Державний вищий навчальний заклад
“Прикарпатський національний університет імені Василя Стефаника”
Кафедра інформаційних технологій

УДК 004

ДИПЛОМНИЙ ПРОЕКТ

Тема Розробка програмного забезпечення для перевірки складу продуктових товарів

Спеціальність 121 Інженерія програмного забезпечення

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДП.ІПЗ-22.ПЗ

Рецензент	Студент
доц. Ткачук В.М.	ІПЗ-41 Мацьків Т.М.
(посада) (підпис) (дата) (розшифровка підпису)	(шифр групи) (підпис) (дата) (розшифровка підпису)
Нормоконтролер	Керівник дипломного проекту
доц. Ткачук В.М.	проф. Кузь М.В
(посада) (підпис) (дата) (розшифровка підпису)	(посада) (підпис) (дата) (розшифровка підпису)

Допускається до захисту

Завідувач кафедри

доц. Козленко М.І.
(посада) (підпис) (дата) (розшифровка підпису)

2020

Державний вищий навчальний заклад
«Прикарпатський національний університет імені Василя Стефаника»
Факультет математики та інформатики Кафедра інформаційних технологій
Спеціальність інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Завідувач кафедри Козленко М.І.
студенту Мацьків Тетяні Михайлівній
"_____”_____ 20___р.

ЗАВДАННЯ
НА ВИКОНАННЯ ДИПЛОМНОГО ПРОЕКТУ

Студенту Мацьків Тетяні Михайлівній
1. Тема проекту Розробка програмного забезпечення для перевірки складу продуктових товарів

Затверджена розпорядженням факультету математики та інформатики від 25 жовтня 2019р. №7
2. Термін здачі студентом закінченого проекту 22 травня 2020р.
3. Вихідні дані до дипломного проекту категорії продуктів, поживна цінність продукту, інформація про потенційно небезпечні продукти та алергени, статистика захворювань по Україні через нездоровий спосіб харчування. Технології розробки – Java, Android.

4. Зміст пояснювальної записки (перелік питань, що їх належить опрацювати)
1. Аналіз області здорового харчування та постановка задачі
2. Проектування додатку
3. Програмна реалізація програмного забезпечення
4. Бізнес-план розробки додатку «EatBetter.»
5. Перелік графічного матеріалу (з точним забезпеченням обов’язкових креслень) «Purpose and novelty», «Similar products», «Key features», «Main functionality», «Development technologies», «Development patterns», «Database structure», «Application demonstration», «Business case», «Conclusions».

6. Дата видачі завдання 11 вересня 2020р.

Керівник _____ Кузь М.В.
(підпис) (розшифровка підпису)

Завдання прийняв до виконання _____ Мацьків Т.М.
(підпис) (розшифровка підпису)

КАЛЕНДАРНИЙ ПЛАН

Номер і назва етапів дипломного проекту	Термін виконання етапів проекту	Примітка
1. Аналіз області здорового харчування та постановка задачі	02.12.2019	
2. Проектування додатку	15.02.2020	
3. Програмна реалізація мобільного застосунку	17.04.2020	
4. Бізнес-план розробки додатку «EatBetter.»	11.05.2020	
5. Оформлення пояснювальної записки	18.05.2020	

Студент _____ Мацьків Т.М.
(підпис) (розшифровка підпису)

Керівник проекту _____ Кузь М.В.
(підпис) (розшифровка підпису)

РЕФЕРАТ

Пояснювальна записка: 86 сторінок (без додатків), 82 рисунка, 2 таблиці, 25 джерел, 1 додаток на 12 сторінках.

Ключові слова: здорове харчування, сканування, мобільний додаток, Android, Java, Firebase.

Об`єктом дослідження є мобільний застосунок для користувачів операційної системи Android для перевірки складу продуктових товарів.

Мета роботи: розробити програмний продукт, що забезпечить швидке сканування та точний аналіз складу продуктових товарів.

Стислий опис тексту пояснювальної записки: у ході виконання дипломного проекту було проведено аналіз предметної області здорового харчування та існуючих аналогів програмного продукту. Розроблено дизайн інтерфейсу, спроектована база даних і архітектура додатку, і розроблено бізнес-логіку. Також виконано економічне обгрунтування розробки мобільного застосунку.

ABSTRACT

Explanatory note: 86 pages (without appendix), 82 figures, 2 tables, 25 references, 1 appendix on 12 pages.

Key words: healthy eating, scanning, mobile app, Android, Java, Firebase.

Object of study: mobile app for Andorid users for checking the ingredients of food products.

Purpose of study to develop software which will provide fast scanning and precise analyze of food products' ingredients.

Brief description of the text of the explanatory note: during the implementation of the diploma work the analysis of the healthy eating field and existing analogues of the software were held. The design of user interface was designed, database and architecture of application were developed as well as business logic. The business case of mobile application was also developed.

ЗМІСТ

ВСТУП

Харчування – це невід'ємна частина правильного функціонування організму людини. Ми просто не можемо не вживати їжу більше, ніж декілька днів. Також спосіб харчування впливає на якість життя, тому саме як ви харчуєтесь є дуже важливим.

Останніми роками починає набувати популярності так званий тренд на здорове, збалансоване харчування, яке включає в себе: вживати корисну їжу, мати в раціоні достатню кількість клітковини та поживних речовин. Для того, щоб харчуватись правильно потрібно і слідкувати за продуктами, що ми купуємо в магазинах, за їхнім складом. Адже саме склад визначає корисність продукту і його вплив на організм. Для того, щоб не запам'ятовувати списки складників, що вживати не рекомендується, щоб не витрачати зайвий час на самостійний аналіз складу продукту потрібно завжди з собою мати «помічника». В такому випадку ідея розробки мобільного додатку «EatBetter.» стає актуальною. Додаток дозволить швидко сканувати та потім аналізувати склад продуктового товару, що значно полегшить процес покупки продуктів.

Предмет роботи - мобільний застосунок для користувачів операційної системи Android для перевірки складу продуктових товарів.

Мета роботи – спроектувати та розробити мобільний додаток для сканування штрих-коду товару та аналізу його складу.

Для того, щоб мета роботи була досягнута було поставлені такі цілі:

- дослідити предметну область здорового харчування;
- зробити огляд існуючих аналогів;
- визначити вимоги до програмного забезпечення та функціоналу;
- обрати технології розробки;
- спроектувати архітектуру та базу даних;
- спроектувати та відмалювати дизайн інтерфейсу користувача;
- реалізувати визначений функціонал додатку;

– забезпечити якість програмного продукту за допомогою тестування.

Результат роботи – мобільний застосунок на базі операційної системи Android, що дозволить користувачам аналізувати склад продуктових товарів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ЗДОРОВОГО ХАРЧУВАННЯ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Дослідження предметної області

Всі ми знаємо, що харчування важливе для нашого організму. У поєднанні з фізичною активністю, правильне харчування - це відмінний спосіб допомогти нашому тілу залишатися міцним і здоровим.

Здорове, правильне, збалансоване харчування - це харчування, яке забезпечує ріст, нормальний розвиток і життєдіяльність людини, що також сприяє зміцненню його здоров'я та профілактиці захворювань [1].

Останнім часом здорове харчування стало трендом і це можна пояснити більшою обізнаністю щодо шкоди деяких продуктів харчування. Люди почали цікавитись складом продуктів, процесом їх виготовлення та навіть процесами на виробництві [2]. Також набули популярності нутриціологія, дієтологія та багато інших наук, що пов’язані з дослідженням продуктів та їх взаємодії з нашим організмом.

Хоча, сфера харчування за останні роки під впливом тренду на здорове харчування змінилась, ми все одно бачимо багато продуктів з шкідливими добавками та вживаємо їх щодня. Тому важливо розуміти які наслідки несе за собою наявність таких продуктів у нашому раціоні.

Та й не на всіх людей вплинув цей тренд на здорове та збалансоване харчування. Більшість людей в Україні харчуються не найкращим чином. У 2018 році Україна посіла «почесне» перше місце серед країн Європи за смертністю від неправильного харчування. Учені проаналізували дані 51 країни європейського регіону за період з 1990 по 2016 рік. За даними дослідження, у 2016 році майже 40% всіх смертей в Україні були тією чи іншою мірою пов'язані з неправильним харчуванням.

Рисунок 1.1 – Обсяг споживання продуктів в Україні та інших країнах [3]

І хоча за калорійністю щоденне меню українця практично не поступається меню середньостатистичного поляка або француза, складаючи трохи більше 2 тис. калорій, його складно назвати збалансованим (див. рис. 1.1) [3]. Мабуть, у зв'язку з такою статистикою в Україні у 2017 році Уляна Супрун, діючий на той час міністр охорони здоров'я, видала загальні рекомендації щодо здорового харчування дорослих.

Також в цьому ж році було затверджено «тарілку здорового харчування», в якій відображено рекомендації щодо споживання продуктів (див. рис. 1.2).

Рисунок 1.2 – Тарілка здорового харчування [4]

Як видно з рисунку 1.2 МОЗ рекомендує вживати овочі та бобові щодня, що має складати більшу частину щоденного раціону людини. У цьому є певний сенс, адже саме овочі містять в собі багато клітковини, яка потрібна для хорошого травлення. Також людина має вживати фрукти, молочну продукцію низької та помірної жирності, корисні олії та жири, цільнозернові продукти, і також важливим компонентом у раціоні є м'ясні страви і яйця.

Проте, це всього лиш рекомендації і людина може формувати свій тижневий раціон враховуючи свої вподобання, потреби організму та особливості території, де вона проживає. Зважаючи на те, що вподобання та потреби організму у різних людей різні існують різноманітні способи харчування. Наприклад, є вегетаріанський спосіб харчування, що виключає вживання м'ясних та рибних продуктів. Також існує і веганський спосіб харчування, який крім невживання м'яса та риби, виключає вживання молочних, кисломолочних продуктів та яєць [5]. Загалом є багато різних видів/способів харчування, але основні з них такі:

- вегетаріанський;
- веганський ;
- сироїдський;
- без глютену;
- без сої;
- без лактози.

Ще один важливий момент, на який потрібно звертати увагу при складанні раціону – це алергени. Алергени – це продукти або складники, що викликають алергічну реакцію в організмі людини. Часто, вживання продуктів з алергенами в їхньому складі призводить до плачевних наслідків, тому аналізувати склад продуктів людям-алергікам – обов'язково, щоб уникнути наслідків. Є вісім найпоширеніших алергенів [6]:

- яйця;
- риба;

– горіхи;

– злаки;

– морепродукти;

– мед;

– шоколад;

– лактоза.

Також в наш час багато людей відмовляються від цукру, глутамату натрію та ряду інших продуктів, які вважають небезпечними чи некорисними для свого організму. І варто зауважити, що саме цукор є у більшості продуктів харчування.

Дослідивши предметну область, можна зробити висновки, що аналізувати склад, якщо ви притримуєтесь здорового харчування, конкретного способу харчування чи є алергіком, є необхідним.

1.2 Огляд та аналіз існуючих аналогів

Огляд та аналіз існуючих аналогів перед початком роботи над власним додатком є необхідним кроком. Адже вивчення та порівняння додатків, які вже є на ринку, допоможе уникнути помилок, що були допущені під час розробки існуючих програмних продуктів. А також, найголовніше, врахувати всі недопрацювання та зробити свій продукт унікальним.

Для огляду я обрала три схожі за метою та функціоналом мобільні додатки. Це OpenFoodFacts, EcoEd та Сканер Їжі. Всі додатки виконують одну і ж ту функцію – сканують штрих-код продуктового товару та видають інформацію про його склад. Деякі з них, наприклад, OpenFoodFacts та EcoEd мають додаткову корисну інформацію про здорове харчування та інгредієнти.

Розглянемо кожен додаток окремо.

1.2.1 OpenFoodFacts

OpenFoodFacts – один із найпопулярніших додатків у предметній області здорового харчування (див. рис. 1.3).

Рисунок 1.3 – Логотип та іконка додатку OpenFoodFacts [7]

Він безкоштовний для всіх користувачів та має велику базу продуктів з усього світу, яка постійно поповнюється. Дизайн даного додатку простий, зрозумілий та відносно відповідає теперішнім «трендам» в розробці дизайну для мобільних додатків. Також в додатку є функція реєстрації та авторизації, що дозволяє створити свій профіль та зберігати історію сканувань та списки продуктів [7].

Для кожного продукту у цій базі даних зберігається його назва, тип упаковки, бренд, категорія, місце виробництва та переробки, країни та магазини, де продається продукт, перелік інгредієнтів (включаючи «небезпечні», наприклад, алергени, харчові добавки). До бази даних може додавати будь-який зареєстрований користувач, при цьому заповнюючи всю необхідну для аналізу складу інформацію, а також фото упаковки.

На рисунку 1.4 можна побачити головний «Home» екран додатку OpenFoodFacts. На ній відображаються вкладки «Сканувати», «Compare» («Порівняти»), «Історія» та «Ваші списки».

Рисунок 1.4 – Головний екран OpenFoodFacts

Власне, кожна з цих вкладок відображає ту чи іншу функцію, що може виконувати цей додаток. Основна функція – сканування штрих-кодів. Після сканування штрих-коду харчового продукту відкривається екран з основною інформацією про цей продукт, за умови якщо він є в базі. Інформація, що надається про продукт (див. рис. 1.5):

– показник поживних речовин;
– харчова цінність на 100г продукту;
– склад;
– алергени;
– добавки.

Рисунок 1.5 – Екран продукту

Також є цікава та відносно унікальна функція для додатків даної предметної області – порівняння. Ви можете порівнювати склад, харчову цінність та показник поживних речовин (див. рис. 1.6).

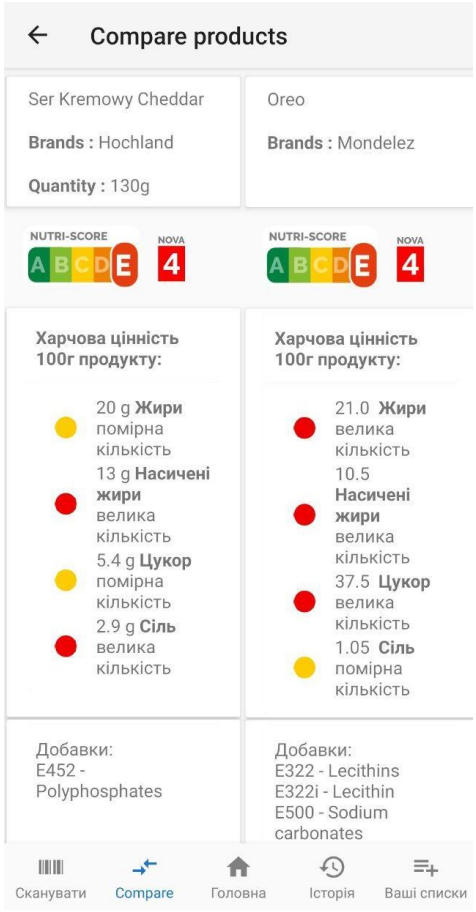

Рисунок 1.6 – Екран порівняння двох продуктів

Проте, окрім великої кількості корисних та потрібних функцій цей додаток має один великий недолік для мешканців України – база українських, та більшості країн СНД, продуктів майже відсутня. Деякі продукти, якщо і є в базі цього додатку, то є тільки фото упаковки і зовсім ніякої інформації про продукт. Тобто функціями цього додатку ви просто не зможете скористатись. На рисунку 1.7 видно результат сканування продукту українського виробника і це один з безлічі випадків. Звичайно, користувач може самостійно заповнити інформацію про цей продукт, але це займає досить багато часу.

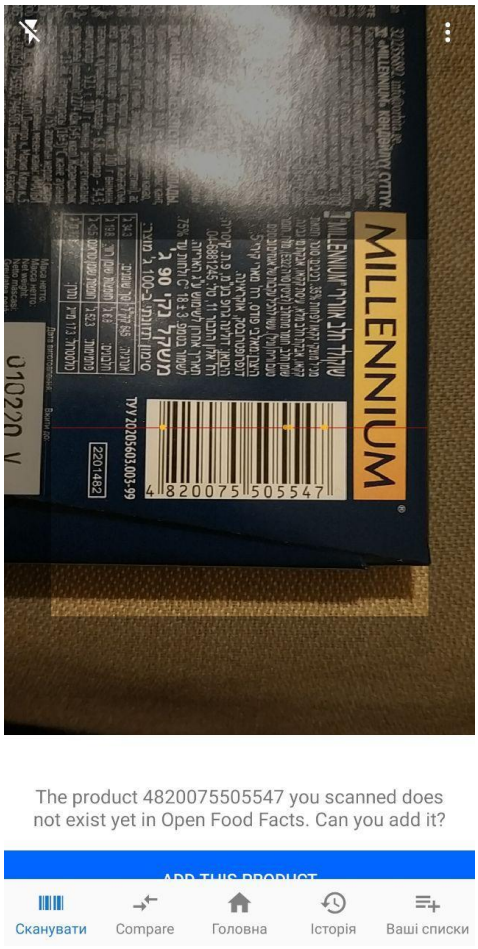

Рисунок 1.7 – Результат сканування продукту українського виробника

1.2.2 EcoEd

EcoEd – додаток російської компанії, з символічним логотипом зеленого яблука (див. рис. 1.8).

Рисунок 1.8 – Логотип та іконка додатку EcoEd [8]

Програмний продукт не є популярним, його скачали всього 500+ користувачів [8]. Проте, я вибрала цей додаток для огляду та аналізу, адже це продукт розробника з країни СНД, що робить можливим наявність бази продуктів українських виробників.

Дизайн продукту є неактуальним і не user-friendly - користувач не зможе, або ж буде важко, користуватись додатком інтуїтивно. На головному екрані є кнопка, яка виконує основну функцію додатку – сканує штрих-код (див. рис. 1.9). Проте додаток не є стабільним і в більшості разів потрібно вводити штрих-код вручну, бо просканувати штрих-код не можливо через помилку.

Рисунок 1.9 – Головний екран додатку EcoEd

Додаток також не оправдав надій щодо бази даних продуктів українських виробників. Для того, щоб перевірити як працює аналіз складу продукту мені

прийшлось шукати в мережі Інтернет продукт російського виробництва і вводити його штрих-код вручну.

