Для досягнення персоналізації в додатку використовуються такі функції, як спливаючі або діалогові вікна, пошук рекомендацій по місцю розташування, нагороди або знижки[4].

Push-повідомлення використовуються для відправки персоналізованих повідомлень клієнтам про поточні акції та нові надходження. Повідомлення постійно нагадують клієнтам про програму, продукти, які вони додали в кошик, і наявності доданого товару.Персоналізовані нагороди, такі як безкоштовні товари, додаткові знижки або Кешбек, - це ще один спосіб завоювати лояльність клієнтівх[4].

Штучний інтелект робить мобільні додатки ще більш привабливими. Завдяки чат-ботам користувачі можуть швидко отримувати відповіді на свої запити 24/7, 365 днів у році. Користувачі також можуть використовувати візуальний пошук, в якому продукт шукається за допомогою візуальних елементів (наприклад, зображень).

Чат-бот сьогодні також використовується не тільки для відповіді на запити клієнтів, але і для надання рекомендацій по стилю і продукту. Ставлячи різні запитання, чат-бот збирає інформацію про стильові переваги користувача і пропонує потрібний продукт на основі зібраних даних.Крім того, деякі програми вже створюють віртуальні примірочні - і це теж завдяки штучному інтелекту[5].

Близько дві третини трафіку електронної комерції безпосередньо надходить з мобільних додатків, і близько 89% користувачів із задоволенням використовують мобільні додатки, коли мова заходить про оформлення замовлень в інтернеті.

Таким чином, всі ці цифри показують, що мобільні додатки більш зручні, ніж веб-сайти, і дозволяють більше продавати інтернет-магазинам.Крім того, мобільні додатки скоротили розрив між бізнесом і клієнтами, наблизивши їх один до одного.

1.3. Вибір ОС для розробки Web-додатку.

Вже на стадії проектування мобільного додатку важливо розуміти, яку мову вигідніше використовувати для конкретного проекту. Поряд з нативною розробкою (наприклад, для iOS - Swift або Objective-C, для Android - Java або Kotlin), використовуються Кросплатформені фреймворки, такі як React Native і Flutter. У світі вже близько п'яти мільярдів смартфонів, за різними оцінками, до 80% з них використовують операційну систему Android, і менше 20% - iOS. І все ж у кожній країні є свої особливості, так, в США більше 65% смартфонів працюють на iOS. При створенні мобільних додатків

найчастіше потрібно випустити версії як на iOS, так і на Android. Для цього можна звернутися до нативної або кроссплатформенной («гібридної») розробки[6].

Нативна розробка - це класичне рішення, яке вимагає писати програми під кожну платформу окремо, використовуючи різні мови і з огляду на особливості кожної платформи. При створенні декількох версій над проектом одночасно працюють кілька команд.

Завдяки кросплатформнним фреймворками, з'явилася можливість «вбити двох зайців» разом і підготувати версії для iOS і Android за допомогою одного інструмента. Серед фреймворків особливо широкого поширення набули:

- React Native від Facebook
    - для додатків iOS, Android і Windows
    - використовує мову JavaScript і бібліотеку React.js як основний засіб розробки.
- Flutter від Google
    - для додатків Android, IOS і Fuchsia
    - використовує мову Dart, який також служить для веб-програмування.[6]

Як нативна, так і кросплатформенна розробка мають свої особливості. У числі переваг нативної розробки можна відзначити наступні:

- стабільна і швидка робота програми;
- максимальний термін життя додатка;
- додаток більш гнучкий і масштабується, завдяки використанню «рідних» інструментів;
- менше обмежень в архітектурі та функціях;
- інтерфейс в точності відповідає платформі.

Кросплатформенні фреймворки «підганяють» додаток під кілька операційних систем, тому немає необхідності створювати унікальні елементи для кожної платформи. В результаті:

- потрібно менше фахівців;
- йде менше часу і ресурсів;
- швидкість розробки підвищується.

Якщо логіка додатка однакова на всіх платформах, а інтерфейс простий, гібридна розробка допомагає швидше вивести продукт на ринок. React Native і Flutter рекомендуються, за таких умов:

- якщо в стислі терміни потрібно написати прототип додатка для декількох платформ;
- якщо додаток бере всю інформацію з сервера, логіка реалізована на сервері, а анімація і інтерфейс не мають принципового значення;
- якщо використовується мінімальна кількість нативних можливостей, таких як push повідомлення, вбудовані покупки, геолокація і т.п .;
- якщо допустима скромна продуктивність, а контент завантажується з Інтернету.

Є декілька категорій по яким можна порівняти обидва варіантів розробки, які будуть наведені далі.

Нативні технології потрібні, коли додаток розрахований на тривалу роботу (інакше кажучи, "термін життя"). Також це вигідно при наявності потреби у високій продуктивності, складному інтерфейсі і анімації, низькому енергоспоживанні, інтеграції зі сторонніми ресурсами (API і ін.). Нативні додатки більш вигідні в перспективі за рахунок зниження витрат на технічну підтримку[7].

Кросплатформені технології використовуються для швидкої перевірки гіпотез, розробки прототипів і додатків з явним обмеженням терміну експлуатації, наприклад, розроблених для певних заходів.

Час виведення на ринок (time-to-market) визначається, в першу чергу, розміром доступної команди і особливостями мобільного додатка.

Робота з кросплатформеними фреймворками може виявитися вигідніше і швидше, якщо додаток простий, з однаковим UI, без платформо-специфічних деталей, таких як доступ до камери, робота з файловою

системою і відбитками пальців, runtime permissons. Тут гібридна розробка дозволяє заощадити час за рахунок вторинного використання практично всього коду і UI під дві платформи. Однак, при необхідності створення складних кастомних view кросплатформенна розробка сповільнюється.

Для бізнесу - при виборі мови важливо оцінити їїнадійність і безпеку на сьогоднішній день з технічної точки зору, а також перспективи подальшого розвитку, ризики старіння.

У питанні надійності нативна розробка випереджає всіх своїх конкурентів. Розвиток основних бібліотек йде не один рік, в них вже виправили більшість багів, нативні мови - такі, як Java, Objective-C, Swift, Kotlin - постійно розвиваються. Серед мобільних розробників можна почути думки, що в 2020-х роках нативн розробку на Android частково витіснить Flutter, але поки це всього лише припущення.

React Native надає всі інструменти для створення безпечних мобільних додатків, приклад тому - Skype, Instagram, Facebook і інші відомі продукти. Побоювання за безпеку можливі тільки при використанні сторонніх модулів при розробці. При цьому JavaScript активно розвивається, випускаються нові фічі, в доступному для огляду майбутньому ризик старіння мінімальний.

У разі розробки на Flutter ризики вищі, оскільки фреймворк молодий, реліз вийшов тільки в грудні 2018 року. Поки що бувають проблеми, наприклад, в той чи інший момент складання бібліотек доступна тільки під одну платформу, трапляються збої в Android Studio, є баги в деяких плагінах і бібліотеках. З іншого боку, все це допрацьовують і виправляють. Не можна виключати ризик, що Google припинить підтримку Flutter, як це вже було з іншими проектами компанії. Однак, на Flutter написана Fuchsia OS, в якій деякі розробники бачать заміну Android.

Вище були описані чинники, які враховує як бізнес, так і виконавець. Також мають місце технологічні критерії, якими опікується, в першу чергу, менеджер проекту. наприклад:

1. Рівень знання нативних мов і переваги команди. Кожна мобільна студія має свої переваги у виборі технологій. Нативна розробка вимагає максимально повного знання відповідних мов. Однак, завдяки використанню нативних засобів систем, менше обмежень і складності при кастомізації або здійсненні доступу до платформо-специфічним інструментам (на відміну від React Native і Flutter). При наявності досвіду в JavaScript мобільний розробник може досить легко перейти на React Native (не потрібно додатково вивчати мову Dart, як у випадку з Flutter) або на Dart (великим плюсом буде знання TypeScript).

React Native під капотом задіє нативні модулі. Як наслідок, якщо виникає потреба в кастомізації (і це не підтримується з «коробки»), необхідно працювати з модулями native.

Flutter, на відміну від React Native, виділяється власним графічним движком. З одного боку, це дозволяє при розробці простих додатків взагалі не торкатися native. З іншого боку, при необхідності звернення до native це означає додаткові складності (наприклад, обмін повідомленнями з елементарними типами даних і JSON) і неможливість використання графічних компонентів native[7].

2. Поріг входження

Flutter і React Native постійно розвиваються, у них є активне професійне співтовариство і хороша документація. При цьому нативна розробка випереджає фреймворки, завдяки більшому ком'юніті і більшій кількості навчальних матеріалів і форумів, де описані процеси розробки складних компонентів. [7]

Наступна порівняльна таблиця допоможе спростити вибір і відповісти на питання, в яких випадках той чи інший спосіб реалізації підходить (або не підходить) для створення мобільного додатку.

	React Native	Flutter	Native
MVP або простий додаток	+	+	Не рекомендується
Додаток без складного UI та анімації	+	+	Не рекомендується
Використання камери, відбитків пальців, робота з файловою системою	Потрібно заложити великі ризики	Не рекомендується	+
Підтримка старих версій Android та IOS	Не рекомендується		+
3Д графіка	Не рекомендується		
Важлива стабільність додатку	Не рекомендується		+
Віджети	Не рекомендується		+
Робота для якої потрібна тяжкі операції	Не рекомендується	+	+

Таблиця 1.3 Порівняння мов розробки додатку

Виходячи з цієї таблиці та з бажання залучити більшу кількість користувачів, для розробки Web-додатку буде використовуватися Native розробка на Android, на мові програмування Kotlin.

Розділ 2. РОЗРОБКА МОДЕЛЕЙ ДОДАТКУ ТА БАЗИ ДАНИХ. ПІДБІР ІНСТРУМЕНТІВ ДЛЯ РОЗРОБКИ WEB-ДОДАТКУ.

2.1. Опис концепції додатку (MVVM)

Патерн MVVM (Model-View-ViewModel) дозволяє відокремити логіку додатки від візуальної частини (подання). Даний патерн є архітектурним, тобто він задає загальну архітектуру програми.

Даний патерн був представлений Джоном Госсманом (John Gossman) в 2005 році як модифікація шаблону Presentation Model і був спочатку націлений на розробку додатків в WPF. І хоча зараз цей патерн вийшов за межі WPF і застосовується в самих різних технологіях, в тому числі при розробці під Android, iOS, проте WPF є досить показовою технологією, яка розкриває можливості даного патерну.

MVVM складається з трьох компонентів: моделі (Model), моделі подання (ViewModel) і уявлення (View).

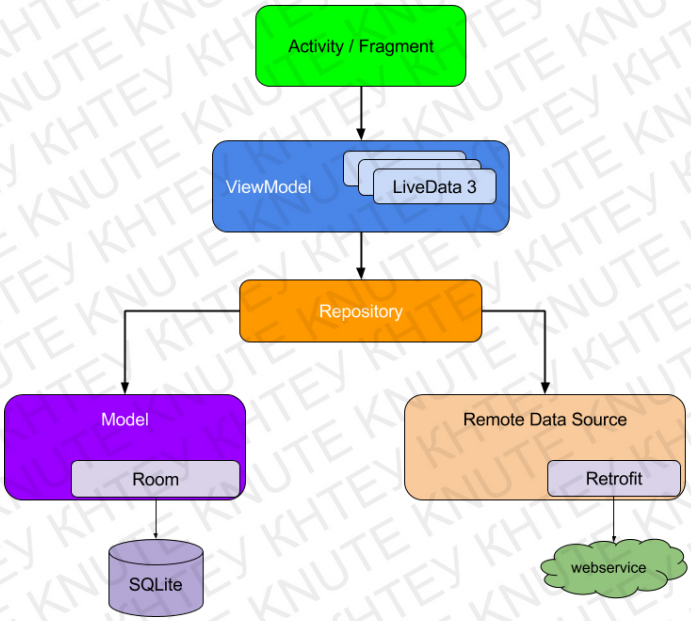

Рис. 2.1. Архітектура MVVM

Model - Модель являє собою легкий компонент, який не знає ні про яку іншу компоненті. Його роль полягає в збереженні стану системи та надання інформації про це в ViewModel.

