МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ  
Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ   
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационных систем и технологий

Специальность 1-98 01 03 «Программное обеспечение информационной безопасности мобильных систем»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА КУРСОВОГО ПРОЕКТА**

по дисциплине «Программирование мобильных систем»

Тема «Мобильное приложение E-Commerce App»

**Исполнитель**

студент 3 курса 8 группы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Фадеев А.Д.

подпись, дата

**Руководитель**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Уласевич Н.И.

должность, ученая степень, ученое звание подпись, дата

Допущен(а) к защите \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

дата, подпись

Курсовой проект защищен с оценкой

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Уласевич Н.И.

подпись дата инициалы и фамилия

**Содержание**

[**Введение** 3](#_Toc154131759)

[**1. Анализ требований к программного средству** 4](#_Toc154131760)

[1.1 Аналитический обзор аналогов 6](#_Toc154131761)

[**1.1.1 Программное средство «Amazon»** 6](#_Toc154131762)

[**1.1.2 Программное средство «eBay»** 7](#_Toc154131763)

[**1.1.3 Программное средство «Alibaba»** 8](#_Toc154131764)

[**1.2 Вывод анализа требований** 8](#_Toc154131765)

[**2. Анализ требований к программному средству и разработка функциональных требований** 9](#_Toc154131766)

[2.1 Сущности в базе данных и связи между ними 1](#_Toc154131767)1

**2.2 Серверная часть приложения** …………………………………………………..12

**2.2.1 Уровни. API, Service, Repository** …………………………………………...…12

**2.2.2 Объекты, использующиеся на различных уровнях, и их особенности** ….13

**2.2.3 Безопасность. Аутентификация и авторизация** ……………………………13

**2.2.4 Хранение фото** …………………………………………………………………..14

**2.3 Клиентская часть приложения** …………………………………………………14

**2.3.1 Виджеты, управление состоянием виджетов, декларативный UI, фрагменты и диалоги, навигация** …………………………………………………..15

**2.3.2 Внедрение зависимостей** ……………………………………………………….16

**2.3.3 Дизайн и расположение элементов** …………………………………………...16

**2.4 Итоги раздела** ……………………………………………………………………..17

**3 Реализация приложения** …………………………………………………………...18

**3.1 Серверная часть** …………………………………………………………………..18

**3.1.1 Конфигурация** …………………………………………………………………..18

**3.2.1 Реализация REST API. Контроллеры** ………………………………………..18

**3.1.3 Сервисы** ………………………………………………………………………….19

**3.1.4 Репозитории. Взаимодействие с Firestore** ……………………………………19

**3.1.5 DTO. Конвертация разных типов объектов** ………………………………...20

**3.1.6 Аутентификация и авторизация** ……………………………………………...20

**3.2 Клиентская часть** …………………………………………………………………21

**3.2.1 Виджеты, разметка, диалоги** …………………………………………………..21

**3.2.3 Внедрение зависимостей** ……………………………………………………….21

**3.3 Итоги раздела** ……………………………………………………………………...22

**4. Тестирование, проверка работоспособности и анализ полученных данных** .23

**5. Руководство по использованию** ………………………………………………….24

**Заключение** ……………………………………………………………………………28

**Список литературы** …………………………………………………………………..29

**Приложение А** …………………………………………………………………………30

**Приложение В** …………………………………………………………………………33

# **Введение**

В эпоху цифровизации и онлайн-торговли мобильные приложения становятся ключевым инструментом для предпринимателей, желающих создать эффективный и привлекательный канал продаж. Особенно важным является разработка приложений для электронной коммерции (E-commerce), которые обеспечивают удобный доступ к товарам и услугам, удовлетворяя запросы современного потребителя. В этом контексте разработка мобильного приложения E-commerce на основе фреймворка Flutter представляет собой актуальное и перспективное направление в области мобильной разработки.

Целью данного курсового проекта является изучение и практическое применение принципов разработки мобильных приложений E-commerce на платформе Flutter и языка программирования Dart. Flutter, разработанный Google, представляет собой мощный инструмент для создания кроссплатформенных приложений с использованием единой кодовой базы. Его преимущества включают быструю разработку, высокую производительность и возможность создания красивого и адаптивного пользовательского интерфейса.

Этот курсовой проект предоставляет уникальную возможность применить полученные знания и навыки в области мобильной разработки на практике, а также ознакомиться с особенностями создания приложений для электронной коммерции. В результате успешного завершения проекта полностью функциональное приложение электронной коммерции, которое будет разработано с использованием современных технологий.

1. **Анализ требований к программного средству**

Целью данного проекта является разработка кроссплатформенного мобильного приложения для управления таксопарком на основе фреймворка Flutter.

Для достижения поставленной цели функционально должны быть выполнены следующие требования:

* Гость может зарегистрироваться в приложении, создав свой учетный профиль.
* Пользователь и гость имеет возможность проходить авторизацию, используя ранее созданный аккаунт.
* Подтверждение электронной почты для новых пользователей обеспечивает безопасность и подтверждает их идентичность.
* Просмотр товаров: Пользователь может просматривать доступный ассортимент товаров в приложении.
* Поиск товаров: Пользователь может выполнять поиск товаров по ключевым словам или категориям.
* Сортировка товаров: Пользователь имеет возможность сортировать товары по различным критериям, таким как цена или популярность.
* Фильтрация товаров: Пользователь может применять фильтры для уточнения результатов поиска в соответствии с их предпочтениями.
* Добавление товаров в список пожеланий: Пользователь может добавлять понравившиеся товары в список пожеланий для последующего просмотра или покупки.
* Добавление товаров в корзину: Пользователь может добавлять выбранные товары в корзину для последующего оформления заказа.
* Удаление товаров из корзины: Пользователь имеет возможность удалять товары из корзины перед оформлением заказа.
* Использование промо кодов: Пользователь может вводить промо коды для получения скидок или специальных предложений.
* Оформление заказа: Пользователь может оформлять заказ, выбирая нужные товары, указывая адрес доставки и выбирая способ оплаты.
* Оплата заказа: Пользователь может производить оплату заказа онлайн, используя доступные способы оплаты.
* Отслеживание заказов: Пользователь может отслеживать статус и местоположение своих заказов в режиме реального времени.
* Редактирование профиля: Пользователь имеет возможность редактировать свой профиль, включая изменение личной информации и настроек безопасности.

Диаграмма вариантов использования для пользователя и гостя должна включать все действия, указанные выше, и доступные им, которые будут закрывать все их потребности в рамках приложения электронной коммерции.

На рисунке 1.1 представлена диаграмма вариантов использования для гостя приложения.

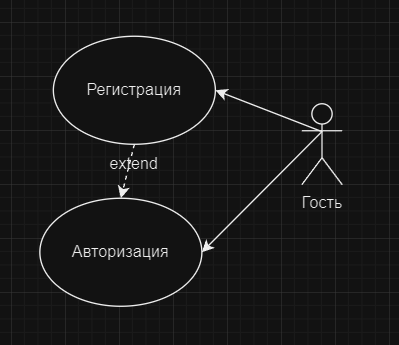


Рисунок 1.1 — Диаграмма вариантов использования для гостя

На рисунке 1.2 представлена диаграмма вариантов использования для пользователя приложения.

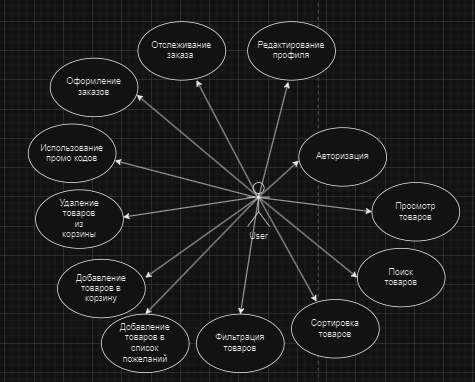


Рисунок 1.2 — Диаграмма использования для пользователя

**Вывод по постановке задачи:**

В результате анализа требований к программному средству были определены основные функциональные требования к мобильному приложению электронной коммерции на основе фреймворка Flutter. Это включает в себя не только основные функции приложения, такие как регистрация пользователей, управление их данными, но и необходимость обеспечения высокой производительности, безопасности и удобства использования.