На рисунку 1.10 показано результат «сканування» та аналізу продукту. Інформація, яка видається після аналізу:

– назва продукту;

– оцінка;

– склад;

– список харчових добавок.

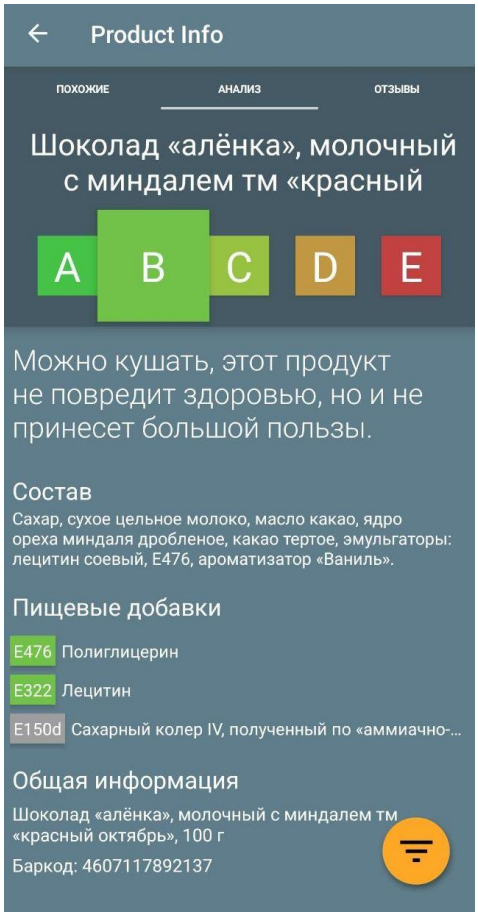

Рисунок 1.10 – Екран продукту

Також в додатку користувач може зареєструватись, налаштовувати параметри та отримувати бали за редагування інформації про продукти (див. рис. 1.11).

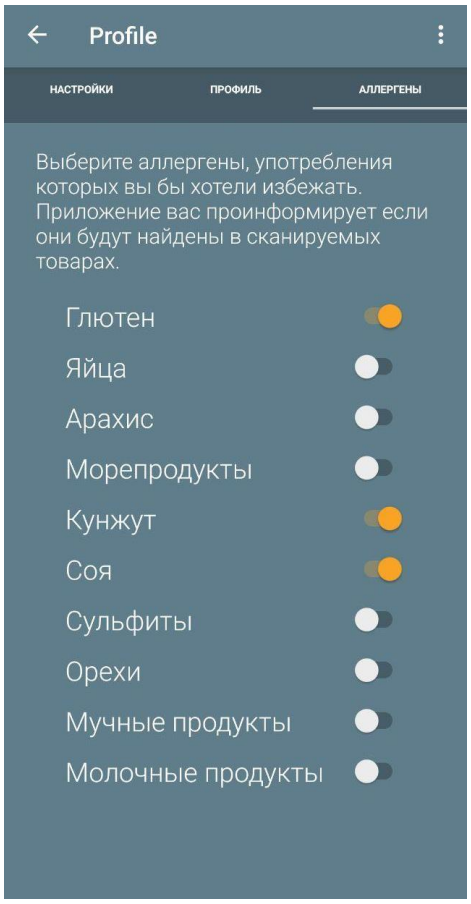

Рисунок 1.11 – Екран налаштування алергенів в їжі

1.2.3 Сканер їжі (Food Scanner)

Food Scanner – американський безкоштовний додаток від Netpeak Mobile [9]. Додаток має лаконічний та стильний дизайн та айдентику (див. рис. 1.12), добре продуманий app-flow.

Рисунок 1.12 – Логотип та іконка додатку Food Scanner [9]

Після встановлення додатку на мобільний пристрій та його першого запуску ви побачите екрани, де вам потрібно вказати що ви їсте і що ви не їсте (див. рис. 1.13). За цими параметрами потім буде аналізуватись склад продуктів, які ви скануєте.

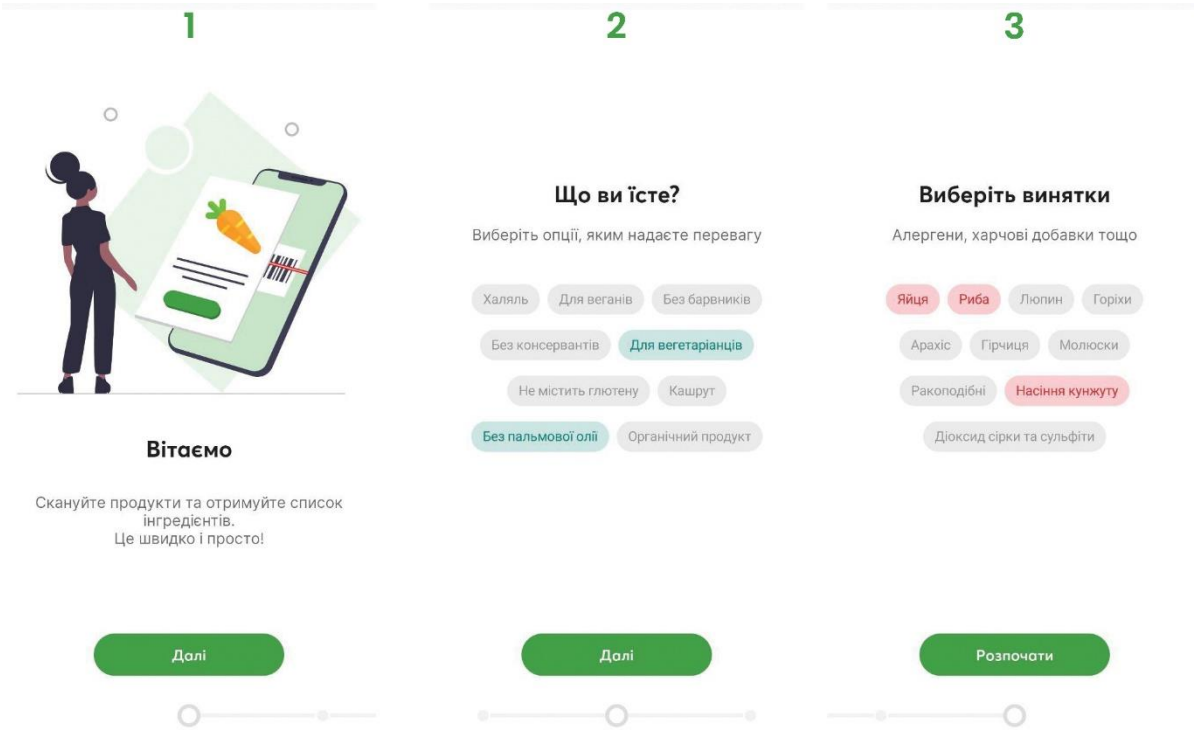

Рисунок 1.13 – Екрани налаштування параметрів

Сканер їжі має досить обмежені функції, в порівнянні з двома попередніми додатками. За допомогою даного програмного продукту користувачі зможете тільки сканувати штрих-код і бачити результат аналізу і більше нічого. Звичайно, користувач має можливість зареєструвати свій профіль, але це не надає ніяких нових функцій. Перелік функцій, що доступні користувачам зображено на рисунку 1.14.

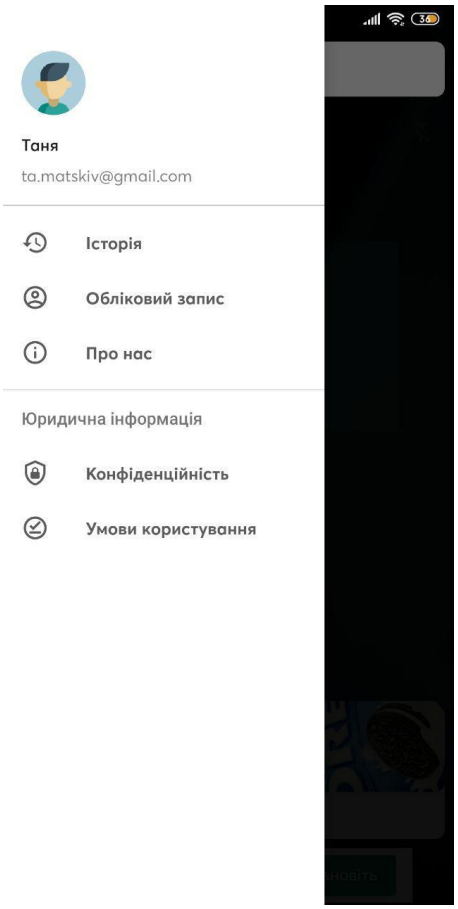

Рисунок 1.14 – Перелік доступних функцій додатку FoodScanner

База даних заповнюється користувачами. Для того, щоб додати новий продукт в базу користувачу потрібно заповнити певну «анкету» для товару. Користувач має надати наступну інформацію (див. рис. 1.15):

– назва продукту;
– фото продукту та інгредієнтів;
– тип продукту;
– алергени;
– бренд;
– харчова цінність.

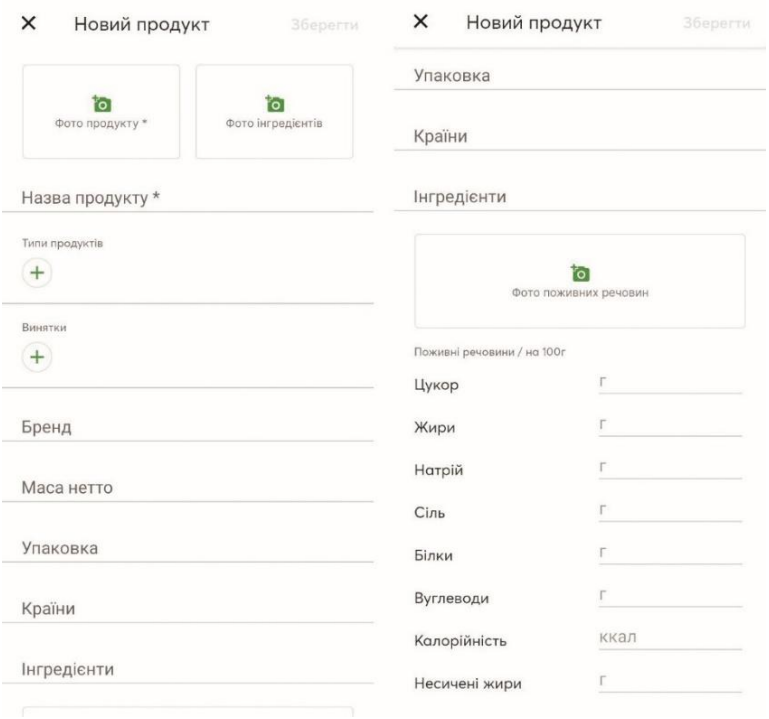

Рисунок 1.15 – Екрани додавання нового продукту

На рисунку 1.16 зображено екрани доданого користувачем продукту. Проте, не всі продукти описано так детально або й взагалі відсутні. Особливо, це стосується продуктів українського виробництва.

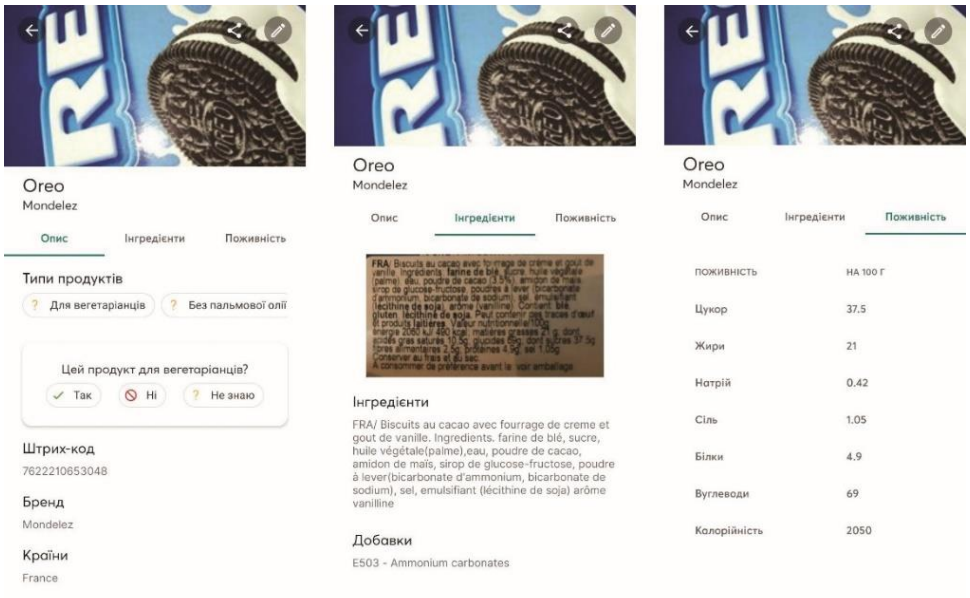

Рисунок 1.16 – Екран інформації про продукт

Детальний огляд та аналіз даних програмних продуктів допоміг визначити основні цілі та функції, що потрібно реалізувати під час розробки мобільного додатку для перевірки складу продуктових товарів в межах дипломного проекту.

1.3 Постановка задачі та вимоги до додатку

Отже, після дослідження предметної області, огляду та аналізу існуючих аналогів потрібно окреслити завдання, що мають бути виконані в межах дипломного проекту, а також чітко визначити вимоги до розроблюваного програмного продукту.

Основне завдання – розробити мобільний додаток на базі OS Android для сканування та перевірки складу продуктового товару.

Загальний опис додатку:

- мобільний додаток має бути розроблений мовою програмування Java в середовищі розробки Android Studio;
- класи і характеристики користувачів:
    - гість: анонімно авторизований користувач, який має доступ практично до всіх функцій додатку, окрім збереження даних після видалення програми;
    - користувач: зареєстрований та авторизований користувач з доступом до усіх функцій та можливостей додатку;
    - модератор: верифікація доданих в базу даних користувачами продуктів.
- робоче середовище:
    - мобільний пристрій з операційною системою Android (версія API: 26 і вище). Користувачі можуть знаходитись будь-де з під’єднаним до мережі Інтернет мобільним пристроєм. Сервер бази даних надається платформою FireBase (хмарне сховище).

Вимоги до системи:

- інтерфейс користувача:
    - інтерфейс повинен відповідати нормам та стандартам, що визначені для розробки мобільних додатків [10]. Також має бути використано колірну схему, яка буде приємна для людського ока і відповідати предметній області харчування. Інтерфейс повинен мати інтуїтивно зрозумілі можливості та відображати функціонал додатку;
- функціональні вимоги:
    - програмний продукт повинен правильно розпізнавати штрих-код, визначати складники продуктового товару, відображати список продуктів, коректно виконувати функції реєстрації та авторизації [11];
- вимоги збереженості даних:
    - всі користувацькі дані, що були отримані чи надіслані мають обов'язково бути збережені [12]. Для зареєстрованих користувачів інформація зберігається навіть після видалення додатку з мобільного пристрою;
- вимоги до безпеки системи:
    - користувач не має мати доступу до програмного коду, усі змінні, записи бази даних та інше мають бути захищені від стороннього доступу та модифікації [12].

2 ПРОЄКТУВАННЯ ДОДАТКУ

2.1 Технології розробки

В останньому підпункті першого розділу були описані вимоги до додатку і те, що очікується від нього. Отже, наступним кроком після постановки задачі є пошук інструментів та технологій за допомогою яких можна реалізувати поставлене завдання.

Оскільки, додаток розрахований на користувачів операційної системи Android, то мовою програмування була обрана Java. Java, незважаючи на те, що була випущена ще у 1995 році досі знаходиться на перших сходинках в рейтингах мов програмування. У 2019 році, наприклад, вона зайняла перше місце серед мов програмування, які найчастіше використовують для програмування [13, 14]. Також однією з переваг мови Java є високий рівень безпеки, який вона забезпечує за рахунок її архітектури. Тобто серед мов, за допомогою яких можна розробити Android-додаток, Java однозначно є лідером [15].

Середовищем розробки було обрано Android Studio. Дана IDE надзвичайно легка для розуміння і освоєння, а також в неї вбудовано багато хелперів, що значно спрощують розробку програмного забезпечення. Розробники даного середовища розробки розробили ідеальний інструмент, який допоможе розробляти програмне забезпечення без додаткової витрати часу. Вони постійно оновлюють функції, які доступні для використання в Android Studio і на даний момент доступні наступні функції (перераховані лише деякі):

- розширений редактор макетів WYSIWYG;
- збір застосунку за допомогою Gradle;
- рефакторинг коду;
- шаблони основних макетів і компонентів Android;
- вбудована підтримка Google Cloud Platform.

Також середовище надає можливість тестувати розроблений програмний продукт та підтримує систему версій Git [16].

Бібліотека ButterKnife – інструмент, який використовується для зв'язування елементів та об'єктів у фрагментах. Він використовує анотації для генерації шаблонного коду. Основне завдання цієї бібліотеки – зробити код «чистішим», більш читабельним і зменшити кількість множинного використання. Найкориснішими анотаціями цієї бібліотеки є @BindView, @OnClick, а також метод findById, які включають в себе роботу з віджетами в Activity, у фрагментах, роботу з ресурсами та обробкою подій [17].

Google Cloud Vision API – API, який дозволяє аналізувати зображення і контекстні дані, використовуючи модель машинного навчання, яка самонавчається [18]. Оскільки, додаток передбачає сканування штрих-кодів продукту, то цей API надзвичайно важливий для полегшення імплементації. Та, окрім сканування штрих-кодів і розпізнавання тексту із зображення, він ще може розпізнавати популярні логотипи, лиця, кольори та багато іншого.

Firebase – платформа для розробки програмних застосунків, яка включає в себе необхідні рішення для розроблюваного додатку в межах дипломного проекту таких, як Cloud Storage, Cloud Firestore та Firebase Auth. FirebaseAuth забезпечує бекенд-сервіси, SDK, що легко використовувати, а також готові бібліотеки інтерфейсів для аутентифікації користувачів. Також FirebaseAuth підтримує аутентифікацію, використовуючи пароль, номери телефонів та соціальні мережі [19].