Модель часто створюється з використанням шаблону Repository. Об'єкти доступу до даних (DAO), які забезпечують операції Create-Read-Update-Delete (CRUD), можуть допомогти в зберіганні дискового простору. Для Android команда Google випустила для цієї мети Бібліотеку Room в Jetpack.

View - У MVVM вид дуже простий, оскільки в ньому майже немає логіки. Viewпов'язано з призначеним для користувача інтерфейсом. В Android це зазвичай дія або фрагмент. Його роль полягає в тому, щоб показати, що він отримує від ViewModel, і переслати його клієнту.

ViewModel- об'єкт, в якому описується логіка поведінки View в залежності від результату роботи Model. Можна назвати його моделлю поведінки View. Це може бути як форматування тексту, так і логіка управління видимістю компонентів або відображення станів, таких як завантаження, помилка, порожні екрани і т.д.

Repository - репозиторії є абстракцією над джерелами даних, яку додаток може використовувати для отримання даних та їх кешування. Ця абстракція корисна з двох основних причин:

1) код не залежить від конкретної реалізації сховища даних ;

2) в наслідок попередньої причини, ми можемо легко змінювати конкретні реалізації сховища даних, наприклад, для тестування.

Data Service - це шар, який містить код для кешування (наприклад, використовуючи SQLite), а також код для завантаження даних (наприклад, завантаження даних з backend api за допомогою Retrofit).[8]

На перший погляд, MVVM здається дуже схожим на шаблон Model-View-Presenter, оскільки обидва вони чудово справляються із абстракцією стану та поведінки виду. Презентаційна модель абстрагує подання, незалежне від певної платформи користувальницького інтерфейсу, тоді як шаблон MVVM був створений для спрощення програмування користувальницьких інтерфейсів.

Якщо шаблон MVP означав, що Presenter безпосередньо повідомляв представлення, що відображати, у MVVM - ViewModel виставляє потоки подій, до яких View можуть прив'язуватися. Так, ViewModel більше не потрібно містити посилання на View, як в Presenter.

Представлення також повідомляють ViewModel про різні дії. Таким чином, шаблон MVVM підтримує двостороннє прив'язку даних між View і ViewModel, також між View і ViewModel існує взаємозв'язок "багато до одного". View має посилання на ViewModel, але ViewModel не має інформації про View. Споживач даних повинен знати про виробника, але виробник - ViewModel - не знає і не хвилює, хто споживає дані.

MVVMмає значну кількість переваг:

- Цей підхід підвищує зручність роботи в командах, так якнаприклад один член команди працює над компоновкою та стилізацією екрану - інший, в цей час, описує логіку отримання даних та їх обробки[8];
- Тестування. Така структура ролегшує написання тестів і процес створення макетів. Крім того, у більшості випадків відпадає потреба в автоматизованому UI-тестуванні, адже можна обернути unit-тестами сам ViewModel[8];
- Розділення логіки. За рахунок більшої кількості обмежень код стає більш гнучким і простим у підтримці, не говорячи про його читабельність. Кажен модуль відповідає за конкретну функцію і лише[8].

Також у MVVM є певні недоліки :

- Для невеликих проектів цей підхід може бути зайвим.
- Якщо логіка прив'язки даних занадто важка - відкласти додаток буде трохи складніше.

2.2. Модель бази даних та хмарного сховища.

CloudFirestore - це гнучка, масштабована база даних для розробки мобільних пристроїв від Firebase та GoogleCloudPlatform. Подібно до бази даних Firebase у реальному часі, вона синхронізує дані у клієнтських програмах через слухачі реального часу та пропонує автономну підтримку для мобільних пристроїв, щоб розробник могли створювати адаптивні програми, які працюють незалежно від затримки мережі або підключення до інтернету. CloudFirestore також пропонує безперебійну інтеграцію з іншими продуктами Firebase та GoogleCloudPlatform, включаючи хмарні функції[9]. Основні переваги Cloud Firestore :

- Гнучкість - Модель даних Cloud Firestore підтримує гнучкі ієрархічні структури даних. Дані зберігаються у документах, організованих у колекції. Документи можуть містити складні вкладені об'єкти як додаток до підколекцій.
- Виразні запити - У Cloud Firestore ви можете використовувати запити для отримання окремих, конкретних документів або для отримання всіх документів у колекції, які відповідають параметрам запиту. Запити можуть включати декілька ланцюжкових фільтрів та поєднувати фільтрацію та сортування. Вони також індексуються за замовчуванням, тому продуктивність запитів пропорційна розміру набору результатів, а не набору даних.
- Оновлення в реальному часі - Як і база даних у реальному часі, Cloud Firestore використовує синхронізацію даних для оновлення даних на будь-якому підключеному пристрої. Однак він також розроблений для ефективного одноразового отримання запитів.
- Підтримка в автономному режимі Cloud Firestore кешує дані, які додаток активно використовує, тому програма може писати, читати, слухати та запитувати дані, навіть якщо пристрій у режимі офлайн.

Коли пристрій повертається в Інтернет, Cloud Firestore синхронізує всі локальні зміни назад до Cloud Firestore.

- Розроблений для масштабування Cloud Firestore пропонує вам найкраще з потужної інфраструктури Google Cloud Platform: автоматична регіональна реплікація даних, надійні гарантії узгодженості, атомні пакетні операції та реальна підтримка транзакцій. Cloud Firestore був розроблений, щоб обробляти найскладніші робочі навантаження з найбільших програм у світі.

Cloud Firestore - це розміщена у хмарі база даних NoSQL, до якої iOS, Android та веб-додатки можуть отримувати безпосередній доступ через власні SDK. Cloud Firestore також доступний у власних Node.js, Java, Python, Unity, C ++ та Go SDK, на додаток до REST та RPC API.

Дотримуючись моделі даних Cloud Firestore NoSQL, дані зберігаться в документах, що містять поля, що відображають значення. Ці документи зберігаються у колекціях, які є контейнерами для документів, які використовуються для впорядкування даних та побудови запитів. Документи підтримують багато різних типів даних, від простих рядків і чисел до складних, вкладених об'єктів. Також можна створювати підколекції в документах та будувати ієрархічні структури даних, які масштабуються в міру зростання бази даних. Модель даних Cloud Firestore підтримує будь-яку структуру даних, яка найкраще підходить для додатка.

Крім того, запити в Cloud Firestore є виразними, ефективними та гнучкими. Створюються поверхневі запити для отримання даних на рівні документа без необхідності отримувати всю колекцію або будь-які вкладені підколекції. Додавання сортування, фільтрацію та обмеження до своїх запитів або курсорів, відбувається щоб перенести результати на сторінки. Щоб дані у програмах залишались актуальними, не отримуючи всю базу даних кожного разу, коли відбувається оновлення, використовуються слухачі реального часу. Додавання слухачів у реальному часі до додатку сповіщає

про зміну даних, коли дані, які прослуховують клієнтські програми, змінюються, отримуючи лише нові зміни.

Cloud Firestore не має схем, тому є повна свободу щодо того, які поля вкладаються в кожен документ і які типи даних зберігаються в цих полях. Документи в одній колекції можуть містити різні поля або зберігати різні типи даних у цих полях.

Колекція містить документи і нічого іншого. Він не може безпосередньо містити необроблені поля зі значеннями, а також не може містити інші колекції.

Назви документів у колекції унікальні. Можна надати власні ключі, такі як ідентифікатори користувачів, або можна дозволити Cloud Firestore автоматично створювати випадкові ідентифікатори для додатку.Не потрібно "створювати" або "видаляти" колекції. Після створення першого документа у колекції колекція існує. Якщо видалити всі документи з колекції, вона більше не існує.

Хмарне сховище для Firebase - це потужна, проста та економічна послуга зберігання об’єктів, створена Google. Пакети SDK Firebase для хмарного сховища додають безпеку Google для завантаження файлів для програм Firebase, незалежно від якості мережі. Можнс використовувати SDK для зберігання зображень, аудіо, відео чи іншого вмісту, створеного користувачами. На сервері можна використовувати Google Cloud Storage для доступу до тих самих файлів.

Основні переваги Firebase Cloud Storage :

- Надійні операційні пакети SDK Firebase для хмарного сховища виконують завантаження незалежно від якості мережі. Завантаження надійні, тобто вони перезапускаються там, де зупинились, заощаджуючи час та пропускну здатність користувачів.
- Сильні захисні пакети SDK Firebase для хмарного сховища інтегровані з автентифікацією Firebase, щоб забезпечити просту та інтуїтивно зрозумілу автентифікацію для розробників. Можна використовувати

декларативну модель безпеки, щоб дозволити доступ на основі імені файлу, розміру, типу вмісту та інших метаданих.

- Хмарне сховище для Firebase із високою масштабованістю створено для збільшення масштабу, коли додаток стає вірусним. Легко переходити від прототипу до виробництва, використовуючи ту саму інфраструктуру, яка забезпечує Spotify та Google Photos. [10]

Cloud Storage зберігає файли в хмарі Google Cloud Storage, роблячи їх доступними як через Firebase, так і через Google Cloud. Це дозволяє гнучко завантажувати файли з мобільних клієнтів за допомогою пакетів SDK Firebase, а також виконувати обробку на стороні сервера, таку як фільтрація зображень або перекодування відео за допомогою Google Cloud Platform. Cloud Storage масштабується автоматично, це означає, що немає необхідності переходити до будь-якого іншого постачальника[10].

Пакети SDK Firebase для хмарного сховища безперешкодно інтегруються з автентифікацією Firebase для ідентифікації користувачів, і також використовується декларативна мова безпеки, яка дозволяє встановлювати елементи керування доступом до окремих файлів або груп файлів, щоб була можливість робити файли загальнодоступними або приватними.

2.3. Підбір інструментів для розробки мобільного web-додатку

В сучасній розробці додатків використовуються такі інструменти :

- AndroidStudio
- Мова програмування Kotlin
- Система збірки проектів Gradle
- ConstraintLayout (UI)
- Бібліотека звязки даних Data Binding
- Hilt (DI)
- Coroutines (Multithreading)

AndroidStudio - програма, що створює середовище розробки програм для мобільних платформ Android. AndroidStudioмає такі переваги над іншими конкурентами в сфері розробки:

- середовище розробки підтримує роботу з декількома мовами програмування, до яких відносяться найпопулярніші - C / C ++, Java.
- редактор коду, з яким зручно працювати;
- дозволяє розробляти додатки не тільки для смартфонів / планшетів, а також для портативних ПК, приставки для телевізорів Android TV, пристроїв Android Wear, новомодних мобільних пристроїв з необ'єктивним співвідношенням сторін екрану;
- тестування коректності роботи нових ігор, утиліт, їх продуктивність на тій чи іній системі, відбувається безпосередньо в емуляторі;
- рефакторінг уже готового коду;
- достатньо велика бібліотека з готовими шаблонами та компонентами для розробки ПО;
- розробка додатків для Android R - сама остання версія операційної системи;
- попередня перевірка вже створеного додатку на предмет помилки в нім;
- великий набір засобів для тестування кожного елемента додатків, ігор;
- для починаючих розробників є спеціально створене керівництво за використанням Android Studio, розміщене на офіційному веб-сайті утиліти.
- Коли створюється додаток та утиліт для операційної системи Android, програма забезпечення користувачів може перейти за змінами в проекті, в режимі реального часу.[11]

Kotlin - це відносно молода мова розробки від російської компанії JetBrains. Зявилася в 2011 році. На конференції Google I / O 2017 команда

розробників Android повідомляла, що Kotlin отримав офіційну підтримку для розробки Android-додатків.