* 1. **Аналитический обзор аналогов**

В наши дни множество подобных программных продуктов можно найти на сайтах с тематикой по данному профилю. Перед тем как приступить к работе требовалось провести анализ и прочитать соответствующие статьи по данной теме. В ходе поиска данной литературы мною были найдены прототипы приложений E-commerce.

1. **Программное средство «Amazon»**

На рисунке 1.1 представлен скриншоты мобильного приложения Amazon.

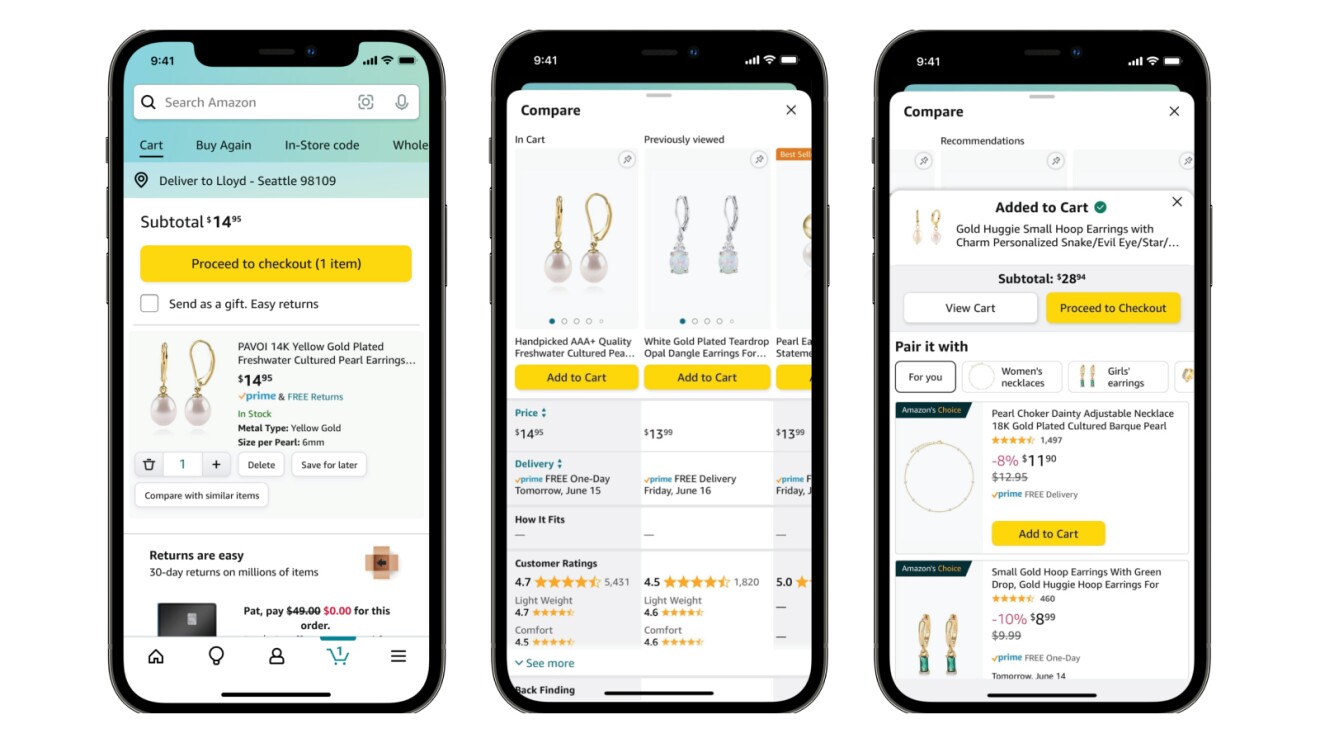


Рисунок 1.1 – Скриншоты аналога мобильного приложения «Amazon»

Проанализировав мобильное приложение «Amazon», можно выделить некоторые плюсы и минусы программного средства.

Основные плюсы:

* Удобная навигация. Приложение обеспечивает простую и интуитивно понятную навигацию по категориям и подкатегориям товаров.
* Отзывы пользователей. Пользователи могут оставлять отзывы о товарах, что помогает другим пользователям сделать правильный выбор.
* Персонализация. Amazon предлагает рекомендации товаров на основе предыдущих покупок и просмотров, что улучшает пользовательский опыт.

Основные минусы:

* Загроможденный интерфейс. Иногда интерфейс приложения может казаться перегруженным из-за большого количества информации.
* Сложности с поиском: Некоторые пользователи могут испытывать трудности с поиском товаров из-за большого объема предложений.

1. **Программное средство «eBay»**

На рисунке 1.2 представлен скриншоты мобильного приложения «eBay».

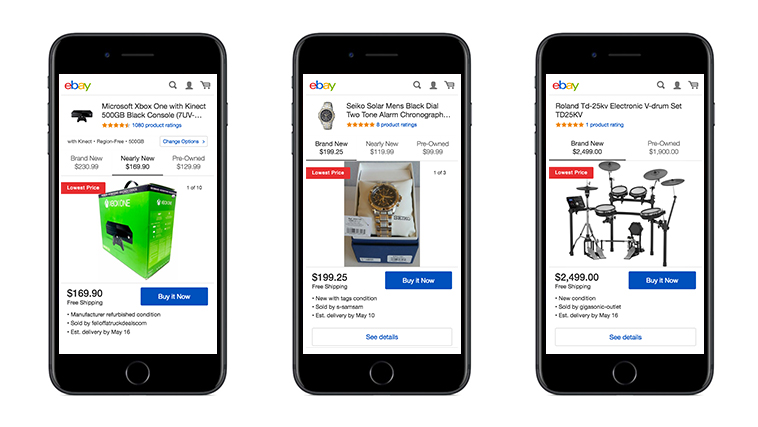


Рисунок 1.2 – Скриншот аналога веб-приложения «eBay»

Проанализировав мобильное приложение «eBay», можно выделить некоторые плюсы и минусы программного средства.

Основные плюсы приложения:

* Вариативность предложений: Пользователи могут найти как новые, так и б/у товары, что дает больше выбора.
* Интерактивность: Возможность участвовать в аукционах добавляет элемент игры и взаимодействия для пользователей.
* Площадка также интегрируют технологии обработки больших данных для обработки большого объема транзакций.

Основные минусы приложения:

* Сложности с навигацией. Интерфейс eBay может иногда казаться менее интуитивным, особенно для новых пользователей.
* Неоднородность качества. Из-за того, что товары могут быть как новыми, так и б/у, качество их может варьироваться.

1. **Программное средство «Alibaba»**

На рисунке 1.3 представлены скриншоты мобильного приложения «Alibaba».



Рисунок 1.3 – Скриншоты аналога мобильного приложения «Alibaba»

Проанализировав «Alibaba», можно выделить некоторые плюсы и минусы программного средства.

Основные плюсы:

* Они также интегрируют большие данные для анализа предпочтений покупателей и оптимизации поиска товаров.
* Alibaba активно использует технологии блокчейна для обеспечения безопасности транзакций и подтверждения подлинности товаров.

Основные минусы:

* Сложности с качеством: Качество товаров на Alibaba может быть неоднородным, что может создавать риски для покупателей.
* Ограничения для мелких покупателей: Некоторые товары могут иметь минимальные заказы, что не всегда удобно для малых бизнесов или частных покупателей.
  1. **Вывод анализа требований**

При анализе трех приложений e-commerce конкурентов, можно заметить, что каждое из них имеет свои уникальные плюсы и минусы с точки зрения пользовательского опыта (UX) и технологических аспектов. Общими положительными моментами являются интуитивный интерфейс, персонализация рекомендаций, широкий выбор товаров и использование технологий машинного обучения для улучшения опыта пользователей. Однако недостатки могут включать в себя сложности с навигацией, неоднородность качества товаров и ограничения для некоторых типов пользователей. Важно учитывать это при разработке собственного приложения, чтобы создать удобное и привлекательное для пользователей приложение, которое будет конкурентоспособным на рынке e-commerce.

1. **Анализ требований к программному средству и разработка функциональных требований**

Прежде чем приступить к формулировке функциональных требований, проведем теоретический анализ предметной области - системы управления задачами.