2.2 База даних та її структура

Під час розробки додатку важливим етапом є проектування бази даних, яка буде забезпечувати зберігання даних і можливість здійснювати операції над ними. Для реалізації бази даних було обрано Cloud Firestore та Cloud Storage. Як

вже згадувалось вище, Firebase є платформою, яка забезпечує рішеннями для швидкої розробки програмного продукту.

Основною особливістю Firebase Cloud Firestore є те, що вона є NoSQL базою даних. NoSQL база даних – нереляційна база даних, що має відмінний від реляційних баз даних механізм зберігання даних. Цей підхід включає спрощене горизонтальне масштабування на колекції та контроль над доступністю [20]. Cloud Firestore підтримує дані, що синхронізуються між усіма пристроями користувача у реальному часі та надає підтримку без підключення до мережі Інтернет. Також ця база даних пропонує інтеграцію з іншими компонентами Firebase та продуктами Google Cloud Platfrom [21].

Також окрім Cloud Firestore буде використовуватись Cloud Storage. Це вбудований компонент, який забезпечує зберігання користувацького контенту, наприклад фото або відео. Використання цього компоненту є обов'язковим та одним з ключових для розроблюваного додатку, адже база даних повинна мати в собі інформацію про вигляд продуктових товарів [22].

Структура бази даних складається з колекцій та документів. Є дві колекції – Products, тобто колекція з продуктами та Users, тобто колекція з усіма користувачами, враховуючи і анонімних користувачів. Кожна колекція містить собі документи, кількість яких залежить від кількості продуктів і користувачів відповідно. Тобто один продукт – один документ в колекції Продуктів, так само і з користувачами – один користувач – один документ в колекції Користувачі. Кожен документ може містити в собі підколекцію, так як наприклад документ User має в собі підколекцію history, яка в свою чергу теж містить документи з продуктами, і так далі (див. рис. 2.1).

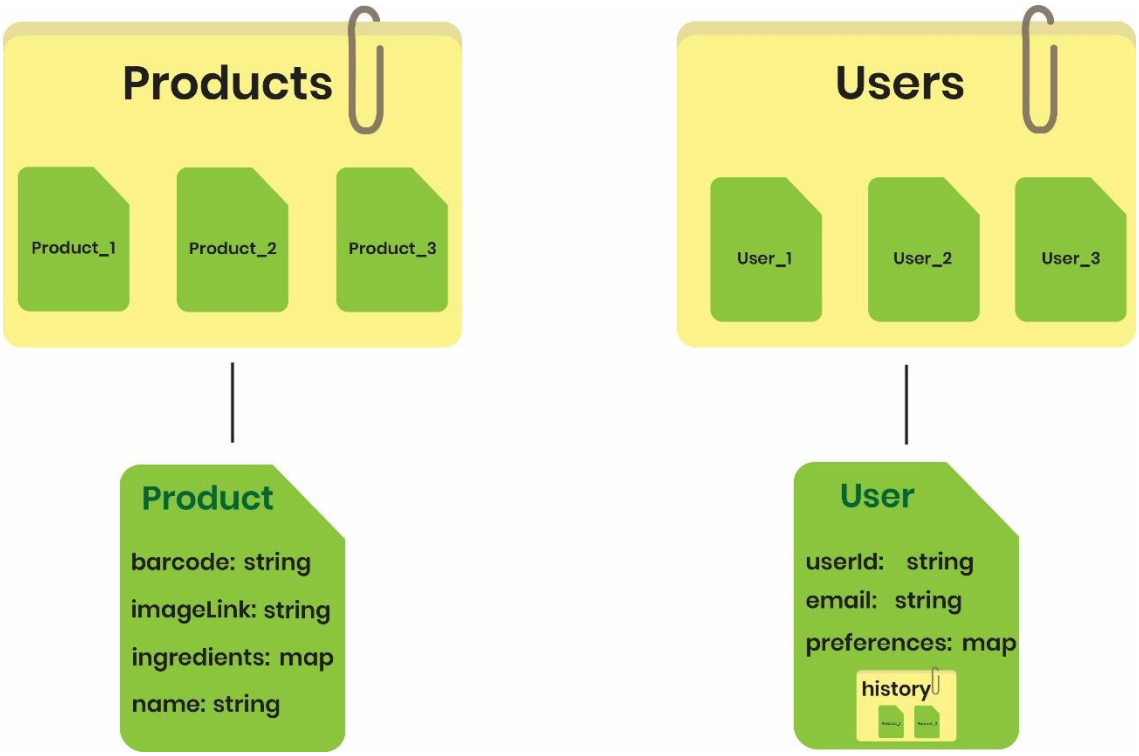

Рисунок 2.1 – Схема структури бази даних

Розглянемо також структуру кожного документа.

Документ Product призначений для збереження всієї необхідної інформації про продукт (див. рис. 2.2). Кожне поле відповідає запису в базі даних:

– barcode – поле типу String, яке зберігає в собі штрих-код відповідного продукту;
– imageLink – поле типу String, в якому зберігається посилання на фотографію, що зробив користувач (саме фото зберігається в Cloud Storage);
– ingredients – поле map (колекція ключ-значення), що містить зв’язку String-boolean. Поле призначене для зберігання інформації про наявність того чи іншого складника;
– name – поле типу String, в якому зберігається назва продукту.

Рисунок 2.2 – Документ продукту

Документ User призначений для збереження даних користувачів та містить такі поля (див. рис. 2.3):

- userId – поле типу String, яке містить згенерований під час авторизації вираз, який є унікальним для кожного користувача і містить в собі 28 символів;
- email – поле типу String, яке зберігає в собі електронну пошту користувача;
- preferences – поле map, що містить колекцію типів String-boolean та відповідає за інформацію щодо вподобань та винятків у складниках продуктових товарів;
- history – підколекція, яка зберігає в собі усі просканові товари даного користувача.

Рисунок 2.3 – Документ користувача

Дані у Cloud Firestore зберігаються наступним чином (див. рис. 2.4):

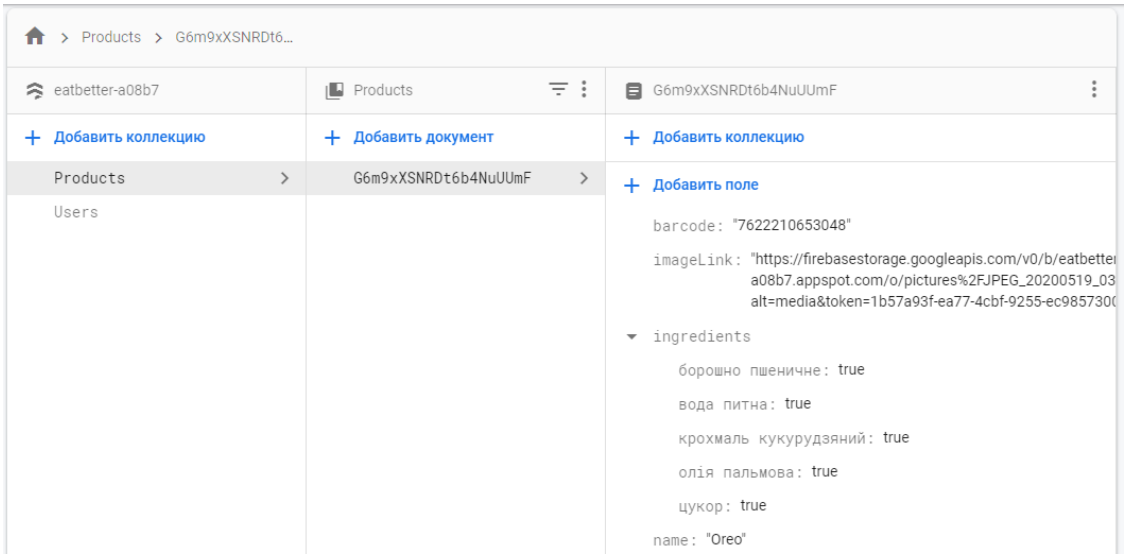

Рисунок 2.4 – Приклад збереження даних в Cloud Firestore

2.3 Архітектура додатку

Проектування архітектури додатку є одним з найважливіших етапів розробки програмного продукту. Адже саме архітектура впливає на багато важливих факторів такі, як швидкодія, передача даних, їх обробка та багато іншого, що звичайно ж впливає на загальне враження від користування додатком.

Додаток «EatBetter.» буде розроблений на основі шаблонів MVP та MVVM. Ці шаблони є одними з найпопулярніших шаблонів розробки та ефективні в реалізації.

Шаблон MVP (Model-View-Presenter) дозволяє створити таке поняття як представлення. Для цього створюється інтерфейс презентера зі своїм набором властивостей та методів. Даний шаблон має три основні складові (див. рис. 2.5):

– Model (модель) представляє собі набір класів, що описують дані та бізнес-логіку застосунку. Також тут визначається правила для даних, що описують як дані можуть бути змінені та які операції над ними можна виконувати;

– View (вигляд) – компонент, який напряму взаємодіє з користувачем, використовуючи такі речі XML, Activity та Fragment. Основною особливістю даного компонента є те, що в ньому не може бути описано ніякої бізнес-логіки, лише вигляд застосунку;
– Presenter (презентер) – компонент, який служить як «регулювальник» між моделлю і виглядом. Він отримує вхідну інформацію від користувачів через View, обробляє ці дані за допомогою моделі і віддає результати обробки вигляду. Презентер взаємодіє з виглядом напряму через інтерфейс. Також він виконує операції в моделі та оновлює вигляд відповідно до результатів [23].

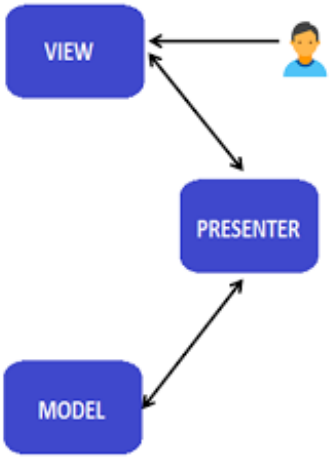

Рисунок 2.5 – Схема взаємодії за шаблоном MVP [23]

Шаблон MVVM (Model-View-ViewModel) також складається з трьох основних компонентів – Model (модель), View (вигляд) та ViewModel. Цей шаблон підтримує двостороннє зв’язування даних між моделлю і виглядом, що забезпечує автоматичне оновлення даних. ViewModel використовує шаблон observer для відслідковування змін в моделі та вигляді.

– ViewModel – компонент, що відповідальний за виявлення методів, команд та інший властивостей, що допомагає підтримувати стан вигляду відповідним до змін в моделі. View має уявлення та посилання

на ViewModel, проте останній не має посилання на вигляд. У даному випадку ми маємо своєрідно багато-до-багатьох з'єднання між виглядом і ViewModel, тобто багато виглядів може бути прив'язано до одного компонента ViewModel [23].

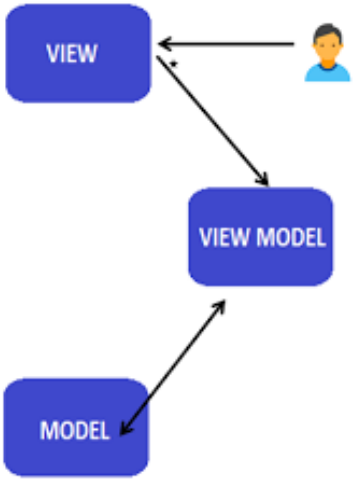

Рисунок 2.6 - Схема взаємодії за шаблоном MVVM [23]

На рисунку 2.6 біля стрілки зв'язку між виглядом та ViewModel є позначення зірочки, що означає Data Binding, тобто автоматичне зв'язування даних, про яке згадувалось вище.

Для розробки додатку було обрано два шаблони, тому що повинні бути реалізовані різні функції, які потребують різних шаблонів реалізації. Проте, основним шаблоном є MVP, а шаблон MVVM використовується для реалізації RecyclerView, адже в цьому випадку потрібний постійний моніторинг даних і їхнє відповідне представлення. Тобто властивості даного шаблону, такі як data binding та LiveData необхідні для цього.

Варто зазначити, що Firebase містить в собі більшість бізнес-логіки, що є вбудованою в цей сервіс. Тобто більшість методів, наприклад як реєстрація, авторизація та ін, вже реалізовані і готові до використання.

2.4 Дизайн інтерфейсу користувача

Наступним етапом проектування додатку є розробка дизайну інтерфейсу. Для того, щоб розробити інтерфейс, який гарно виглядає, є правильним з точки зору дизайну та інтуїтивно зрозумілий для користувача потрібно використати принципи UX та UI дизайну.

Абревіатура UX розшифровується як User eXperience, тобто UX дизайн відповідає за те, яке враження отримає користувач від користування програмним продуктом. Основними завданнями є: спроектувати інтерфейс користувача таким, що він буде максимально простим і зручним у використанні. У свою чергу, абревіатура UI – це User Interface, тобто це те, як буде виглядати програмний продукт. Основними завданнями цього напряму дизайну є: розробити інтерфейс таким, щоб він виглядав цілісним, красивим і зрозумілим [24]. Тобто, як видно з опису вище обидва напрями дизайну важливі у створенні дизайну інтерфейсу для додатку і виконують свої відповідні функції.

Отже, обов'язкові етапи на шляху до створення дизайну інтерфейсу користувача [25]:

- зібрати інформацію про сам проект, його суть та мету, а також інформацію про цільову аудиторію;
- спроектувати користувацькі сценарії;
- розробити індивідуальний стиль (підібрати колірну схему, шрифт та ін.);
- створити макети та прототип.

Для створення макетів та прототипу я використовувала векторний сервіс Figma. Сервіс надає можливість створення макетів екранів, а також прототипування.

Оскільки, предметна область, в якій розробляється додаток – здорове харчування, я обрала колірну схему з основним кольором – зеленим, що у всіх людей асоціюється з чимось хорошим, правильним та корисним. Отже, основні

кольори – темний зелений (#284818) та світлий зелений (#879C52), другорядні кольори – жовтий (#FBBF45) та персиковий (#F39F5F) (див. рис. 2.7).

Рисунок 2.7 – Колірна схема додатку

Логотип додатку також був розроблений, враховуючи предметну область здорового харчування та колірну схему. Оскільки, першими асоціаціями після почутих слів «здорове харчування» в більшості випадків є: салат, броколі, яблуко та зелений колір я обрала броколі як логотип для додатку (див. рис. 2.8). Адже навіть сама назва вказує на те, що користуючись цим додатком – ви зможете харчуватись якісніше та корисніше.

Рисунок 2.8 – Логотип додатку «EatBetter.»

Тепер почнемо розглядати основі екрани та сценарії взаємодії користувача з додатком.

Початковим екраном для додатку майже завжди є екран-індикатор його завантаження. Прелоадер може бути як і анімацією, так і просто зображенням. Цей екран виконує важливу функцію – інформує користувача, що додаток «готується до роботи», а також забезпечує взаємодію з користувачем. Екран попереднього завантаження для розроблюваного додатку зображено на рисунку 2.9.

Рисунок 2.9 – Екран попереднього завантаження додатку

Далі завантажується наступна послідовність екранів:

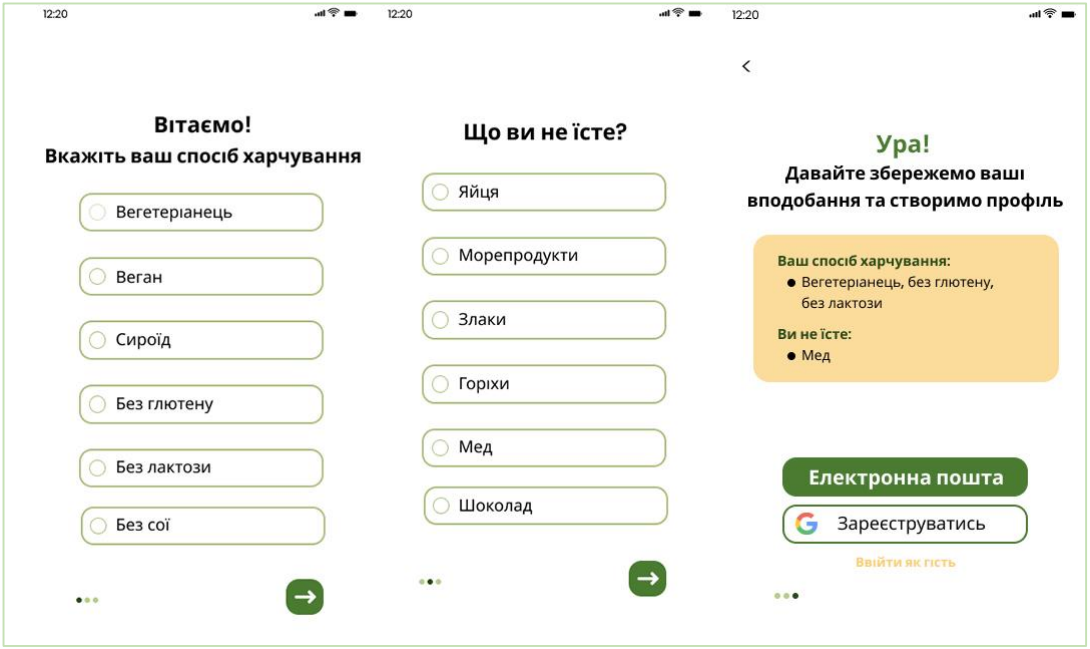

Рисунок 2.10 – Послідовність екранів для заповнення даних

Дана послідовність була розроблена з метою отримати всі потрібні дані для подальшого аналізу сканованих продуктів харчування під конкретного користувача. Кожен користувач, за умови, що він ще не зареєстрований, незалежно від того, чи він буде проходити етап реєстрації чи увійде як гість має обов'язково пройти це «опитування». Дизайн екранів був розроблений з урахуванням психології людини та того, яку траєкторію проходять очі користувача. Як видно на рисунку 2.10 користувач має вибрати, за допомогою чек-боксів, свої вподобання в їжі, які буду занесені в базу даних як preferences. Останнім екраном в цій послідовності є екран, що підсумовує все вибране та пропонує користувача зареєструватись двома способами: використовуючи електронну пошту, або ж за допомогою Google. Також, якщо користувач не хоче реєструватись, то він може увійти як гість.