Як і Java, C і C ++, Kotlin - це статистично типова мова. Вона підтримує як об'єктно-орієнтоване, так і процедурне програмування.

- компілюється в байткод JVM або в JavaScript;
- програми можуть використовувати всі існуючі Java-фреймворки та бібліотеки. Kotlin можна інтегрувати з Maven, Gradle та іншими системами збірки;
- мова дуже прост для вивчення;
- початковий код відкритий;
- в IntelliJ доступна автоматична конвертація Java-коду в Kotlin і навпаки;
- мова null-безпечна - NullPointerException залишилися в Java.
- Легкий синтаксис не складає проблеми при перегляді коду[12].

Gradle - це інструмент автоматизації збірки з відкритим кодом, орієнтований на гнучкість та продуктивність. Сценарії збірки Gradle пишуться із використанням DSL Groovy або Kotlin. Основні переваги Gradle такі :

- Gradle змодельований таким чином, що його можна налаштувати та розширити найбільш фундаментальними способами.
- Gradle швидко виконує завдання, використовуючи повторно вихідні дані попередніх виконань, обробляючи лише введені дані та паралельно виконуючи завдання.
- Gradle - це офіційний інструмент збірки для Android, який має підтримку багатьох популярних мов та технологій[13].

ConstraintLayout - представляє новий тип контейнерів, який розробляє RelativeLayout та дозволяє створювати гнучкі та масштабовані інтерфейси. Constraints - це лінії, на основі яких розташовується view всередині ConstraintLayout. Constraints можуть бути прив'язані до сторін самого ConstraintLayout або до сторонам інших view всередині ConstraintLayout[14].

DataBinding (Зв'язування даних) - це процес, який встановлює з'єднання між UI (призначеним для користувача інтерфейсом) програми та бізнес-логікою. Якщо настроювання та повідомлення встановлені правильно, дані відображають зміни, коли вони зроблені. Це може також означати, що коли UI змінюється, дані, які лежать в його основі будуть відображати ці зміни. У Android для цього є своя бібліотек (Data Binding Library) - це бібліотека підтримки, яка дозволяє прив'язувати компоненти інтерфейсу у своїх макетах до джерел даних у програмі, використовуючи декларативний формат, а не програмно. [15]

Hilt - це бібліотека яка допомагає реалізувати «впровадження залежності.
Плюси Hilt:

- Доводиться писати менше шаблонного коду.
- Допомагає структурувати залежності.
- Значною мірою знижує кількість коду коли залежностей багато
- Код стає простим для читання

МінусиHilt:

- Відсутність докладної документації
- Hilt намагається зрозуміти наміри розробника за допомогою анотацій. Це ускладнюється коли він вас не правильно розуміє
- Hilt генерує код, помилки в яких так само складно визначити

Ін’єкція залежності (ін’єкція або введення залежності) - це залежність одного класу від іншого. тобто для повноцінної роботи одного класу необхідна ініціалізація іншого класу.[16]

Спільна програма - це шаблон одночасного дизайну, який ви можете використовувати на Android для спрощення коду, який виконується асинхронно. Спільні програми були додані до Kotlin у версії 1.3 і базуються на усталених концепціях з інших мов.[20]

Coroutines- це легкі потоки. Легкий потік означає, що він не прив'язується до запуску потоку, тому він не вимагає переключення

контексту на процесорі, тому він швидше. Android Coroutinesдопомагають керувати тривалими завданнями, які в іншому випадку можуть заблокувати основний потік і спричинити невідповідність програми. Понад 50% професійних розробників, які використовують спільні програми, повідомили про збільшення продуктивності.[17]ОсобливостіCoroutines:

- Легкі: Можна запускати безліч програм на одному потоці завдяки підтримці підвіски, яка не блокує потік, де працює програма. Призупинення економить пам'ять над блокуванням, підтримуючи багато одночасних операцій.
- Менше витоків пам'яті: використовуйте структуровану паралельність для запуску операцій у межах області.
- Вбудована підтримка скасування: Скасування розповсюджується автоматично через діючу ієрархію корутин.
- Інтеграція Jetpack: Багато бібліотек Jetpack містять розширення, що забезпечують повну підтримку програм. Деякі бібліотеки також надають власну спільну програму, яку ви можете використовувати для структурованої паралельності.[19]

РОЗДІЛ 3. ПРОЕКТУВАННЯ ТА РОЗРОБКА МОБІЛЬНОГО WEB-ДОДАТКУ ДЛЯ ЕЛЕКТРОННОЇ ТОРГІВЛІ

3.1. Проектування архітектурі та опис функціоналу внутрішніх модулів додатку

Відштовхуючись від потреб додатку він матиме декілька екранів які будуть взаємодіяти між собою. Відповідно до потреб висловлених в попередніх розділах проект додатку буде мати такий вигляд

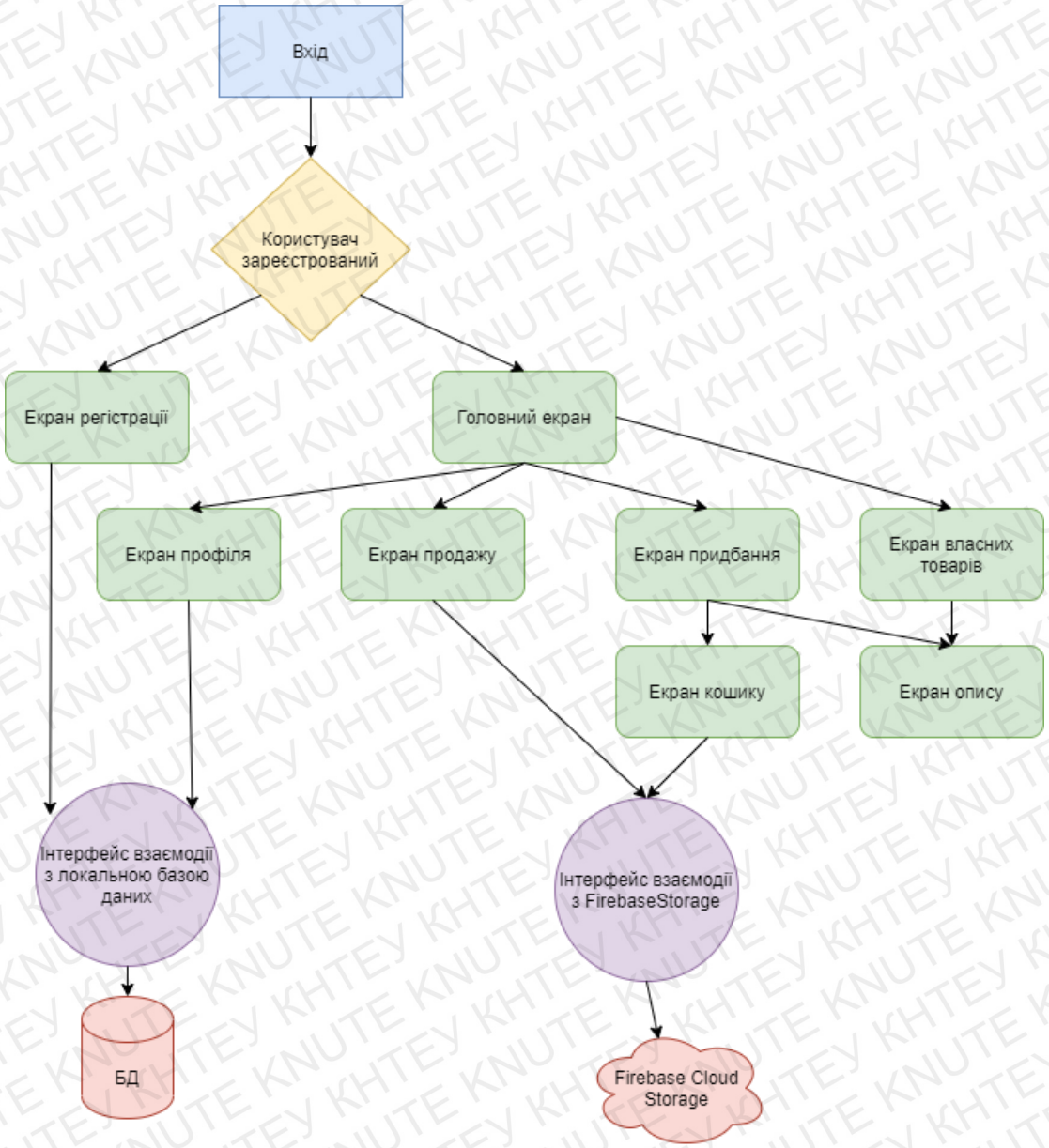

Рис. 3.1. Архітектура додатку

Екран регістрації – виконує функцію регістрації користувача в системі та збереження його даних для подальшого використання.

Головний екран – виконує функцію навігатора в системі, через який користувач може використовувати весь функціонал додатку.

Екран профіля – виконує функцію зміни та збереження нових даних клієнта.

Екран продажу – надає можливість додати всю інформацію про новий товар та завантажити його на сервер.

Екран придбання – надає можливість вибрати товари які користувач бажає придбати.

Екран власних товарів – надає модливість переглянути власні товари, та побачити їх покупців, для початку комунікації з ними.

Екран кошику – надає можливість придбати товари.

Екран опису – надає можливість переглянути деталі товару.

Інтерфейс взаємодії з локальною базою даних – виконує функцію посередника між базою даних та представлення, виконуючи вирахування в окремих потоках.

Інтерфейс взаємодії з FirebaseStorage – виконує функцію посередника між FirebaseCloudStorage та представленням, виконуючи вирахування в окремих потоках.

База даних (БД) – локальна база даних, якак знаходиться на смартфоні. FirebaseCloudStorage – серверна база даних для збереження фото та інформації про продукти продажу.

Таким чином був створений проект додаткувідповідно до якого можна розділити дотаток на модулі, які будуть легко піддаватися автоматичним тестам, та не будуть звязані один з одним. Відповідно до архітектури MVVM додаток був розбитий на такі модулі (в випадку Android- модулі є пакетами додатку) :

- UI
- ViewModel

- Repository
- LocalDataStorage
- RemoteDataStorage

Взаємодія між ними буде відбуватися через інтерфейсизгідно принципів SOLID, а саме DI (Hilt -інєкція залежностей).

3.2. Програмна реалізація додатку та бази даних. Опис класів.

Шаблоном розробки додатку був обраний Single-Activity – додаток має одну вхідну точку (MainActivity.kt), та декілька змінючих один одного екранів. Це зменшує навантаження на процессор, та поліпшує взаємодіію з користувачем.[18]

Екран регістрації – включає в себе – відображення інтерфейса (RegistrationFragment.kt, RegistrationViewModel.kt, fragment_registration.xml) валідацію полів які заповнює користувач (реалізовано в NameValidator.kt, PhoneValidator.kt, EmailValidator.kt, PasswordValidator.kt) та збереження користувача в локальну базу даних (доступ до бази даних LocalDataStorage.kt, база даних – DataStorageImpl.kt). Після регістрації користувач має доступ до всіх функції додатку.

Головний екран - включає в себе – відображення інтерфейса (MainFragment.kt, fragment_main.xml), екран виконує роль навігатора по додатку для користувача.

Екран профіля – включає в себе – відображення інтерфейса (ProfileFragment.kt, ProfileViewModel.kt, fragment_profile.xml), через екран користувач може змінити інформацію на своєму додатку, плюсом є те, що особиста інформація користувача не відправляється на сервер без потреби в цьому. Зміна інформації відбувається в те ж сховище що й при регістрації, тобто попередня інформація видаляється для збереження пам’яті (доступ до бази даних LocalDataStorage.kt, база даних – DataStorageImpl.kt). Користувач

може вийти з додатку на цьому екрані. Інформація видаляється тільки при видаленні додатку.