Анализ требований к программному средству и разработка функциональных требований планировщика задач являются важными этапами процесса разработки программного обеспечения. Ниже приведены основные шаги, которые могут помочь нам выполнить поставленную задачу.

Функции пользователя включают:

* Пользователь может зарегистрироваться в приложении, создав свой учетный профиль.
* Пользователь имеет возможность проходить авторизацию, используя ранее созданный аккаунт.
* Подтверждение электронной почты для новых пользователей обеспечивает безопасность и подтверждает их идентичность.
* Просмотр товаров: Пользователь может просматривать доступный ассортимент товаров в приложении.
* Поиск товаров: Пользователь может выполнять поиск товаров по ключевым словам или категориям.
* Сортировка товаров: Пользователь имеет возможность сортировать товары по различным критериям, таким как цена или популярность.
* Фильтрация товаров: Пользователь может применять фильтры для уточнения результатов поиска в соответствии с их предпочтениями.
* Добавление товаров в список пожеланий: Пользователь может добавлять понравившиеся товары в список пожеланий для последующего просмотра или покупки.
* Добавление товаров в корзину: Пользователь может добавлять выбранные товары в корзину для последующего оформления заказа.
* Удаление товаров из корзины: Пользователь имеет возможность удалять товары из корзины перед оформлением заказа.
* Использование промо кодов: Пользователь может вводить промо коды для получения скидок или специальных предложений.
* Оформление заказа: Пользователь может оформлять заказ, выбирая нужные товары, указывая адрес доставки и выбирая способ оплаты.
* Оплата заказа: Пользователь может производить оплату заказа онлайн, используя доступные способы оплаты.
* Отслеживание заказов: Пользователь может отслеживать статус и местоположение своих заказов в режиме реального времени.
* Редактирование профиля: Пользователь имеет возможность редактировать свой профиль, включая изменение личной информации и настроек безопасности.

1. **Сущности в базе данных и связи между ними**

Изучение предметной области и составленный впоследствии функционал показали, что можно выделить 9 коллекций, которые позволяют полностью охватить обозначенный функционал (рис. 2.1). Из них 5 основных:

* пользователь (Users) – коллекция, предназначенная для хранения основных данных о пользователе, таких как имя, телефон, электронная почта, пароль;
* заказы (Orders) – коллекция, которая является подколлекцией коллекции пользователей, предназначенная для хранения заказов;
* продукт (Products) – коллекция, предназначенная для хранения данных о продукте;
* производитель (Brands) – в этой коллекции хранятся данные о производителях;
* категории (Categories) – коллекция, предназначенная для хранения информации о категориях продуктов;

Остальные коллекции вспомогательные, и нужны для того, чтобы поддерживать целостность информации

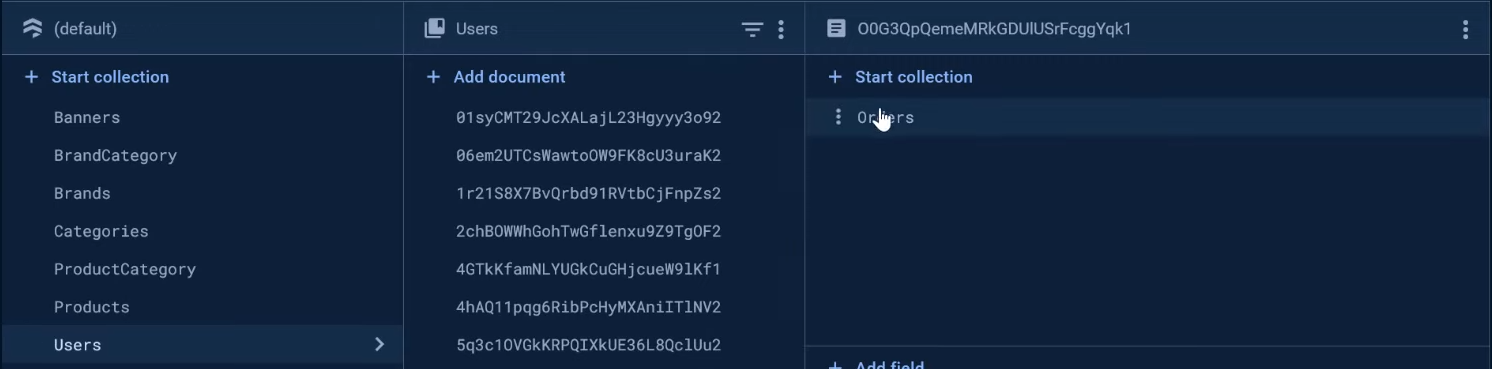


Рисунок 2.1 — Схема коллекций

Отдельно стоит упомянуть способ хранения изображений в базе данных Firestore. Вместо хранения изображений в виде относительных ссылок на файловую систему или абсолютных ссылок на внешние ресурсы, Firestore предлагает использовать Firebase Storage для хранения и обработки изображений.

При загрузке изображений в Firestore, каждое изображение сохраняется в Firebase Storage, которое является облачным хранилищем отдельно от базы данных. В Firestore сохраняется только ссылка на конкретное изображение в Firebase Storage.

В итоге, для хранения изображений в базе данных Firestore используется подход, при котором сами изображения сохраняются в Firebase Storage, а в Firestore хранятся ссылки на них, позволяя эффективно управлять изображениями и обеспечивая их доступность для клиентского приложения.

В результате была спроектирована необходимая модель базы данных. Была определена информация, которая будет хранится в таблицах.

2.2 Серверная часть приложения

При проектировании классов и интерфейсов для серверной части приложения на базе Flutter (используя язык Dart) и Firestore, важно стремиться к минимальной избыточности и связности между компонентами, а также обеспечивать корректность кода, поскольку это существенно влияет на работу всех подключенных клиентов. Для достижения этой цели можно использовать хорошо продуманное распределение логики сервера по разным уровням, определять формат объектов, используемых на каждом уровне, и выполнять преобразования объектов при их передаче между уровнями. Также полезно применять соответствующие паттерны проектирования.

В данном случае выбран архитектурный стиль REST для обмена данными между клиентами и сервисами. Он широко использует возможности протокола HTTP и, следовательно, является состоянием (stateless). REST является современным подходом, применяемым во многих веб-сервисах, и имеет множество реализаций различных функций на его основе, таких как аутентификация, выполнение операций CRUD и т. д. В REST-приложениях наиболее распространенным форматом передачи данных является JSON, поэтому предлагается использовать его для обмена данными между клиентами и сервером.

При разработке серверной части приложения на Flutter с использованием Firestore, следует учесть особенности Firestore и его API. Firestore - это гибкая, масштабируемая облачная база данных, обеспечивающая реальное время и синхронизацию данных между клиентами.

2.2.1 Уровни. API, Service, Repository

Приняв во внимание вышенаписанное, лучше всего разбить серверную логику на следующие уровни API, Service и Repository.

API-уровень. Это самый верхний уровень сервера, и на нём будут располагаться классы-контроллеры, методы которых будут описывать URL-адреса, на которые можно будет сделать вызов, и их характеристику. Также эти методы будут преобразовывать данные из формата, пригодного для сериализации, в формат, пригодный для сохранения данных в Firestore и работы с ними как с бизнес-объектами.

Service-уровень. На этом уровне находятся классы, ответственные за бизнес-логику, среди них как те, которые просто делегируют вызов к репозиторию, так и те, которые выполняют более сложную логику.

Repository-уровень. Этот уровень содержит классы-репозитории, которые используются для выполнения операций с Firestore. Методы этих классов могут обращаться к Firestore API для создания, чтения, обновления и удаления данных. Репозитории могут выполнять автоматически сгенерированные запросы к Firestore или использовать пользовательские запросы в зависимости от требований приложения.

Пример класса-контроллера приведен в приложении Б.