Якщо користувач обрав опцію «Зареєструватись, використовуючи електронну пошту», то наступним екраном він побачить цей, що зображений на рисунку 2.11.

Рисунок 2.11 – Екран реєстрації через електронну пошту

Користувач повинен заповнити стандартні дані, що потрібні для реєстрації, а саме вказати електронну пошту та пароль (див. рис. 2.12). Після чого натиснути на кнопку «Зареєструватись». Після натиску на кнопку користувач потрапляє на Головний екран додатку. Якщо ж користувач обрав опцію «Увійти як гість», то він одразу потрапляє на головний екран.

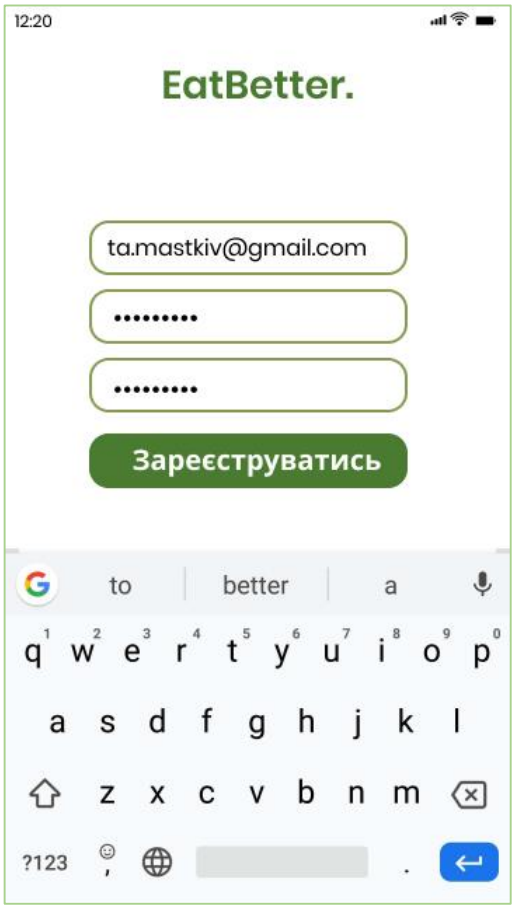

Рисунок 2.12 – Екран заповнення інформації для реєстрації

Після того як реєстрація була успішно здійснена після першого запуску додатку, всі наступні рази користувач буде мати змогу авторизуватись. Екран авторизації з полями «Електронна пошта» та «Пароль» зображено на рисунку 2.13. Користувач має можливість скористатись функцією авторизації через Google, а також функцією «Забули пароль?», яка допоможе відновити його, використовуючи електронну пошту.

Рисунок 2.13 – Екран авторизації

Головний екран додатку – це те, що користувач буде бачити більшість часу, тому головний екран має бути з ненав'язливим дизайном, що відповідає загальному стилю, проте цікавим. На рисунку 2.14 зображено головний екран додатку «EatBetter.». На ньому є панель з трьома основними кнопками: «Сканувати», «Профіль» та «Головна». Також головний екран містить два фрейми:

– перший фрейм «Рекомендовано для вас» передбачає показ продуктів, які рекомендовані до перегляду користувачам. Даний фрейм має горизонтальне прокручування, одночасно видно тільки два продукти;
– другий фрейм «Просканові товари» - фрейм з історією усіх товарів, що сканувались раніше. Містить назву товару на кнопку для перегляду інформації про товар. Фрейм має вертикальну прокрутку, одночасно видно три недавні продукти.

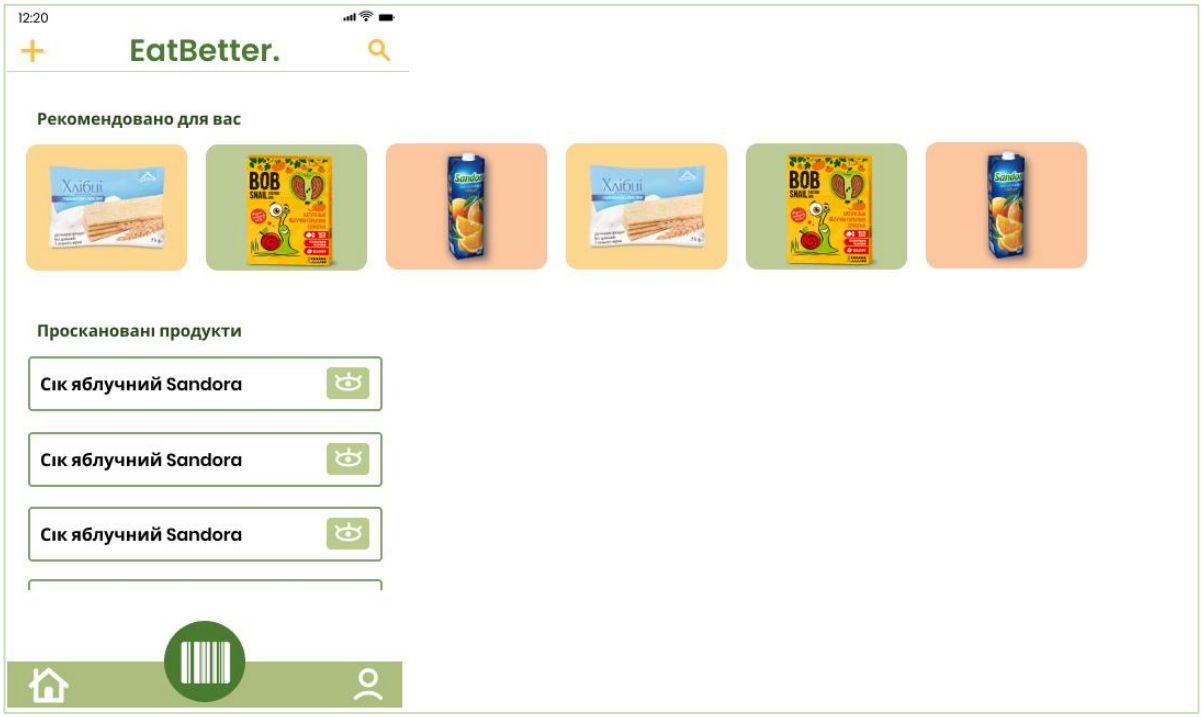

Рисунок 2.14 – Головний екран додатку

Також на головному екрані відображено назву додатку «EatBetter.», а також кнопки для пошуку та додавання нового продукту.

Екран профілю містить в собі інформацію про користувача за допомогою якого користувач увійшов в додаток, а також налаштування вподобань в їжі. Вподобання оформлені окремим фреймом, який має вертикальне прокручування. Всі вподобання в їжі редагуються, видаляються та додаються нові. Видалити вибране вподобання можна за свайпом в будь-яку сторону (див. рис. 2.15).

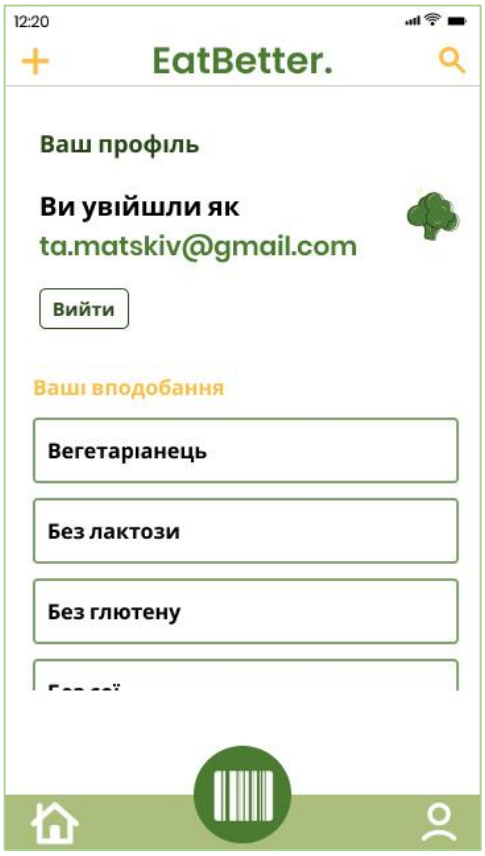

Рисунок 2.15 – Екран профілю користувача

Як видно з рисунку 2.16 додавання нового продукту не займає багато часу, оскільки інформації, яку потрібно ввести не є так багато, а також інтерфейс інтуїтивно зрозумілий. Для того, щоб перейти на екран додавання нового продукту користувачу потрібно всього лиш натиснути на «Плюс» на головному екрані. Також потрібно зробити фотографію упаковки продукту (передню та задню частини). ввести назву продукту, його виробника та ввести всі інгредієнти. Користувач також має можливість, за бажанням, сфотографувати склад. Після заповнення всіх даних користувачу потрібно натиснути на кнопку «Зберегти» і продукт буде автоматично доданий в базу даних, а користувач перенаправлений на головний екран. Користувач також має можливість закрити екран додавання нового продукту просто натиснувши на «Хрестик» в лівому верхньому куті.

Рисунок 2.16 – Екран додавання нового продукту в базу даних

Після сканування штрих-коду та ідентифікації його в базі користувач побачить спливаючий діалог про результат сканування. Даний діалог існує в двох варіаціях: перша – якщо продукт безпечний для вживання для конкретного користувача і друга – якщо продукт не рекомендований до вживання (див. рис. 2.17).

Рисунок 2.17 – Варіанти спливаючого діалогу після сканування продукту

На рисунку 2.18 зображено екран продукту, що є в базі. Цей екран появляється, якщо після сканування користувач натисне на кнопку «Дивитись». На екрані відображається фотографія продукту та основні характеристики: назва, штрих-код, виробник та інгредієнти.

Рисунок 2.18 – Екран існуючого продукту

Розроблений дизайну інтерфейсу дозволить користувачам вільно і без зайвого напруження користуватись додатком, що однозначно є важливим критерієм оцінки існуючих програмних продуктів.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ МОБІЛЬНОГО ЗАСТОСУНКУ

3.1 Конфігурація проекту

Хоча Android Studio забезпечує всім потрібним для ефективної розробки програмного продукту, середовище все одно потрібно налаштувати під певний проект. Також це потрібно для того, щоб і застосунок працював адекватно та без збоїв.

Першим кроком потрібно створити порожній проект. Для цього ми вибираємо порожню активність, переходимо на наступну сторінку, вказуємо назву поточного проекту, ім’я пакету, мову розробки і мінімальну потрібну версію SDK (див. рис. 3.1). Від вибору SDK залежить які пристрої будуть підтримувати розроблюваний додаток і з якими версіями операційної системи Android він буде сумісний.

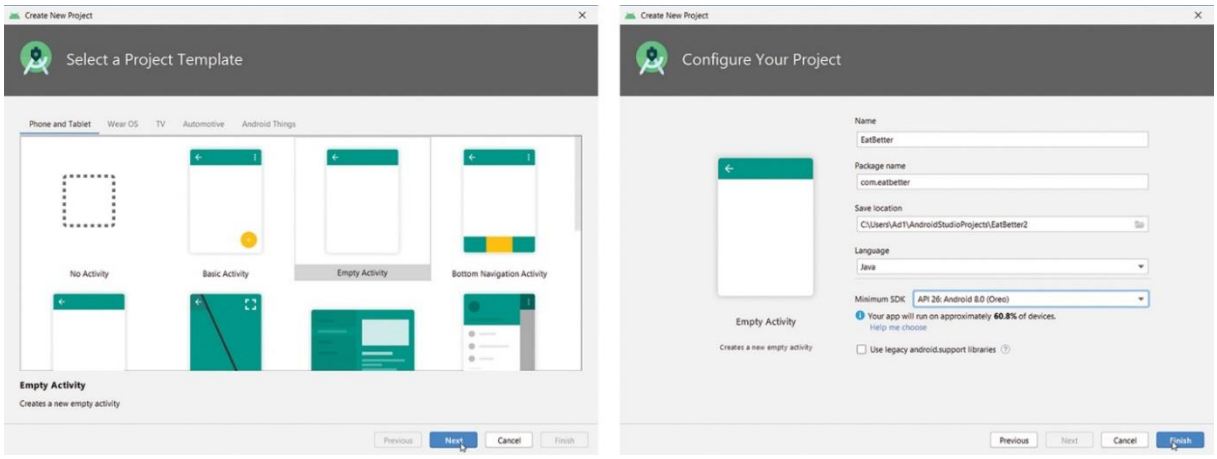

Рисунок 3.1 – Створення нового проекту та вибір конфігурацій

Наступним важливим кроком є налаштування файлу AndroidManifest.xml. Основними налаштуваннями є:

– дозволи, які потрібні для коректної роботи додатку (наприклад, доступ до мережі Інтернет, робота камери, зчитування та запис файлів) (див. рис. 3.2 (а));

– активність, яка буде запускатись при старті проекту (див. рис. 3.2 (б)).

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.eatbetter">

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission-sdk-23 android:name="android.permission.INTERNET" />

    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission-sdk-23 android:name="android.permission.CAMERA" />

    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

а

```
<activity android:name=".auth.StartActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

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

б

Рисунок 3.2 – Налаштування файлу AndroidManifest.xml

Один проект може містити декілька модулів і часто є так, що кожен модуль повинен використовувати різні бібліотеки для реалізації функцій. Проект «EatBetter.» має тільки один модуль. Далі потрібно прописати усі бібліотеки, що стосуються виключно цього модуля та що будуть використовуватись надалі, у файл build.gradle(module: app). Це робиться для того, щоб додаток працював правильно.

На рисунку 3.3 зображено весь список бібліотек, що використовуються для реалізації програмного забезпечення. Опишемо основні з них:

– ViewModel – бібліотека для реалізація шаблону MVVM;
– Livedata використовується для того, щоб у ViewModel завжди була актуальна інформація про дані;
– Navigation component – бібліотека, що забезпечує обробку старту активіті та перехід по фрагментах;

– Google vision services – основна бібліотека, яка надає можливість використовувати функції для виявлення багатьох типів 2D та 3D штрих-кодів;
– ButterKnife спрощує роботу з ініціалізацією елементів;
– Glide використовується для спрощення роботи з фотографіями та медіа-файлами;
– Сервіси Firebase:
    – Auth – бібліотека для авторизації;
    – UI Auth використовується для з'єднання Firebase Auth з такими способами авторизації як через Google, Facebook та інші;
    – Firestore – хмарна база даних NoSQL;
    – Storage – середовище (своєрідна база даних) для збереження фотографій.

```
dependencies {
    def lifecycle_version = "2.2.0"
    def nav_version = "2.3.0-alpha06"
    implementation "androidx.lifecycle:lifecycle-viewmodel:$lifecycle_version"
    implementation "androidx.lifecycle:lifecycle-livedata:$lifecycle_version"
    implementation 'androidx.legacy:legacy-support-v4:1.0.0'
    implementation 'androidx.lifecycle:lifecycle-extensions:2.2.0'
    //noinspection LifecycleAnnotationProcessorWithJava8
    annotationProcessor "androidx.lifecycle:lifecycle-compiler:$lifecycle_version"

    //noinspection AnnotationProcessorOnCompilePath
    implementation 'com.jakewharton:butterknife:10.1.0'
    annotationProcessor 'com.jakewharton:butterknife-compiler:10.0.0'
    implementation 'com.github.bumptech.glide:glide:4.11.0'
    annotationProcessor 'com.github.bumptech.glide:compiler:4.11.0'