Екран продажу –включає в себе – відображення інтерфейса (SellFragment.kt, SellViewModel.kt, fragment_sell.xml), через екран користувач створює новий продукт, поля якого також проходять валідацію (ProdNameValidator.kt, ProdDescValidator.kt, ProdPhotoValidator.kt), після чого інформація відправлється на сервер (RemoteRepository.kt, FirebaseStorage.kt).

Екран придбання – включає в себе – відображення інтерфейса (BuyFragment.kt, BuyViewModel.kt, fragment_buy.xml),екран відображає всі доступні товари на даний час, також є можливість їх відфільтрувати та додати в кошик, для пізнішого придбання. Завантаження відбувається через RemoteRepository.kt таFirebaseStorage.kt.

Екран власних товарів – – включає в себе – відображення інтерфейса (SelfProdFragment.kt, fragment_self_prod.xml), відображає товари які належать користувачу (доступ до бази даних LocalDataStorage.kt, база даних – DataStorageImpl.kt).

Екран кошику – включає в себе – відображення інтерфейса (BasketFragment.kt, BasketViewModel.kt, fragment_basket.xml), відображає товари які користувач бажає придбати, при нажаті на кнопку придбати запит буде відправлений на сервер(доступ до бази даних LocalDataStorage.kt, база даних – DataStorageImpl.kt).

Екран опису – включає в себе – відображення інтерфейса (DetailsFragment.kt, DetailsViewModel.kt, fragment_details.xml), відображає детальну інформацію товару.

Інтерфейс взаємодії з локальною базою даних – реалізований в LocalDataStorage.ktпотрібен для збереження слабозвязності окремих модулів додатку та легкої заміни їх в майбутньому. Виконує доступ до локальної бази даних, та інкапсулює її реалізацію.

Інтерфейс взаємодії з FirebaseStorage – – реалізований в RemoteRepository.kt, FirebaseStorage.kt, потрібен для збереження

слабозвязності окремих модулів додатку та легкої заміни їх в майбутньому. Виконує доступ до віддаленої бази даних, та інкапсулює її реалізацію.

Моделі користувача та товару зберігаються в класах User.ktta Product.ktвідповідно

База даних (БД) – локальна база даних, яка знаходиться на девайсі, доступ до неї відбувається асинхронно, тобто не блокує інтерфейс користувача під час запиту – реалізовано в DataStorageImpl.kt. Так як збирається не значна кількість інформації, використовується стандартне рішення Android–DataStore, без додаткової ORM.

FirebaseCloudStorage – доступ до серверної бази даних відбувається через FirebaseStorage.kt – запити виконуються асинхронно.Підключення до бази даних відбувається через файл google-services.json, що містить в собі ключі та додаткову інформацію для Firebase. База даних має формат NoSql. На рис. 3.2.1 зображено базу даних цього проекта.

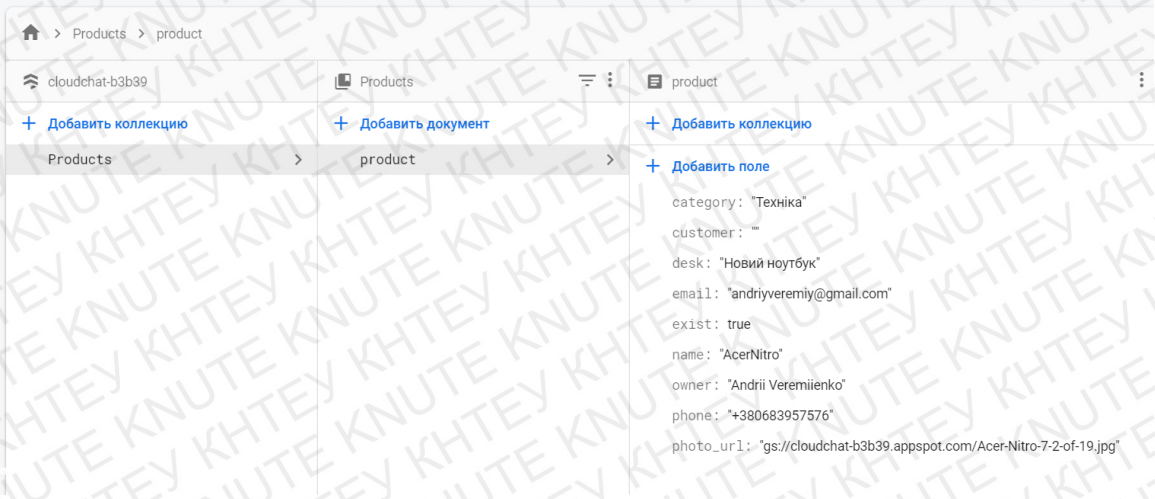

Рис. 3.2 Внутрішня структура бази даних

Фото завантажуються в FirebaseStorage який зображений на рис. 3.2.2, а посилання на нього зберігається в FirebaseCloudStorage

Рис. 3.3 Внутрішня структура бази зберігання фото

3.3. Мануал по використанню мобільного Web-додатку для електронної торгівлі

Запуск додатку відбувається через вхід через іконку SiBS. Після чого буде запущено вікно регістрації (якщо користувач не зареєстрований – зареєстрований користувач перенапрявляється на головну сторінку).

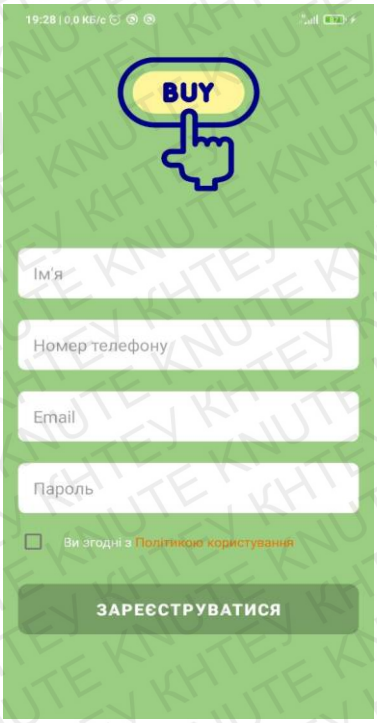

Рис. 3.4. Екран регістрації

Після введення коректної інформації та погодження з правилами користування – кнопка стає активною й користувач може залогінитися.

Рис. 3.5 Екран регістрації заповнений

Наступним екраном є головний екран, який надає доступ до всіх функцій додатку. Так як додаток орієнтований на продаж/купівлю то саме на ці функції виділені на екрані. Також є доступ до профіля.

Рис. 3.6 Головний екран

Важливим екраном є – екран зміни профіля, при вході уже збережена інформація відображається (окрім пароля), кожне поле можна змінити й надалі оновлена інформація буде використовуватися. Вийти з додатку можна по кнопці в правому верхньому куті. Кнопка «зберегти зміни» збереже нову інформацію.

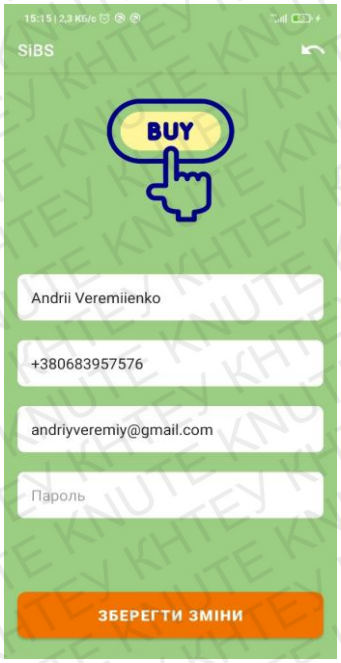

Рис. 3.7 Профіль

Першою головною функцією є додавання нового товару на сервер. На я кому потрібно заповнити текстову інформацію, додати фото, та вибрати категорію товару.

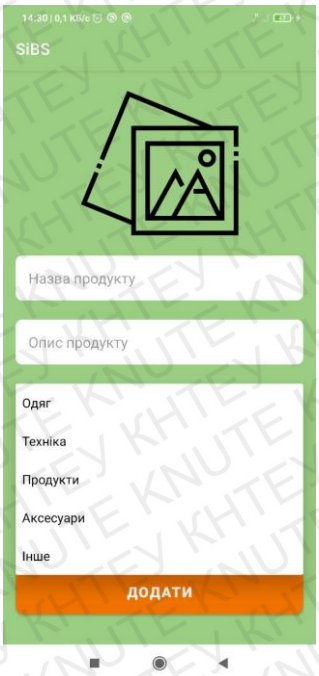

Рис. 3.8 Екран створення продукту

Заповниши всю важливу інформацію про товар – користувач може нажати на кнопку «додати» - після чого інформація буде відправлена на сервер.

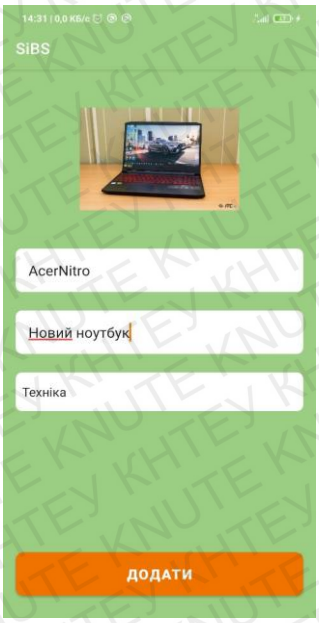

Рис. 3.9 Заповнений екран створення продукту

Друга важлива функція – перегляд всіх товарів та їх придбання. Весь список наявних товарів відображається на одному екрані який можна відсортувати за категоріями вибравши відповідний елемент в правому верхньому куті.

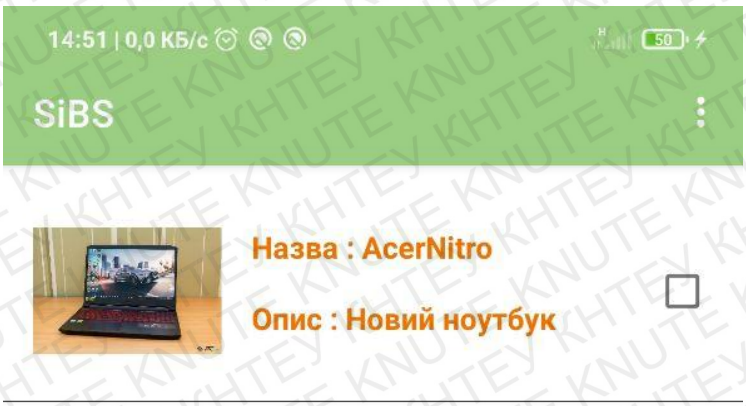

Рис. 3.10 Екран товарів

Деталі товару можна переглянути вибравши його в списку.

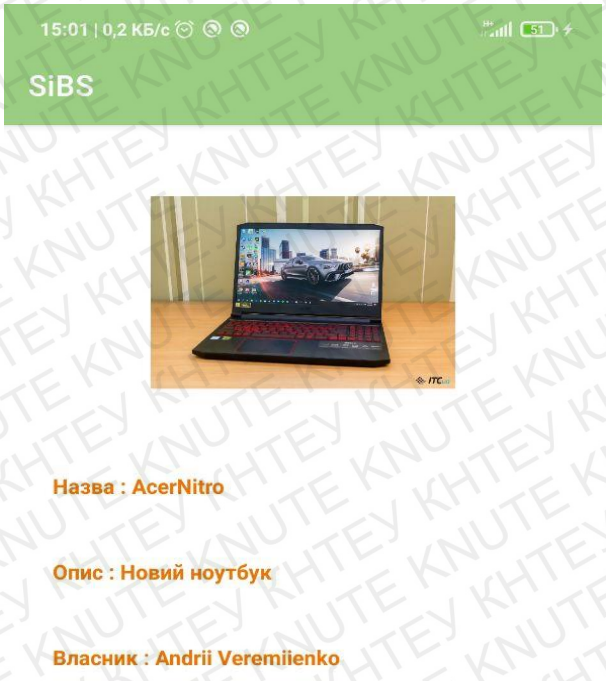

Рис. 3.11 Екран деталей товару

Вибрані товари додаються в кошик в якому можна їх видалити (знявши чекбокс) або замовити, після чого потрібно очікувати на повідомлення від продавця.

