Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Владимирский государственный университет имени Александра Григорьевича и Николая Григорьевича Столетовых»

(ВлГУ)

Кафедра ВТиСУ

КУРСОВАЯ РАБОТА

по дисциплине «Разработка мобильных приложений»

на тему: «Разработка мобильного приложения на платформе Android. Часть 8. Разработка мобильного приложения для магазина компьютеров»

Студент: Кубылин Н.С.

Институт: ИИТЭ

Направление: 09.03.03

Группа: ПИ-121

№ зачётной книжки: 121002353

Руководитель: Шутов Антон Владимирович

Владимир 2024 г.

**Оглавление**

[Введение 3](#_Toc185272803)

[Глава 1. Анализ предметной области и современные технологии мобильной коммерции 5](#_Toc185272804)

[1.1. Современные тенденции мобильной коммерции 5](#_Toc185272805)

[1.2. Обзор существующих решений 6](#_Toc185272806)

[1.3. Выводы по анализу 9](#_Toc185272807)

[Глава 2. Разработка мобильного приложения для магазина по продажи компьютеров 11](#_Toc185272808)

[2.1. Функциональные и нефункциональные требования 11](#_Toc185272809)

[2.2. Функциональные и нефункциональные требования 11](#_Toc185272810)

[2.3 Реализация функционала 14](#_Toc185272811)

[2.3.1. Авторизация и регистрация 14](#_Toc185272812)

[2.3.2. Каталог товаров 23](#_Toc185272813)

[2.3.3. Административная панель 32](#_Toc185272814)

[2.3.4. Корзина 44](#_Toc185272815)

[Заключение 49](#_Toc185272816)

[Список литературы 50](#_Toc185272817)

# **Введение**

Современный мир стремительно движется в сторону цифровизации, и мобильные технологии занимают ключевую роль в этом процессе. Мобильные приложения стали незаменимым инструментом как для пользователей, так и для бизнеса, предлагая доступ к сервисам и продуктам в любое время и из любого места. В сфере электронной коммерции (e-commerce) мобильные приложения занимают особое место, формируя новое направление — мобильную коммерцию (m-commerce). Это направление стремительно растет благодаря доступности смартфонов, быстрому интернету и повышению удобства использования.

На фоне глобального интереса к модным и спортивным товарам, кроссовки приобрели статус важного элемента стиля и культуры. Это делает разработку мобильного приложения для продажи кроссовок актуальной и востребованной. Удобство совершения покупок, персонализация предложений и прозрачность взаимодействия с клиентом становятся ключевыми факторами успешной работы любого интернет-магазина.

Целью данной курсовой работы является создание удобного и функционального мобильного приложения для продажи компьютера, которое включает:

1. Авторизация и регистрация: возможность зарегистрироваться и авторизоваться новому пользователю.
2. Просмотр каталога товаров: пользователи могут изучать ассортимент компьютеров с описаниями и изображениями.
3. Добавление товаров в корзину и оформление заказа
4. Административная панель: инструмент для управления контентом приложения, включая добавление новых товаров, редактирование цен и управление заказами.

**Задачи работы:**

Для достижения поставленной цели необходимо выполнить следующие задачи:

1. Провести анализ предметной области, изучить существующие решения и определить потребности целевой аудитории.
2. Разработать удобный интерфейс для пользователей и администраторов, ориентированный на интуитивное взаимодействие.
3. Реализовать авторизацию и регистрацию пользователей.
4. Разработать функционал административной панели для управления ассортиментом и обработкой заказов.

Практическая значимость проекта заключается в создании программного продукта, который обеспечивает:

1. Простоту покупки для конечного пользователя.
2. Удобство управления магазином для администратора.
3. Повышение вовлеченности клиентов благодаря персонализации и интеграции современных технологий.

# **Глава 1. Анализ предметной области и современные технологии мобильной коммерции**

## **1.1. Современные тенденции мобильной коммерции**

Мобильная коммерция (m-commerce) представляет собой современное направление электронной коммерции, ориентированное на использование мобильных устройств для покупок. В последние годы m-commerce стал неотъемлемой частью повседневной жизни многих пользователей благодаря быстрому распространению смартфонов, доступности высокоскоростного интернета и удобству, которое предлагают мобильные приложения. По статистике, более 60% всех онлайн-покупок сейчас осуществляется через мобильные устройства, что подчеркивает значимость этой области для бизнеса и пользователей.