2.2.2 Объекты, использующиеся на различных уровнях, и их особенности

Использование различных типов объектов для предоставления данных является хорошей практикой при разработке любых типов программных продуктов, а при наличии разных уровней в пределах одного приложения, без такого подхода обойтись попросту невозможно. Для этого использовались два паттерна одновременно: DTO-объект и Immutable-объект. Следуя первому паттерну, мы будем проектировать классы, содержащие только базовые типы, либо те типы, которые легко сериализуются. Его преимущество фактически уже было названо: понятная и предсказуемая сериализация, то есть разработчик всегда будет знать, как экземпляр класса будет выглядеть после превращения его в JSON-строку. Второй паттерн было решено применить по причине как раз-таки тех ограничений, что накладывает первый паттерн. DTO-объекты содержат в себе данные исключительно для передачи, более с ними никакие операции совершать не рекомендуется. Вывод о том, что такой объект должен быть неизменяемым, очевиден, более того, по мере роста проекта это поможет защитить программиста от дальнейших ошибок.

Второй тип объекта, которые будет использоваться на сервере – Entity-объект. Entity-объекты представляют собой объекты, которые включают данные, хранящиеся в Firestore или полученные из него. Они могут быть использованы на всех уровнях серверной логики. API-уровень может преобразовывать DTO-объекты в Entity-объекты для передачи их на Service-уровень, например, при обработке POST-запросов, а также преобразовывать Entity-объекты в DTO-объекты при запросе данных. Service-уровень использует Entity-объекты для выполнения бизнес-логики и взаимодействия с Repository-уровнем. Repository-уровень отвечает за операции с данными Firestore и может работать с Entity-объектами.

При работе с различными типами объектов и их взаимодействии важно определить, как будет осуществляться конвертация между ними. Для этого можно использовать специальные классы, называемые "мапперами" или "конвертерами", которые содержат логику преобразования данных из одного типа объекта в другой. В Flutter (Dart) с Firestore существует выбор библиотек, которые помогают автоматизировать и упростить этот процесс конвертации данных между различными объектами.

2.2.3 Безопасность. Аутентификация и авторизация

Важнейшим аспектом проектирования серверной части приложения является то, как пользователи будут входить в систему и как система будет понимать, что очередной присланный запрос прислан уже находящимся в системе пользователем.

Firebase Authentication предоставляет набор инструментов для аутентификации пользователей, включая поддержку различных провайдеров, таких как Google, Facebook, Twitter и другие. Он также позволяет создавать пользователей с помощью адреса электронной почты и пароля. Firebase Authentication обеспечивает безопасность ваших пользователей, управление сеансами пользователей и позволяет легко интегрировать аутентификацию в ваши клиентские приложения.

После того, как пользователь успешно пройдет процесс аутентификации с помощью Firebase Authentication и Google Auth, серверная часть может использовать JWT-токен (JSON Web Token) для поддержания состояния пользователя. JWT-токен представляет собой компактную и самодостаточную форму представления информации об аутентифицированном пользователе. Он содержит зашифрованную информацию о пользователе, такую как идентификатор пользователя, разрешения и срок действия токена.

Когда пользователь выполняет запрос к серверу, он должен предоставить JWT-токен в заголовке запроса или в другом безопасном месте. Серверная часть может проверить подлинность токена и его срок действия, а затем разрешить или отклонить запрос в зависимости от результата проверки. Это позволяет поддерживать состояние пользователя без хранения его данных на сервере и обеспечивает безопасность запросов.

2.2.4 Хранение фото

Реализовать функционал хранения фотографий можно как и вручную, храня их на собственном сервере, так и воспользовавшись уже готовым решением в облаке, такими как Firestore Database. В рамках работы над данным курсовым проектом, был выбран данный вариант, так как он предполагает наименьшие трудозатраты, однако соответствующие классы на серверной части приложения сделаны таким образом, что хранилище фотографий можно будет при необходимости легко сменить с облачное на удалённое либо локальное.

2.3 Клиентская часть приложения

Клиентская часть приложения не менее важна, чем серверная. Это то, с чем будет контактировать пользователь, то, от чего в первую очередь зависит будут ли впечатления от использования положительными или отрицательными. Здесь стоит понимать, исходя из специфики проекта, в первую очередь разрабатывается мобильное приложение и самые строгие требования со стороны пользователя будут предъявляться именно к нему.

Точно так же, как и серверная часть приложения, как и практически любой другой проект, который имеет риск сильно разрастись в будущем, лучше всего разбить логику клиентской части приложения на части таким образом, чтобы на этой основе можно было каждую часть реализовывать последовательно и независимо от других частей. Приняв во внимание все требования к функционалу, а также лучшие практики и паттерны, использующиеся при разработке Android-приложений, логику клиентской части приложения можно разделить на две категории: включающую в себя взаимодействие с пользовательским интерфейсом и ту, которая связана с бизнес-логикой.

В первую категорию будут входить:

* виджеты и разметка экранов;
* виджеты и разметка диалогов;
* адаптеры списков;
* стили, иконки.

Во вторую:

* классы, ответственные за внедрение зависимостей;
* функции, отправляющие запросы к API;
* прочие дополнительные и вспомогательные классы.

Понятно, что иерархически классы из категории с пользовательским интерфейсом стоят ниже, чем классы из категории с бизнес-логикой, и именно они будут пользоваться классами из второй категории.

2.3.1 Виджеты, управление состоянием виджетов, декларативный UI, фрагменты и диалоги, навигация

Так как было принято решение разрабатывать приложение на Flutter, были изучены различные способы управления состояниями виджетов, так как все объекты во флаттере являются виджетами, которые изменяют свое состояние и перерисовываются во время работы приложения. В больших проектах есть смысл в использовании такого паттерна проектирования как BLoC, который является посредником между уровнем представления и базой данных. В данном приложении было принято решение использовать метод setState() для контроля состояний.

Виджетами является всё то, что связано с интерфейсом: код, реагирующий на нажатия кнопок и на остальные события, код инициализации состояния, очистки состояния, валидация. Они по большей части выполняют работу по отображению пользователю какого-либо динамического контента, путём взаимодействия с соответствующей ему моделью, а также в ответ на действия пользователя. Важно отметить, что приложение будет построено целиком с использование Stateless и Statefull виджетах. Это стандартная практика при разработке приложений на Flutter, так как такая архитектура позволяет легко использовать одни и те же виджеты по нескольку раз, появляется возможность применять продвинутые библиотеки с навигацией и манипулировать их жизненным циклом.

Разметка в разработке на Flutter представляет собой декларативный UI, в которых с помощью классов виджетов описываются различные элементы интерфейса. Все сложные виджеты строятся на основе простых путем наследования одного от другого.

Навигация – очень важный для мобильного приложения элемент, при её проектировании необходимо поддерживать логическую связь между виджетами, давать возможность перейти на какой-либо виджет из одних виджетов и не позволять это делать из других. Необходимый список виджетов будет таким:

* с главным экраном, который будет содержать навигационное меню;
* со списком производителей;
* со списком товаров;
* с любимыми товарами;
* с корзиной пользователя;
* с профилем;
* с детальным описанием продукта;
* и некоторое количество диалогов, которые, по сути, тоже являются виджетами и участвуют в навигации наравне с ними.

Для управления навигацией в приложении используется пакет Get, который предоставляет удобные инструменты для навигации между виджетами. Он предоставляет более простой и декларативный подход к навигации, чем стандартный класс Navigator. С его помощью можно будет также строить граф переходов, где будут отображены стрелками направления переходов.

2.3.2 Внедрение зависимостей

Одной из самых распространённых практик при разработке мобильных приложений – это использование различных библиотек, внедряющихся виде зависимостей в pubsec файл. Для Flutter существует множество библиотек, предоставляющих различный функционал.

Во время разработки приложения были использованы такие библиотеки как:

* firebase\_core, firebase\_auth, cloud\_firestore и firebase\_storage, необходимые для выполнения запросов от приложения к Firebase;
* connectivity\_plus, для проверки соединения с Firestore;

2.3.3 Дизайн и расположение элементов

В виду своей кроссплатформенности, во Flutter присутствуют элементы Material Design для устройств, работающих по ОС Android и Cupertino для IOS, так как все приложения в App Store должны иметь схожий дизайн и не путать пользователей.

