Министерство образования РМ
ГБПОУ РМ «Саранский государственный промышленно-экономический колледж»

СОГЛАСОВАНО
Зав. отделением
по специальности
_____ Л.А.Терентьева
«10» июня 2023 года

К защите допускается
Заместитель директора
по учебной работе
_____ Ю.Г. Кудаева
«10» июня 2023 года

ДИПЛОМНАЯ РАБОТА

Тема Разработка мобильного приложения по продаже компьютерной техники

Обозначение дипломной работы
СПЭК.09.02.07.23.30ДР

Автор дипломной работы	_____ (подпись)	_____ (дата)	С.В.Копейкин (инициалы, фамилия)
Руководитель дипломной работы	_____ (подпись)	_____ (дата)	И.П. Фирсова (инициалы, фамилия)
Рецензент	_____ (подпись)	_____ (дата)	А.А. Горьков (инициалы, фамилия)

Дата защиты _____

Протокол № _____

Оценка _____

Саранск
2023

Министерство образования РМ
ГБПОУ РМ «Саранский государственный промышленно-экономический колледж»

Дата выдачи задания
«02» марта 2023 года
Дата сдачи дипломной работы
«10» июня 2023 года
Специальность 09.02.07

УТВЕРЖДАЮ
Заместитель директора
по учебной работе
_____ Ю.Г. Кудаева
«02» марта 2023 года

ЗАДАНИЕ
НА ДИПЛОМНУЮ РАБОТУ

Тема Разработка мобильного приложения по продаже компьютерной техники

Утверждена приказом по колледжу от № 37 от 02.03.2023г.

Автор дипломной работы С.В. Копейкин

Студент группы ИСИП4А19

Содержание дипломной работы:

Введение

1 Анализ предметной области

1.1 Понятие архитектуры приложений

1.1.1 Понятие слоя UI

1.1.2 Понятие слоя данных

1.1.3 Понятие доменного слоя

1.2 Среды разработки мобильных приложений

2 Разработка приложения

2.1 Постановка задачи

2.2 Описание разработки мобильного приложения

Заключение

Список использованных источников

Заведующий отделением	_____	02.03.2023	Л.А.Терентьева
	(подпись)	(дата)	(инициалы, фамилия)
Руководитель дипломной работы	_____	02.03.2023	И.П.Фирсова
	(подпись)	(дата)	(инициалы, фамилия)
Задание принял к исполнению	_____	02.03.2023	С.В.Копейкин
	(подпись)	(дата)	(инициалы, фамилия)

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

На сегодняшний день смартфоны прочно вошли в быт людей. Множество программных продуктов в эру цифровых технологий, разрабатывается именно для мобильных телефонов. Вместе с этим растущая популярность мобильных устройств позволяет людям использовать свои смартфоны и планшеты в повседневной жизни для различных целей. Это может быть доставка еды, вызов такси, выбор одежды, заказ билетов, контроль и учет рабочей деятельности, и так далее. Однако эффективность и функциональность «мини» компьютеров не была бы доведена до столь высокого уровня без специализированных приложений. Поэтому именно для полноценного использования приложение по меньшей мере должно быть удобным, доступным, и функциональным. Тем самым с активным развитием рынка все больше предпринимателей и пользователей телефонов нуждаются в разработке качественных приложений для мобильных устройств.

Такая необходимость возникает из-за того, что большинство пользователей смартфонов являются молодежью и людьми среднего возраста, как правило, в возрасте от 14 до 40 лет, которые отдают предпочтение использованию сайтов через смартфоны и планшеты, нежели через стационарные компьютеры.

Также в виду того, что в последнее время все большее количество людей отдает предпочтение покупкам через интернет остро стоит вопрос создания максимально удобных приложений, позволяющих пользователям легко делать покупки.

Обычно использование сайта с мобильного устройства создает ряд неудобств. Пользователю необходимо помнить адрес сайта или выполнить поиск в одной из поисковых систем. Далее на сайте среди большого текста и огромного количества ссылок ищите нужный раздел. В связи с этим было решено разработать мобильное приложение, а именно, приложение сайта по продаже электронных устройств и техники, которое бы полностью включало

информацию с сайта, а также предлагало пользователю дополнительную интерактивность.

Таким образом, объектом исследования настоящей работы будет являться структура мобильного приложения на системе Android.

Предметом исследования - разработка мобильного приложения магазина по продаже компьютеров и их комплектующих.

Цель исследования состоит в том, чтобы разработать новое актуальное мобильное приложение интернет-магазина по продаже компьютеров и их комплектующих на базе операционной системы Android.

Для достижения цели работы были поставлены следующие задачи:

1) обозначить теоретические аспекты структуры приложений на системе Android;
2) определить требования к мобильному приложению;
3) разработать новое приложение на базе Android, которое удовлетворяло бы целевой аудитории пользователей смартфонов;
4) провести тестирование работоспособности созданного приложения
5) выделить практические аспекты создания нового приложения.

Работа состоит из введения, трех глав, заключения, списка использованных источников.

В первой главе описывается основная архитектура мобильного приложения на базе Android.

Во второй главе представлена постановка задачи и определение требований к новому мобильному приложению интернет-магазина по продаже компьютерной техники и его разработка

В заключении подводится итог, проделанный автором работы. В него также входят все использованные в ходе исследования источники.

1. Анализ предметной области

1.1 Понятие архитектуры приложений

Архитектура Android-приложения немного сложнее, нежели десктопной программы, хотя бы потому что компьютерное приложение в основном имеет одну точку входа с рабочего экрана или из меню «Пуска» приложений, а далее оно работает по принципу монолитного процесса. В то время как у Android-приложения архитектура выглядит по-другому.

В виду того, что мобильные устройства в большинстве своем ограничены в ресурсах: операционная система в любое время может отключить какой-нибудь процесс приложения, чтобы освободить место для новых. Не исключено и то, что компоненты приложения будут запускаться по отдельности или в неправильном порядке [3].

Именно поэтому, если компоненты приложений не рекомендуется использовать для хранения данных и их состояний, важно создать архитектуру, которая позволит масштабировать приложения, сделает их надёжнее и облегчит тестирование.

Архитектура приложения устанавливает границы между разными частями приложения и их ответственностями. Согласно общепринятым архитектурным принципам, каждое приложение Android состоит из трех слоёв:

– Слой UI, который отображает данные приложения на экране.

– Слой данных, который содержит бизнес-логику приложения и открывает доступ к данным приложения.

– Доменный слой, помогающий упростить и переиспользовать взаимодействия между слоями UI и данных [4]. Схема такой архитектуры находится на рисунке 1.

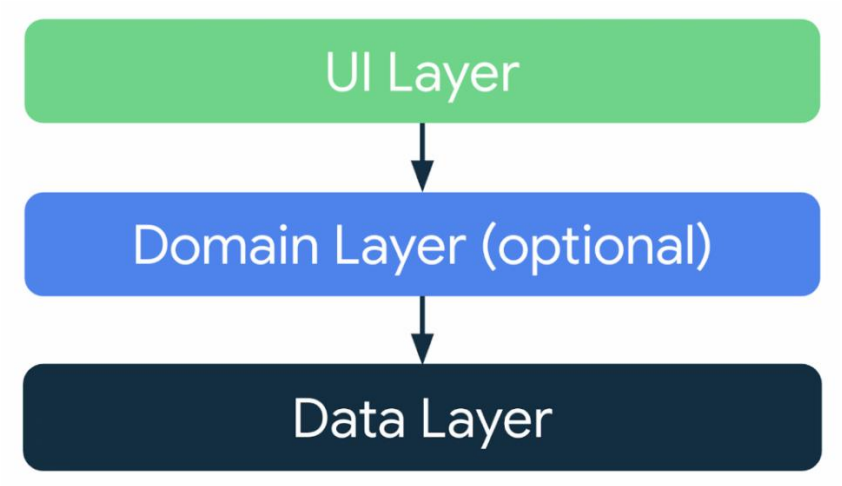


Рисунок 1 – схема классической архитектуры приложений

1.1.1 Понятие слоя UI

UI отображает данные приложения на экране: это основное место взаимодействия с пользователями приложений. Именно этот слой привязывается к экранам и помогает организовать взаимодействие со слоем бизнес-логики и работу с данными. Если отображаемые данные меняются — из-за взаимодействия с пользователем (например, нажатия кнопки) или внешнего воздействия (например, отклика сети) — UI должен своевременно обновиться и отразить изменения [2].

Стоит отметить, что этот слой может быть реализован с использованием любого предпочитаемого паттерна, к примеру, MVC, MVP, MVVM и других. Под самим термином UI подразумеваются UI-элементы (например, Activity и Fragment), которые отображают данные: вне зависимости от того, какими API они пользуются для этой цели, поэтому схему из рисунка 2 можно использовать на любую модель.

Поэтому слой UI должен:

– Принимать данные приложения и преобразовывать их в данные, которые UI легко сможет отрисовать.

– Принимать данные, которые может отрисовать UI, и преобразовывать их в элементы UI, которые будут показаны пользователю.

– Принимать события о вводе данных пользователем от элементов UI из пункта 2 и отражать в данных UI изменения, которые они вносят.

– Повторять шаги 1–3 необходимое количество раз [2].

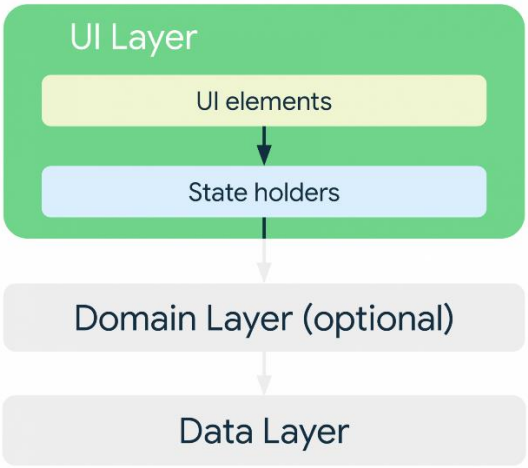


Рисунок 2 - Роль слоя UI в архитектуре приложения

1.1.2 Понятие слоя данных

Слой данных в свою очередь отвечает в первую очередь за получение данных из различных источников и их кэширование. Он реализуется за счет паттерна Repository. Схему слоя данных можно найти на рисунке 3.