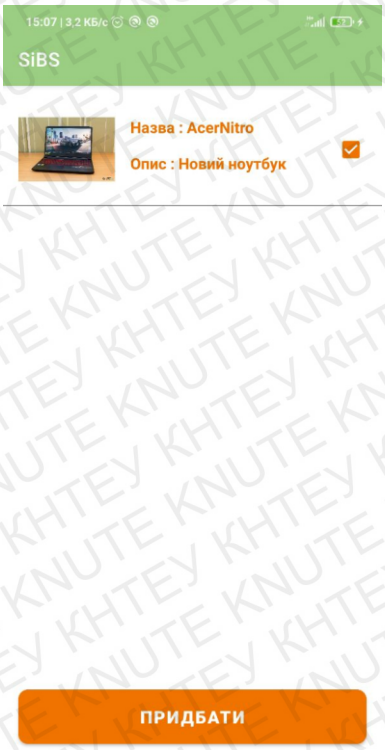

Рис. 3.12 Екран кошику товарів

Після того як товар замовлений – у продавця з’являється можливість, зателефонувати користувачу для уточнення інформації по замовленню й кнопка дзвінка стає активною.

Рис. 3.10 Екран товарів

Таким чином спроектовано і реалізовано мобільний Web-додаток для електронної торгівлі, який можна легко підтримувати та модернізувати під потреби підприємства. Також у додатку зручний та зрозмілий інтерфейс, тобто правила UI/UXдотримувалися під час створення. Кастомізація додатку може продоводитися лише одним користувачем рівня Junior (початківець). Впровадження додатку потребує лише зміни електронної почти та файлу google-services.json. Кастомізація відбувається в файлах .xml та strings.xml. За допомогою додатку можна легко продати, або ж купити товар, за потребую підприємства функція продажу може бути вимкнута. Додаток не потребує значних витрат коштів і часу, що є основною перевагою для підприємств, що тільки розпочинають.

ВИСНОВКИ ТА РЕЗУЛЬТАТИ

У випускній кваліфікаційній роботі представлено проектування та розробку мобільного Web-додатку для електронної торгівлі на базі ОС Android.

Були розглянуті теоретичні засади e-commerce, її види, а також переваги та недоліки. Була досліджена мобільна розробка в e-commerce, а саме – m-commerce. Сучасним підтриємствам, для поширення та популяризації своїх продуктів, а також своєї торгової марки, потрібно користуватися найліпшими способами маркетингу. В даний час – таким способом є m-commerce, завдяки своїй персоналізації та ряду других переваг. Виявлено її сильні сторони на фоні інших представників сфери. Також були проаналізовані мобільні ОС та сучасні типи розробки мобільних додатків серед яких вибрано найліпший варіант під потреби Українського ринку – Android та нативна розробка на мові програмування Kotlin.

Описана сучасна концепція додатку - MVVM її структурні компоненти та їх взаємодія. Також була розглянуте хмарне сховище даних від Firebase – Firestore та хмарне сховище файлів – Storage. Вибрані та проаналізовані сучасні інструменти та засоби для розробки, які будуть використовуватися в додатку або ж його розробці (AndroidStudio, Мова програмування Kotlin, Система збірки проектів Gradle, ConstraintLayout, Бібліотека звязки даних Data Binding, Hilt (DI), Coroutines (Multithreading))

Була спроектована сучасна архітектура мобільних додатків, на основі потреб додатку, та описані основні модулі взаємодії з користувачем. Завдяки свої простоті додаток може бути легко впроваджений в будь-яке підтриємство або стартап, для аналізу ринку чи повноцінної роботи. Функціональну підтримку та кастомізацію додатку може виконувати один розробник рівня Junior–тобто не потрібно витрачати час та кошти на розробку.

За результатами був створений додаток, який повністю відповідав вимогам розробки а саме -реалізовує функції регістрації, продажу та придбання. Також додаток був протестований і показав свою працездатність, як на стандартних емуляторах, взятих з SDK Android (версії API 22 та 29), так і на реальних пристроях на платформі Android (Api 29).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. GrafA., SchneiderH., StobbeM.The E-Commerce Book: About a Channel that became an Industry / Alexander Graf, Prof Holger Schneider, Mareike Stobbe - London: CreateSpace Independent Publishing Platform, 2016 – 460 p.
2. Larsson T.Ecommerce Evolved/Tanner Larsson - London: CreateSpace Independent Publishing Platform, 2016 – 333 p.
3. Skeldon P.M-Commerce: Boost Your Business with the Power of Mobile Commerce / Paul Skeldon - Bath: Crimson Publishing, 2011 – 300 p.
4. Duhan P., Singh A.Experiencing the Phygital Retail / Punita Duhan, Anurag Singh - Florida: Apple Academic Press, 2019 – 444 p.
5. Blake Meike B.Inside the Android OS: Building, Customizing, Managing and Operating Android System Services/Blake Meike- Boston: Addison-Wesley Professional, 2020 – 350 p.
6. Lewis S., Dunn M. Native Mobile Development: A Cross-Reference for iOS and Android / Shaun Lewis, Mike Dunn- Sebastopol: O'Reilly Media, 2019 – 394 p.
7. https://www.zeolearn.com/magazine/native-vs-cross-platform-apps-youll-be-the-winner - NativevsCrosPlatform [Електронний ресурс] – Режим доступу: 19.11.2020
8. Cheng Y., Domínguez A. O. Native Advanced Android App Architecture / Yun Cheng, Aldo Olivares Domínguez -McGaheysville: Razeware LLC, 2019 – 273p.
9. Moroney L. The Definitive Guide to Firebase: Build Android Apps on Google's Mobile Platform / Laurence Moroney -New York: Apress, 2017 – 288p.
10. Smyth N. Firebase Essentials - Android Edition / Neil Smyth –London: CreateSpace Independent Publishing Platform, 2017 – 534p.
11. Hagos T.Learn Android Studio 4: Efficient Java-Based Android Apps Development / Ted Hagos–New York: Apress, 2020 – 344p.
12. Jemerov D., Isakova S. Native Advanced Android App Architecture / Dmitry Jemerov, Svetlana Isakova -Shelter Island: Manning Publications, 2017 – 360p.

13. Muschko B. Gradle in Action/ Benjamin Muschko-Shelter Island: Manning Publications, 2014 – 480p.
14. https://www.raywenderlich.com/9475-constraintlayout-tutorial-for-android-complex-layouts - ConstraintLayout [Електронний ресурс] – Режим доступу: 19.11.2020
15. https://www.raywenderlich.com/7711166-data-binding-in-android-getting-started - DataBinding [Електронний ресурс] – Режим доступу: 19.11.2020
16. https://dagger.dev/hilt/ - Hilt (DI) [Електронний ресурс] – Режим доступу: 19.11.2020
17. https://kotlinlang.org/docs/reference/coroutines-overview.html - Coroutines (Multi-threading) [Електронний ресурс] – Режим доступу: 19.11.2020
18. https://proandroiddev.com/part-3-single-activity-architecture-514791724172 - SingleActivity[Електронний ресурс] – Режим доступу: 19.11.2020
19. https://developer.android.com/jetpack - JetpackCompose[Електронний ресурс] – Режим доступу: 19.11.2020
20. https://www.geeksforgeeks.org/suspend-function-in-kotlin-coroutines/- SuspendFunction [Електронний ресурс] – Режим доступу: 19.11.2020

ДОДАТОК

Програмний код реалізації Web-додатку

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifestxmlns:android="http://schemas.android.com/apk/res/android"
package="com.vero.sibs">

<application
android:name=".SibsApplication"
android:allowBackup="true"
android:icon="@mipmap/ic_launcher"
android:label="@string/app_name"
android:roundIcon="@mipmap/ic_launcher_round"
android:supportsRtl="true"
android:theme="@style/AppTheme">

<activityandroid:name=".ui.MainActivity">
<intent-filter>
<actionandroid:name="android.intent.action.MAIN"/>

<categoryandroid:name="android.intent.category.LAUNCHER"/>
</intent-filter>
</activity>
</application>

</manifest>
```

Build.gradle

```
apply plugin:'com.android.application'
apply plugin:'kotlin-android'
apply plugin:'kotlin-android-extensions'
apply plugin:'kotlin-kapt'
apply plugin:'androidx.navigation.safeargs.kotlin'
apply plugin:'dagger.hilt.android.plugin'
apply from:'../dependencies.gradle'

android {
    compileSdkVersion 30
    buildToolsVersion "30.0.2"

    defaultConfig {
        applicationId "com.vero.sibs"
        minSdkVersion 21
        targetSdkVersion 30
        versionCode 1
        versionName "1.0"

        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }
```

```
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-
optimize.txt'), 'proguard-rules.pro'
        }
    }

    buildFeatures {
        dataBinding = true
    }

    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }

    kotlinOptions {
        jvmTarget = JavaVersion.VERSION_1_8.toString()
    }

}

dependencies {
    implementation fileTree(dir:"libs", include: ["*.jar"])
    implementation "org.jetbrains.kotlin:kotlin-stdlib:$versions.kotlin"
    implementation 'androidx.core:core-ktx:1.3.2'
    implementation 'androidx.appcompat:appcompat:1.2.0'

//hilt
    implementation "com.google.dagger:hilt-
android:$versions.daggerHiltCoreVersion"
    implementation "androidx.hilt:hilt-common:$versions.daggerHiltVersion"
    implementation "androidx.hilt:hilt-lifecycle-
viewmodel:$versions.daggerHiltVersion"
    kapt "com.google.dagger:hilt-android-
compiler:$versions.daggerHiltCoreVersion"
    kapt "androidx.hilt:hilt-compiler:$versions.daggerHiltVersion"

//ui
    implementation
"com.google.android.material:material:$versions.materialVersion"
    implementation
"androidx.constraintlayout:constraintlayout:$versions.constraintVersion"

//navigation
    implementation 'androidx.navigation:navigation-fragment-ktx:2.3.1'
    implementation 'androidx.navigation:navigation-ui-ktx:2.3.1'

// unit test
    testImplementation "com.google.truth:truth:$versions.truthVersion"
    testImplementation "junit:junit:$versions.junitVersion"
    testImplementation "androidx.test:core:$versions.androidxTest"
    testImplementation
"org.robolectric:robolectric:$versions.robolectricVersion"
    testImplementation "androidx.arch.core:core-
testing:$versions.coreExecutorTestingVersion"
```

```
    testImplementation "org.jetbrains.kotlinx:kotlinx-coroutines-
test:$versions.kotlinCoroutinesTestVersion"
    testImplementation "io.mockk:mockk:$versions.mockkVersion"

//ui test
    androidTestImplementation "com.google.truth:truth:$versions.truthVersion"
    androidTestImplementation "junit:junit:$versions.junitVersion"
    androidTestImplementation "androidx.test:core:$versions.androidxTest"
    androidTestImplementation
"androidx.test.ext:junit:$versions.androidxJRunner"
    androidTestImplementation "androidx.arch.core:core-testing:2.1.0"
    androidTestImplementation "com.linkedin.dexmaker:dexmaker-mockito:2.12.1"
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.3.0'
    androidTestImplementation "org.mockito:mockito-core:2.21.0"

    androidTestImplementation "com.google.dagger:hilt-android-
testing:$versions.daggerHiltCoreVersion"
    kaptAndroidTest "com.google.dagger:hilt-android-
compiler:$versions.daggerHiltCoreVersion"
    androidTestAnnotationProcessor "com.google.dagger:hilt-android-
compiler:$versions.daggerHiltCoreVersion"

    debugImplementation "androidx.fragment:fragment-testing:1.2.5"
    debugImplementation("androidx.test:core:1.3.0")
    debugImplementation("androidx.test:rules:1.3.0")
    debugImplementation("androidx.test:runner:1.3.0")