    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'com.google.android.gms:play-services-vision:20.0.0'
    implementation 'com.google.android.material:material:1.1.0'
    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
    implementation 'androidx.cardview:cardview:1.0.0'
    implementation 'androidx.appcompat:appcompat:1.1.0'
    implementation "androidx.navigation:navigation-fragment:$nav_version"
    implementation "androidx.navigation:navigation-ui:$nav_version"

    implementation 'com.google.firebase:firebase-analytics:17.4.1'
    implementation 'com.google.firebase:firebase-core:17.4.1'
    implementation 'com.google.firebase:firebase-auth:19.3.1'
    implementation 'com.firebaseui:firebase-ui-auth:6.2.1'
    implementation 'com.google.firebase:firebase-firestore:21.4.3'
    implementation 'com.google.firebase:firebase-storage:19.1.1'
    implementation 'com.firebaseui:firebase-ui-firestore:6.2.1'
    implementation 'com.google.android.gms:play-services-auth:18.0.0'
```

Рисунок 3.3 – Список всіх бібліотек, що використовуються

3.2 Структура проекту

Для того, щоб було легко та зручно працювати на розробкою проекту, створювати новий функціонал та розширювати наявний потрібно коректно організувати структуру проекту.

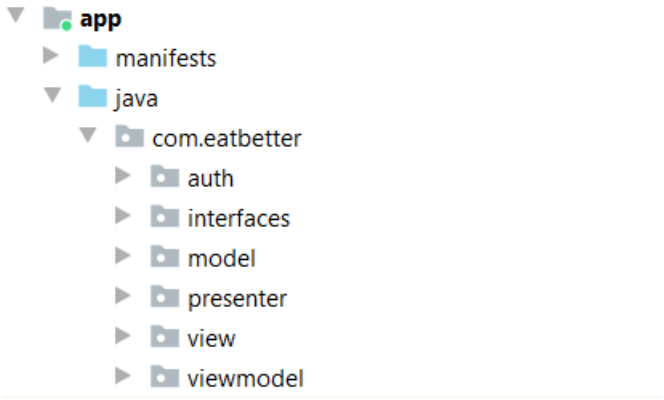

Рисунок 3.4 – Структура проекту в середовищі Android Studio

Як видно з рисунку 3.4, в проекті є 6 основних пакетів:

– Auth – пакет, в якому знаходяться всі вигляди, що стосуються реєстрації користувачів;
– Interfaces – усі інтерфейси, що використовуються для виклику презентерами у класах View;
– Model – пакет з моделями для роботи з базою даних, а також репозиторії для ViewModel;
– Presenter – тут знаходяться усі презентери;
– View – в цьому пакеті знаходяться усі вигляди, окрім вигляду реєстрації, що знаходиться у пакеті auth;
– ViewModel – оскільки у додатку буде частково реалізована livedata, то деякі сторінки будуть реалізовані саме за допомогою вмісту цього пакету.

3.3 Основний функціонал додатку

3.3.1 Активіті, фрагменти та навігація між ними

Додаток «EatBetter.» має всього дві активіті і все інше – це фрагменти. Одна активіті для початку роботи додатку та реєстрації користувача, а друга відповідає за весь інший функціонал. За перехід (навігацію) між цими компонентами відповідає Navigation Component.

Для того, щоб створити цей компонент потрібно у пакеті res створити пакет navigation (див. рис. 3.5), який власне і буде зберігати в собі файли для навігації з усіма даними (напрямами та аргументами).

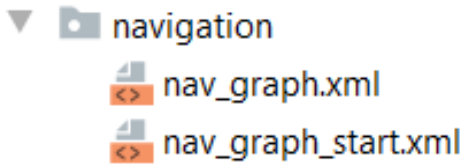

Рисунок 3.5 – Створений пакет navigation

Документ nav_graph.xml створений для навігації користувачів, що вже авторизовані, а документ nav_graph_start.xml – для користувачів, що ще не авторизовані. Сам файл графу навігаціє виглядає так як зображено на рисунку 3.6. Саме в цьому середовищі ми додаємо всі потрібні фрагменти, між якими власне і буде відбуватись навігація.

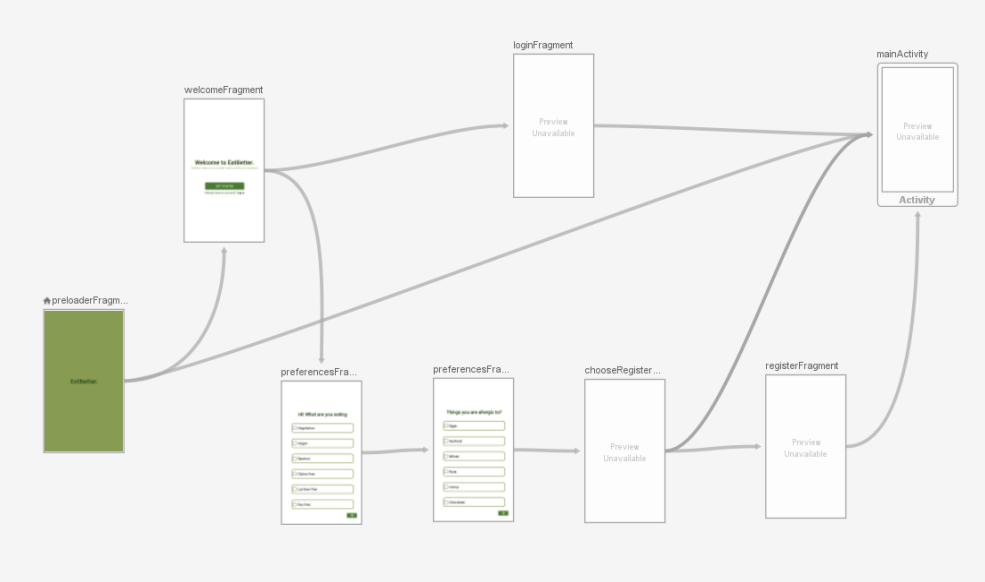

Рисунок 3.6 – Вигляд графу навігації

В активіті, що відповідає за навігацію нічого не потрібно прописувати, тому що цю всю роботу на себе бере Navigation Component. Все, що потрібно зробити – це в layout файлі поточної активіті прописати фрагмент, що вказує на граф навігації (див. рис. 3.7).

```
<fragment
    android:id="@+id/fragment_container_start"
    android:name="androidx.navigation.fragment.NavHostFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_marginBottom="0dp"
    app:defaultNavHost="true"
    app:navGraph="@navigation/nav_graph_start" />
```

Рисунок 3.7 – Layout файл з прописаним фрагментом, що вказує на граф навігації

Контролер, що відповідає за навігацію ініціалізується всього лише одним рядком коду (див. рис. 3.8).

```
NavController navController = Navigation.findNavController(view);
```

Рисунок 3.8 – Ініціалізація контролера навігації

Для того, щоб зв’язати сторінки (фрагменти) між собою і в подальшому могти переходити між ними потрібно реалізувати один метод (див. рис. 3.9), де Id – це назва стрілки у файлі навігації.

```
public void onLogInButtonClick() {
    navController.navigate(R.id.action_welcomeFragment_to_loginFragment);
}
```

Рисунок 3.9 – Реалізація метод зв’язування фрагментів між собою

Також через Navigation Component можна передавати не тільки фрагменти, а і дані у вигляді аргументів. Для реалізації цієї функції потрібно вибрати екран destination (виділено синім на рисунку 3.10), і справа ввести тип даних, що має отримувати даний фрагмент.

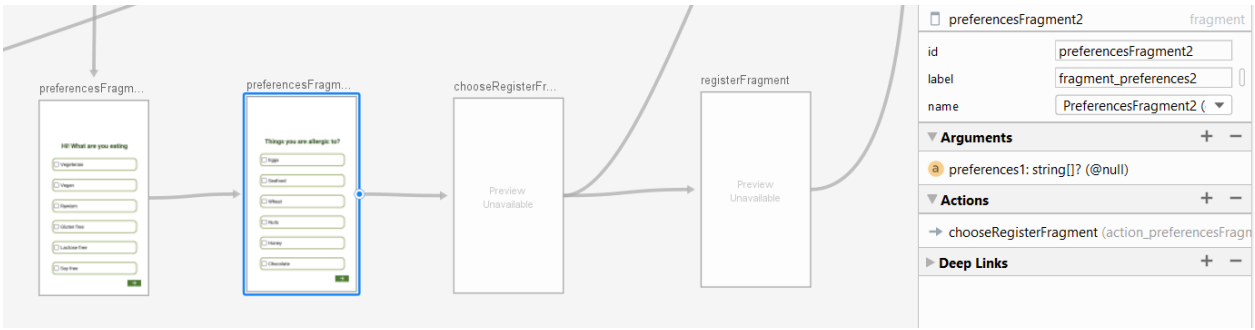

Рисунок 3.10 – Приклад задання передавання аргументів за допомогою Navigation Component

На рисунку вище зображено передавання даних на PreferencesFragment_2 масив String з попереднього екрану Для того, щоб передати дані з одного фрагменту в інший потрібно викликати автозгенерований клас, що буде мати назву View класу + Directions, після чого викликати метод set + назву аргументу (див. рис. 3.11). Для того, щоб це виконалось успішно потрібну ще раз збілдити проект після встановлення всіх напрямків і задання аргументів.

```
@Override
public void passPreferencesToNextPage(String... preferences) {
    PreferencesFragment1Directions.ActionPreferencesFragment1ToPreferencesFragment2 action =
            PreferencesFragment1Directions.actionPreferencesFragment1ToPreferencesFragment2();
    action.setPreferences1(preferences);
    navController.navigate(action);
}
```

Рисунок 3.11 – Метод для передачі аргументів

А для того, щоб певні дані отримати потрібно викликати клас, в якому безпосередньо знаходиться аргумент + Args (див. рис. 3.12).

```
@Override
public void passPreferencesToNextPage(String... preferences) {
    String[] preferences1 = PreferencesFragment2Args.fromBundle(getArguments()).getPreferences1();
```

Рисунок 3.12 – Метод для отримання аргументів

Таким чином, як показано вище, побудована уся навігація і передача даних з одного фрагмента в інший у додатку, що розробляється.

Також в даному підпункті буде доцільно згадати про бібліотеку, яка використовується для полегшення реалізації ініціалізації елементів – ButterKnife. Дякуючи цій бібліотеці, нам не потрібно самому реалізовувати деякі методи, продумувати логіку для них. Наприклад, findViewById і setOnClickListener. Проте, обов'язковим кроком для успішного використання усіх можливостей цієї бібліотеки потрібно виконати команду bind, щоб ініціалізувати всі елементи (див. рис. 3.13). Для цього достатньо написати анотацію @BindView для оголошення елементів і @OnClick для ініціалізації лістенера (див. рис. 3.14).

```
ButterKnife.bind( target: this, view);
```

Рисунок 3.13 – Ініціалізація елементів бібліотеки

```
@BindView(R.id.fieldEmailRegister)
EditText edtEmail;

/unused/
@OnClick(R.id.emailCreateAccountButton)
public void onCreateButtonClick() {
    mPresenter.trySignUpWithEmail(edtEmail.getText().toString(), edtPassword.getText().toString(),
            edtConfirmPassword.getText().toString(), RegisterFragmentArgs.fromBundle(getArguments()).getPreferencesAll());
}
```

Рисунок 3.14 – Анотації для оголошення та ініціалізації

3.3.2 Початок роботи

Як тільки користувач запускає додаток з робочого стола мобільного пристрою з'являється екран попереднього завантаження, презентер якого перевіряє чи даний користувач авторизований чи ні. Якщо користувач авторизований, то презенетер «перенаправляє» користувач на екран home (див. рис. 3.15). Якщо ж користувач не авторизований – то на початкову сторінку.

```
public void init() {
    if (firebaseAuth.getCurrentUser() != null)
        mAuthStartView.openHomeFragment();
    else
        mAuthStartView.openWelcomeFragment();
}
```

Рисунок 3.15 – Метод, що перевіряє чи авторизований користувач

Якщо користувач не авторизований, йому буде запропоновано авторизуватись, якщо він вже має створений аккаунт, або ж пройти «опитування» для того, щоб зареєструватись. App flow з «опитування» буде активований для кожного користувача, який не зареєстрований і це його перший раз користування додатком.

Отже, після натиску користувачем на кнопку «Почати» йому один за одним з'являться два екрани персоналізованих налаштувань. На першому екрані користувач повинен відзначити чекбокси, які будуть відображати його спосіб

харчування, а на другому – алергени, або ж винятки в їжі. Є можливість вибрати декілька варіантів – мультичекбокс.

Після налаштування всіх вподобань в їжі користувач натисне кнопку «далі», яка в інтерфейсі відображена як стрілка. В цей самий час всі дані, що користувач відмітив передаються у вигляді значень типу int вибраних чекбоксів презентеру (див. рис. 3.16).

```
/unused/
@OnClick(R.id.continueIngredientsButton)
public void onContinueButtonClick() {
    for(int i = 0; i<prefCheckboxes.size(); i++){
        if(prefCheckboxes.get(i).isChecked()){
            preferencesSelected.add(i);
        }
    }
    prefPresenter.passSelectedPreferences(preferencesSelected);
}
```

Рисунок 3.16 – Реалізований метод передавання значень, вибраних користувачем

Ці дані отримує PreferencesPresenter, після чого він витягує їх значення з enum по індексу і передає далі до сторінки реєстрації через метод інтерфейсу PreferencesView – passPreferencesToNextPage. Цей метод реалізований в самому фрагменті і передає через аргументи навігації масив String до наступної сторінки (див. рис. 3.17).

```
public class PreferencesPresenter {

    private final PreferencesView mPreferencesView;
    private List<String> preferencesList;

    public PreferencesPresenter(@NonNull PreferencesView preferencesView) {
        mPreferencesView = preferencesView;
        preferencesList = new ArrayList<>();
    }

    public void passSelectedPreferences() { mPreferencesView.passPreferencesToNextPage(); }

    public void passSelectedPreferences(List<Integer> checkedResult) {
        for(Integer checked : checkedResult){
            preferencesList.addAll(Arrays.asList(PreferencesList.values()[checked].getPreferences()));
        }
        mPreferencesView.passPreferencesToNextPage(preferencesList.toArray(new String[0]));
    }
}
```

Рисунок 3.17 – Реалізований метод передачі даних на наступну сторінку

3.3.3 Реєстрація та авторизація

У додатку «EatBetter.» користувач буде мати можливість зареєструватись трьома способами:

– використовуючи електронну пошту;

– за допомогою Google аккаунту;

– гість (анонімний користувач).

Ці всі реалізовані способи реєстрації надає нам Firebase. Точніше сказати цей сервіс надає дуже багато варіантів реєстрації (див. рис. 3.18), але в даному програмному продукті буде використовуватись тільки три.

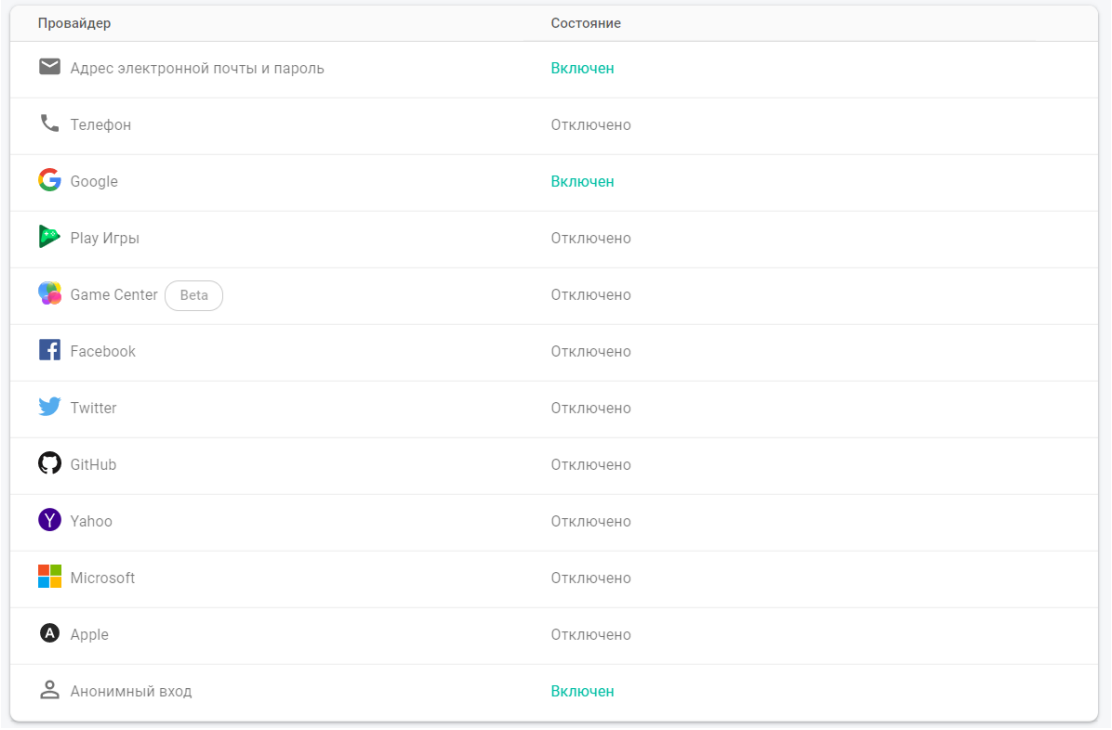

Рисунок 3.18 – Всі можливі способи реєстрації, представлені Firebase

В залежності від того, який спосіб реєстрації вибере користувач викликається один з трьох методів (див. рис. 3.19):

1) реєстрація за допомогою Google аккаунту: запускається активіті Google, в якій користувач повинен обрати за допомогою якого аккаунту він зареєструється в додатку «EatBetter.». Якщо реєстрація пройде успішно, то викликається метод з ChooseRegisterPresenter trySignUpWithGoogleAccount, одним з параметрів якого буде токен;
2) реєстрація, використовуючи електронну пошту: користувач перенаправляється на екран реєстрації через електронну пошту з переданими значеннями, які були обрані раніше на екранах налаштувань;
3) гість (анонімний користувач): викликається метод trySignUpAnonymously.

Перший і третій методи приймають масив вибраних користувачем налаштувань як аргумент.

```
/unused/
@OnClick(R.id.to_email_register_btn)
public void onEmailButtonClick() { prefPresenter.passSelectedPreferences(); }

/unused/
@OnClick(R.id.image_google_button)
public void onGoogleButtonClick() {
    GoogleSignInOptions gso = new GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)
            .requestIdToken("312810911093-a5glka1ns08ep19c9on2uplr9oqqf4k6.apps.goog...")
            .requestEmail()
            .build();
    mGoogleSignInClient = GoogleSignIn.getClient(getActivity(), gso);
    Intent signInIntent = mGoogleSignInClient.getSignInIntent();
    startActivityForResult(signInIntent, RC_SIGN_IN);
}

/unused/
@OnClick(R.id.continue_as_a_guest)
public void onContinueAsGuestClick() {
    chooseRegisterPresenter.trySignUpAnonymously(ChooseRegisterFragmentArgs.fromBundle(getArguments()).getPreferences2());
}
```

Рисунок 3.19 – Три методи реєстрації користувача

Варто зазначити, що Firebase зручний тим, що багато роботи він виконує за розробників. Наприклад, для того, щоб отримати користувача, який зараз авторизований достатньо просто викликати метод getInstance() класу FirebaseAuth і після цього викликати .getCurrentUser(). Ми отримаємо об’єкт типу FirebaseUser, з якого далі буде можливість взяти всі необхідні нам дані. Дані, які потрібні: email та uid (унікальний ідентифікатор користувача). Таким же самим чином ініціалізується екземпляр бази даних – для того, щоб звернутись до колекції, потрібно просто вказати її назву (див. рис. 3.20).

```
public ChooseRegisterPresenter(@NonNull RegisterView registerView) {
    mRegisterView = registerView;
    mAuth = FirebaseAuth.getInstance();
    userRef = FirebaseFirestore.getInstance().collection( collectionPath: "Users");
}
```

Рисунок 3.20 – Ініціалізація презентера (екземпляри авторизації та колекції користувача)

Отже, якщо користувач успішно пройшов реєстрацію і був зареєстрований, створюється об’єкт класу User, з даними, які ми отримали (uid, email та preferences) (див. рис. 3.21).