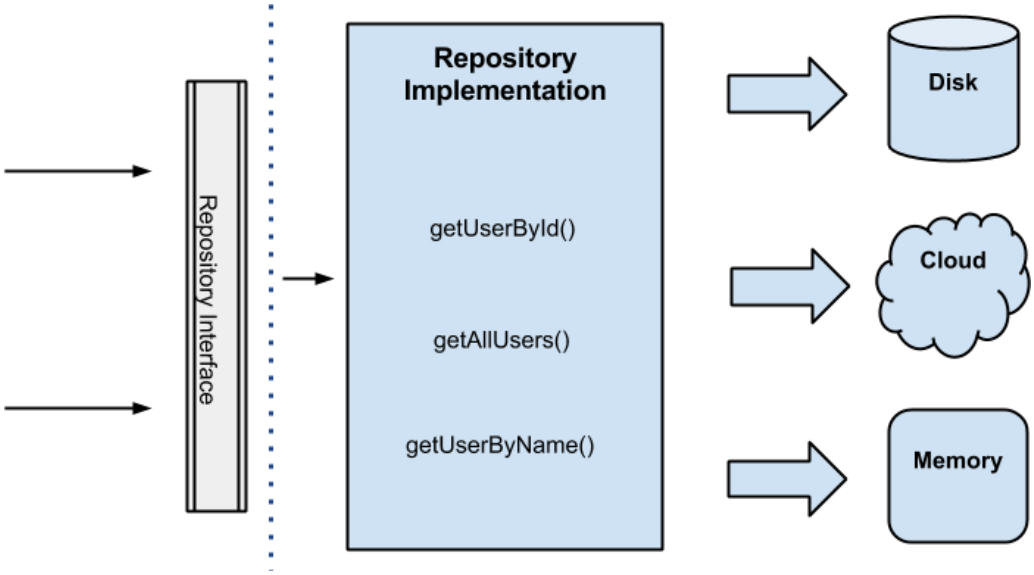


Рисунок 3 – Схема слоя данных

В каждом классе Repository может содержаться множество классов DataSource или в то же время не быть ни одного. Отдельный класс Repository следует создать для каждого уникального типа данных в приложении. Например, может быть класс MoviesRepository для данных, связанных с фильмами, или класс PaymentsRepository для данных, связанных с платежами.

Классы Repository выполняют следующие задачи:

– Предоставляют доступ к данным для остальных элементов приложения.

– Централизуют изменения в данных.

– Разрешают конфликты между несколькими классами DataSource.

– Абстрагируют источники данных от остальных элементов приложения.

– Содержат бизнес-логику.

К примеру, когда необходимо получить какого-то пользователя по имени, реализация репозитория проверяет наличие этого пользователя в локальном хранилище, при отсутствии сохраненного пользователя она отправляет запрос на сервер и получает ответ, который сохраняется в локальное хранилище и возвращается. Или, к примеру, репозиторий может всегда обращаться к серверу, а уже в случае ошибки возвращать сохраненный результат [3].

Стоит отметить, что остальные слои в иерархии ни в коем случае не должны получать доступ к классам data source напрямую. Переход на слой данных всегда должен осуществляться через Repository классы. Используя Repository-классы в качестве точек входа, различным слоям архитектуры приложения дается возможность масштабироваться независимо от остальных слоёв.

1.1.3 Понятие доменного слоя

Говоря о последнем из слоев, схему которого можно увидеть на рисунке 4, то, как правило, доменный слой отвечает за инкапсуляцию сложной или

простой бизнес-логики, которую пере используют несколько ViewModel. Этот слой не является обязательным, так как он требуется не всем приложениям. Его следует внедрять только когда существует такая необходимость при работе со сложной логикой или для переиспользования этой самой логики.

Классы этого слоя, как правило, называют UseCase или Interactor. Каждый UseCase должен отвечать только за одну функциональность. Чтобы классы оставались облегчёнными, каждый класс UseCase должен отвечать только за одну функциональность и не должен содержать изменяемых данных. С изменяемыми данными лучше работать в слоях UI или данных.

К примеру, в приложении может быть один класс GetTimeZoneUseCase и несколько ViewModel, которые в зависимости от часового пояса отображают соответствующее сообщение на экране [2].

Главными преимуществами доменного слоя являются:

– Помощь в избежании дублирования кода.

– Улучшение читаемости кода в классах, которые используют классы из доменного слоя.

– Облегчение тестирование приложения.

– Добавляет возможность избежать «раздутых» классов благодаря разделению ответственностей.

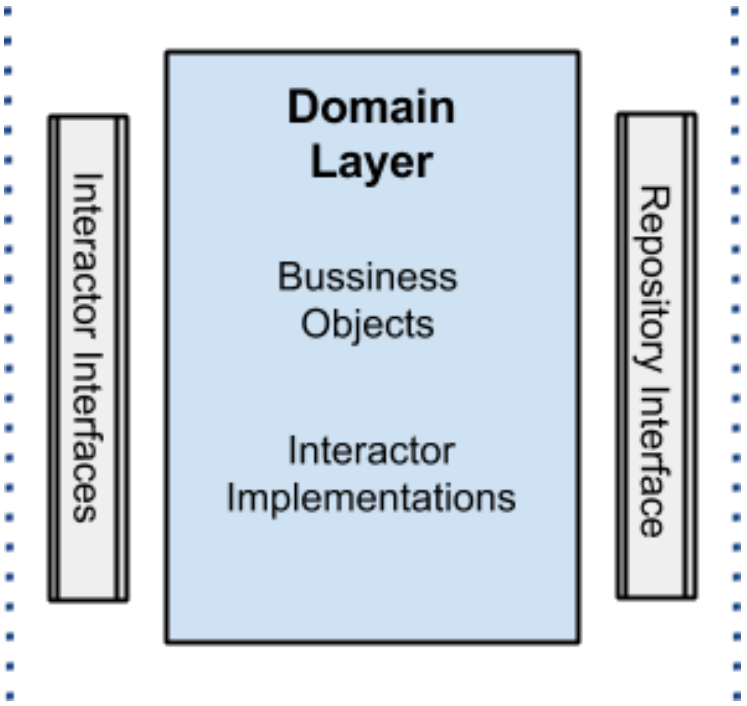


Рисунок 4 – Схема доменного слоя

1.2 Среды разработки мобильных приложений

1) Android Studio

Android Studio предоставляет все необходимые инструменты и функции для создания, отладки и тестирования мобильных приложений под Android. Она основана на популярной IntelliJ IDEA от JetBrains и обладает широким набором функциональных возможностей, специально адаптированных для разработки Android-приложений.

Достоинства

- Официальная среда разработки для Android-приложений, предлагаемая Google.
- Полная интеграция с Android SDK и множеством инструментов для разработки.
- Обширные возможности отладки и профилирования приложений.
- Большое сообщество разработчиков и обширная документация.

Недостатки

– Используется в основном для разработки приложений под платформу Android, не поддерживает разработку для других платформ.

– Может потребовать значительных ресурсов системы.

2) Xcode

Xcode позволяет разрабатывать приложения как для мобильной платформы iOS, так и для настольной платформы macOS. Она поддерживает различные языки программирования, включая Objective-C и Swift, и предоставляет инструменты для создания пользовательского интерфейса, обработки данных, работы с сетью и других функций.

Достоинства

– Официальная среда разработки для iOS-приложений, предлагаемая Apple.

– Интегрированный набор инструментов для разработки, отладки и профилирования.

– Поддержка языков программирования Swift и Objective-C.

– Хорошая интеграция с другими инструментами Apple, такими как Simulator и TestFlight.

Недостатки

– Доступна только для macOS, что ограничивает разработку iOS-приложений на других операционных системах.

– Ограничена поддержкой только для платформы iOS.

3) React Native

Основной средой разработки для React Native является любая интегрированная среда разработки (IDE) или текстовый редактор, который поддерживает разработку JavaScript-приложений.

Достоинства

– Позволяет разрабатывать приложения под Android и iOS, используя общий код на JavaScript.

– Поддерживает множество готовых компонентов и библиотек для ускорения разработки.

– Быстрая разработка и горячая перезагрузка для быстрого просмотра изменений.

– Хорошая производительность благодаря использованию нативных компонентов.

Недостатки

– Возможны некоторые ограничения в функциональности или производительности по сравнению с полностью нативными приложениями.

– Иногда требуется дополнительная настройка для обеспечения полной совместимости с платформами.

4) Flutter

Основной средой разработки для Flutter является Flutter SDK, который предоставляет все необходимые инструменты для создания кросс-платформенных мобильных приложений.

Достоинства:

– Позволяет разрабатывать кросс-платформенные приложения для Android и iOS с использованием одного и того же кода на языке Dart.

– Быстрая разработка и горячая перезагрузка для мгновенного просмотра изменений.

– Высокая производительность и возможность создания красивого пользовательского интерфейса.

– Широкий выбор готовых виджетов и библиотек.

Недостатки

– В некоторых случаях может потребоваться настройка для полной совместимости с платформами.

– Некоторые функции могут быть доступны с задержкой по сравнению с нативными разработками.

Безусловно все вышеперечисленные среды программирования хороши и подходят для разработки мобильного приложения. На основе проведенного анализа платформ, Android Studio, исходя из своих плюсов и незначительных недостатков по сравнению с другими средами программирования, была выбрана для достижения цели, поставленной в дипломной работе.

2. Разработка приложения

2.1 Постановка задачи

На сегодняшний день мобильное приложение — мощный маркетинговый инструмент для интернет-магазина. Оно помогает торговой площадке не только увеличить продажи, но и проще, быстрее, удобнее взаимодействовать с клиентами.