//storage
    implementation "androidx.datastore:datastore-
preferences:$versions.dataStorageVersion"

}
```

Activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayoutxmlns:android="http://schem
as.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context=".MainActivity">

<fragment
android:layout_width="match_parent"
android:layout_height="match_parent"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```

Fragment_backet.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayoutxmlns:android="http://schem
as.android.com/apk/res/android"
```

```
android:layout_width="match_parent"
android:layout_height="match_parent"
xmlns:app="http://schemas.android.com/apk/res-auto">

<ImageView
android:layout_width="100dp"
android:layout_height="100dp"
android:id="@+id/ivImageView1"
android:layout_margin="16dp"
android:src="@drawable/photo1"
app:layout_constraintTop_toTopOf="parent"
app:layout_constraintStart_toStartOf="parent"/>

<TextView
android:id="@+id/nameProd"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_marginStart="16dp"
android:layout_marginTop="16dp"
android:textColor="@color/colorAccent"
android:textSize="16sp"
android:textStyle="bold"
app:layout_constraintStart_toEndOf="@id/ivImageView1"
app:layout_constraintTop_toTopOf="@id/ivImageView1"/>

<TextView
android:id="@+id/descProd"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_marginStart="16dp"
android:layout_marginTop="16dp"
android:textColor="@color/colorAccent"
android:textSize="16sp"
android:textStyle="bold"
app:layout_constraintStart_toEndOf="@id/ivImageView1"
app:layout_constraintTop_toBottomOf="@id/nameProd"/>

<CheckBox
android:layout_width="32dp"
android:layout_height="32dp"
android:layout_marginEnd="16dp"
android:checked="true"
app:layout_constraintBottom_toBottomOf="@id/ivImageView1"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintTop_toTopOf="@id/ivImageView1"/>

<View
android:layout_width="match_parent"
android:layout_height="0.5dp"
android:layout_marginTop="8dp"
android:background="@android:color/black"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toBottomOf="@id/ivImageView1"/>

<com.google.android.material.button.MaterialButton
android:id="@+id/btnSignUp"
android:layout_width="match_parent"
android:layout_height="wrap_content"
```

```
android:layout_marginStart="16dp"
android:layout_marginEnd="16dp"
android:layout_marginBottom="32dp"
android:backgroundTint="@drawable/default_material_button_background"
android:padding="16dp"
android:text="@string/buy"
android:textColor="@android:color/white"
android:textSize="18sp"
android:textStyle="bold"
app:cornerRadius="10dp"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintStart_toStartOf="parent"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```

Fragment_buy.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayoutxmlns:android="http://schem
as.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent"
xmlns:app="http://schemas.android.com/apk/res-auto">

<ImageView
android:layout_width="100dp"
android:layout_height="100dp"
android:id="@+id/ivImageView1"
android:layout_margin="16dp"
android:src="@drawable/photo1"
app:layout_constraintTop_toTopOf="parent"
app:layout_constraintStart_toStartOf="parent"/>

<TextView
android:id="@+id/nameProd"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_marginStart="16dp"
android:layout_marginTop="16dp"
android:textColor="@color/colorAccent"
android:textSize="16sp"
android:textStyle="bold"
app:layout_constraintStart_toEndOf="@id/ivImageView1"
app:layout_constraintTop_toTopOf="@id/ivImageView1"/>

<TextView
android:id="@+id/descProd"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_marginStart="16dp"
android:layout_marginTop="16dp"
android:textColor="@color/colorAccent"
android:textSize="16sp"
android:textStyle="bold"
app:layout_constraintStart_toEndOf="@id/ivImageView1"
app:layout_constraintTop_toBottomOf="@id/nameProd"/>
```

```
<CheckBox
android:layout_width="32dp"
android:layout_height="32dp"
android:layout_marginEnd="16dp"
app:layout_constraintBottom_toBottomOf="@id/ivImageView1"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintTop_toTopOf="@id/ivImageView1"/>

<View
android:layout_width="match_parent"
android:layout_height="0.5dp"
android:layout_marginTop="8dp"
android:background="@android:color/black"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toBottomOf="@id/ivImageView1"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```

Fragment_details.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayoutxmlns:android="http://schem
as.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:background="@android:color/white"
xmlns:app="http://schemas.android.com/apk/res-auto">

<ImageView
android:id="@+id/ivImageView"
android:layout_width="200dp"
android:layout_height="200dp"
android:layout_margin="16dp"
android:src="@drawable/photo1"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toTopOf="parent"
app:layout_constraintEnd_toEndOf="parent"/>

<TextView
android:id="@+id/etName"
android:layout_width="0dp"
android:layout_height="wrap_content"
android:layout_marginStart="16dp"
android:layout_marginTop="8dp"
android:layout_marginEnd="16dp"
android:padding="16dp"
android:textColor="@color/colorAccent"
android:textStyle="bold"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toBottomOf="@id/ivImageView"/>

<TextView
android:id="@+id/etDescription"
android:layout_width="0dp"
```

```
android:layout_height="wrap_content"
android:layout_marginStart="16dp"
android:layout_marginTop="8dp"
android:layout_marginEnd="16dp"
android:padding="16dp"
android:text="Опис : Новий ноутбук"
android:textColor="@color/colorAccent"
android:textStyle="bold"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toBottomOf="@id/etName"/>

<TextView
android:id="@+id/spinner"
android:layout_width="0dp"
android:layout_height="wrap_content"
android:layout_marginStart="16dp"
android:layout_marginTop="8dp"
android:layout_marginEnd="16dp"
android:padding="16dp"
android:background="@drawable/default_edit_text_background"
android:textColor="@color/colorAccent"
android:textStyle="bold"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toBottomOf="@id/etDescription"/>

<TextView
android:id="@+id/owner"
android:layout_width="0dp"
android:layout_height="wrap_content"
android:layout_marginStart="16dp"
android:layout_marginTop="8dp"
android:layout_marginEnd="16dp"
android:padding="16dp"
android:background="@drawable/default_edit_text_background"
android:textColor="@color/colorAccent"
android:textStyle="bold"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toBottomOf="@id/etDescription"/>

<TextView
android:id="@+id/customer_phone"
android:layout_width="0dp"
android:layout_height="wrap_content"
android:layout_marginStart="16dp"
android:layout_marginTop="8dp"
android:layout_marginEnd="16dp"
android:padding="16dp"
android:background="@drawable/default_edit_text_background"
android:textColor="@color/colorAccent"
android:textStyle="bold"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toBottomOf="@id/owner"/>

<TextView
android:id="@+id/customer"
```

```
android:layout_width="0dp"
android:layout_height="wrap_content"
android:layout_marginStart="16dp"
android:layout_marginTop="8dp"
android:layout_marginEnd="16dp"
android:padding="16dp"
android:background="@drawable/default_edit_text_background"
android:textColor="@color/colorAccent"
android:textStyle="bold"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toBottomOf="@id/customer_phone"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```

Fragment_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<layoutxmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:android="http://schemas.android.com/apk/res/android">

<androidx.constraintlayout.widget.ConstraintLayout
android:layout_width="match_parent"
android:layout_height="match_parent"
android:background="@color/colorPrimary"
android:paddingTop="?attr/actionBarSize">

<ImageView
android:id="@+id/ivLogo"
android:layout_marginTop="32dp"
android:layout_width="150dp"
android:layout_height="150dp"
android:src="@drawable/ic_buy"
app:layout_constraintTop_toTopOf="parent"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintEnd_toEndOf="parent"/>

<androidx.constraintlayout.widget.ConstraintLayout
android:layout_width="150dp"
android:layout_height="150dp"
android:layout_marginStart="16dp"
android:layout_marginBottom="72dp"
android:background="@drawable/oval_bg"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintStart_toStartOf="parent">

<ImageButton
android:id="@+id/btnSell"
android:layout_width="90dp"
android:layout_height="90dp"
android:adjustViewBounds="true"
android:background="?android:selectableItemBackground"
android:scaleType="fitCenter"
android:src="@drawable/ic_sell_main"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintEnd_toEndOf="parent"
```

```
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toTopOf="parent"/>

<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/sell"
android:textColor="@android:color/white"
android:textSize="16sp"
android:textStyle="bold"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toBottomOf="@id/btnSell"/>

</androidx.constraintlayout.widget.ConstraintLayout>

<androidx.constraintlayout.widget.ConstraintLayout
android:layout_width="150dp"
android:layout_height="150dp"
android:layout_marginEnd="16dp"
android:layout_marginBottom="72dp"
android:background="@drawable/oval_bg"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintEnd_toEndOf="parent">

<ImageButton
android:id="@+id/btnBuy"
android:layout_width="90dp"
android:layout_height="90dp"
android:adjustViewBounds="true"
android:background="?android:selectableItemBackground"
android:scaleType="fitCenter"
android:src="@drawable/ic_buy_main"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toTopOf="parent"/>

<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/buy"
android:textColor="@android:color/white"
android:textSize="16sp"
android:textStyle="bold"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toBottomOf="@id/btnBuy"/>

</androidx.constraintlayout.widget.ConstraintLayout>


</androidx.constraintlayout.widget.ConstraintLayout>

</layout>
```

Fragment_profile.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayoutxmlns:android="http://schem
as.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
android:background="@color/colorPrimary"
android:layout_width="match_parent"
android:layout_height="match_parent">

<ImageView
android:id="@+id/ivLogo"
android:layout_width="150dp"
android:layout_height="150dp"
android:src="@drawable/ic_buy"
android:layout_marginTop="32dp"
app:layout_constraintTop_toTopOf="parent"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintEnd_toEndOf="parent"/>

<EditText
android:id="@+id/etName"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:background="@drawable/default_edit_text_background"
android:layout_marginTop="64dp"
android:layout_marginStart="16dp"
android:layout_marginEnd="16dp"
android:padding="16dp"
android:hint="@string/name"
app:layout_constraintTop_toBottomOf="@id/ivLogo"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintEnd_toEndOf="parent"/>

<EditText
android:id="@+id/etPhone"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:background="@drawable/default_edit_text_background"
android:layout_marginTop="24dp"
android:layout_marginStart="16dp"
android:layout_marginEnd="16dp"
android:padding="16dp"
android:hint="@string/phone_number"
app:layout_constraintTop_toBottomOf="@id/etName"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintEnd_toEndOf="parent"/>

<EditText
android:id="@+id/etEmail"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:background="@drawable/default_edit_text_background"
android:layout_marginTop="24dp"
android:layout_marginStart="16dp"
android:layout_marginEnd="16dp"
android:padding="16dp"
android:hint="@string/email"
app:layout_constraintTop_toBottomOf="@id/etPhone"
```

```
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintEnd_toEndOf="parent"/>

<EditText
android:id="@+id/etPassword"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_marginStart="16dp"
android:layout_marginTop="24dp"
android:layout_marginEnd="16dp"
android:background="@drawable/default_edit_text_background"
android:hint="@string/password"
android:inputType="textPassword"
android:padding="16dp"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toBottomOf="@id/etEmail"/>

<com.google.android.material.button.MaterialButton
android:id="@+id/btnSignUp"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_marginStart="16dp"
android:layout_marginEnd="16dp"
android:layout_marginBottom="24dp"
android:backgroundTint="@drawable/default_material_button_background"
android:padding="16dp"
android:text="@string/save"
android:textColor="@android:color/white"
android:textSize="18sp"
android:textStyle="bold"
app:cornerRadius="10dp"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintStart_toStartOf="parent"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```