```
public class User {

    private String documentId;
    private String userId;
    private String email;
    private Map<String, Boolean> preferences;

    public User() {

    }

    public User(String userId, Map<String, Boolean> preferences) {
        this.userId = userId;
        this.preferences = preferences;
    }

    public User(String userId, String email, Map<String, Boolean> preferences) {
        this.userId = userId;
        this.email = email;
        this.preferences = preferences;
    }
```

Рисунок 3.21 – Клас User

Преференси, зазначені користуваче, подаються у вигляді Map<String, boolean>, де String – це назва складника, а boolean – значення true. Цей спосіб є одним з найзручніших для організації взаємодії з базою даних Firestore. Далі викликається метод add референсу колекції та поміщаємо туди об’єкт користувача. Після успішного створення користувача в базі даних викликається метод інтерфейсу signUpGoogleSuccessful, який перенаправляє користувача за допомогою Navigation Component на головний екран.

```
public void trySignUpWithGoogleAccount(String idToken, String... preferencesArray) {
    AuthCredential credential = GoogleAuthProvider.getCredential(idToken, s1: null);
    mAuth.signInWithCredential(credential)
            .addOnCompleteListener(new OnCompleteListener<AuthResult>() {
                @Override
                public void onComplete(@NonNull Task<AuthResult> task) {
                    if (task.isSuccessful()) {
                        Log.d(TAG, msg: "signInWithGoogleCredential:success");
                        Map<String, Boolean> preferencesMap = getPreferencesMap(preferencesArray);
                        User user = new User(mAuth.getCurrentUser().getUid(), mAuth.getCurrentUser().getEmail(), preferencesMap);
                        userRef.add(user)
                                .addOnCompleteListener(new OnCompleteListener<DocumentReference>() {
                                    @Override
                                    public void onComplete(@NonNull Task<DocumentReference> task) {
                                        if (task.isSuccessful()) {
                                            mRegisterView.signUpGoogleSuccessful();
                                        } else {
                                            Log.w(TAG, msg: "addUserToFirestore:failure", task.getException());
                                            mRegisterView.addUserToDatabaseNotSuccessful();
                                        }
                                    }
                                });
                    } else {
                        Log.w(TAG, msg: "googleSignIn:failure", task.getException());
                        mRegisterView.signUpGoogleNotSuccessful();
                    }
                }
            });
}
```

Рисунок 3.22 – Реалізація реєстрації через Google аккаунт

Реєстрація анонімного користувача відбувається точно так само, як і через Google аккаунт (див. рис. 3.22), тільки в цьому випадку викликається метод signInAnonymously() класу FirebaseAuth і для додавання користувача в базу даних використовується конструктор з двома параметрами – uid та preferences.

Реєстрація, використовуючи електронну пошту теж реалізується як через Google аккаунт, проте замість токена передається значення електронної пошти і пароль, що зчитується з полів, які заповнюються користувачем. Після цього викликається метод createUserWithEmailAndPassword (див. рис. 3.23).

```
firebaseAuth.createUserWithEmailAndPassword(email, password)
```

Рисунок 3.23 – Метод створення користувача через електронну пошту

Оскільки, логіка авторизації теж реалізована за допомогою Firebase, то вона схожа до методу реєстрації, відмінність – не створюється документ користувача в колекції користувачів, оскільки він вже там є (див. рис. 3.24).

```
firebaseAuth.signInWithEmailAndPassword(email, password)
```

Рисунок 3.24 – Метод авторизації користувача

3.3.4 Сканер

Для створення сканеру використовується бібліотека Google Vision. Для того, щоб ініціалізувати роботу камери потрібно ініціалізувати детектор і джерело камери. За допомогою методу setBarcodeFormats вибираються потрібні формати штрих-кодів, і оскільки для реалізації основного функціоналу потрібен тільки штрих-код продуктового товару ми обираємо EAN_13 (див. рис. 3.25).

```
BarcodeDetector detector = new BarcodeDetector.Builder(getContext()).setBarcodeFormats(Barcode.EAN_13).build();
detector.setProcessor(new Detector.Processor<Barcode>() {
```

Рисунок 3.25 – Метод вибору формату штрих-коду

Оскільки, нам потрібно, щоб камера була наче вбудована в застосунок ми поміщаємо джерело камери у SurfaceView та ініціалізуємо її (див. рис. 3.26).

```
cameraSource = new CameraSource.Builder(getContext(), detector).setRequestedPreviewSize( i: 1024,  i1: 768)
        .setRequestedFps(25f).setAutoFocusEnabled(true).build();
svBarcode.getHolder().addCallback(new SurfaceHolder.Callback2() {
    @Override
    public void surfaceRedrawNeeded(SurfaceHolder holder) {

    }

    @Override
    public void surfaceCreated(SurfaceHolder holder) {
        if (ContextCompat.checkSelfPermission(getActivity(),
                Manifest.permission.CAMERA) == PackageManager.PERMISSION_GRANTED) {
            try {
                cameraSource.start(holder);
            } catch (IOException e) {
                e.printStackTrace();
            }
        } else {
            ActivityCompat.requestPermissions(getActivity(),
                    new String[]{Manifest.permission.CAMERA}, CAMERA_REQUEST_CODE);
        }
    }
```

Рисунок 3.26 – Метод ініціалізації камери

Також важливим моментом є – запит користувача на дозвіл користуванням камерою перед початком роботи самої камери. Відповідно, від вибору користувача залежить чи буде запущена камера чи ні. Припустимо, що користувач дав дозвіл на використання камери. Тоді ми відкриваємо її і наводимо на штрих-код продукту, коли детектор розрізняє штрих-код – запускається завдання в потоці ui. Враховуючи те, що Firebase усі запити виконує в окремих асинхронних потоках, то немає сенсу самостійно ініціалізовувати окреми потік, тому все виконується в ui потоці. Після визначення штрих-коду камера зупиняється і викликається метод презентера tryToVerifyProduct, в який ми передаємо значення штрих-коду як аргумент (див. рис. 3.27).

```
@Override
public void receiveDetections(Detector.Detections<Barcode> detections) {
    final SparseArray<Barcode> barcodes = detections.getDetectedItems();
    if (barcodes != null && barcodes.size() > 0) {
        view.post(new Runnable() {
            @Override
            public void run() {
                cameraSource.stop();
                mPresenter.tryToVerifyProduct(barcodes.valueAt( index: 0).displayValue);
            }
        });
    }
}
```

Рисунок 3.27 – Метод, що визначає отриманий штрих-код

У даному презентері обов’язковим моментом є ініціалізація колекції користувачів та продуктів, оскільки буде виконуватись порівняння преференсів користувача зі складниками продуктового товару. Також тут потрібний FirebaseAuth для того, щоб отримати id користувача, що є авторизованим на момент перевірки (див. рис. 3.28).

```
public ScannerPresenter(ScannerView scannerView) {
    mAuth = FirebaseAuth.getInstance();
    FirebaseFirestore db = FirebaseFirestore.getInstance();
    mScannerView = scannerView;
    productRef = db.collection( collectionPath: "Products");
    userRef = db.collection( collectionPath: "Users");
}
```

Рисунок 3.28 – Реалізований метод презентера сканера

Перевірка вхідних даних відбувається наступним чином: спочатку витягується продукт, в якого поле «штрих-код» співпадає зі значенням, яке було проскановане. Якщо змінна task менше одиниці (що означає, що продукт відсутній в базі даних), то презентер перенаправляє користувача на сторінку створення нового продукту, з попередньо заповненим полем штрих-коду (див. рис. 3.29).

```
public void tryToVerifyProduct(String barcode) {
    mScannerView.showLoading();
    productRef.whereEqualTo( field: "barcode", barcode)
            .limit(1)
            .get()
            .addOnCompleteListener(new OnCompleteListener<QuerySnapshot>() {
                @Override
                public void onComplete(@NonNull Task<QuerySnapshot> task) {
                    if (task.isSuccessful()) {
                        if (task.getResult().getDocuments().size() < 1) {
                            mScannerView.onProductNotExist(barcode);
                        } else {
                            String productDocumentId = task.getResult().getDocuments().get(0).getId();
                            Product product = task.getResult().getDocuments().get(0).toObject(Product.class);
                            userRef.whereEqualTo( field: "userId", mAuth.getCurrentUser().getUid())
                                    .get()
                                    .addOnCompleteListener(new OnCompleteListener<QuerySnapshot>() {
                                        @Override
                                        public void onComplete(@NonNull Task<QuerySnapshot> task) {
                                            if (task.isSuccessful()) {
                                                if (task.getResult().getDocuments().size() > 0) {
                                                    String userDocumentId = task.getResult().getDocuments().get(0).getId();
                                                    userRef.document(userDocumentId).collection( collectionPath: "history")
                                                            .document(productDocumentId).set(product, SetOptions.merge());
                                                    boolean result = findMatch(product, task.getResult().getDocuments().get(0).toObject(User.class));
                                                    if (result) {
                                                        mScannerView.onProductContains(product);
                                                        mScannerView.hideLoading();
                                                    } else {
                                                        mScannerView.onProductNotContains(product);
                                                        mScannerView.hideLoading();
                                                    }
```

Рисунок 3.30 – Реалізований метод верифікації продукту

Якщо продукт є в базі, то наступним кроком є витягнення користувача по uid з бази даних, потім відбувається порівняння його преференсів зі складниками

просканованого продукту. В цей самий час даний продуктовий товар поміщається в підколекцію history документа User, після чого використовується команда merge. Це виконується для того, щоб уникнути повтору в історії, наприклад якщо продукт вже є в цій колекції, то він так і залишається там, а якщо він змінений, то дані будуть оновлені.

Далі ми порівнюємо результат сканування методом findMatch, який приймає як аргументи продукт і об'єкти користувача та повертає результат типу boolean, який у свою чергу передається у спливаюче діалогове вікно (див. рис. 3.30).

```
private boolean findMatch(Product product, User user) {
    return product.getIngredients().entrySet().stream()
            .anyMatch(e -> e.getValue().equals(user.getPreferences().get(e.getKey())));
}
```

Рисунок 3.30 – Метод порівняння отриманого результату з даними в базі даних

У результаті ми маємо наступні опції: якщо преференси містять хоча б один складник, то виводиться як результат червоне спливаюче вікно з назвою продукту та його штрих-кодом. Якщо ж немає ні одного складника, який користувач попередньо зазначив як виняток, то спливаюче вікно буде зеленим. Також у спливаючому діалоговому вікні для користувача надається можливість перейти на сторінку продукту, натиснувши на кнопку «Дивитись», або ж продовжити далі сканувати продукти – «Продовжити сканування», або користувач має натиснути будь-де за межами спливаючого вікна, щоб вийти (див. рис. 3.31). Якщо користувач обрав опцію «Продовжити сканування» або ж натиснув будь-де за межами вікна додаток автоматично відновлює джерело камери і вона знову готова до сканування штрих-код (див. рис. 3.32).

```
@NonNull
@Override
public Dialog onCreateDialog(@Nullable Bundle savedInstanceState) {
    AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());

    LayoutInflater layoutInflater = getActivity().getLayoutInflater();
    View view = layoutInflater.inflate(R.layout.comparison_result_popup,  root: null);

    builder.setView(view)
            .setTitle("Scanner result")
            .setNegativeButton( text: "Continue", new DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog, int which) {
                    try {
                        listener.onContinueClicked();
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                }
            })
            .setPositiveButton( text: "View product", new DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog, int which) {
                    listener.onViewProductClick(product);
                }
            });
```

Рисунок 3.31 – Метод виклика та генерації спливаючого діалогового вікна

```
@Override
public void onContinueClicked() throws IOException {
    cameraSource.start(svBarcode.getHolder());
}
```

Рисунок 3.32 – Відновлення джерела камери після дії користувача

3.3.5 Преференси користувача та історія

В цьому підпункті ми розберемо роботу списку преференсів користувача та історію, оскільки вони реалізовані використовуючи RecyclerView разом з LiveData і ViewModel.

Для початку потрібно створити репозиторій, який буде надавати оновлені актуальні дані і який буде виконувати команду додавання преференсів. При створенні нового об’єкту репозиторія відразу потрібно ініціалізувати FirebaseAuth і колекцію стандартним чином, але відразу витягується userId, з якого отримується id самого документа, щоб в інших методах була можливість звертатись одразу безпосередньо до документа поточного користувача.

При створенні нового об’єкту UserPreferencesViewModel одразу ініціалізовується створений раніше репозиторій, і коли в ньому ідентифікатор документа буде присвоєно змінній – викликається метод інтерфейсу documentIdAssigned, який в свою чергу витягує з репозиторія усі преференси, що присвоєні певному користувачу (див. рис. 3.33). Метод, шо присвоює ідентифікатор onFirestoreTaskComplete – це інтерфейс, що оголошений в репозиторії, і який реалізуватиме ViewModel (див. рис. 3.34).

```
private void assignUserDocId() {
    usersRef.whereEqualTo( field: "userId", mAuth.getUid())
            .limit(1)
            .get()
            .addOnCompleteListener(new OnCompleteListener<QuerySnapshot>() {
                @Override
                public void onComplete(@NonNull Task<QuerySnapshot> task) {
                    if (task.isSuccessful()) {
                        userDocumentId = task.getResult().getDocuments().get(0).getId();
                        onFirestoreTaskComplete.documentIdAssigned();
                    } else {
                        onFirestoreTaskComplete.onError(task.getException());
                    }
                }
            });
}
```

Рисунок 3.33 – Метод, що присвоює ідентифікатор документу

```java
public class UserPreferencesViewModel extends ViewModel implements UserPreferencesRepository.OnFirestoreUserTaskComplete {

    private MutableLiveData<List<String>> preferencesListModelData = new MutableLiveData<>();
    private UserPreferencesRepository db;

    public UserPreferencesViewModel() { db = new UserPreferencesRepository( onFirestoreTaskComplete: this); }

    public LiveData<List<String>> getAllPreferences() { return preferencesListModelData; }

    public void addPreference(String preference) { db.addPreference(preference); }

    public void deletePreference(String preference) { db.deletePreference(preference); }

    @Override
    public void preferenceListDataAdded(List<String> productListModelsList) {
        preferencesListModelData.setValue(productListModelsList);
    }

    @Override
    public void documentIdAssigned() { db.getUserPreferences(); }

    @Override
    public void onError(Exception e) {

    }
```

Рисунок 3.34 – Реалізована ViewModel

Як вже було вказано вище, ми звертаємось до документа напряму, використовуючи його ідентифікатор і разом з цим додаємо SnapshotListener. Суть даного лістенера в тому, що він завжди перевіряє і повертає актуальні (up-to-date) дані. Це означає, що як тільки в базі даних зміняться якісь дані, то лістенер одразу викликає метод preferenceListDataAdded, але вже з оновленими даними. Метод preferenceListDataAdded, в свою чергу, задає livedata полю preferencesListModelData нові значення (див. рис. 3.35).

```java
public void getUserPreferences() {
    usersRef.document(userDocumentId)
            .addSnapshotListener(new EventListener<DocumentSnapshot>() {
                @Override
                public void onEvent(@Nullable DocumentSnapshot documentSnapshot, @Nullable FirebaseFirestoreException e) {
                    if (e != null) {
                        onFirestoreTaskComplete.onError(e);
                    }
                    if (documentSnapshot.exists()) {
                        ArrayList<String> prefList = new ArrayList<>(documentSnapshot.toObject(User.class)
                                .getPreferences().keySet());
                        onFirestoreTaskComplete.preferenceListDataAdded(prefList);
                    }
```

Рисунок 3.35 – Метод лістенера SnapshotListener

Останнім кроком в реалізації RecyclerView для історії – створення у певному фрагменті ViewModel та створити адаптер з метою відображення даних

так, як це потребує RecyclerView. Для цього створюється layout файл, який буде відображати один пункт (див. рис. 3.36).

```
@NonNull
@Override
public PreferencesHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
    View itemView = LayoutInflater.from(parent.getContext())
            .inflate(R.layout.preference_item, parent,  attachToRoot: false);
    return new PreferencesHolder(itemView);
}
```

Рисунок 3.36 – Створення адаптера

Далі потрібно реалізувати: метод, який буде, власне, встановлювати преференс користувача; метод для визначення позиції преференса; метод, щоб отримувати кількість преференсів та клас ViewHolder (див. рис. 3.37).

```
@Override
public void onBindViewHolder(@NonNull PreferencesHolder holder, int position) {
    holder.textViewPreferenceName.setText(preferences.get(position));
}

@Override
public int getItemCount() { return preferences.size(); }

public void setPreferences(List<String> preferences) { this.preferences = preferences; }

public String getPreferenceAt(int position) { return preferences.get(position); }

class PreferencesHolder extends RecyclerView.ViewHolder {

    private TextView textViewPreferenceName;

    public PreferencesHolder(@NonNull View itemView) {
        super(itemView);
        textViewPreferenceName = itemView.findViewById(R.id.text_preference_name);
    }
}
```

Рисунок 3.37 – Три реалізовані методи

Після успішної реалізації методів, що були потрібні та створення класу – все, що потрібне для відображення преференсів готове, потрібно тільки використати це все у фрагменті (див. рис. 3.38). Ініціалізовуємо RecyclerView(listView) і присвоюємо йому адаптер. Після цього, коли активіті ініціалізована присвоюємо їй ViewModel (див. рис. 3.39).

```
listView.setLayoutManager(new LinearLayoutManager(getContext()));
listView.setHasFixedSize(true);

userPreferencesAdapter = new UserPreferencesAdapter();
listView.setAdapter(userPreferencesAdapter);
```

Рисунок 3.38 – Використання методів у фрагменті

```
@Override
public void onActivityCreated(@Nullable Bundle savedInstanceState) {
    super.onActivityCreated(savedInstanceState);

    userPreferencesViewModel = new ViewModelProvider(getActivity()).get(UserPreferencesViewModel.class);
    userPreferencesViewModel.getAllPreferences().observe(getViewLifecycleOwner(), new Observer<List<String>>() {
        @Override
        public void onChanged(List<String> preferences) {
            userPreferencesAdapter.setPreferences(preferences);
            userPreferencesAdapter.notifyDataSetChanged();
        }
    });
```

Рисунок 3.39 – Ініціалізація RecyclerView та присвоєння адаптера

Також буде реалізована функція видалення преференсу за допомогою свайпу, для цього буде використано ItemTouchHelper (див. рис. 3.40). Тут вказуються напрями через логічний оператор «|». Метод onMove (керує drag-and-drop actions) залишається незмінним, а метод onSwiped буде зміненим. Викликається метод з ViewModel і передається преференс елемента, що був свайпнутим і одночасно це прикріпляється до RecyclerView. У репозиторії витягується map, видаляється значення за ключем і оновлюється поле prefernces у базі даних (див. рис. 3.41).