С его помощью клиент сможет выбирать товары, ставить оценки, изучать характеристики, проверять состояние доставки или бонусного счета – функционал индивидуален, он разрабатывается с оглядкой на потребности бизнеса и аудитории. Приложения распространяются бесплатно, они требуют простейшей регистрации по номеру телефона или коду участника программы лояльности, а также через аккаунт в социальных сетях [7].

Несмотря на то, что функциональные возможности веб-сайтов существенно отличаются от ресурсов мобильного приложения и, как правило, обладают расширенным функционалам, людям зачастую проще совершать покупки и оформлять доставку через смартфон, где многие данные сохраняются автоматически или заполняются с использованием функции геолокации. Так как обычно сам телефон всегда находится под рукой в отличие от лэптопа и тем более стационарного компьютера. Также мобильные приложения существенно повышают эффективность коммуникации с теплой и горячей клиентской базой. При правильной реализации они могут иметь высокий индекс окупаемости, а также позволят сократить маркетинговый бюджет, что особенно актуально для работы со старыми клиентами [6].

Так как целью данной дипломной работы является разработка актуального мобильного приложения интернет-магазина по продаже техники на базе операционной системы Android, которое позволит пользователю облегчить процесс покупки техники.

Тем самым, для того чтобы создать максимально качественное приложение для интернет-покупок необходимо поставить следующие задачи:

– Обеспечить достижение высокой конверсии, путем упрощения интерфейса приложения;

– перенести весь основной функционал веб-страницы (к примеру, сравнение определенных единиц техники на усмотрение пользователя);

– создать приятный для глаза дизайн приложения;

– предоставить качественную оптимизацию приложения;

– создать качественную и удобную структуру приложения.

Таким образом, можно сделать вывод, что разработка мобильного приложения интернет-магазина по продаже техники, который будет занимать уверенные позиции на рынке, иметь постоянных клиентов и хорошую выручку – это трудоемкий и сложный процесс, так как различные ошибки, неприятный дизайн, неудобный юзабилити, непродуманная структура могут легко оттолкнуть потенциальных пользователей.

2.2 Описание разработки мобильного приложения

Данный пункт посвящён непосредственно созданию мобильного приложения “CondencedGeek”, темой которого является “магазин по продаже компьютеров и их комплектующих”.

Первым пунктом будет создание шапки, которая отображается на каждой странице приложения, были добавлены такие объекты как TextView “Главная”, TextView “Condenced”, TextView “Geek”, ImageView с логотипом приложения и ConstraintLayout, на котором располагается непосредственно шапка. Её можно увидеть на рисунке 5

Рисунок 6 – Шапка приложения “Condenced Geek”

Также были добавлены TextView “Категории” и RecyclerView с перечислением категорий, создание и вывод которых идёт через код, расположенный на рисунке 7, и кнопка, отвечающая за перенос пользователя на страницу с корзиной, код для которой находится на рисунке 8.

```
public class categoriyaadapter extends RecyclerView.Adapter<categoriyaadapter.CategoryViewHolder> {

    Context context;
    List<Caterogiya> categories;



    public categoriyaadapter(Context context, List<Caterogiya> categories) {
        this.context = context;
        this.categories = categories;
    }

    @NonNull
    @Override
    public CategoryViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
        View categoryItems = LayoutInflater.from(context).inflate(R.layout.categoryitem, parent,  attachToRoot: false);
        return new CategoryViewHolder(categoryItems);
    }

    @Override
    public void onBindViewHolder(@NonNull categoriyaadapter.CategoryViewHolder holder, int position) {
        holder.CategoryText.setText(categories.get(position).getTitle());

        holder.itemView.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                MainActivity.showCoursesByCat(categories.get(position).getId());
            }
        });



    }
```

Рисунок 7 – Перечисление категорий

После также был добавлен ещё один RecyclerView под товары на странице. Код, относящийся к нему, можно найти на рисунке 8.

```
public class CourseAdapter extends RecyclerView.Adapter<CourseAdapter.CourseViewHolder> {

    Context context;
    List<Courses> courses;

    public CourseAdapter(Context context, List<Courses> courses) {
        this.context = context;
        this.courses = courses;
    }

    @NonNull
    @Override
    public CourseViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
        View courseItems = LayoutInflater.from(context).inflate(R.layout.course_item, parent,  attachToRoot: false);
        return new CourseAdapter.CourseViewHolder(courseItems);
    }

    @Override
    public void onBindViewHolder(@NonNull CourseViewHolder holder, int position) {

        int imageId = context.getResources().getIdentifier(courses.get(position).getImg(),  defType: "drawable", context.getPackageName());
        holder.courseImage.setImageResource(imageId);

        holder.CourseTitle.setText(courses.get(position).getTitle());
        holder.CourseDesc.setText(courses.get(position).getdescrip());
        holder.CoursePrice.setText(courses.get(position).getprice());

        holder.itemView.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent intent = new Intent(context, CoursePage.class);

                intent.putExtra( name: "courseImage", imageId);
                intent.putExtra( name: "CourseTitle", courses.get(position).getTitle());
                intent.putExtra( name: "CourseDesc",courses.get(position).getdescrip());
                intent.putExtra( name: "CoursePrice", courses.get(position).getprice());
                intent.putExtra( name: "courseText", courses.get(position).getText());
                intent.putExtra( name: "courseId", courses.get(position).getId());
                context.startActivity(intent);
            }
        });
    }
```

Рисунок 8 – Перечисление товаров

```
public void openCart(View view) {

    Intent intent = new Intent( packageContext: this, OrderPage.class);
    startActivity(intent);

}
```

Рисунок 9 – Код для кнопки переноса

Рисунок 9 – Тестирование добавленных функций

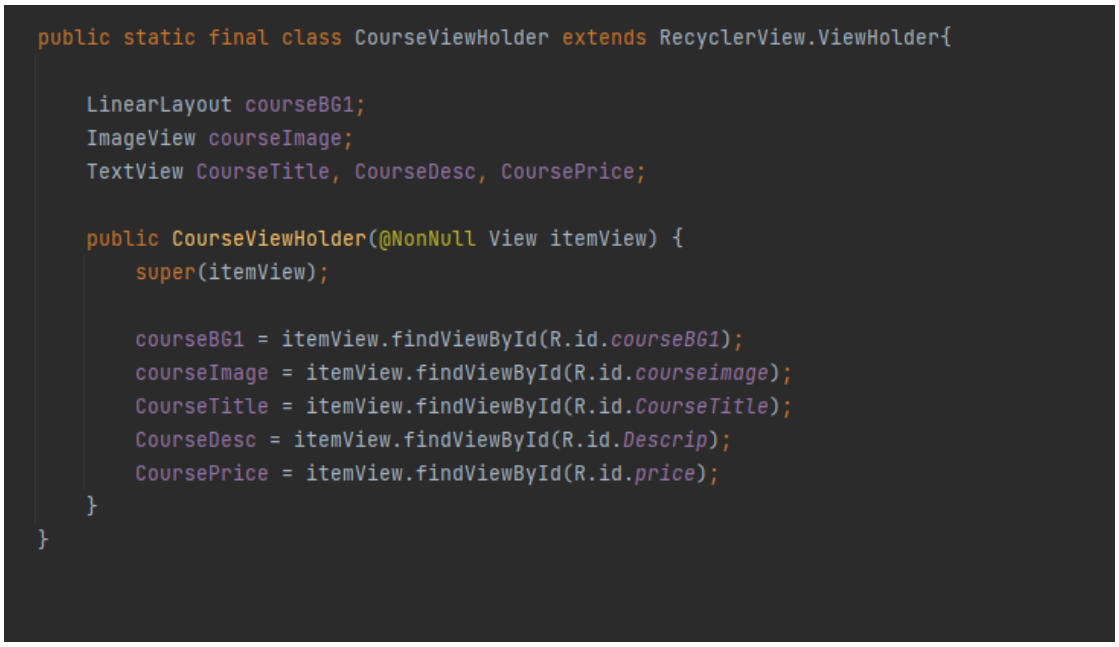
Тестирование вышеперечисленных функций прошло успешно, результат можно наблюдать на рисунке 9.

Как видно из рисунка 10, товары и категории создаются с помощью кода, который очень легко переделывать и использовать. Находится он в классе MainActivity.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    List<Caterogiya> categoryList = new ArrayList<>();
    categoryList.add(new Caterogiya( id: 1,  title: "Сборки"));
    categoryList.add(new Caterogiya( id: 2,  title: "Видеокарты"));
    categoryList.add(new Caterogiya( id: 3,  title: "Процессоры"));
    categoryList.add(new Caterogiya( id: 4,  title: "Переферия"));
    categoryList.add(new Caterogiya( id: 5,  title: "Прочее"));

    setCategoryRecycler(categoryList);


    coursesList.add(new Courses( id: 1,  img: "komp",  title: "HYPERPC LUMEN",  descrip: "Компьютер HYPERPC LUMEN — это синоним надежности, потому
    coursesList.add(new Courses( id: 2,  img: "proc",  title: "proc",  descrip: "adsasda",  price: "30,000p" ,  text: "TestText",  category: 3));
    coursesList.add(new Courses( id: 3,  img: "vidahu",  title: "vidyaha",  descrip: "shik",  price: "130,000p",  text: "TestText",  category: 2 ));
    coursesList.add(new Courses( id: 4,  img: "proc",  title: "proc",  descrip: "adsasda",  price: "30,000p" ,  text: "TestText",  category: 3));
    coursesList.add(new Courses( id: 5,  img: "komp",  title: "pc",  descrip: "adsasda",  price: "30,000p",  text: "TestText",  category: 1 ));
    coursesList.add(new Courses( id: 6,  img: "proc",  title: "proc",  descrip: "adsasda",  price: "30,000p" ,  text: "TestText",  category: 3));

    fullcoursesList.addAll(coursesList);

    setCourseRecycler(coursesList);
}
```

Рисунок 10 – Создание товаров и категорий товаров

Данный код создаёт два листа, один с категориями, второй с товарами, после чего они добавляются в RecyclerView1 и RecyclerView2 соответственно, на рисунке 11 код добавления товаров, на рисунке 12 добавление категорий.