Fragment_registration.xml

```
<?xml version="1.0" encoding="utf-8"?>
<layoutxmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:android="http://schemas.android.com/apk/res/android">

<androidx.constraintlayout.widget.ConstraintLayout
android:background="@color/colorPrimary"
android:layout_width="match_parent"
android:layout_height="match_parent">

<ImageView
android:id="@+id/ivLogo"
android:layout_width="150dp"
android:layout_height="150dp"
android:src="@drawable/ic_buy"
android:layout_marginTop="32dp"
app:layout_constraintTop_toTopOf="parent"
```

```
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintEnd_toEndOf="parent"/>

<EditText
android:id="@+id/etName"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:background="@drawable/default_edit_text_background"
android:layout_marginTop="64dp"
android:layout_marginStart="16dp"
android:layout_marginEnd="16dp"
android:padding="16dp"
android:hint="@string/name"
app:layout_constraintTop_toBottomOf="@id/ivLogo"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintEnd_toEndOf="parent"/>

<EditText
android:id="@+id/etPhone"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:background="@drawable/default_edit_text_background"
android:layout_marginTop="24dp"
android:layout_marginStart="16dp"
android:layout_marginEnd="16dp"
android:padding="16dp"
android:hint="@string/phone_number"
app:layout_constraintTop_toBottomOf="@id/etName"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintEnd_toEndOf="parent"/>

<EditText
android:id="@+id/etEmail"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:background="@drawable/default_edit_text_background"
android:layout_marginTop="24dp"
android:layout_marginStart="16dp"
android:layout_marginEnd="16dp"
android:padding="16dp"
android:hint="@string/email"
app:layout_constraintTop_toBottomOf="@id/etPhone"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintEnd_toEndOf="parent"/>

<EditText
android:id="@+id/etPassword"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_marginStart="16dp"
android:layout_marginTop="24dp"
android:layout_marginEnd="16dp"
android:background="@drawable/default_edit_text_background"
android:hint="@string/password"
android:inputType="textPassword"
android:padding="16dp"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toBottomOf="@id/etEmail"/>
```

```
<CheckBox
android:id="@+id/checkbox"
android:layout_width="32dp"
android:layout_height="32dp"
android:layout_marginStart="16dp"
android:layout_marginTop="16dp"
android:backgroundTint="@color/colorAccent"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toBottomOf="@id/etPassword"/>

<TextView
android:id="@+id/termsOfUse"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_marginStart="16dp"
android:text="@string/youAgree"
android:textColor="@android:color/white"
app:layout_constraintBottom_toBottomOf="@+id/checkbox"
app:layout_constraintStart_toEndOf="@id/checkbox"
app:layout_constraintTop_toTopOf="@+id/checkbox"/>

<TextView
android:id="@+id/termsOfUse2"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/terms"
android:textColor="@color/colorAccent"
app:layout_constraintBottom_toBottomOf="@+id/termsOfUse"
app:layout_constraintStart_toEndOf="@id/termsOfUse"
app:layout_constraintTop_toTopOf="@+id/termsOfUse"/>

<com.google.android.material.button.MaterialButton
android:id="@+id/btnSignUp"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:padding="16dp"
android:textStyle="bold"
android:textSize="18sp"
android:enabled="false"
android:layout_marginStart="16dp"
android:layout_marginEnd="16dp"
android:layout_marginTop="32dp"
android:text="@string/sign_up"
android:textColor="@android:color/white"
android:backgroundTint="@drawable/default_material_button_background"
app:cornerRadius="10dp"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintTop_toBottomOf="@id/termsOfUse"/>
</androidx.constraintlayout.widget.ConstraintLayout>

</layout>
```

Fragment_self.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayoutxmlns:android="http://schem
as.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:background="@color/colorPrimary"
xmlns:app="http://schemas.android.com/apk/res-auto">

<ImageView
android:id="@+id/ivImageView"
android:layout_width="200dp"
android:layout_height="200dp"
android:layout_margin="16dp"
android:src="@drawable/photo1"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toTopOf="parent"
app:layout_constraintEnd_toEndOf="parent"/>

<EditText
android:id="@+id/etName"
android:layout_width="0dp"
android:layout_height="wrap_content"
android:layout_marginStart="16dp"
android:layout_marginTop="16dp"
android:layout_marginEnd="16dp"
android:background="@drawable/default_edit_text_background"
android:hint="@string/name_prod"
android:padding="16dp"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toBottomOf="@id/ivImageView"/>

<EditText
android:id="@+id/etDescription"
android:layout_width="0dp"
android:layout_height="wrap_content"
android:background="@drawable/default_edit_text_background"
android:layout_marginTop="24dp"
android:layout_marginStart="16dp"
android:layout_marginEnd="16dp"
android:padding="16dp"
android:hint="@string/desc_prod"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toBottomOf="@id/etName"/>

<Spinner
android:id="@+id/spinner"
android:layout_width="0dp"
android:layout_height="50dp"
android:layout_marginTop="24dp"
android:layout_marginStart="16dp"
android:layout_marginEnd="16dp"
android:background="@drawable/default_edit_text_background"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toBottomOf="@id/etDescription"/>

<com.google.android.material.button.MaterialButton
```

```
android:id="@+id/btnSignUp"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_marginStart="16dp"
android:layout_marginEnd="16dp"
android:layout_marginBottom="32dp"
android:backgroundTint="@drawable/default_material_button_background"
android:padding="16dp"
android:text="@string/send"
android:textColor="@android:color/white"
android:textSize="18sp"
android:textStyle="bold"
app:cornerRadius="10dp"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintStart_toStartOf="parent"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```

ActivityMain.kt

```
packagecom.vero.sibs.ui

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import com.vero.sibs.R
import dagger.hilt.android.AndroidEntryPoint

@AndroidEntryPoint
classMainActivity : AppCompatActivity() {
overridefunonCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
}
}
```

Fragment_registration.kt

```
packagecom.vero.sibs.ui.registration

import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.appcompat.app.AppCompatActivity
import androidx.databinding.DataBindingUtil
import androidx.fragment.app.Fragment
import androidx.fragment.app.viewModels
import androidx.navigation.fragment.findNavController
import com.vero.sibs.R
import com.vero.sibs.databinding.FragmentRegistrationBinding
import dagger.hilt.android.AndroidEntryPoint
```

```
@AndroidEntryPoint
classRegistrationFragment : Fragment(R.layout.fragment_registration) {

privateval viewModel: RegistrationViewModel by viewModels()
private lateinit var binding: FragmentRegistrationBinding

overridefunonCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        binding = DataBindingUtil.inflate(inflater,
R.layout.fragment_registration, container,false)
        binding.lifecycleOwner = this
        binding.viewModel = viewModel

return binding.root
    }

overridefunonResume() {
        super.onResume()
        (requireActivity() as AppCompatActivity).supportActionBar?.hide()
    }

overridefunonViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        initObservers()
    }

privatefuninitObservers() {
        viewModel.isAuthorize.observe(viewLifecycleOwner, { isRegister ->
if (isRegister)
                findNavController().navigate(

RegistrationFragmentDirections.actionRegistrationFragment2ToMainFragment2()
)
        })
    }
}
```

RegistrationViewModel.kt

```
packagecom.vero.sibs.ui.registration

import androidx.hilt.lifecycle.ViewModelInject
import androidx.lifecycle.MediatorLiveData
import androidx.lifecycle.MutableLiveData
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.vero.sibs.utils.NameValidator
import com.vero.sibs.utils.PhoneValidator
import com.vero.core_db_impl.api.LocalStorage
import com.vero.core.model.User
import kotlinx.coroutines.launch

classRegistrationViewModel @ViewModelInject constructor (
```

```
privateval phoneValidator: PhoneValidator,
privateval nameValidator: NameValidator,
privateval localStorage: LocalStorage
) : ViewModel() {

val phoneNumber = MutableLiveData("")
val name = MutableLiveData("")
val isAuthorize = MutableLiveData(false)

val valid = MediatorLiveData<Boolean>().apply {
        addSource(phoneNumber) {
            value = phoneValidator.phoneIsValid(it) &&
nameValidator.nameIsValid(name.value ?: "")
        }
        addSource(name) {
            value = nameValidator.nameIsValid(it) &&
phoneValidator.phoneIsValid(phoneNumber.value ?: "")
        }
    }

funsignUpUser() {
        viewModelScope.launch {
            localStorage.saveUser(
                User(
                    username = name.value ?: "",
                    phone = phoneNumber.value ?: "")
)
            isAuthorize.postValue(true)
}
}
}
```

MainFragment.kt

```
packagecom.vero.sibs.ui.main

import android.os.Bundle
import android.view.*
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import androidx.databinding.DataBindingUtil
import androidx.fragment.app.Fragment
import com.vero.sibs.R
import com.vero.sibs.databinding.FragmentMainBinding
import dagger.hilt.android.AndroidEntryPoint

@AndroidEntryPoint
classMainFragment : Fragment(R.layout.fragment_main) {

private lateinit var binding: FragmentMainBinding

overridefunonCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
```

```
        setHasOptionsMenu(true)
        binding = DataBindingUtil.inflate(inflater, R.layout.fragment_main,
container,false)
        binding.lifecycleOwner = this

return binding.root
    }

overridefunonViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        (requireActivity() as AppCompatActivity).supportActionBar?.show()
        initListeners()
    }

overridefunonCreateOptionsMenu(menu: Menu, inflater: MenuInflater) {
        super.onCreateOptionsMenu(menu, inflater)
        inflater.inflate(R.menu.menu_main_fragment, menu)
    }

overridefunonOptionsItemSelected(item: MenuItem): Boolean {
when (item.itemId) {
            R.id.miOpenProfile ->
RegistrationFragmentDirections.actionRegistrationFragment2ToMainFragment2()

        }
return super.onOptionsItemSelected(item)
    }

privatefuninitListeners() {
        binding.btnBuy.setOnClickListener {
RegistrationFragmentDirections.actionRegistrationFragment2ToMainFragment2()
}
        binding.btnSell.setOnClickListener {
RegistrationFragmentDirections.actionRegistrationFragment2ToMainFragment2()
}
    }
}
```

ProfileViewModel.kt

```
packagecom.vero.sibs.ui.profile

import androidx.hilt.lifecycle.ViewModelInject
import androidx.lifecycle.MediatorLiveData
import androidx.lifecycle.MutableLiveData
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.vero.core.model.User
import com.vero.core_db_impl.api.DataStorage
import com.vero.sibs.utils.NameValidator
import com.vero.sibs.utils.PhoneValidator
import kotlinx.coroutines.launch

classProfileViewModel @ViewModelInject constructor(
privateval dataStorage: DataStorage,
privateval phoneValidator: PhoneValidator,
```

```
privateval nameValidator: NameValidator
) : ViewModel() {

var phoneNumber = MutableLiveData("")
var name = MutableLiveData("")
var successLogOut = MutableLiveData(false)

val fieldIsValid = MediatorLiveData<Boolean>().apply {
        addSource(phoneNumber) {
            value = phoneValidator.phoneIsValid(it) &&
nameValidator.nameIsValid(name.value ?: "")
        }
        addSource(name) {
            value = nameValidator.nameIsValid(it) &&
phoneValidator.phoneIsValid(phoneNumber.value ?: "")
        }
    }

funchangeUserData() {
        viewModelScope.launch {
            dataStorage.saveUser(
                User(
                    username = name.value ?: "",
                    phone = phoneNumber.value ?: "")
            )
            fieldIsValid.postValue(true)
        }
    }

funlogOut() {
viewModelScope.launch {
            dataStorage.logOut()
}
}
}
```