**Ключевые Тенденции в M-Commerce**

1. **Рост популярности мобильных платежей**

Одной из главных тенденций в мобильной коммерции является значительное увеличение использования мобильных платежных систем. Технологии, такие как Apple Pay, Google Pay и QR-коды, делают процесс оплаты более удобным и безопасным. Пользователи могут осуществлять платежи одним касанием или сканированием, что ускоряет процесс покупок и предоставляет высокую степень безопасности данных. Это способствует росту доверия потребителей к m-commerce, так как они чувствуют себя более защищёнными при использовании этих технологий.

1. **Персонализация пользовательского опыта**

Современные мобильные приложения все чаще обращаются к персонализации, используя данные о поведении пользователей для создания рекомендаций и предложений, которые соответствуют их индивидуальным предпочтениям. Это включает в себя анализ истории покупок, предпочтений в категориях товаров и даже местоположения пользователей. Персонализированный подход помогает бизнесу не только сохранять интерес клиентов, но и увеличивать количество успешных продаж, так как пользователи видят предложения, которые действительно могут их заинтересовать.

1. **Интеграция с социальными сетями**

Сегодня многие пользователи предпочитают совершать покупки через социальные сети, такие как Instagram и TikTok. Эти платформы предоставляют возможность купить товар прямо во время просмотра контента, что значительно упрощает процесс. Интеграция функций для покупок в социальных сетях позволяет брендам расширять свою аудиторию и улучшать взаимодействие с клиентами. Функции, такие как «шоппинг-экран» в Instagram, делают процесс покупок более органичным и захватывающим.

1. **Мобильные уведомления**

Push-уведомления являются ещё одним инструментом, который позволяет компаниям мгновенно информировать пользователей о новых товарах, акциях, скидках и новостях. Эти уведомления могут быть персонализированными и сегментированными, что повышает их эффективность. Когда пользователи получают актуальную информацию, они становятся более склонными совершать покупки, что увеличивает конверсию и лояльность клиентов.

1. **Применение M-Commerce в бизнесе**

Мобильные приложения, особенно в индустрии техники, таких как персональные компьютеры, открывают новые возможности для бизнеса. Они позволяют не только улучшить лояльность клиентов, предлагая уникальные продукты и услуги, но также оптимизировать внутренние процессы. Бренды могут провести эффективные маркетинговые кампании, анализируя поведение пользователей в реальном времени и адаптируя свои предложения к текущим трендам. Например, с использованием мобильных приложений можно проводить специальные акции, предлагать эксклюзивные коллекции и организовывать мероприятия для лояльных клиентов.

**1.2. Обзор существующих решений**

Обзор мобильных приложений для магазинов по продаже персональных компьютеров в России включает в себя ведущие платформы, которые предлагают покупателям широкий выбор техники, комплектующих и аксессуаров. Рассмотрим несколько наиболее заметных приложений на российском рынке.

1. **Ситилинк**

**Описание:**

Ситилинк — один из крупнейших российских онлайн-ретейлеров электроники и компьютерной техники.

**Ключевые функции:**

Широкий ассортимент: Пользователи могут найти как сборные системы, так и комплектующие от разных производителей.

Поиск по параметрам: Удобная фильтрация по параметрам, включая цену, бренд, характеристики и наличие.

Отслеживание статуса заказа: Возможность отслеживания статуса доставки в реальном времени.

Программа лояльности: Бонусы и скидки для постоянных клиентов.

1. **М.видео**

**Описание:**

М.видео — один из крупнейших ритейлеров бытовой техники и электроники, который представил мобильное приложение для покупки компьютерной техники.

**Ключевые функции:**

Доступ к каталогу: Удобный интерфейс для просмотра продукции, включая новые поступления и распродажи.

Подборки и рекомендации: Персонализированные рекомендации на основе предпочтений пользователя.