```
public static final class CourseViewHolder extends RecyclerView.ViewHolder{

    LinearLayout courseBG1;
    ImageView courseImage;
    TextView CourseTitle, CourseDesc, CoursePrice;

    public CourseViewHolder(@NonNull View itemView) {
        super(itemView);

        courseBG1 = itemView.findViewById(R.id.courseBG1);
        courseImage = itemView.findViewById(R.id.courseimage);
        CourseTitle = itemView.findViewById(R.id.CourseTitle);
        CourseDesc = itemView.findViewById(R.id.Descrip);
        CoursePrice = itemView.findViewById(R.id.price);
    }
}
```

Рисунок 11 – Код, отвечающий размещение товара на странице

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

public static final class CategoryViewHolder extends RecyclerView.ViewHolder {

    TextView CategoryText;


    public CategoryViewHolder(@NonNull View itemView) {
        super(itemView);

        CategoryText = itemView.findViewById(R.id.CategoryText);
    }
}
```

Рисунок 12 – Код, отвечающий за добавление категорий

При нажатии на товар открывается автоматически созданная страница с этим товаром, его описанием, ценой и кнопкой, при нажатии на которую товар добавляется в корзину.

На этой странице была создана шапка приложения, описанная выше, ImageView с картинкой товара, TextView с наименованием товара, и CardView, в котором находится описание товара и его цена. Увидеть готовый результат можно на рисунке 13, а на рисунке 14 располагается код создания этой страницы.

Рисунок 13 – Страница-пример, по которой создаются остальные страницы с товаром

```java
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_course_page);

    ImageView courseImage = findViewById(R.id.imageView3);
    TextView courseTitle = findViewById(R.id.textView);
    TextView courseDesc = findViewById(R.id.Descrip);
    TextView price = findViewById(R.id.Price);


    courseImage.setImageResource(getIntent().getIntExtra( name: "courseImage",  defaultValue: 0));
    courseTitle.setText(getIntent().getStringExtra( name: "CourseTitle"));
    courseDesc.setText(getIntent().getStringExtra( name: "CourseDesc"));
    price.setText(getIntent().getStringExtra( name: "CoursePrice"));
}
```

Рисунок 14 – Код, отвечающий за создание страницы с товаром

Кнопка “Добавить в корзину” записывает товар, выбранный пользователем в корзину.

На страницу с корзиной можно перейти, кликнув на логотип приложения, который находится в шапке приложения список ListView с товарами, добавленными пользователем. Страницу корзины можно увидеть на рисунке 15, а код этой страницы на рисунках 16,17.

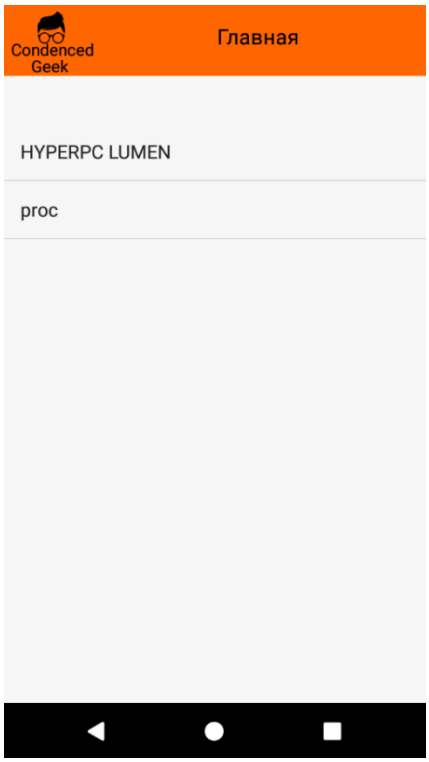


Рисунок 15 – Страница-корзина

```
public void AddToCart(View view){
    int id = getIntent().getIntExtra( name: "courseId",  defaultValue: 0);
    Order.ItemsId.add(id);
    Toast.makeText( context: this,  text: "Добавлено в корзину", Toast.LENGTH_LONG).show();
}
```

Рисунок 16 – Код, отвечающий за добавление товара в корзину

```
public class OrderPage extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_order_page);

        ListView ordersList = findViewById(R.id.orders_list);

        List<String> coursesTitle = new ArrayList<>();
        for(Courses c : MainActivity.fullcoursesList) {
            if (Order.ItemsId.contains(c.getId()))
            coursesTitle.add(c.getTitle());
        }

        ordersList.setAdapter(new ArrayAdapter<>( context: this, android.R.layout.simple_list_item_1, coursesTitle));

    }
    public void OpenMain(View view){

        Intent intent = new Intent( packageContext: this, MainActivity.class);
        startActivity(intent);

    }
}
```

Рисунок 17 – Код, отвечающий за заполнение листа покупок

Также было добавлено отображение товаров по категориям, каждому товару присваивается Id, которому соответствует своя категория. Метод в коде находится на рисунке 18.

```
public static void showCoursesByCat (int Category){

    coursesList.clear();
    coursesList.addAll(fullcoursesList);

    List<Courses> filterCourses = new ArrayList<>();
    for(Courses c: coursesList){
        if(c.getCategory() == Category)
            filterCourses.add(c);
    }

    coursesList.clear();
    coursesList.addAll(filterCourses);

    courseAdapter.notifyDataSetChanged();

}
```

Рисунок 18 – Метод сортировки товаров по категориям

В конце разработки был добавлен текст и цвета в соответствующие XML документы “Strings” и “Colors”, документы представлены на рисунках 19 и 20.

```
<resources>
    <string name="app_name">CondencedGeek</string>
    <string name="SalesIcon">Sales</string>
    <string name="Contacts_tag">Контакты</string>
    <string name="Us_tag">О нас</string>
    <string name="Main_tag">Главная</string>
    <string name="Name1">Condenced</string>
    <string name="Name2">Geek</string>
    <string name="Categories">Категории:</string>
    <string name="CategoryItem">Кружок</string>
    <string name="CategoryText">Готовые Сборки</string>
    <string name="VDescription">GIGABYTE AORUS GeForce RTX 3050 ELITE обеспечива
    <string name="Vname">GIGABYTE AORUS GeForce RTX 3050</string>
    <string name="Desc">Описание</string>
    <string name="Name">Название</string>
    <string name="pcname">САМЫЙ ЛУЧШИЙ ПК 2022</string>
    <string name="cpdesc">Компьютер HYPERPC LUMEN – это синоним надежности, пото
    <string name="trash">Купить!</string>
</resources>
```

Рисунок 19 – XML документ “Strings”

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="purple_200">#FFBB86FC</color>
    <color name="purple_500">#FF6200EE</color>
    <color name="purple_700">#FF3700B3</color>
    <color name="teal_200">#FF03DAC5</color>
    <color name="teal_700">#FF018786</color>
    <color name="black">#FF000000</color>
    <color name="white">#FFFFFFFF</color>
    <color name="mainbgcolor">#F6F6F6</color>
    <color name="sidecolor">#FF6600</color>
</resources>
```

Рисунок 20 – XML документ “Colors”

Далее было принято решение добавить на страницу корзины кнопку, отвечающую за оформление заказа, которая будет отправлять данные в базу данных и хранить их для дальнейшего использования. Её можно увидеть на рисунке 21.

Рисунок 21– Кнопка оформления заказа

Для того, чтобы кнопка функционировала как было задумано, был написан код, он находится на рисунке 22.

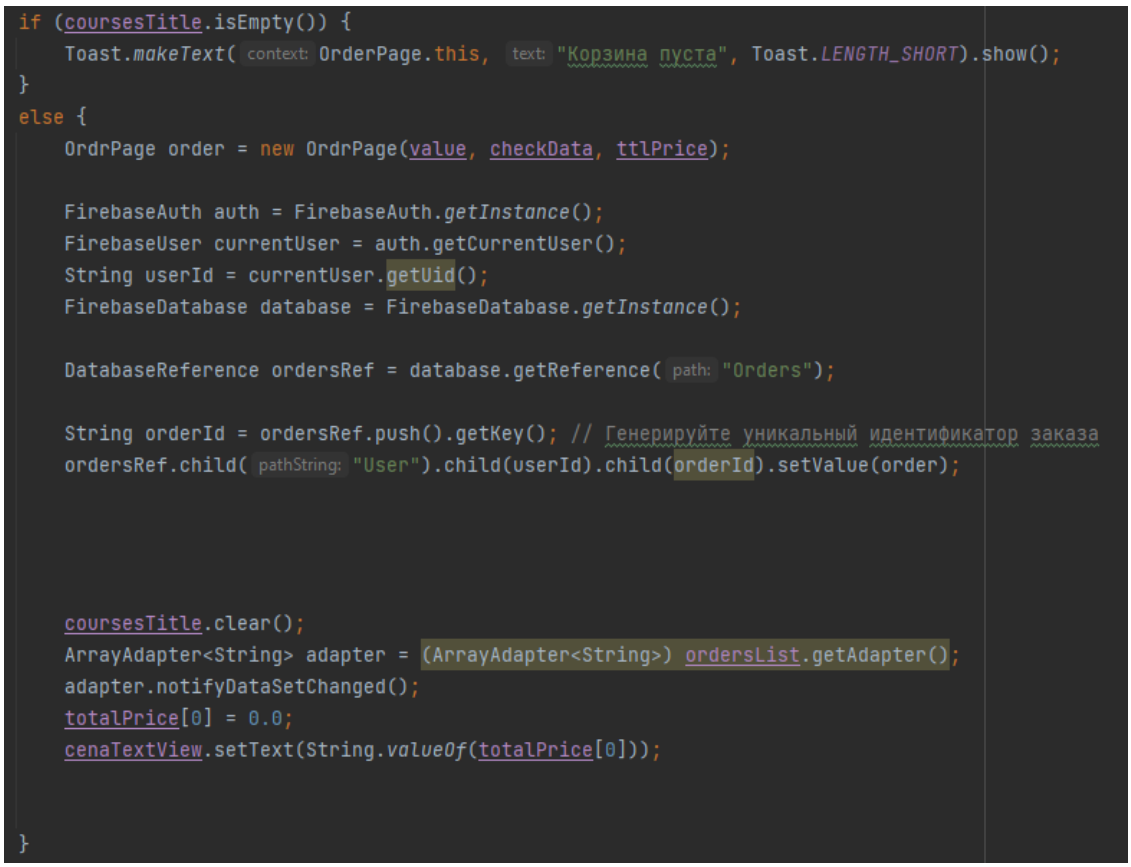


Рисунок 22 – Код, отвечающий за оформление заказа

Для полного функционала понадобилось сделать дополнительный код в методе OnCreate.

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_order_page);

    ordrButton = findViewById(R.id.Trash);

    SharedPreferences Prefo = getSharedPreferences( name: "MyPrefs", MODE_PRIVATE);
    String value = Prefo.getString( key: "pochta",  defValue: "");

    ListView ordersList = findViewById(R.id.orders_list);

    List<String> coursesTitle = new ArrayList<>();
    for(Courses c : MainActivity.fullcoursesList) {
        if (Order.ItemsId.contains(c.getId()))
            coursesTitle.add(c.getTitle());
    }
    ordersList.setAdapter(new ArrayAdapter<>( context: this, android.R.layout.simple_list_item_1, coursesTitle));

    List<String> coursesPrice = new ArrayList<>();
    for (Courses c: MainActivity.fullcoursesList) {
        if (Order.ItemsId.contains(c.getId()))
            coursesPrice.add(c.getprice());
    }

    final double[] totalPrice = {0.0};
    for (String price : coursesPrice) {
        double coursePrice = Double.parseDouble(price);
        totalPrice[0] += coursePrice;
    }
    TextView cenaTextView = findViewById(R.id.Cena);
    cenaTextView.setText(String.valueOf(totalPrice[0]));
    double ttlPrice = totalPrice[0];

//Список покупок
    StringBuilder stringBuilder = new StringBuilder();
    for (String item : coursesTitle ) {
        stringBuilder.append(item).append("\n");
    }
    String checkData = stringBuilder.toString();
    SharedPreferences sharedPreferences = getSharedPreferences( name: "MyPrefs", MODE_PRIVATE);
    SharedPreferences.Editor editor = sharedPreferences.edit();
    editor.putString("checkData", checkData);
    editor.apply();
```