Приложение имеет в себе 4 главных экрана, являющиеся отправными точками для переходов на все остальные виджеты: экран с категориями, продуктами, брендами и с профилем. Для того, чтобы перемещаться между ними, используется ряд с кнопками в нижней части экрана, сделано это по той причине, что на рынке сейчас доминируют телефоны с большой диагональю, и дотянуться до нижней части экрана всегда проще всего. Перемещение по фрагментам назад осуществляется либо с помощью аппаратной кнопки назад, либо с помощью его программного аналога в левом верхнем углу.

При выборе цветовой гаммы приложения и вида интерфейса важно помнить для какой аудитории делается приложения и какие цели те или иные цвета и геометрические формы преследуют. Основной цвет приложения изменяется в зависимости от примененной пользователем системной расцветки.

В приложении используются различного вида кнопки, как имеющие заливку, так и нет. Первый тип используется для тех кнопок, которые делают какие-то существенные действия: добавление продуктов, новостей, выполняют вход в приложение и т.д. Второй же тип используется для менее важных действий, например, добавления товара в избранное или переход на новый экран. Такое разделение позволяет заострить внимание пользователя на важных действиях и не распылять его внимание на менее существенные.

Наличие виджетов, которые содержат в себе несколько типов полей ввода, а также имею несколько способов предоставлять пользователю информации, предоставляет дополнительные сложности при проектировании их дизайна, и при работе над ними нужно быть особенно внимательным, их нужно составлять таким образом, чтобы уместить достаточно количество информации на экран мобильного телефона, при этом не запутав пользователя. Для решения такой проблемы чаще всего используется группировка элементов, она может осуществляться, например, с помощью использования разделителей. Следуя последним трендам в дизайне, решено было использовать разделение контента на своеобразные «карточки» с закруглёнными углами. При этом стоит учитывать, что более натурально будет выглядеть, если карточка сверху будет иметь закруглённые углы только внизу, карточка посередине – и внизу, и вверху, а карточка внизу – только вверху.

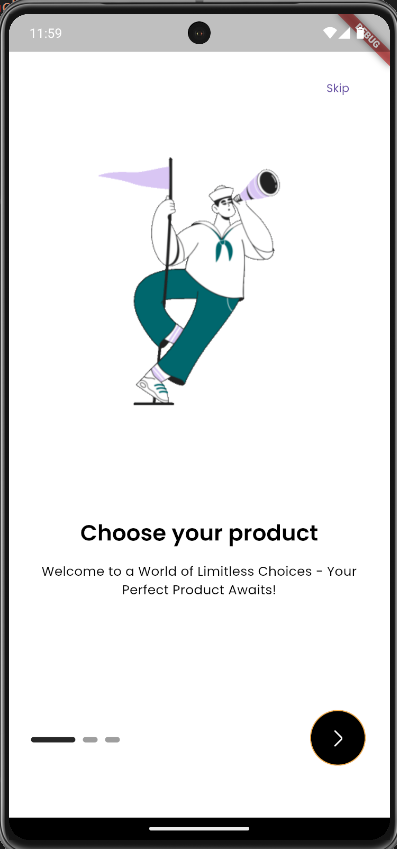
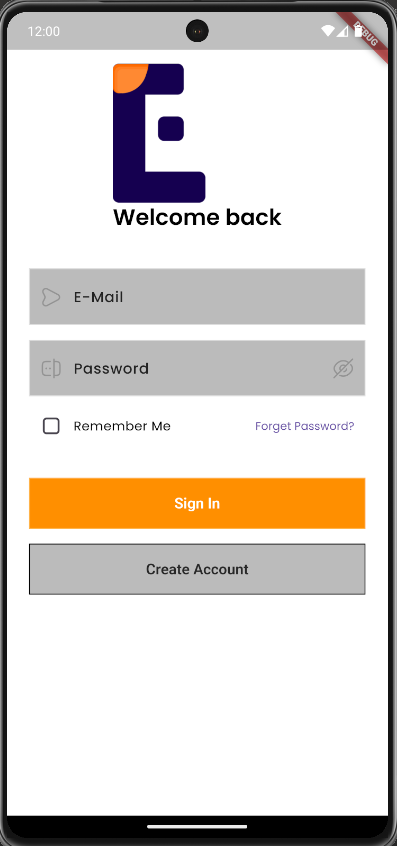
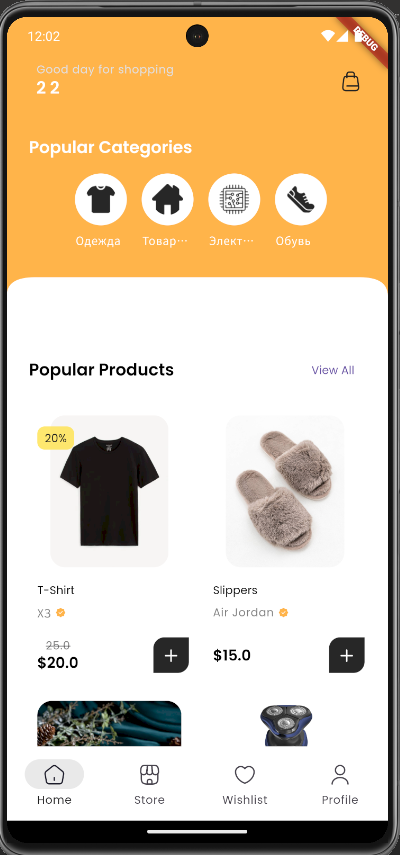
  

Рисунок 2.3.3 — Основные экраны

2.4 Итоги раздела

По итогу данного раздела, у нас имеется детально проработанный план разработки приложения, как серверной и клиентской части, так и веб-приложения. Проработана внутренняя архитектура самих частей приложения, проработаны вопросы того, какого вида классы там будут использоваться, а также вопросы связи всех частей между собой и вопросы, связанные с безопасностью. Проект готов к реализации.

3 Реализация приложения

3.1 Серверная часть

Для разработки серверной части приложения был использован Firebase, платформа, предоставляемая Google, которая обеспечивает облачные услуги для разработки и развертывания приложений. Firebase предоставляет широкий набор инструментов и сервисов, которые упрощают разработку серверной части приложения и управление базой данных.

3.1.1 Конфигурация

Конфигурация серверной части приложения с использованием Firestore осуществляется через несколько ключевых компонентов и настроек.

Необходимо установить и настроить Firebase Admin SDK. Этот SDK предоставляет API для работы с различными сервисами Firebase, включая Firestore. Инициализация и аутентификация Firebase Admin SDK выполняются в начале серверного приложения с использованием файла ключа службы Firebase, полученного из Firebase Console. Этот файл содержит секретные ключи для аутентификации сервера и доступа к Firestore.

После инициализации Firebase Admin SDK следует настроить подключение к Firestore. Конфигурационные параметры, такие как адрес сервера Firestore, учетные данные аутентификации и другие настройки, могут быть указаны в файле .env или другом безопасном месте, чтобы защитить конфиденциальные данные.

Отдельно стоит уделить внимание форматам даты и времени, которые будут использоваться при сериализации. Это важный момент, так как очевидно, что тип, представляющий время и дату, является сложным, а значит, что на разных платформах, в разных языках программирования, этот тип может быть интерпретирован совершенно по-разному, поэтому возникает необходимость введения определённого единого стиля. Таковым был выбран стандарт ISO 8601, так как он является самым распространённым и имеет поддержку во всех используемых языках программирования. Он выглядит так: «YYYY-MM-DDThh:mm:ss».

Вся эта конфигурация позволит взаимодействовать с Firestore из вашего серверного приложения и выполнять операции чтения и записи данных в базу данных Firestore.

3.1.2 Реализация REST API. Контроллеры

Классы-контроллеры в приложении, использующем Firestore, представляют API. У них есть методы, которые вызываются при поступлении HTTP-запросов на сервер. Шаблон описания каждого контроллера включает декораторы, которые указывают его характеристики. В данном случае, мы будем использовать декоратор, чтобы указать, что это REST-контроллер. Внутри класса контроллера определяются методы, каждый из которых помечен аннотацией, указывающей тип HTTP-метода, данные, принимаемые методом, и данные, возвращаемые им.