Наличие товара: Опция проверки наличия товара в магазинах близлежащих к пользователю.

Оплата и доставка: Разнообразные способы оплаты и доставки, включая самовывоз из магазина.

1. **DNS**

**Описание:**

Интернет-магазин DNS предлагает большой выбор компьютерной и бытовой техники.

**Ключевые функции:**

Сравнение товаров: Возможность сравнивать различные модели и характеристики прямо в приложении.

Отзывы и оценки: Пользователи могут оставлять свои мнения о товарах, что помогает другим в выборе.

Гибкие условия возврата: Удобные условия возврата и обмена товаров.

Специальные предложения и акции: Регулярные скидки и специальные предложения для мобильных пользователей.

1. **Юлмарт**

**Описание:**

Юлмарт — крупный онлайн-магазин электроники с хорошим выбором ПК и комплектующих.

**Ключевые функции:**

Удобный интерфейс: Простое и быстрое создание заказа через приложение.

Поддержка клиентов: Онлайн-консультанты, которые готовы ответить на вопросы и помочь с выбором.

История заказов: Функция просмотра предыдущих покупок для удобства повторного заказа.

Интересные предложения: Систематическое размещение акций и распродаж в отдельном разделе приложения.

1. **Технопарк**

**Описание:**

Технопарк — онлайн-магазин, который предоставляет большой выбор компьютерной техники, включая сборку ПК.

**Ключевые функции:**

Конфигуратор ПК: Функция, позволяющая пользователям собирать компьютер по индивидуальным параметрам.

Отображение характеристик: Подробные спецификации каждого товара.

Доставка: Разнообразные варианты доставки, включая экспресс-доставку в крупных городах.

VIP-обслуживание: Услуги по сборке и настройке ПК для клиентов.

## **1.3. Выводы по анализу**

Проведённый анализ показал, что для создания конкурентоспособного продукта важно учитывать следующие аспекты:

1. Удобство интерфейса: приложение должно быть интуитивным и понятным.
2. Быстродействие: высокая скорость загрузки страниц и товаров играет важную роль в удержании пользователей.
3. Функциональность: необходимо реализовать удобный поиск, фильтры, авторизацию и возможность управления заказами.

Приложение для магазина по продажи компьютеров должно сочетать в себе преимущества существующих решений и учитывать их недостатки, предлагая пользователю уникальные возможности. Особое внимание стоит уделить внедрению административной панели для удобного управления ассортиментом и заказами.

# **Глава 2. Разработка мобильного приложения для магазина по продажи компьютеров**

## **2.1. Функциональные и нефункциональные требования**

Для успешной реализации мобильного приложения важно определить функциональные и нефункциональные требования. Они задают рамки разработки и обеспечивают соответствие конечного продукта ожиданиям пользователей.

**Функциональные требования:**

1. Авторизация и регистрация пользователя:

Вход в приложение через логин и пароль

Регистрация новых пользователей с валидацией данных.

1. Просмотр каталога товаров:

Отображение списка компьютера с изображениями, названием, ценой и описанием.

1. Корзина:

Добавление товаров в корзину.

Удаление товара из корзины

1. Административная панель:

Управление каталогом товаров: добавление, удаление и редактирование.

**Нефункциональные требования:**

1. Интуитивно понятный интерфейс для пользователей с минимальным опытом.
2. Кроссплатформенность: приложение ориентировано на Android.

## **2.2. Функциональные и нефункциональные требования**

Проектирование архитектуры приложения — это важный этап разработки, который влияет на стабильность, производительность и масштабируемость вашего приложения. В структуре вашего приложения на Android, использующего Java, а также взаимодействующего с базой данных MySQL через PHP на локальном сервере XAMPP, можно выделить несколько ключевых компонентов. Давайте разберем каждый из них более подробно.

**1. Общая архитектура приложения**

Клиентская сторона (Android-приложение):

UI (Пользовательский интерфейс): Android-приложение будет состоять из различных активити и фрагментов, которые обеспечивают взаимодействие с пользователем.

Модель (Model): Здесь будут обработаны данные приложения, включая структуру классов, которые представляют логические объекты вашего приложения.