Рисунок 23 – Метод onCreate страницы OrderPage

На рисунках 22,23 видно, что предметы, которые пользователь поместил в корзину, обрабатывались в модели OrdrPage, их информация делилась на два списка «Наименование» и «Цена», после чего переводилась в строку. Таким образом, информация, об оформленном заказе записывается в базу данных во вкладке “Orders/Users/userID/orderID”, а в случае, если корзина пуста, сообщает пользователю об этом. Следующим шагом было принято решение протестировать кнопку оформления заказа. На рисунке 24 представлена страница корзины с кнопкой оформления во время тестирования.

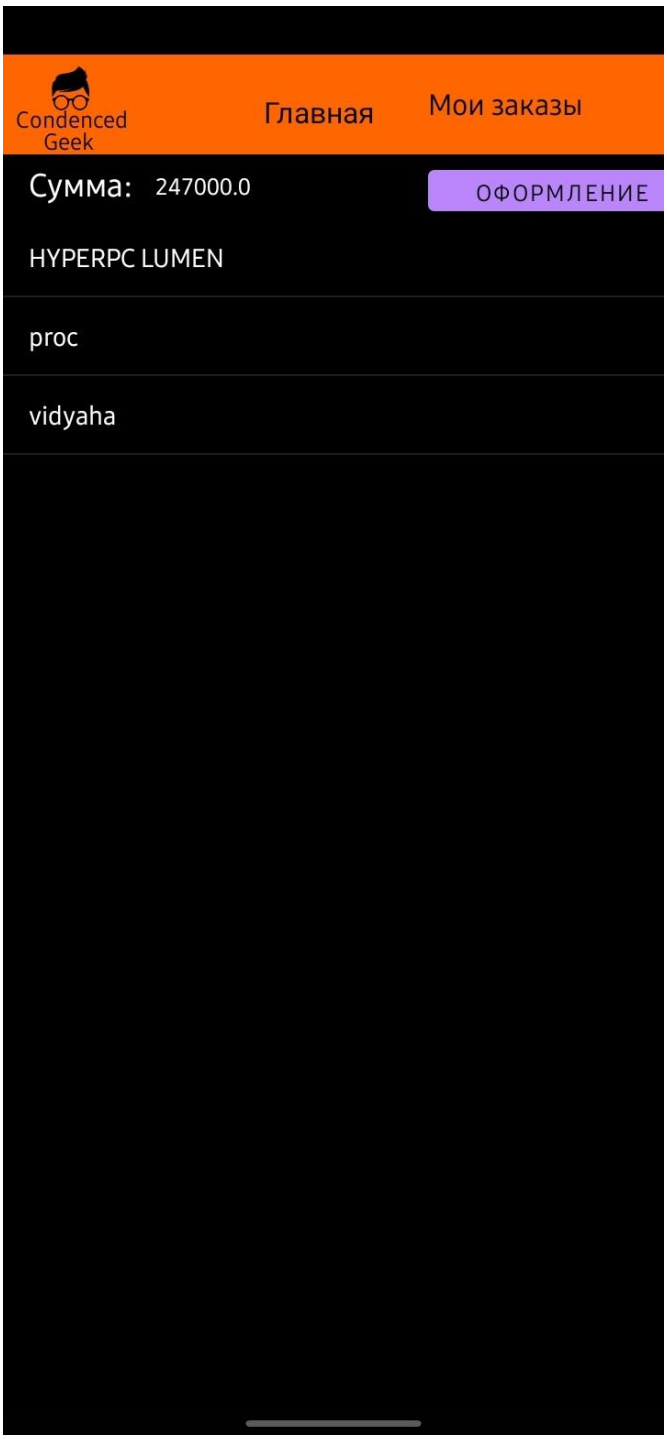


Рисунок 24 – Тестирование страницы корзины

Тестирование прошло успешно. Информация в списке RecyclerView отображается корректно, никаких критических ошибок в процессе тестирования обнаружено не было.

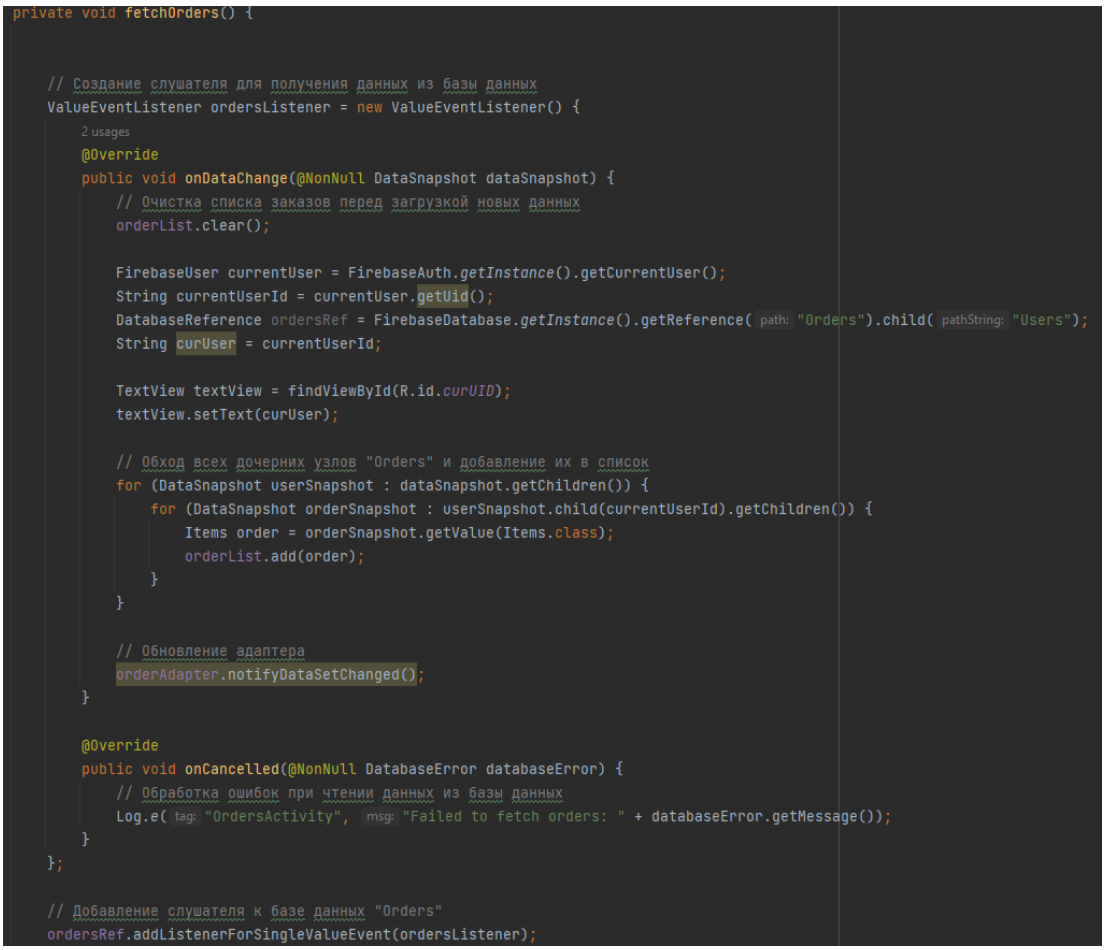
После этого решено было сделать страницу с оформленными заказами пользователя для более удобного отслеживания покупок. Для этого