При использовании Firestore с помощью Firebase Admin SDK, валидация входных данных может быть реализована с помощью встроенных механизмов Firestore. Firestore позволяет определить правила доступа и ограничения на структуру данных в базе. Это позволяет автоматически проверять входные данные на соответствие заданным правилам и возвращать ошибку, если данные не проходят валидацию.

Кроме того, в классы-контроллеры могут быть внедрены сервисы для работы с Firestore. Это позволяет контроллерам использовать сервисы для выполнения операций чтения и записи данных в Firestore. Сервисы могут быть созданы и настроены отдельно, а затем внедрены в контроллеры с помощью механизма Dependency Injection.

Таким образом, классы-контроллеры играют важную роль в организации API и взаимодействии с Firestore, обеспечивая обработку HTTP-запросов и использование сервисов для работы с данными в базе Firestore.

3.1.3 Сервисы.

Сервисы – классы, которым контроллеры делегируют выполнение различных действий, после поступления на них запроса. Они также помечаются своим особым декоратором, но скорее для того, чтобы фреймворк смог создать по ним объекты и включить их в свой пул для последующего внедрения зависимостей.

Как было уже описано ранее, большинство сервисов лишь выполняют различные действия с репозиториями, иногда лишь делая небольшие преобразования в объектах, чтобы ORM-система смогла правильно их обработать. Работа сервисов совершается в неблокирующем асинхронном режиме. В конструктор сервиса так же встраиваются различные модели.

3.1.4 Репозитории. Взаимодействие с Firestore

Для взаимодействия с базой данных Firestore вместо ORM sequelize-typescript используется Firebase Admin SDK и Firestore API. В Firestore данные организованы в коллекции, которые могут содержать документы. Каждый документ представляет собой набор полей со значениями. Для этого создаются Модели – классы, представляющие собой коллекции в Firestore. Эти классы принимают на вход методы репозиториев. Репозиторий в нашем случае – это сами классы моделей, которые позволяют работать с ними как с репозиториями.

При создании коллекции и документа в Firestore, нет необходимости явно определять схему или ограничения для полей. Firestore является гибкой NoSQL базой данных, в которой каждый документ может иметь свою структуру и поля. Ограничения и валидацию данных можно выполнять на уровне приложения, перед отправкой данных в Firestore.

Пример репозитория приведен в приложении А.

3.1.5 DTO. Конвертация разных типов объектов

DTO – важнейшая часть API-системы, так как эти объекты представляют то, как должны будут выглядеть запросы и ответы к ней. Легко понять тот факт, что если запросы будут по большей части одинаковыми, то ответы могут отличаться как по своему наполнению, так и по структуре. Например, если приходит запрос на получение одного пользователя, то не имеет смысла отправлять вместе с ним полную информацию о всех консультациях, в которых он поучаствовал, так как большинство из отправленных данных не будет использоваться получателем и канал передачи будет впустую нагружаться. Одним из решений перечисленных проблем является определение класса для каждого из типов запросов. Главным его недостатком является то, что придётся писать достаточно большое количество мало чем отличающегося кода. Тем не менее, преимуществ такой подход несёт больше, поэтому был выбран именно он. Для реализации этого подхода использовались:

* классы для каждого тела запроса;
* классы, которые наследовали либо все поля либо часть из них.

Пример такого класса можно увидеть в приложении Б.

При валидации используется npm-пакет class-validator.

Так же в приложении реализована загрузка тела с Content-Type двух видов multipart/form-data и application/json. Так как при загрузке информации с телом переданным как multipart/form-data вся информация приходит в виде строки, испольщовался пакет class-transorm для преобразования типа строк в тип указынный в DTO с помощью декораторов @Transform

3.1.6 Аутентификация и авторизация

Firebase предоставляет встроенные функции аутентификации и авторизации, которые облегчают процесс работы с пользователями. Для аутентификации пользователей используется Firebase Authentication, а для авторизации — Firebase Firestore и его правила доступа.

Firebase обрабатывает процесс аутентификации и генерирует уникальный идентификатор пользователя (UID), который можно использовать для идентификации и управления доступом к данным.

Вместо JWT-токенов, Firebase Authentication использует специальные токены Firebase, которые автоматически генерируются и проверяются серверной частью Firebase при выполнении запросов к API.

Для авторизации пользователей и управления доступом к данным в Firestore используются правила доступа (Firestore Security Rules). Правила доступа позволяют определить, какие пользователи имеют доступ к определенным коллекциям и документам в Firestore, основываясь на их UID или других атрибутах.

Таким образом, вместо пакета passport и JWT-токенов, в приложении на основе Firebase и Google Auth используются встроенные функции Firebase Authentication, Google Auth и Firestore Security Rules для обеспечения аутентификации и авторизации пользователей.

3.2 Клиентская часть

Разработка приложений под Android имеет свою специфику и во многом отличается от разработки для других платформ. Изначально в SDK идут только самые базовые компоненты, которые очень сложно использовать для того, чтобы реализовать требуемый функционал, однако существует большое множество библиотек и фреймворков, предоставляемых в том числе и разработчиками Google.

Самые главные сторонних компонентов, которые использовались, это:

* flutter\_http – библиотека, упрощающая совершение вызовов к API;
* Future– библиотека для выполнения асинхронного кода;
* Material и Cupertino – библиотеки, предоставляющая компоненты из Material UI и Cupertino.

Языком программирования был выбран Dart.

3.2.1 Виджеты, разметка, диалоги.

Так как в проекте используется DataBinding, то при создании вида каждого фрагмента приходится использовать специальные классы, которые генерируются этим фреймворком. Для этого, а также для того, чтобы сократить количество остального повторяющегося кода, был создан абстрактный класс StandardBoundFragment. Туда, в качестве generic-параметров, посылается класс ViewModel и класс, который обеспечивает привязку данных. Также в качестве аргументов конструктора туда посылаются методы создания класса с привязками, связывания ViewModel, внедрения зависимостей и уничтожения связанного компонента. Далее в нужных местах эти методы вызываются, а также инициализируются поля, которые, как показала практика, будут использоваться почти в каждом фрагменте: поле с классом связывания и поле с ViewModel. Также этот класс предоставляет два метода для реализации: initElements и subscribeUi. Первый нужен для того, чтобы инициализировать UI-компоненты, а второй, чтобы подписаться на обновления полей из ViewModel.

Диалоги, разработанные в данном проекте, тоже являются виджетами. Они используются при вызове функции удаления элемента для того чтобы подтвердить операцию. Диалоговое окно представляет собой объект класса AlertDialog, который включает виджет Text – для отображения текста сообщения и виджеты MaterialButton – для расположения в нес кнопок отмены и подтверждения. При нажатии кнопку подтвердить вызывается функция delete, в теле которой происходит отправка запроса на удаление. После удаления сервер возвращает удаленный объект и при помощи метода setState() происходит перерисовка виджета.

3.2.2 Внедрение зависимостей

При разработке мобильной части приложения повсеместно используется внедрение зависимостей. Помимо того, что оно предоставляет единое место для создания объектов, он также предоставляет возможности взаимодействия с жизненным циклом фрагментов, что крайне полезно в случае, если необходимо иметь ViewModel, использующуюся более чем в одном фрагменте, например, когда создаётся новое соревнование. Классы для внедрения зависимостей расположены в специально созданной для этого директории и могут разделяться на три типа: компоненты, модули и области видимости. Помимо этого, используется собственный класс, который называется ComponentKeeper, он содержит в себе все компоненты, обёрнутые в класс DestroyableComponent. Всё это нужно для того, чтобы можно было управлять жизненным циклом компонентов, уничтожая их в нужный момент (например, при выходе из фрагмента). Области видимости же нужны для того, чтобы Dagger понимал, когда используется новый компонент, а когда тот же самый. Каждому фрагменту соответствует свой компонент, при этом все они объединены в один большой, принадлежащий всему приложению, который создаётся в классе MyApplication, и через который создаются все остальные компоненты.

3.3 Итоги раздела

В итоге была разработана мобильная и серверная часть приложения, а также проверено и отлажено их взаимодействие. Основная сложность была в грамотной реализации поддержки соревновательных сессий, чтобы одни клиенты исправно отправляли координаты на сервер, а другие клиенты их получали, а также в том, чтобы правильно реализовать обработку этих локаций. Также много сложностей возникало с правильным структурированием всех взаимодействующих классов в мобильном приложении, что было, несмотря на это, успешно выполнено.