ProfileFragment.kt

```
packagecom.vero.sibs.ui.profile

import android.os.Bundle
import android.view.*
import androidx.appcompat.app.AppCompatActivity
import androidx.databinding.DataBindingUtil
import androidx.fragment.app.Fragment
import androidx.fragment.app.viewModels
import androidx.navigation.fragment.findNavController
import com.vero.sibs.R
import com.vero.sibs.databinding.FragmentProfileBinding
import dagger.hilt.android.AndroidEntryPoint

@AndroidEntryPoint
classProfileFragment : Fragment(R.layout.fragment_profile) {

privateval viewModel : ProfileViewModel by viewModels()
private lateinit var binding : FragmentProfileBinding
```

```
overridefunonCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
): View? {
        binding = DataBindingUtil.inflate(inflater,
R.layout.fragment_profile, container, false)
        binding.lifecycleOwner = this
        binding.viewModel = viewModel

return binding.root
    }

overridefunonViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        setHasOptionsMenu(true)
        (requireActivity() as AppCompatActivity).supportActionBar?.show()
        initObservers()
    }
overridefunonCreateOptionsMenu(menu: Menu, inflater: MenuInflater) {
return inflater.inflate(R.menu.menu_profile_fragment, menu)
    }

overridefunonOptionsItemSelected(item: MenuItem): Boolean {
when (item.itemId) {
            R.id.miLogOut -> viewModel.logOut()
        }
return super.onOptionsItemSelected(item)
    }

privatefuninitObservers() {
        viewModel.successLogOut.observe(viewLifecycleOwner, { successLogOut -
>
if (successLogOut)
                findNavController().navigate(

ProfileFragmentDirections.actionProfileFragmentToRegistrationFragment2())
})
    }
}
```

SellFragment.kt

```
packagecom.vero.sibs.ui.sell

import android.app.Activity
import android.content.Intent
import android.os.Bundle
import androidx.activity.viewModels
import com.skydoves.transformationlayout.TransformationCompat
import com.skydoves.transformationlayout.TransformationLayout
import dagger.hilt.android.AndroidEntryPoint

@AndroidEntryPoint
classSellFragment : Fragment() {
```

```
privateval binding: SellFragmentBinding by binding(R.layout.activity_detail)
privateval viewModel: SellViewModel by viewModels()

overridefunonCreate(savedInstanceState: Bundle?) {
        onTransformationEndContainerApplyParams()
        super.onCreate(savedInstanceState)
valitem: Prod = requireNotNull(intent.getParcelableExtra(EXTRA))
        binding.apply {
prod = item
            lifecycleOwner = this@DetailActivity
            vm = viewModel.apply { fetchProductInfo(item.name) }
        }
    }

    companion object {
private const val EXTRA = "EXTRA "

funstartActivity(transformationLayout: TransformationLayout, prod: Product) {
val context = transformationLayout.context
if (context is Activity) {
val intent = Intent(context, DetailActivity::class.java)
          intent.putExtra(EXTRA, prod)
                TransformationCompat.startActivity(transformationLayout,
intent)
            }
        }
    }
}
```

SellViewModel.kt

```
packagecom.vero.sibs.ui.sell

import androidx.annotation.MainThread
import androidx.databinding.ObservableBoolean
import androidx.hilt.lifecycle.ViewModelInject
import androidx.lifecycle.LiveData
import androidx.lifecycle.MutableLiveData
import androidx.lifecycle.asLiveData
import androidx.lifecycle.switchMap
import timber.log.Timber

classSellViewModel @ViewModelInject constructor(
privatevalsellRepository: SellRepository
) : LiveCoroutinesViewModel() {

privatevar FetchingLiveData: MutableLiveData<String> = MutableLiveData()
valinfoLiveData: LiveData<prodInfo?>

val isLoading: ObservableBoolean = ObservableBoolean(false)
val toastLiveData: MutableLiveData<String> = MutableLiveData()

    init {
infoLiveData = fetchingLiveData.switchMap {
            isLoading.set(true)
            launchOnViewModelScope {
this.detailRepository.fetchInfo(
                    name = it,
```

```
                    onSuccess = {isLoading.set(false) },
                    onError = {toastLiveData.postValue(it) }
                ).asLiveData()
            }
        }
    }

    @MainThread
funfetchInfo(name: String) {
 fetchingLiveData.value = name
    }
}
```

BuyFragment.kt

```
packagecom.vero.sibs.ui.buy

import android.os.Bundle
import androidx.activity.viewModels
import com.skydoves.transformationlayout.onTransformationStartContainer
import dagger.hilt.android.AndroidEntryPoint

@AndroidEntryPoint
classBuyFragment : Fragment() {

privateval binding: FragmentBuyBinding by binding(R.layout.activity_main)
privateval viewModel by viewModels<MainViewModel>()

overridefunonCreate(savedInstanceState: Bundle?) {
        onTransformationStartContainer()
        super.onCreate(savedInstanceState)
        binding.apply {
            lifecycleOwner = this@MainActivity
            adapter = ProductAdapter()
            vm = viewModel.apply {
                fetchProductList(0) }
        }
    }
}
```

BuyViewModel.kt

```
packagecom.vero.sibs.ui.buy

import android.util.Log
import androidx.annotation.MainThread
import androidx.databinding.ObservableBoolean
import androidx.hilt.Assisted
import androidx.hilt.lifecycle.ViewModelInject
import androidx.lifecycle.*
import timber.log.Timber

classBuyViewModel @ViewModelInject constructor(
privatevalbuyRepository: BuyRepository,
    @Assisted privateval savedStateHandle: SavedStateHandle
) : LiveCoroutinesViewModel() {
```

```
privatevarproductFetchingLiveData: MutableLiveData<Int> = MutableLiveData()
valproductListLiveData: LiveData<List<Product>>

privateval _toastLiveData: MutableLiveData<String> = MutableLiveData()
val toastLiveData: LiveData<String>get() = _toastLiveData

val isLoading: ObservableBoolean = ObservableBoolean(false)

    init {
        productListLiveData = productFetchingLiveData.switchMap {
            isLoading.set(true)
            launchOnViewModelScope {
this.mainRepository.fetchProductList(
                    page = it,
                    onSuccess = { isLoading.set(false) },
                    onError = { _toastLiveData.postValue(it) }
                ).asLiveData()
            }
        }
    }

    @MainThread
funfetchProductList(page: Int) {
productFetchingLiveData.value = page
    }
}
```

SelfProductFragment.kt

```
packagecom.vero.sibs.ui

import android.os.Bundle
import android.view.View
import com.example.productinfo.R
import com.example.core.extension.getViewModel
import com.example.core.extension.observe
import com.example.core.platform.BaseFragment

class SelfProductFragment : BaseFragment() {

private lateinit var listAdapter: ProductListAdapter
private lateinit var productsViewModel: ProductsViewModel

override fun layoutId() = R.layout.fragment_list

override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        productsViewModel = getViewModel(ProductsViewModel::class.java)
    }

Override fun onActivityCreated(savedInstanceState: Bundle?) {
super.onActivityCreated(savedInstanceState)
        setupListAdapter()

        with(productsViewModel) {
            observe(dataLoading) { value ->
if (value) showProgress() else {
```

```
                hideProgress()
                println("Request completed.")
            }
        }
        observe(items) { value ->
            listAdapter.collection = value.orEmpty()
        }
        observe(noDataEvent) { value ->
            emptyView.visibility = if (value) {
                listAdapter.collection = emptyList()
                View.VISIBLE
            } else View.GONE
        }
    }

    productsViewModel.loadDataWithIntervalUpdate()
}

Private fun setupListAdapter() {
    listAdapter = ProductListAdapter()
    with(recyclerView) {
val layoutManager = androidx.recyclerview.widget.LinearLayoutManager(context)
        setLayoutManager(layoutManager)
        setHasFixedSize(true)
        adapter = listAdapter
    }
    listAdapter.clickListener = { productsViewModel.openNext(it) }
}
}
```

BasketFragment.kt

```
Package com.vero.sibs.ui

import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.Button
import android.widget.TextView
import android.widget.Toast
import androidx.fragment.app.Fragment
import androidx.lifecycle.Observer
import androidx.recyclerview.widget.LinearLayoutManager
import androidx.recyclerview.widget.RecyclerView
import org.koin.android.ext.android.inject

classBasketFragment : Fragment(), BasketAdapter.ItemClick {

private lateinit var basketRecyclerView: RecyclerView
private lateinit var tvPrice: TextView
private lateinit var btnOrder: Button

privateval basketAdapter: BasketAdapter by lazy {
    BasketAdapter(itemClickListener = this)
}
privateval basketViewModel: BasketViewModel by inject()
privateval restaurantDetailsViewModel: RestaurantDetailsViewModel by inject()
```

```
overridefunonCreateView(
        inflater: LayoutInflater, container: ViewGroup?, savedInstanceState:
Bundle?
    ): View? {
return inflater.inflate(R.layout.basket_fragment, container, false)
    }

overridefunonViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        bindViews(view)
        setAdapter()
        getBasketItems()
        observe()
    }

overridefunitemClick(position: Int, item: Dish) {
        Toast.makeText(requireContext(), "hello", Toast.LENGTH_SHORT).show()
    }

overridefunupdateItemAmount(item: Dish?, amount: Int) {
        basketViewModel.changeOrderedDishAmount(item, amount)
    }

privatefunbindViews(view: View) {
        basketRecyclerView = view.findViewById(R.id.basketRecyclerView)
        tvPrice = view.findViewById(R.id.tvPrice)
        btnOrder = view.findViewById(R.id.btnOrder)

        btnOrder.setOnClickListener {
            Toast.makeText(requireContext(), "Order made",
Toast.LENGTH_SHORT).show()
            basketViewModel.clearBasket()
        }
    }

privatefunsetAdapter() {
        basketRecyclerView.layoutManager = LinearLayoutManager(context)
        basketRecyclerView.adapter = basketAdapter
    }

privatefungetBasketItems() {
        basketViewModel.getItems()
}

privatefungetTotalPrice(items: List<Dish>?): Int {
var count = 0
if (!items.isNullOrEmpty()) {
for (item in items) {
                count += item.amount?.let { item.price?.times(it) } ?: 0
            }
        }
return count
    }

privatefunobserve() {
        basketViewModel.liveData.observe(requireActivity(), Observer { result
->
when (result) {
is BasketViewModel.State.Items -> {
```

```
                    basketAdapter.setCartItems(result.items)
                    tvPrice.text = context?.getString(R.string.price,
getTotalPrice(result.items))
                }
is BasketViewModel.State.Inserted -> {
                    basketAdapter.insertItem(result.item)
                }
is BasketViewModel.State.BasketCleared -> {
                    basketAdapter.removeItems()
restaurantDetailsViewModel.itemsDeleted()
                }
            }
        })
    }
}
```

BasketViewModel.kt

```
packagecom.vero.sibs.ui

import androidx.lifecycle.MutableLiveData
import com.example.ramsay.model.Dish
import com.example.ramsay.repository.RestaurantRepository
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch
import kotlinx.coroutines.withContext

classBasketViewModel() : ViewModel() {

val liveData = MutableLiveData<State>()

fungetItems() {
        launch {
val items = withContext(Dispatchers.IO) {
                restaurantRepository.getCartItems()
            }
            liveData.value = State.Items(items)
        }
    }

funinsertItem(item: Dish?) {
if (item != null) {
            launch {
                withContext(Dispatchers.IO) {
                    restaurantRepository.addToCart(item.id)
}
                liveData.value = State.Inserted(item)
            }
        }
    }

funchangeOrderedDishAmount(dish: Dish?, amount: Int) {
        launch {
            withContext(Dispatchers.IO) {
if (dish != null) {
                    restaurantRepository.updateDishAmount(dish, amount)
                }
```

```
            }
            liveData.value = State.DishAmountChanged
        }
    }

    funclearBasket() {
        launch {
            withContext(Dispatchers.IO) {
                restaurantRepository.clearCart()
            }
            liveData.value = State.BasketCleared
        }
    }

    sealedclassState {
        object DishAmountChanged : State()
        object BasketCleared : State()
        data classItems(val items: List<Dish>?) : State()
        data classInserted(val item: Dish) : State()
    }
}
```