понадобилось создать дополнительную страницу “OrdersActivity”, сделать в ней шапку приложения, как на остальных страницах, добавить “RecyclerView” для отображения заказов, также было необходимостью создать модель Items и адаптер “OrderAdapter” для обработки строк, получаемых из базы данных.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_orders);


    // Получение ссылки на базу данных "Orders"
    ordersRef = FirebaseDatabase.getInstance().getReference( path: "Orders");

    // Инициализация RecyclerView
    recyclerView = findViewById(R.id.RecyclerOrders);
    recyclerView.setHasFixedSize(true);

    // Инициализация менеджера макета
    LinearLayoutManager layoutManager = new LinearLayoutManager( context: this);
    recyclerView.setLayoutManager(layoutManager);

    // Инициализация списка заказов
    orderList = new ArrayList<>();

    // Инициализация адаптера
    orderAdapter = new OrderAdapter(orderList);
    recyclerView.setAdapter(orderAdapter);

    // Получение данных из Firebase Realtime Database
    fetchOrders();
}
```

Рисунок 25 – метод onCreate страницы OrdersActivity

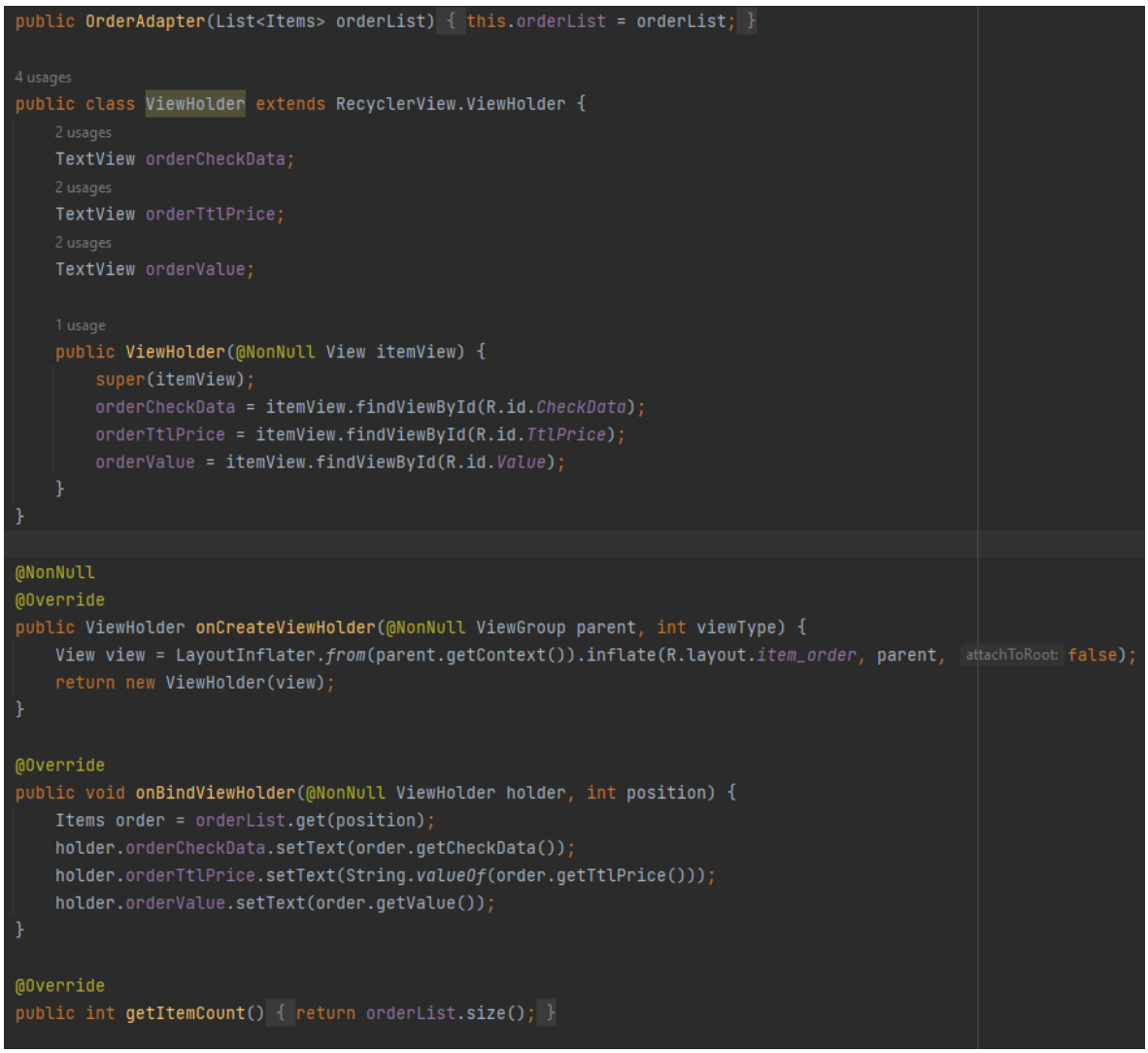
На рисунке 25 при переходе на страницу заказов, приложение начинает искать объекты, находящиеся в папке “Orders”, затем создаётся новый LayoutManager, отвечающий за отображение элементов в RecyclerView, создаёт список ArrayList и адаптер orderAdapter и присоединяет его к RecyclerView, после чего инициализирует метод fetchOrders(), которая находится на рисунке 26.



```
private void fetchOrders() {


    // Создание слушателя для получения данных из базы данных
    ValueEventListener ordersListener = new ValueEventListener() {
        2 usages
        @Override
        public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
            // Очистка списка заказов перед загрузкой новых данных
            orderList.clear();

            FirebaseUser currentUser = FirebaseAuth.getInstance().getCurrentUser();
            String currentUserId = currentUser.getUid();
            DatabaseReference ordersRef = FirebaseDatabase.getInstance().getReference( path: "Orders").child( pathString: "Users");
            String curUser = currentUserId;

            TextView textView = findViewById(R.id.curUID);
            textView.setText(curUser);

            // Обход всех дочерних узлов "Orders" и добавление их в список
            for (DataSnapshot userSnapshot : dataSnapshot.getChildren()) {
                for (DataSnapshot orderSnapshot : userSnapshot.child(currentUserId).getChildren()) {
                    Items order = orderSnapshot.getValue(Items.class);
                    orderList.add(order);
                }
            }

            // Обновление адаптера
            orderAdapter.notifyDataSetChanged();
        }

        @Override
        public void onCancelled(@NonNull DatabaseError databaseError) {
            // Обработка ошибок при чтении данных из базы данных
            Log.e( tag: "OrdersActivity",  msg: "Failed to fetch orders: " + databaseError.getMessage());
        }
    };

    // Добавление слушателя к базе данных "Orders"
    ordersRef.addListenerForSingleValueEvent(ordersListener);
```

Рисунок 26 – метод fetchOrders

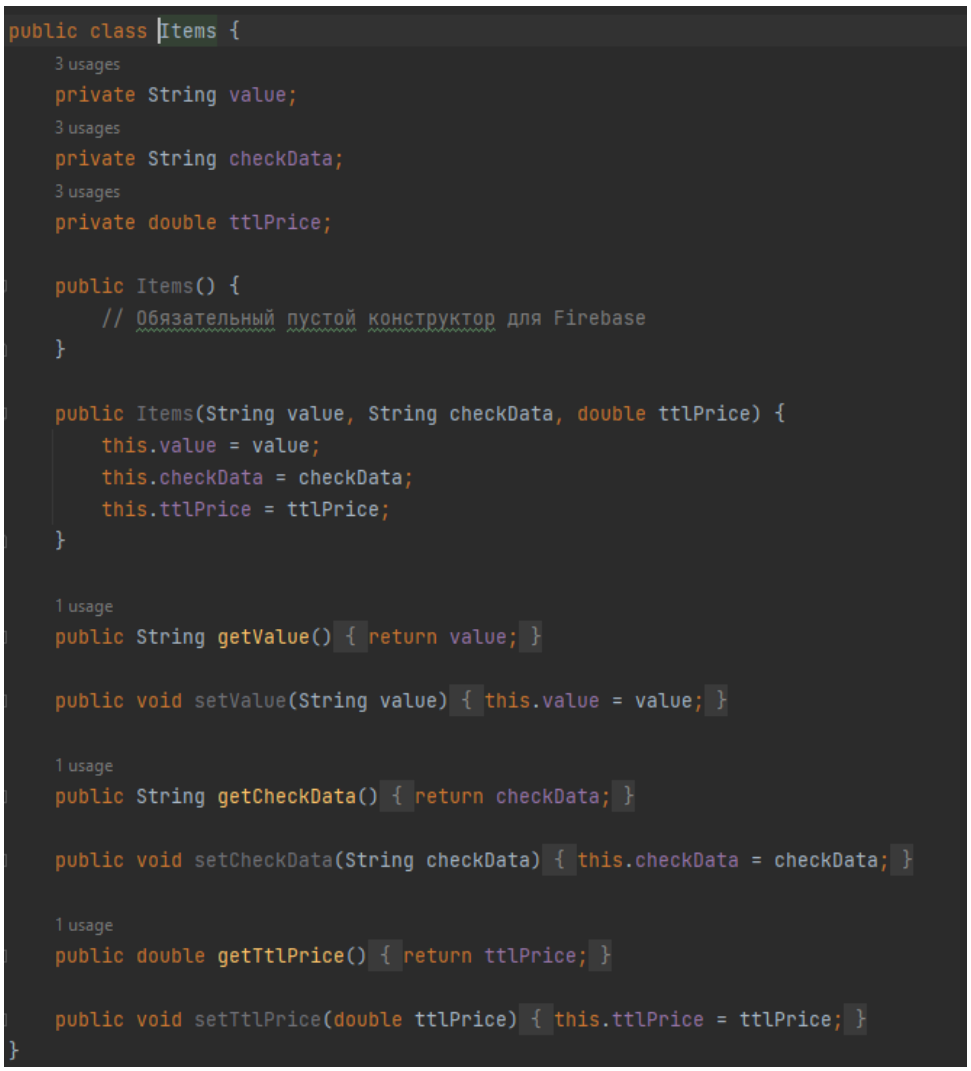
В данном методе идёт постоянное обновление списка для поддержания актуальности информации, находящейся внутри него. Первым делом идёт очистка листа, чтобы впоследствии заполнить его новыми данными, после очистки из базы данных получается ID текущего пользователя и указывается путь, по которому его стоит искать, также этот ID записывается в TextView, находящийся на странице OrdersActivity и служит как секретный код пользователя, чтобы определить заказы покупателя на кассе, после этого из базы данных извлекается информация о заказах и добавляется в список. Далее программа уведомляет адаптер OrderAdapter о том, что данные изменились, а он в свою очередь меняет данные внутри элементов RecyclerView. Код OrderAdapter представлен на рисунке 27.