4 Тестирование, проверка работоспособности и анализ полученных результатов

Тестирование проекта выполнялось в течение разработки посредством таких средств, как отладка и логгирование. После введения нового либо изменения существующего компонента программы, он и зависящий от него код также подвергался тестированию. Анализ полученных результатов происходил по факту получения ошибок и проблемные места сразу исправлялись. После завершения разработки программы по нескольку раз были подвергнуты испытаниям все элементы управления в пользовательском интерфейсе, также они тестировались по ходу написания записки. Если происходили ошибки, то их причина и местоположение в коде легко фиксировались благодаря ведению логов.

После написания приложения также проводились различные тесты, и проверки, такие как проверка на корректное завершение соревнования, проверка на корректную работу таймера, проверка корректного создания соревнования и команды, а также их удаления и изменения.

В итоге приложение стало хорошо отлаженным и готовым к полноценному использованию.

5 Руководство по использованию

Для того, чтобы установить приложение на телефон, Вам необходимо иметь версию Android 7.0 и выше (версию API 22 и более). Версию Вашего телефона можно узнать в настройках.

Если Ваша версия удовлетворяет требованиям, то можно устанавливать приложение на телефон.

После установки приложения на экране появится иконка и подпись “e\_commerce” (рис.5.1)

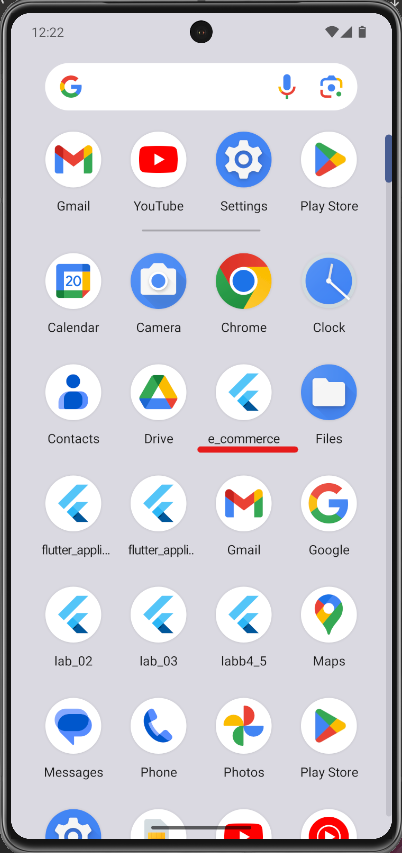


Рис 5.1 Иконка приложения

Изначально открывается экран, на котором находится форма авторизации, которая имеет заголовок, два текстовых поля и две кнопки: регистрации и авторизации(рис.5.2).

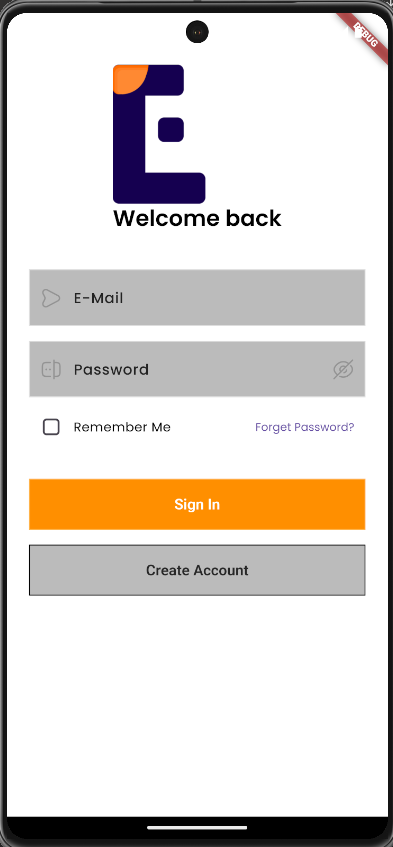


Рисунок 5.2 – Форма авторизации

Перед первым использованием приложения Вам необходимо зарегистрироваться. Для этого нужно заполнить поля логин и пароль валидными данными и нажать на кнопку “Create Account”.

Вы успешно прошли регистрацию и можете зайти в свой аккаунт. Для этого Вы заполняете email (имя пользователя) и вводите пароль (password) на форме входа, после нажатия клавиши “SING IN” Вы переходите в приложение.

Сразу открывается главный экран, где отображаются популярные товары (рис.5.3).

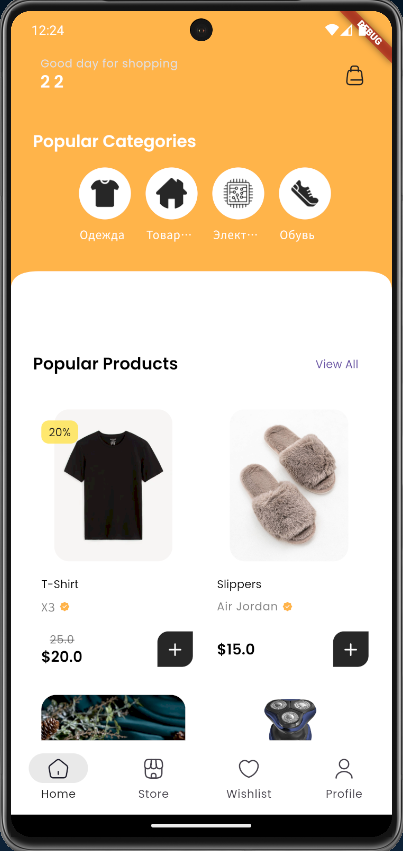


Рисунок 5.3 – Главная страница

Все элементы списка оформлены в виде карточек и при нажатии на определенный элемент списка будет открыт экран с подробным описанием продукта. Также на данном экране можно добавить товар в избранное кликнув на иконку сердечка. (рис. 5.4).

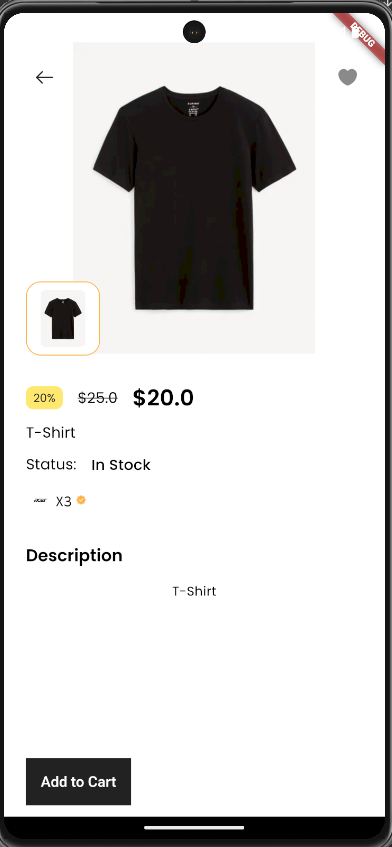


Рисунок 5.4 – Страница просмотра товара

Последним экраном является профиль, на котором просто выводиться информация об авторизованном пользователе.

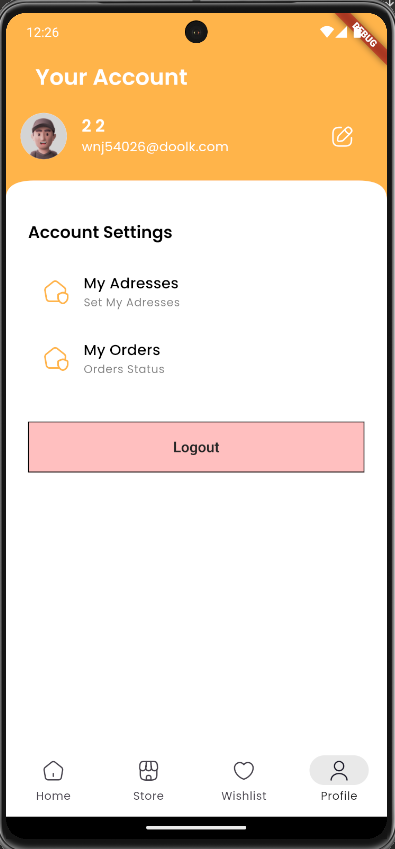


Рисунок 5.5 — Экран настроек пользователя

Заключение

При работе над курсовым проектом было проведено изучение предметной области, выявление в ней проблем и составление функционала программного средства, с их учётом, после чего последовало проектирование и разработка. Также в ходе работы проанализировано и проработано множество сторон и нюансов проектирования и разработки мобильного приложения, сервера, а также веб-приложения, например, выбор подходящих паттернов и технологий, составление и тестирование алгоритмов, структурирование проекта, построение пользовательского интерфейса, синхронизации данных и т.д.