```
new ItemTouchHelper(new ItemTouchHelper.SimpleCallback( dragDirs: 0,
        swipeDirs: ItemTouchHelper.LEFT | ItemTouchHelper.RIGHT) {
    @Override
    public boolean onMove(@NonNull RecyclerView recyclerView, @NonNull RecyclerView.ViewHolder viewHolder,
                          @NonNull RecyclerView.ViewHolder target) {
        return false;
    }

    @Override
    public void onSwiped(@NonNull RecyclerView.ViewHolder viewHolder, int direction) {
       userPreferencesViewModel.deletePreference(userPreferencesAdapter.getPreferenceAt(viewHolder.getAdapterPosition()));
    }
}).attachToRecyclerView(listView);
```

Рисунок 3.40 – Методи ItemTouchHelper

```
public void deletePreference(String preference) {
    usersRef.document(userDocumentId)
            .get()
            .addOnCompleteListener(new OnCompleteListener<DocumentSnapshot>() {
                @Override
                public void onComplete(@NonNull Task<DocumentSnapshot> task) {
                    if (task.isSuccessful()) {
                        DocumentReference reference = task.getResult().getReference();
                        Map<String, Boolean> preferences = task.getResult().toObject(User.class).getPreferences();
                        preferences.remove(preference);
                        reference.update( field: "preferences", preferences);
                    } else {
```

Рисунок 3.41 – Метод видалення преференса користувача

Таким же самим чином організована історія просканованих продуктових товарів, тільки витягується не поле документа користувача, а ціла підколекція history, яка в собі містить документи продуктів. Також адаптер відображає назву продукут і штрих-код.

3.3.6 Сторінка продукту і створення продукту

Сторінка продукту дуже просто організована, адже у Navigation Component передається об’єкт продукту, і все, що залишається це відобразити продукт на сторінці. Для відображення фотографії продукту використовується бібліотека Glide (див. рис. 3.42).

```
@Override
public void onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState) {
    super.onViewCreated(view, savedInstanceState);
    ButterKnife.bind( target: this, view);
    Product product = ProductFragmentArgs.fromBundle(getArguments()).getProduct();
    ArrayList<String> ingretients = new ArrayList<String>(product.getIngredients().keySet());

    productName.setText(product.getName());
    productBarcode.setText(product.getBarcode());
    productManufacturer.setText(product.getManufacturer());
    productIngredients.setText(String.join( delimiter: ", ", ingretients));

    Glide.with(getContext()) RequestManager
            .load(product.getImageLink()) RequestBuilder<Drawable>
            .centerCrop() RequestBuilder<Drawable>
            .placeholder(R.drawable.placeholder_image) RequestBuilder<Drawable>
            .into(productImage);
}
```

Рисунок 3.42 – Методи відображення даних продукту

Для того, щоб потрапити на сторінку створення продукту потрібна одна умова – продукт, що був просканований не виявлений в базі даних. Дана сторінка містить в собі аргумент barcode, що передається зі сторінки сканування, одразу ж заповнюючи поле штрих-коду і вказується, що це поле неможливо відредагувати. Це зроблено для того, щоб користувач не вводив самостійно штрих-код продукту і унеможливити допущення помилки при занесенні даних до бази даних (див. рис. 3.43).

```
navController = Navigation.findNavController(view);
edtBarcode.setText(CreateProductFragmentArgs.fromBundle(getArguments()).getBarcode());
```

Рисунок 3.43 – Отримання інформації про штрих-код

Користувач повинен зробити фотографію упаковки продукту, або ж вибрати вже існуюче фото з галереї мобільного пристрою. У двох випадках запускається активіті результату і поміщається фотографію у placeholder для зображень (див. рис. 3.44).

```
@SuppressWarnings("unused")
@OnClick(R.id.galleryBtn)
public void onGalleryButtonClick() {
    Intent gallery = new Intent(Intent.ACTION_PICK, MediaStore.Images.Media.EXTERNAL_CONTENT_URI);
    startActivityForResult(gallery, GALLERY_REQUEST_CODE);
}
```

Рисунок 3.44 – Startactivityforresult для відкриття галереї

Фотографія завантажується на сервер не одразу, а відправляється туди разом з іншими заповненими користувачем даними, коли останній натисне на кнопку «Зберегти» (див. рис. 3.46). Для зберігання фотографій використовується Firebase Storage, куди фотографія завантажується першочергово, а потім звідти витягується посилання на неї і присвоюємо це значення полю imageLink класу Product (див. рис. 3.45).

```
private void addNewProductWithImage(String barcode, String productName, String manufacturer, String productPirctureLink,
                                    String ingredients) {
    Product product = formProduct(barcode, productName, manufacturer, productPirctureLink, ingredients);
    productRef.add(product)
            .addOnCompleteListener(new OnCompleteListener<DocumentReference>() {
                @Override
                public void onComplete(@NonNull Task<DocumentReference> task) {
                    if (task.isSuccessful()) {
                        mCreateProductView.hideLoading();
                        mCreateProductView.onProductAddedSuccessful();
                    } else {
                        mCreateProductView.hideLoading();
                        mCreateProductView.onError();
                    }
                }
            }).addOnFailureListener(new OnFailureListener() {
        @Override
        public void onFailure(@NonNull Exception e) {
            mCreateProductView.hideLoading();
            mCreateProductView.onError();
        }
    });
}
```

Рисунок 3.45 – Створення нового продукту

```
/unused/
@OnClick(R.id.create_product_button)
public void onCreateButtonClick() {
    mPresenter.tryToAddNewProduct(CreateProductFragmentArgs.fromBundle(getArguments()).getBarcode(),
            edtProductName.getText().toString(), edtProductManufacturer.getText().toString(),
            edtIngredients.getText().toString(), pictureName, contentUri);
}
```

Рисунок 3.46 – Передавання інформації про продукт після натиску на кнопку «Зберегти»

І останнім кроком в заповненні інформації про новий продукт є внесення всіх інгредієнтів складу продуктового товару. Користувач вносить всі дані через кому в окремо виділене для цього поле. Після чого всі інгредієнти відділяються і поміщають у поле типу map (див. рис. 3.47).

```
private Product formProduct(String barcode, String productName, String manufacturer, String productPirctureLink,
                            String ingredients){
    String[] ingredientsArray = ingredients.split( regex: ",");
    Map<String, Boolean> ingredientsMap = new HashMap<>();
    for (String ingredient : ingredientsArray){
        ingredientsMap.put(ingredient.trim(), true);
    }
    return new Product(barcode, productName, manufacturer, productPirctureLink, ingredientsMap);
}
```

Рисунок 3.47 – Відділення інгредієнтів та занесення в поле

Після цього процесу товар є в базі і буде доступний для інший функцій додатку.

4 БІЗНЕС-ПЛАН РОЗРОБКИ ДОДАТКУ «EatBetter.»

4.1 Резюме проекту

Бізнес-ідея полягає в розробці програмного продукту для перевірки складу продуктових товарів. Додаток буде виконувати функцію сканування штрих-коду з подальшою перевіркою та аналізом складу продукту.

Ціль проекту – створення сервісу для швидкої перевірки складу продуктових товарів на алергени та інші шкідливі або небажані складники.

Цільова аудиторія – люди, які притримуються певного способу харчування (вегетаріанці, вегани та ін.), алергіки та люди, які обмежили себе у вживанні певних складників продуктів (наприклад, цукор, глуматат натрію та ін.). Додаток розрахований на користувачів операційної системи Android. Віковий сегмент – від 13 до 99 років.

Додаток буде доступний для завантаження в Google Play Market. Популяризація додатку буде здійснюватись через сторінку в Інстаграмі та таргетовану рекламу.

Головний конкурент – OpenFoodFacts. Але основною відмінністю даного додатку є наявність бази продуктів українського виробництва.

Організаційно-правова форма підприємства: приватний підприємець.

Для реалізації проекту потрібно найняти трьох працівників:

– програміст;

– тестувальник;

– UI/UX дизайнер.

Потреби в стартовому фінансуванні і дохід після реалізації проекту: стартовий капітал – не менше 63 500 грн, дохід після реалізації проекту (за перший місяць) – 46 800 грн (гірший випадок), 114 000 грн (кращий випадок).

4.2 Загальний опис проекту

Тип інвестиційного проекту – створення проекту «з нуля», так як проект новий і попередніх версій не існує.

На момент запуску планується реалізувати додаток по всій території України. Можливо, за умови, що проект буде успішний в межах України, планується запуск додатку в інших країнах СНД, а потім і світу. Також можливий другий варіант – це продаж продукту для його інтеграції в додаток продуктової мережі такої як МЕТРО, Сільпо.

Ціль проекту – створення сервісу для швидкої перевірки складу продуктових товарів на алергени та інші шкідливі складники.

Як зародилась бізнес-ідея: людина, що тільки починає харчуватись правильно та корисно витрачає рази більше часу на вибір та покупку продуктових товарів, ніж до того, тому що треба «створювати» нову продуктову корзину, без небажаних інгредієнтів в складі продуктів харчування; треба читати склад і аналізувати його. Але для того, щоб могти самостійно відфільтровувати непотрібні продукти потрібно запам’ятати багато інформації такої як, небезпечні інгредієнти, шкідливі складники, назви Е-додатків і їх призначення та шкоду. Тому було вирішено розробити додаток, який буде робити все за людей і стане невід’ємним помічником людям, що харчують правильно.

Цінність даного проекту полягає в тому, що мобільний додаток буде забезпечувати якісний аналіз складу продуктового товару за короткий час. Тобто буде економити час, який люди витрачають на перегляд складу і пошуку шкідливих чи небажаних для них складників.

Для досягнення цілі проекту потрібно придбати базу даних товарів, що доступні для продажу в українських продуктових магазинах.

Термін досягнення цілі проекту – від зародження бізнес-ідеї до завантаження готового додатку на Google Play Market – 2 місяці.

4.3 Загальний опис продукту

Мобільний додаток для аналізу складу продуктів харчування «EatBetter.»:

1) швидка і точна оцінка складу продукту, налаштування персоналізації;
2) зручний, інтуїтивно зрозумілий інтерфейс;
3) економія часу, витраченого на покупку продуктів;
4) доступна безкоштовна версія додатку.

Планується розробити дві версії додатку:

1) безкоштовна, але з рекламою;
2) платна, без реклами:
   - місячна підписка: 20грн;
   - річна підписка: 160 грн.

Реалізація додатку буде здійснюватись через Google Play Market.

4.4 Учасники проекту

Основні учасники проекту:

- власник проекту;
- програміст;
- тестувальник;
- UI/UX дизайнер.

Взаємодія учасників в проекті:

- власник проекту разом з командою (програмістом, тестувальником та дизайнером) обговорює всі свої побажання та вимоги щодо функціоналу та вигляду кінцевого продукту;
- дизайнер створює мокапи та прототип додатку, при цьому дизайнер повинен все узгодити з програмістом, щоб не було непорозуміння в подальшій реалізації, а також щоб дизайн додатку не обмежував його реалізацію (на цьому робота дизайнера закінчується);

– програміст імплементує весь функціонал додатку, запрограмовує бізнес-логіку і робить інтерфейс додатку відповідно до дизайну та прототипу;
– паралельно з програмістом працює тестувальник, тобто коли є імплементовані функції вони мають бути протестовані. Паралельна робота програміста і тестувальника має бути для того, щоб в кінці роботи над додатком не було витрачено багато часу на правки;
– програміст з тестувальником проводять демо-презентації імплементованих функцій для власника кожних 4 дні, для того щоб всі вимоги і побажання були враховані;
– власник, в цей час, планує продаж і реалізацію додатку, він проводить багато переговорів з потенційними покупцями товару для його подальшої інтеграції;
– готовий додаток віддається власнику;
– в залежності, від подальшої реалізації додатку можливі два варіанти продовження роботи програміста та тестувальника:
– якщо додаток буде реалізований на платформі Google Play Market, то на цьому їхня робота закінчується;
– якщо додаток буде проданий для інтеграції, то програміст разом з тестувальником, можливо, мають інтегрувати це (за умови, якщо покупець додатку не буде наймати своїх працівників для цього).

4.5 Організаційний план та маркетинг

План розробки додатку:

1) збір команди;
2) оплата послуг програміста та тестувальника;
3) розробка додатку для операційної системи Android;
4) рекламна кампанія;

5) публікація додатку в Google Play Market;
6) розвиток маркетингової стратегії для зацікавлення нових клієнтів;
7) технічна підтримка користувачів.

План реалізації проекту:

1) збір команди та підписання контрактів;
2) розробка і тестування додатку (1.5 місяці, працюють разом);
3) підписання договору з плей маркет (1 тиждень);
4) публікація додатку в плей маркет (1 тиждень);
5) старт продажу і технічна підтримка.

Маркетинговий план реалізації та популяризації додатку:

1) створення тематичного рекламного ролику про розроблений додаток, довжиною 15 секунд (обмеження в 15 секунд є для того, щоб відео можна було завантажити однією історією в Інстаграм-і);
2) поширення за допомогою соціальних мереж, використовуючи таргетовану рекламу;
3) пошук та договір Інстаграм-блогера з аудиторією 50 000+, у якого тематика блогу пов’язана зі сферою правильного харчування;
4) розповсюдження реклами в магазинах здорового харчування, використовуючи різні рекламні засоби для цього. Наприклад, рекламні листівки, банери, оголошення та ін.

4.6 Фінансовий план

Таблиця 4.1 - Витрати на персонал та матеріали для реалізації

Витрати на персонал			
№	Працівник	Години роботи, год	Зарплата, грн/год
1	Програміст	240	187,5
2	Тестувальник	120	75
3	Дизайнер	56	150
Витрати на матеріали			
№	Назва	Ціна, грн	
1	Сервіс FireStore	2 грн/100 000 зчитувань записів	
2	База даних штрих-кодів продуктів харчування	270 грн/5 000 запитів	
Витрати на маркетингову діяльність			
№	Назва	Ціна, грн	
1	Рекламний відеоролик	560	
2	Реклама в Інстаграм-блогера	9000	

Витрати на обладнання та оренду приміщення не передбачені, оскільки персонал буде працювати віддалено, використовуючи своє обладнання.

Загальна сума витрат – 73 060 грн, з яких:

– витрати на персонал: 62 400 грн.;

– витрати на матеріали: 1 100 грн;

– витрати на маркетингову діяльність: 73 060 грн.

Діаграма витрат на весь процес розробки та реалізації додатку зображено на рисунку 4.1.

Рисунок 4.1 – Діаграма витрат на розробку та реалізацію додатку

4.7 План продажу та розрахунок доходу

План продажу:

1) перший місяць після запуску очікується 9 000 скачувань мобільного додатку, з яких 1 600 – з покупкою преміум-версії без реклами;
2) протягом другого – четвертого місяців очікується такий самий попит на продукт, як і в перший місяць після запуску. Рекламна кампанія проводиться так само;
3) після п’ятого місяця очікується спад попиту на додаток, тобто приблизно 7 000 скачувань, з яких 1 000 з покупкою преміум-версії;
4) починаючи з шостого місяця – 6 500 скачувань, з яких 800 з преміум-версією; сьомий – дев’ятий місяці: 5 000 скачувань в місяць, з яких 600 з покупкою преміум-версії;
5) через рік після запуску очікується, що додаток вийде на стабільну кількість скачувань в місяць – 4 000 скачувань, з яких 500 з преміум-версією.

Розрахунок доходів від продажу та реклами за перший місяць:

1) реклама всередині додатку (контекстна):

Внутрішня реклама працює наступним чином: рекламне оголошення відображається на попередньо виділеному місці в інтерфейсі, що дає прибуток за певну кількість переглядів та/або переходів.

Припустимо, що в день один користувач буде здійснювати 15 сканувань штрих-коду. Реклама буде показуватись після кожного 6 сканування. Тобто 2 перегляди реклами в день від одного користувача. За планом продажу, в перший місяць очікується 9 000 завантажень, з яких 7 400 – безкоштовна версія.

Отже, 2*7 400 = 14 800. 14 800 переглядів реклами в місяць.

Один перегляд реклами оцінюється, приблизно в 1 грн. Враховуючи це, за перший місяць від внутрішньої реклами ми отримаємо 14 800 грн.

2) купівля преміум-версії додатку:

за планом, преміум-версію додатку куплять 1 600 користувачів. Якщо 1 600 користувачів придбають місячну підписку, що найбільш ймовірно, то ми отримаємо 20*1 600 = 32 000 грн. Також можливий варіант, що 30% користувачів придбають річну підписку, а 70% користувачів – місячну. Тоді: 480*160 = 76 800 грн – дохід від покупки річної підписки на преміум-версію, 1 120*20 = 22 400 грн – дохід від покупки користувачами місячної підписки. Разом за перший місяць можна виручити 99 200 грн. (див. таб. 4.2).

Підсумовуючи, дохід за перший місяць від реклами, а також від покупок преміум-версії може складати 114 000 грн (в кращому випадку), що повністю покриває витрати на розробку додатку. А в гіршому випадку – 46 800 грн., в такому випадку очікується, що сума доходів покриє суму витрат за півтора місяці.

Таблиця 4.2 – Розрахунок доходу від додатку

№	Назва доходу	Сума доходу в місяць, грн
1	Внутрішня реклама в безкоштовній версії додатку (план продажу – 7 400 скачувань)	14 800
2	Продаж преміум-версії додатку без реклами (план продажу – 1 600 купівель)	32 000 (тільки місячна підписка), 99 200 (30% продаж річної підписки, 70% - місячної)
Разом:		46 800 (гірший випадок), 114 000 (кращий випадок)

4.8 Можливі ризики

- Ризики пов’язані з часом:
  - неправильно встановлені часові межі реалізації;
  - неочікувана зміна масштабу проекту;
- Бюджетні, фінансові ризики:
  - неправильне розподілення бюджету;
  - не відслідковування витрат;
- Процедурні:
  - різні пріоритети в робочому процесі;
  - відсутність або брак командного духу;
  - невідповідальність працівників.

ВИСНОВКИ

Метою даного дипломного проекту була проектування та розробка мобільного додатку для сканування штрих-коду продуктового товару і його аналізу. Завдяки правильно поставленим цілям мету було досягнуто. У ході проектування та розробки програмного продукту було досліджено предметну область здорового харчування, проаналізовано потреби користувачів даного сегменту і на основі цих даних було спроектовано архітектуру додатку.

Для реалізації функціоналу мобільного застосунку було використано передові технології, такі як мова програмування Java, середовище розробки Android Studio, бібліотека Google Vision API та архітектурні шаблони MVP та, частоково, MVVM.

Всі визначені вимоги були виконані та очікуваний функціонал - реалізовано. Додаток готовий до використання користувачами.

Після пілотного запуску мобільного додатку та оцінки попиту очікується розширення функціоналу:

- додавання розпізнавання тексту з фото, для полегшення процесу додавання нового продукту в базу;
- додавання нових корисних функцій: списки покупок, можливість додавати продукти у «Вибране»;
- розширення бази даних продукту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

REFERENCES

1. Здорове харчування – Вікіпедія. URL: https://uk.wikipedia.org/wiki/Здорове_харчування (дата звернення: 18.11.2019)
2. Michael Greger, M.D., FACLM; Gene Stone How Not To Die: Discover the Foods Scientifically Proven to Prevent and Reverse Disease, Flatiron Books, ISBN 978-1-4472-8245-7, 2015.
3. Перше місце України. Чому ми очолили рейтинг країн з найвищою смертністю через неправильне харчування – nv.ua. URL: https://nv.ua/ukr/ukraine/events/pershe-misce-ukrajini-chomu-mi-ocholili-reyting-krajin-z-nayvishchoyu-smertnistyu-cherez-nepravilne-harchuvannya-50006288.html (дата звернення: 18.11.2019)
4. МОЗ України представило рекомендації зі здорового харчування – МОЗ. URL: https://moz.gov.ua/article/news/moz-ukraini-predstavilo-rekomendacii-zi-zdorovogo-harchuvannja (дата звернення: 18.11.2019)
5. Maura Harrigan Gonzalez Way to Eat: A Six-step Path to Lifelong Weight Control, Sourcebooks, Inc., ISBN 978-1-4022-5210-5, 2002.
6. Leo M.L. Nollet; Arjon J. van Henge Food Allergens: Analysis Instrumentation and Methods, CRC Press, ISBN 978-1-4398-1505-2, 2010.
7. Open Food Facts – Play Market. URL: https://play.google.com/store/apps/details?id=org.openfoodfacts.scanner&hl=uk (дата звернення: 25.11.2019)
8. ЭкоЕд – Play Market. URL:

https://play.google.com/store/apps/details?id=com.agnitio&hl=uk
(дата звернення: 25.11.2019)
9. Сканер їжі – Play Market. URL: https://play.google.com/store/apps/details?id=food.scanner&hl=uk (дата звернення: 25.11.2019)
10. Кузь М.В., Соловко Я.Т., Андрейко В.М. Методологія формування узагальненого критерію якості програмного забезпечення в умовах невизначеності. Вісник Вінницького політехнічного інституту. Вінниця, 2015. №5. С. 104-107.
11. Кузь М., Новак В., Новак М., Сільва Р. Модель часових параметрів показників якості програмного забезпечення. Прикладні науково-технічні дослідження: матеріали II міжнар. наук.-практ. конф., м. Івано-Франківськ, 3-5 квіт. 2018 р. Івано-Франківськ, 2018. С.32-33.
12. M. Kozlenko, V. Tkachuk, and M. Dutchak, "Software implementation of microcomputer based intrusion detection and prevention system with binary neural network," in *Proc. 2nd International Scientific-Practical Conference "Problems of Cyber Security of Information and Telecommunication Systems" (PCSITS)*, O. Oksiiuk et al, Eds. Taras Shevchenko National University of Kyiv, Kyiv, Ukraine, Apr. 11-12, 2019, pp. 371-373.
13. Java – Вікіпедія. URL: https://uk.wikipedia.org/wiki/Java (дата звернення: 07.02.2020)
14. Рейтинг мов програмування 2019 – DOU. URL: https://dou.ua/lenta/articles/language-rating-jan-2019/ (дата звернення: 07.02.2020)
15. Ron Hitchens Java NIO, "O'Reilly Media, Inc.", ISBN 978-0-596-00288-6, 2002.
16. Android Studio – Википедия. URL: https://ru.wikipedia.org/wiki/Android_Studio (дата звернення: 07.02.2020)

17. Обзор и пример использования библиотеки ButterKnife в Android – Javadevblog. URL: https://javadevblog.com/obzor-i-primer-ispol-zovaniya-biblioteki-butterknife-v-android.html (дата звернення: 07.02.2020)
18. Применение Google Cloud Vision API в приложении для Android – Anroiddev. URL: https://androiddev.apptractor.ru/primenenie-google-cloud-vision-api-v-prilozhenii-dlya-android/ (дата звернення: 07.02.2020)
19. Firebase – Вікіпедія. URL: https://uk.wikipedia.org/wiki/Firebase (дата звернення: 07.02.2020)
20. NoSQL – Вікіпедія. URL: https://uk.wikipedia.org/wiki/NoSQL (дата звернення: 13.02.2020)
21. Cloud Firestore – Firebase. URL: https://firebase.google.com/docs/firestore?authuser=0 (дата звернення: 13.02.2020)
22. Cloud Storage – Firebase. URL: https://firebase.google.com/docs/storage?authuser=0 (дата звернення: 13.02.2020)
23. MVC, MVP and MVVM Design Pattern – Medium. URL: https://medium.com/@ankit.sinhal/mvc-mvp-and-mvvm-design-pattern-6e169567bbad (дата звернення: 13.02.2020)
24. Карьера в IT: должность UX/UI дизайнер – DOU. URL: https://dou.ua/lenta/articles/ux-ui-designer-position/ (дата звернення: 13.02.2020)
25. Pablo Perea, Pau Giner UX Design for Mobile, Packt Publishing Ltd, ISBN 978-1-78728-359-6, 2017.

ДОДАТОК А

Лістинг програми

Додаток А

Клас RegisterPresenter.java