```
public OrderAdapter(List<Items> orderList) { this.orderList = orderList; }

4 usages
public class ViewHolder extends RecyclerView.ViewHolder {
    2 usages
    TextView orderCheckData;
    2 usages
    TextView orderTtlPrice;
    2 usages
    TextView orderValue;

    1 usage
    public ViewHolder(@NonNull View itemView) {
        super(itemView);
        orderCheckData = itemView.findViewById(R.id.CheckData);
        orderTtlPrice = itemView.findViewById(R.id.TtlPrice);
        orderValue = itemView.findViewById(R.id.Value);
    }
}

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

@Override
public void onBindViewHolder(@NonNull ViewHolder holder, int position) {
    Items order = orderList.get(position);
    holder.orderCheckData.setText(order.getCheckData());
    holder.orderTtlPrice.setText(String.valueOf(order.getTtlPrice()));
    holder.orderValue.setText(order.getValue());
}

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

Рисунок 27 – адаптер OrderAdapter

В адаптер информация поступает через модель Items, в которой она сортируется по трём строкам. Код модели Items представлен на рисунке 28. Модель Items является своего рода сортировщиком данных, а адаптер связующим звеном между моделью и пользовательским интерфейсом.



```
public class Items {
    3 usages
    private String value;
    3 usages
    private String checkData;
    3 usages
    private double ttlPrice;

    public Items() {
        // Обязательный пустой конструктор для Firebase
    }

    public Items(String value, String checkData, double ttlPrice) {
        this.value = value;
        this.checkData = checkData;
        this.ttlPrice = ttlPrice;
    }

    1 usage
    public String getValue() { return value; }

    public void setValue(String value) { this.value = value; }

    1 usage
    public String getCheckData() { return checkData; }

    public void setCheckData(String checkData) { this.checkData = checkData; }

    1 usage
    public double getTtlPrice() { return ttlPrice; }

    public void setTtlPrice(double ttlPrice) { this.ttlPrice = ttlPrice; }
}
```

Рисунок 28 – модель Items

На этом программирование страницы заказов закончено, следующим шагом будет протестировать её. Тестирование страницы представлено на рисунке 29.

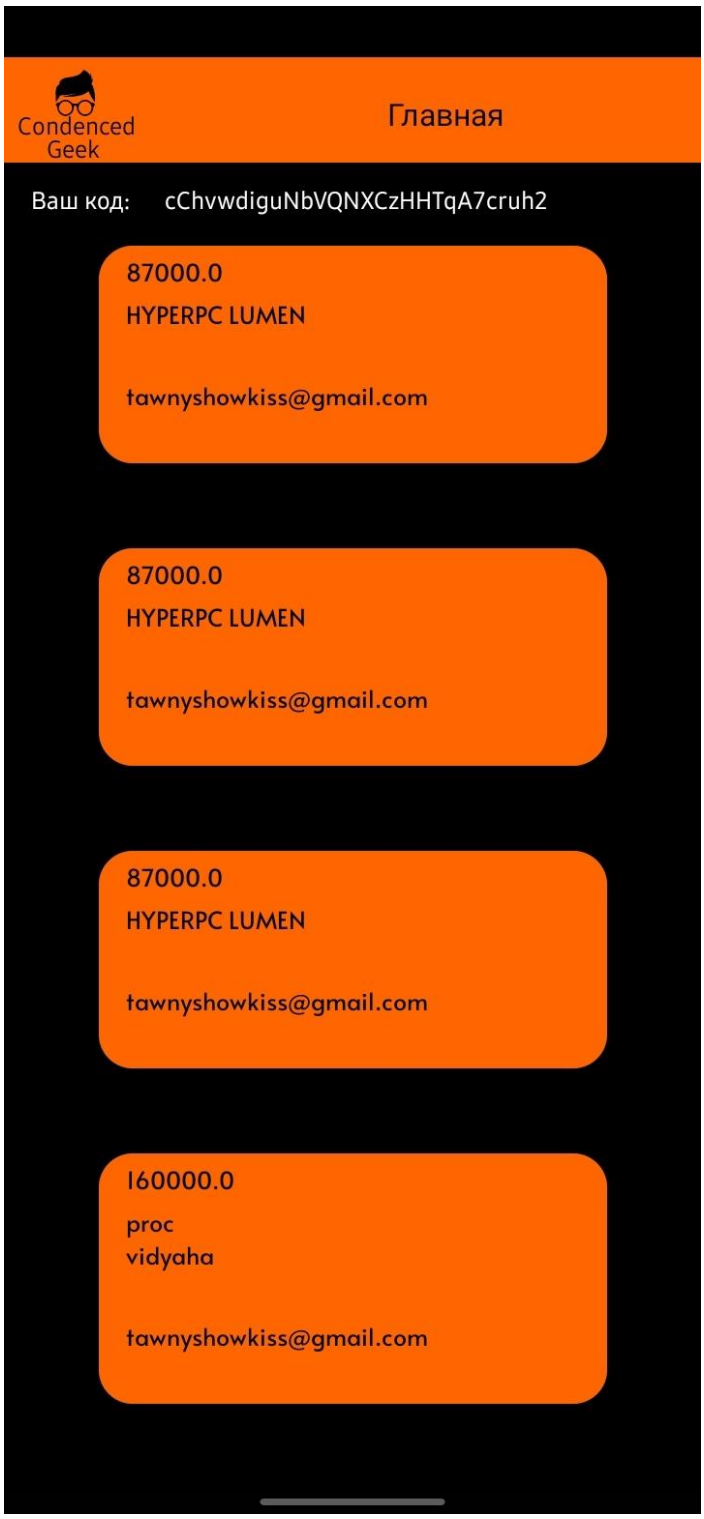


Рисунок 29 – Страница заказов

Тестирование прошло успешно, никаких ошибок или визуальных багов обнаружено не было, RecyclerView отображает все заказы текущего пользователя и не конфликтует с другими элементами пользовательского интерфейса.

После тестирования страницы заказов было принято решение заняться авторизацией пользователей для хранения заказов и информации о пользователях. Для этого понадобилось создать две страницы, страницу входа и регистрации. На странице входа было создано несколько объектов, а именно кнопка «Назад», кнопка «Вход», TextView, оповещающий потенциального покупателя о том, что это страница входа, CardView, внутри которого были помещены два EditText для почты и пароля соответственно. На странице регистрации были созданы кнопка «Назад», кнопка «Регистрация», TextView, оповещающий пользователя, что это страница регистрации, CardView, содержащий в себе три поля EditText с логином, почтой и паролем и кнопка «Уже есть аккаунт?», перенаправляющая пользователя на страницу входа.

После создания пользовательского интерфейса была начата разработка функционала созданного интерфейса на странице регистрации. Первым делом был скрыт ActionBar, всем объектам пользовательского интерфейса была предоставлена переменная для дальнейшего создания функционала. Также были объявлены переменные с базой данных и был создан ProgressDialog, который бы оповещал пользователя о том, что создание аккаунта в процессе. Этот код можно найти на рисунке 30.

```
ActionBar actionBar = getSupportActionBar();
if (actionBar != null) {
    actionBar.hide();
}


btnSignUp=findViewById(R.id.singUP);
name = findViewById(R.id.login);
email = findViewById(R.id.email);
password = findViewById(R.id.password);
AccountExists = findViewById(R.id.lin);
backbutton = findViewById(R.id.imageView);

auth = FirebaseAuth.getInstance();
database = FirebaseDatabase.getInstance();

ProgressDialog = new AlertDialog.Builder( context: LoginActivity.this);
ProgressDialog.setMessage("Подождите, идёт создание аккаунта");
```

Рисунок 30 – Основной код метода OnCreate

После объявления переменных был добавлен OnClickListener для кнопки регистрации. Код кнопки представлен на рисунке 31.

```
ProgressDialog.show();

String edtName = name.getText().toString();
String edtEmail = email.getText().toString();
String edtPassword = password.getText().toString();

auth.createUserWithEmailAndPassword(edtEmail, edtPassword)
        .addOnCompleteListener(new OnCompleteListener<AuthResult>() {
            @Override
            public void onComplete(@NonNull Task<AuthResult> task) {
                SharedPreferences preferences = getSharedPreferences( name: "MyPrefs", Context.MODE_PRIVATE);
                SharedPreferences.Editor editor = preferences.edit();
                String pochta = edtEmail;
                editor.putString("pochta", pochta);
                editor.apply();

                if (task.isSuccessful()) {
                    UserModel model = new UserModel(edtName, edtEmail, edtPassword);
                    FirebaseUser user = task.getResult().getUser();
                    if (user != null) {
                        String id = user.getUid();
                        database.getReference().child( pathString: "Users").child(id).setValue(model);

                        Intent intent = new Intent( packageContext: LoginActivity.this, MainActivity.class);
                        startActivity(intent);
                    } else {
                        Toast.makeText( context: LoginActivity.this,  text: "Failed to create user", Toast.LENGTH_SHORT).show();
                    }
                } else {
                    Exception exception = task.getException();
                    if (exception != null && exception.getMessage() != null && exception.getMessage().contains("email address is already in use")) {
                        // Почта уже есть в системе
                        Toast.makeText( context: LoginActivity.this,  text: "Email is already registered", Toast.LENGTH_SHORT).show();
                    } else {
                        // Handle other errors
                        Toast.makeText( context: LoginActivity.this,  text: "Failed to create user: " + exception.getMessage(), Toast.LENGTH_SHORT).show();
                    }
                }
            }
        });
```

Рисунок 31 – Код кнопки регистрации

Так как это действие кнопки, было принято решение показать ProgressBar, созданный на рисунке 30, после этого текст из полей, введённых пользователем переводился в строку для дальнейшей записи в базу данных и создавался метод регистрации с почтой и паролем, который срабатывал в случае, если все данные были введены корректно. Внутри метода был добавлен SharedPreference для хранения почты, введённой пользователем на устройстве пользователя, без root-прав пользователю в директорию с этими данными доступ запрещён. Хранение почты нужно для того, чтобы при оформлении заказа добавлять её в сам заказ для лучшего определения пользователя. Далее, если данные были успешно введены, они записывались в модель, после чего добавлялись в базу данных по пути «Users/userID». В ином случае, если данные были введены некорректно, пользователь получал одну из ошибок.