В итоге были реализован весь заявленный функционал и готовое приложение вполне можно назвать подходящим для реального использования. Как мобильная, так и серверная части проекта имеют хороший потенциал для будущих модификаций во многом благодаря грамотному структурированию, внимательностью при написании кода, а также выборе известных и проверенных сопутствующих технологий. Было получено большое количество опыта по работе с такими фреймворками как Flutter, а также Firebase, улучшены навыки по проектированию, а также применению и связыванию друг с другом различных технологий.

Список литературы

1. REST-API [Электронный ресурс]. – Режим доступа: <https://habr.com/ru/post/483202/>. – Дата доступа: 16.05.2024.
2. StudFiles [Электронный ресурс]. – Режим доступа: https://studfiles.net/preview/page:2/. – Дата доступа: 16.05.2024.
3. Трехуровневая архитектура — Википедия [Электронный ресурс] – Режим доступа: https://ru.wikipedia.org/wiki/Трехуровневая\_архитектура. – Дата доступа: 16.05.2024.
4. Документация по Android [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://developers.google.com/android/> – Дата доступа: 16.05.2024;
5. Мартин, Р. Чистая архитектура и искусство разработки программного обеспечения / Р. Мартин. – Санкт-Петербург: Питер, 2019. – 352 с
6. Metanit Java [Электронный ресурс]. – Режим доступа: <https://metanit.com/java/android/>. – Дата доступа: 16.05.2024
7. Голощапов А. Google Android: программирование для мобильных устройств / Голощапов А. — СПб.: БХВ-Петербург, 2010. —

Приложение А

import 'dart:io';

import 'package:cloud\_firestore/cloud\_firestore.dart';

import 'package:flutter/services.dart';

import 'package:get/get.dart';

import 'package:strikeshop/data/services/firebase\_cloud\_storage.dart';

import 'package:strikeshop/features/shop/models/product\_model.dart';

import 'package:strikeshop/utils/constants/enums.dart';

import 'package:strikeshop/utils/exceptions/firebase\_exceptions.dart';

import 'package:strikeshop/utils/exceptions/platform\_exceptions.dart';

class ProductRepository extends GetxController {

static ProductRepository get instance => Get.find();

final \_db = FirebaseFirestore.instance;

Future<List<ProductModel>> getFeaturedProducts() async {

try {

final snapshot = await \_db

.collection('Products')

.where('IsFeatured', isEqualTo: true)

.limit(4)

.get();

return snapshot.docs.map((e) => ProductModel.fromSnapshot(e)).toList();

} on FirebaseException catch (e) {

throw SFirebaseException(e.code).message;

} on PlatformException catch (e) {

throw SPlatformException(e.code).message;

} catch (e) {

throw 'Something went wrong. Please try again.';

}

}

Future<List<ProductModel>> getAllFeaturedProducts() async {

try {

final snapshot = await \_db

.collection('Products')

.where('IsFeatured', isEqualTo: true)

.get();

return snapshot.docs.map((e) => ProductModel.fromSnapshot(e)).toList();

} on FirebaseException catch (e) {

throw SFirebaseException(e.code).message;

} on PlatformException catch (e) {

throw SPlatformException(e.code).message;

} catch (e) {

throw 'Something went wrong. Please try again.';

}

}

Future<List<ProductModel>> fetchProductsByQuery(Query query) async {

try {

final querySnapshot = await query.get();

final List<ProductModel> productList = querySnapshot.docs

.map((doc) => ProductModel.fromQuerySnapshot(doc))

.toList();

return productList;

} on FirebaseException catch (e) {

throw SFirebaseException(e.code).message;

} on PlatformException catch (e) {

throw SPlatformException(e.code).message;

} catch (e) {

throw 'Something went wrong. Please try again.';

}

}

Future<void> uploadProductData(List<ProductModel> products) async {

try {

final storage = Get.put(SFirebaseStorageService());

for (var product in products) {

final thumbnail =

await storage.getImageDataFromAssets(product.thumbnail);

final url = await storage.uploadImageData(

'Products/Images', thumbnail, product.thumbnail.toString());

product.thumbnail = url;

if (product.images != null && product.images!.isNotEmpty) {

List<String> imagesUrl = [];

for (var image in product.images!) {

final assetImage = await storage.getImageDataFromAssets(image);

final url = await storage.uploadImageData(

'Products/Images', assetImage, image);

imagesUrl.add(url);

}

product.images!.clear();

product.images!.addAll(imagesUrl);

}

if (product.brand != null) {

final brandImage =

await storage.getImageDataFromAssets(product.brand!.image);

final brandImageUrl = await storage.uploadImageData(

'Brands', brandImage, product.brand!.name);

product.brand!.image = brandImageUrl;

}

if (product.productType == ProductType.variable.toString()) {

for (var variation in product.productVariations!) {

final assetImage =

await storage.getImageDataFromAssets(variation.image);

final url = await storage.uploadImageData(

'Products/Images', assetImage, variation.image);

variation.image = url;

}

}

await \_db.collection("Products").doc(product.id).set(product.toJson());

}

} on FirebaseException catch (e) {

throw e.message!;

} on SocketException catch (e) {

throw e.message;

} on PlatformException catch (e) {

throw e.message!;

} catch (e) {

throw e.toString();

}

}

}

Приложение Б

import 'package:get/get.dart';

import 'package:strikeshop/data/repositories/products/product\_repository.dart';

import 'package:strikeshop/features/shop/models/product\_model.dart';

import 'package:strikeshop/utils/constants/enums.dart';

import 'package:strikeshop/utils/popups/loaders.dart';

class ProductController extends GetxController {

static ProductController get instance => Get.find();

final isLoading = false.obs;

final productRepository = Get.put(ProductRepository());

RxList<ProductModel> featuredProducts = <ProductModel>[].obs;

@override

void onInit() {

fetchFeaturedProducts();

super.onInit();

}

void fetchFeaturedProducts() async {

try {

isLoading.value = true;

//Fetch Products

final products = await productRepository.getFeaturedProducts();

//Assign Products

featuredProducts.assignAll(products);

} catch (e) {

SLoaders.errorSnackBar(title: 'Error', message: e.toString());

} finally {

isLoading.value = false;

}

}

Future<List<ProductModel>> fetchAllFeaturedProducts() async {

try {

//Fetch Products

final products = await productRepository.getAllFeaturedProducts();

return products;

} catch (e) {

SLoaders.errorSnackBar(title: 'Error', message: e.toString());

return [];

}

}

String getProductPrice(ProductModel product) {

double smallestPrice = double.infinity;

double largestPrice = 0.0;

if (product.productType == ProductType.single.toString()) {

return (product.salePrice > 0 ? product.salePrice : product.price)

.toString();

} else {

for (var variation in product.productVariations!) {

double priceToConsider =

variation.salePrice > 0.0 ? variation.salePrice : variation.price;

if (priceToConsider < smallestPrice) {

smallestPrice = priceToConsider;

}

if (priceToConsider > largestPrice) {

largestPrice = priceToConsider;

}

}

if (smallestPrice.isEqual(largestPrice)) {

return largestPrice.toString();

} else {

return '$smallestPrice - \$$largestPrice';

}

}

}

String? calculateSalePercentage(double originalPrice, double? salePrice) {

if (salePrice == null || salePrice <= 0.0) {

return null;

}

if (originalPrice <= 0) {

return null;

}

return (((originalPrice - salePrice) / originalPrice) \* 100.0)

.toStringAsFixed(0);

}

String getProductStockStatus(int stock){

return stock > 0 ? 'In Stock' : 'Out of Stock';