```
package com.eatbetter.presenter;

import android.text.TextUtils;
import android.util.Log;

import androidx.annotation.NonNull;

import com.eatbetter.interfaces.RegisterEmailView;
import com.eatbetter.model.User;
import com.google.android.gms.tasks.OnCompleteListener;
import com.google.android.gms.tasks.OnFailureListener;
import com.google.android.gms.tasks.Task;
import com.google.firebase.auth.AuthResult;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.firestore.CollectionReference;
import com.google.firebase.firestore.FirebaseFirestore;

import java.util.HashMap;
import java.util.Map;

public class RegisterPresenter {

    private static final String TAG = "Register";
    private final RegisterEmailView mRegisterView;
    private final FirebaseAuth firebaseAuth;
    private final CollectionReference userRef;

    public RegisterPresenter(@NonNull RegisterEmailView registerView) {
        mRegisterView = registerView;
        firebaseAuth = FirebaseAuth.getInstance();
        userRef = FirebaseFirestore.getInstance().collection("Users"); }
```

```
    public void trySignUpWithEmail(@NonNull String email, @NonNull String password,
@NonNull String confirmPassword, String... preferencesArray) {
        if (TextUtils.isEmpty(email)) {
            mRegisterView.showLoginError();
        } else if (TextUtils.isEmpty(password)) {
            mRegisterView.showPasswordError();
        } else if (!password.equals(confirmPassword)) {
            mRegisterView.showConfirmPasswordError();
        }else {
            firebaseAuth.createUserWithEmailAndPassword(email, password)
                    .addOnCompleteListener(new OnCompleteListener<AuthResult>() {
                        @Override
                        public void onComplete(@NonNull Task<AuthResult> task) {
                            if (task.isSuccessful()) {
                                Log.d(TAG, "createUserWithEmail:success");
                                Map<String, Boolean> preferencesMap = new HashMap<>();
                                for (String preference : preferencesArray){
                                    preferencesMap.put(preference, true);
                                }
                                String uid = firebaseAuth.getCurrentUser().getUid();
                                User user = new User(uid, email, preferencesMap);
                                userRef.add(user);
                                mRegisterView.openHomeFragment();

                            } else {
                                Log.w(TAG, "createUserWithEmail:failure",
task.getException());
                                mRegisterView.showAuthenticationFailed();
                            }

                        }
                    })
                    .addOnFailureListener(new OnFailureListener() {
                        @Override
                    public void onFailure(@NonNull Exception e) {
                            Log.w(TAG, "createUserWithEmail:failure", e.getCause());
                        }               });      }   } }
```

Продовження додатку А

Клас RegisterFragment.java

```
package com.eatbetter.auth;

import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.EditText;
import android.widget.Toast;

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.fragment.app.Fragment;
import androidx.navigation.NavController;
import androidx.navigation.Navigation;

import com.eatbetter.R;
import com.eatbetter.interfaces.RegisterEmailView;
import com.eatbetter.presenter.RegisterPresenter;

import butterknife.BindView;
import butterknife.ButterKnife;
import butterknife.OnClick;

/**
 * A simple {@link Fragment} subclass.
 */
public class RegisterFragment extends Fragment implements RegisterEmailView {

    @BindView(R.id.fieldEmailRegister)
    EditText edtEmail;

    @BindView(R.id.fieldPasswordRegister)
    EditText edtPassword;
```

Продовження додатку А

```
 @BindView(R.id.fieldPasswordConfirm)
    EditText edtConfirmPassword;

    public RegisterFragment() {
        // Required empty public constructor
    }

    private RegisterPresenter mPresenter;
    private NavController navController;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        return inflater.inflate(R.layout.fragment_register, container, false);
    }

    @Override
    public void onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);
        ButterKnife.bind(this, view);
        navController = Navigation.findNavController(view);

mPresenter = new RegisterPresenter(this);
    }

    @SuppressWarnings("unused")
    @OnClick(R.id.emailCreateAccountButton)
    public void onCreateButtonClick() {
        mPresenter.trySignUpWithEmail(edtEmail.getText().toString(),
edtPassword.getText().toString(),
                edtConfirmPassword.getText().toString(),
RegisterFragmentArgs.fromBundle(getArguments()).getPreferencesAll());
    }

    @Override
    public void showConfirmPasswordError() {
```

Продовження додатку А

```
 edtConfirmPassword.setError("Required.");
    }

    @Override
    public void openHomeFragment() {
        navController.navigate(R.id.action_registerFragment_to_mainActivity);
    }

    @Override
    public void showLoginError() {
        edtEmail.setError("Required.");
    }

    @Override
    public void showPasswordError() {
        edtPassword.setError("Required.");
    }

    @Override

 public void showAuthenticationFailed() {
        Toast.makeText(getContext(), "Registration failed. Please try again",
                Toast.LENGTH_SHORT).show();
    }

    @Override
    public void showLoading() {

    }

    @Override
    public void hideLoading() {

    }
}
```

Продовження додатку А

Клас PreferencesPresenter.java

```
package com.eatbetter.presenter;

import androidx.annotation.NonNull;

import com.eatbetter.interfaces.PreferencesList;
import com.eatbetter.interfaces.PreferencesView;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class PreferencesPresenter {

    private final PreferencesView mPreferencesView;

private List<String> preferencesList;

    public PreferencesPresenter(@NonNull PreferencesView preferencesView) {
        mPreferencesView = preferencesView;
        preferencesList = new ArrayList<>();
    }

    public void passSelectedPreferences(){
        mPreferencesView.passPreferencesToNextPage();
    }

    public void passSelectedPreferences(List<Integer> checkedResult) {
        for(Integer checked : checkedResult){

preferencesList.addAll(Arrays.asList(PreferencesList.values()[checked].getPreferences()))
;
        }
        mPreferencesView.passPreferencesToNextPage(preferencesList.toArray(new
String[0]));
    }}
```

Продовження додатку А

Клас ScannerPresenter.java

```
package com.eatbetter.presenter;

import androidx.annotation.NonNull;

import com.eatbetter.interfaces.ScannerView;
import com.eatbetter.model.Product;
import com.eatbetter.model.User;
import com.google.android.gms.tasks.OnCompleteListener;
import com.google.android.gms.tasks.Task;
import com.google.firebase.auth.FirebaseAuth;

import com.google.firebase.firestore.CollectionReference;
import com.google.firebase.firestore.FirebaseFirestore;
import com.google.firebase.firestore.QuerySnapshot;
import com.google.firebase.firestore.SetOptions;

public class ScannerPresenter {

    private final ScannerView mScannerView;
    private FirebaseAuth mAuth;
    private CollectionReference productRef;
    private CollectionReference userRef;

    public ScannerPresenter(ScannerView scannerView) {
        mAuth = FirebaseAuth.getInstance();
        FirebaseFirestore db = FirebaseFirestore.getInstance();
        mScannerView = scannerView;
        productRef = db.collection("Products");
        userRef = db.collection("Users");
    }

    public void tryToVerifyProduct(String barcode) {
        mScannerView.showLoading();
        productRef.whereEqualTo("barcode", barcode)
```

Продовження додатку А

```
                .limit(1)
                .get()
                .addOnCompleteListener(new OnCompleteListener<QuerySnapshot>() {
                    @Override
                    public void onComplete(@NonNull Task<QuerySnapshot> task) {
                        if (task.isSuccessful()) {
                            if (task.getResult().getDocuments().size() < 1) {
                                mScannerView.onProductNotExist(barcode);
                            } else {


                         String productDocumentId =
task.getResult().getDocuments().get(0).getId();
                                Product product =
task.getResult().getDocuments().get(0).toObject(Product.class);
                                userRef.whereEqualTo("userId",
mAuth.getCurrentUser().getUid())
  .get()
                                        .addOnCompleteListener(new
OnCompleteListener<QuerySnapshot>() {
@Override
                                            public void onComplete(@NonNull
Task<QuerySnapshot> task) {
if (task.isSuccessful()) {
                                                    if
(task.getResult().getDocuments().size() > 0) {
                                                        String userDocumentId =
task.getResult().getDocuments().get(0).getId();

userRef.document(userDocumentId).collection("history")

.document(productDocumentId).set(product, SetOptions.merge());
                                                        boolean result =
findMatch(product, task.getResult().getDocuments().get(0).toObject(User.class));
if (result) {

mScannerView.onProductContains(product);
                            mScannerView.hideLoading();
                              }     else {

mScannerView.onProductNotContains(product);
mScannerView.hideLoading();   }   }   }   }   });  } }  }  });  }
```

```
    private boolean findMatch(Product product, User user) {
        return product.getIngredients().entrySet().stream()
                .anyMatch(e ->
e.getValue().equals(user.getPreferences().get(e.getKey())));
    }

}
```

Клас UserHistoryRepository.java

```
package com.eatbetter.viewmodel;

import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;

import android.widget.TextView;

import androidx.annotation.NonNull;
import androidx.recyclerview.widget.RecyclerView;

import com.eatbetter.R;
import com.eatbetter.model.Product;

import java.util.ArrayList;
import java.util.List;

public class UserHistoryAdapter extends
RecyclerView.Adapter<UserHistoryAdapter.ProductsHolder> {

    private List<Product> products = new ArrayList<>();
    private OnProductListItemClickedListener listener;

    public UserHistoryAdapter(OnProductListItemClickedListener
onProductListItemClickedListener) {
        this.listener = onProductListItemClickedListener;   }
```

Продовження додатку А

```
@NonNull
    @Override
    public ProductsHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
        View itemView = LayoutInflater.from(parent.getContext())
                .inflate(R.layout.product_item, parent, false);
        return new ProductsHolder(itemView);
    }

    @Override
    public void onBindViewHolder(@NonNull ProductsHolder holder, int position) {
        Product currentProduct = products.get(position);
        holder.textViewName.setText(currentProduct.getName());
        holder.textViewBarcode.setText(currentProduct.getBarcode());
    }
    @Override
    public int getItemCount() {
        return products.size();
    }

    public void setProducts(List<Product> products) {

        this.products = products;
        notifyDataSetChanged();
    }

    class ProductsHolder extends RecyclerView.ViewHolder {
        private TextView textViewName;
        private TextView textViewBarcode;
        private Button viewProductBtn;

        public ProductsHolder(@NonNull View itemView) {
            super(itemView);
            textViewName = itemView.findViewById(R.id.text_view_product_name);
            textViewBarcode = itemView.findViewById(R.id.text_view_barcode);
```

```
            viewProductBtn = itemView.findViewById(R.id.button_to_product);
             viewProductBtn.setOnClickListener(new View.OnClickListener() {
                 @Override
                 public void onClick(View v) {
                     int position = getAdapterPosition();
                     if (listener != null && position != RecyclerView.NO_POSITION)
                         listener.onItemClicked(products.get(position));
                 }
             });
        }
    }

    public interface OnProductListItemClickedListener {
        void onItemClicked(Product product);
    }}
```

Клас UserHistoryViewModel.java

```
package com.eatbetter.viewmodel;


import androidx.lifecycle.LiveData;
import androidx.lifecycle.MutableLiveData;
import androidx.lifecycle.ViewModel;
import com.eatbetter.model.UserHistoryRepository;
import com.eatbetter.model.Product;


import java.util.List;


public class UserHistoryViewModel extends ViewModel implements
UserHistoryRepository.OnFirestoreTaskComplete {
    private MutableLiveData<List<Product>> productListModelData = new
MutableLiveData<>();
    private UserHistoryRepository db = new UserHistoryRepository(this);


    public UserHistoryViewModel() {
        db.getProductsData();}
```

Продовження додатку А

```
public LiveData<List<Product>> getProductListModelData() {
        return productListModelData;
    }

    @Override
    public void productListDataAdded(List<Product> productListModelsList) {
        productListModelData.setValue(productListModelsList);
    }
  @Override
    public void onError(Exception e) {}}
```