После этого был добавлен код для кнопок «Назад» и «Уже есть аккаунт?». Код для этих кнопок представлен на рисунке 32.

```
backbutton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent( packageContext: LoginActivity.this, SingInActivity.class);
        startActivity(intent);
    }



});

AccountExists.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent( packageContext: LoginActivity.this, SingInActivity.class);
        startActivity(intent);
    }
});
```

Рисунок 32– Кнопки «Назад» и «Уже есть аккаунт?»

В коде, приведённом на рисунке 32 был реализован переход со страницы регистрации на страницу входа через кнопку «Назад» и через кнопку «Уже есть аккаунт?» для более удобного пользования. После создания страницы было решено решение провести её тестирование. Тестирование отображено на рисунке 33.

Рисунок 33 – Тестирование страницы регистрации

На рисунке 33 видно, что пользовательский интерфейс расположен корректно, объекты не пересекаются, как и должны. Со стороны кода тестирование прошло успешно, реквизиты вводятся успешно, регистрация проходит без проблем и данные заносятся в базу данных корректно. Для тестирования было принято решение зарегистрировать новый аккаунт. На рисунке 34 можно увидеть этот аккаунт в базе данных.

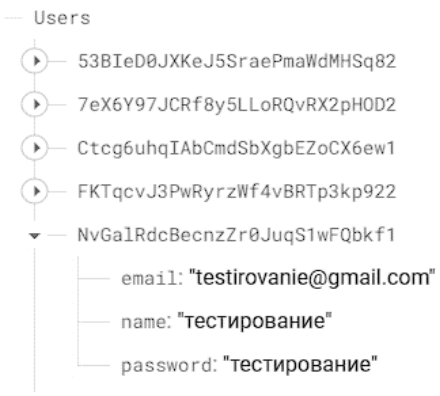


Рисунок 34 – Зарегистрированный для тестирования аккаунт

После окончания работ над страницей регистрации было решено начать работу над страницей входа. Код этой страницы очень похож на код страницы регистрации, но в упрощенном варианте. Код показан на рисунке 35.

```
auth = FirebaseAuth.getInstance();

button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        String edtEmail = email.getText().toString();
        String edtPass = password.getText().toString();

        SharedPreferences preferences = getSharedPreferences( name: "MyPrefs", Context.MODE_PRIVATE);
        SharedPreferences.Editor editor = preferences.edit();
        String pochta = edtEmail;
        editor.putString("pochta", pochta);
        editor.apply();



        auth.signInWithEmailAndPassword(edtEmail, edtPass).addOnCompleteListener(new OnCompleteListener<AuthResult>() {
            @Override
            public void onComplete(@NonNull Task<AuthResult> task) {
                if (task.isSuccessful()) {



                    Intent intent = new Intent( packageContext: SingInActivity.this, MainActivity.class);
                    startActivity(intent);
                } else {
                    Toast.makeText( context: SingInActivity.this,  text: "Sign-in failed", Toast.LENGTH_SHORT).show();
                }
            }
        });
    }
});
```

Рисунок 35 – Код страницы входа

Первым делом переменной auth была присвоена аутентификация для базы данных, после этого на кнопке входа был вызван метод onClick, внутри которого данные, введённые пользователем проверялись в базе данных и в случае, если

эти данные действительно находились в базе данных, пользователя перенаправляло бы на главную страницу приложения, в ином же случае пользователю выдавало ошибку аутентификации. После создания кода на этой странице, работа над приложением окончена и можно перейти к тестированию.

В процессе тестирования проекта была выявлена одна ошибка, а именно, интерфейс пользователя на разных устройствах отображается по-разному, из-за чего у одного пользователя приложение может работать корректно, в то время, как у другого некоторые объекты могут не отображаться или отображаться некорректно. Поэтому стоит отметить, что приложение делалось под устройство Samsung 30s с 6,4-дюймовым дисплеем и разрешением 720 x 1560 пикселей. В дальнейшем имеется возможность изменения и улучшения приложения.

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной дипломной работы было спроектировано новое мобильное приложение по продаже компьютерной техники, позволяющее пользователю просматривать ассортимент товаров и добавлять их в корзину для последующего оформления покупки и просмотра заказов, оформленных пользователем.

Детально исследована не только архитектура приложений на системе Android, но и рассмотрены все три основных слоя, которые являются ее главными составляющими.

Рассмотрена необходимость создания мобильного приложения для интернет-магазинов, и их влияние на взаимодействие с клиентской базой.

Были рассмотрены задачи и требования к новому мобильному приложению по продаже компьютеров и их комплектующих, которые полностью бы соответствовали современным тенденциям и положительно повлияли на дальнейшее развитие магазина.

Было разработано мобильное приложение на ОС Android, представлен подробный разбор его функционала и программного кода, а также проведено последующее тестирование приложения.

В заключение можно сказать, что разработанный автором в ходе исследования проект удовлетворяет всем требования технического задания. Поэтому следует считать, что задачи дипломной работы полностью решены и цель исследования достигнута.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Актуальность разработки мобильных приложений для определенного вида бизнеса // [Электронный ресурс] // © 2010-2022 XenForo Ltd - URL: https://www.safezone.cc/threads/aktualnost-razrabotki-mobilnyx-prilozhenij-dlja-opredelennogo-vida-biznesa.39890/ (дата обращения 12.12.22)

2 Архитектура Android-приложений… Правильный путь? // [Электронный ресурс] // teletype.in - URL: https://teletype.in/@skillbranch/HkSIs5bzL (дата обращения 10.12.22)

3 Гайд по архитектуре приложений для Android. Часть 4: доменный слой // [Электронный ресурс] // © 2006–2022, Habr - URL: https://habr.com/ru/company/surfstudio/blog/653673/ (дата обращения 11.12.22)

4 Каким должно быть мобильное приложение интернет-магазина, чтобы его скачивали // [Электронный ресурс] // © 2012-2022 - URL: https://new-retail.ru/marketing/kakim_dolzhno_byt_mobilnoe_prilozhenie_internet_magazina_chtoby_ego_skachivali2717/ (дата обращения 09.12.22)

5 Лекция 4 по архитектуре андроид приложения. Clean Architecture // [Электронный ресурс] // © 2022 Fandroid.info - URL:https://www.fandroid.info/lektsiya-4-po-arhitekture-android-prilozheniya-clean-arcitecture/ (дата обращения 12.12.22)

6 Мобильное приложение для интернет-магазина: плюсы, минусы и эффективность // [Электронный ресурс] // insales.com - URL: https://www.techyourchance.com/mvc-android-1/ (дата обращения 11.12.22)

7 Что вам нужно знать о разработке мобильного приложения для интернет-магазина слой // [Электронный ресурс] // vc.ru - URL: https://vc.ru/trade/163177-chto-vam-nuzhno-znat-o-razrabotke-mobilnogo-prilozheniya-dlya-internet-magazina (дата обращения 11.12.22)

8 Web MVC framework. [Электронный ресурс] - URL: https://docs.spring.io/spring/docs/current/spring-frameworkreference/html/mvc.html (дата обращения 10.12.22)

9 Android Developers. [Электронный ресурс] – URL: https://developer.android.com/ (дата обращения 01.06.23)

10 Apple Developer Documentation. [Электронный ресурс] – URL: https://developer.apple.com/documentation/ (дата обращения 01.06.23)

11 React Native Documentation. [Электронный ресурс] – URL: https://reactnative.dev/docs/ (дата обращения 01.06.23)

12 Flutter Documentation. [Электронный ресурс] – URL: https://flutter.dev/docs/ (дата обращения 01.06.23)

13 Coursera. [Электронный ресурс] – URL: https://www.coursera.org/ (дата обращения 02.06.23)

14 Stack Overflow [Электронный ресурс] – URL: https://stackoverflow.com/ (дата обращения 02.06.23)

15 Android Developers Community [Электронный ресурс] – URL: https://developer.android.com/community (дата обращения 05.06.23)

16 Android Studio [Электронный ресурс] – URL: https://developer.android.com/studio (дата обращения 05.06.23)

17 Firebase [Электронный ресурс] – URL: https://firebase.google.com/ (дата обращения 05.06.23)

18 GitHub [Электронный ресурс] – URL: https://github.com/ (дата обращения 07.06.23)

19 Android Developers Blog [Электронный ресурс] – URL: https://android-developers.googleblog.com/ (дата обращения 09.06.23)

20 Android Authority [Электронный ресурс] – URL: ttps://www.androidauthority.com/ (09.06.23)