Контроллер (Controller): Фрагменты или активити, содержащие логику управления пользовательским интерфейсом и взаимодействием с моделью.

Сетевой слой: Этот слой будет обрабатывать HTTP-запросы к вашему серверу. Он будет запускать асинхронные операции для взаимодействия с сервером без блокировки UI.

Серверная сторона (PHP-сервер с доступом к MySQL):

PHP-скрипты: Серверная логика будет реализована в PHP. Эти скрипты будут обрабатывать входящие HTTP-запросы от клиента, взаимодействовать с базой данных и возвращать результаты клиенту в формате JSON.

База данных (MySQL): Ваша база данных будет хранить данные приложения. Структура базы данных должна быть оптимизирована для повышения производительности.

**2. Взаимодействие компонентов**

UI: Пользователь взаимодействует с интерфейсом приложения. При выполнении определенных действий (например, нажатии кнопки) будет отправлен HTTP-запрос.

HTTP-запрос: Android-приложение использует библиотеку, такую как HttpURLConnection, для отправки запросов на сервер. Запросы могут быть GET (для получения данных) или POST (для отправки данных).

PHP-скрипт: Сервер получает HTTP-запрос, обрабатывает его, обращаясь к базе данных MySQL и подготавливает ответ.

База данных: PHP-скрипт взаимодействует с MySQL для выполнения операций, таких как выборка или вставка данных.

Ответ на клиент: Сервер отправляет ответ в формате JSON обратно на клиент, который затем разбирает ответ и обновляет UI.

**3. Структура базы данных MySQL**

Users: хранит информацию о пользователях

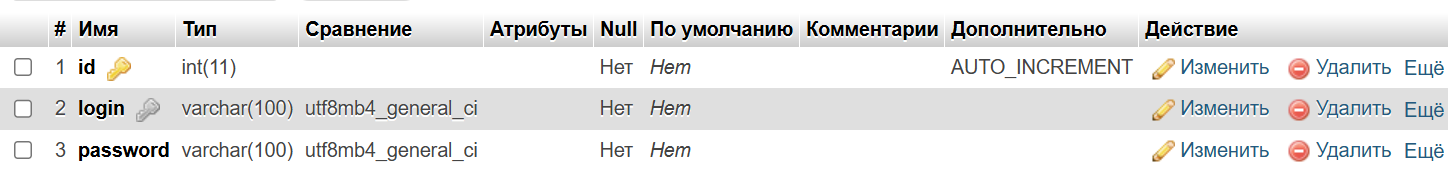


Рис.1 – Структура таблицы Users

Products: содержит данные о продуктах



Рис.2 – Структура таблицы Products

Orders: хранит информацию о заказах

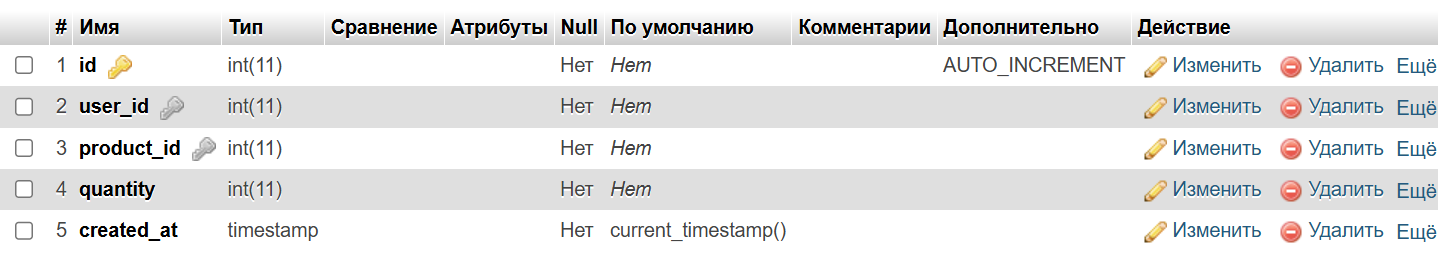


Рис.3 – Структура таблицы Orders

**4. Обработка ошибок**

Необходимо предусмотреть обработку ошибок как на клиентской стороне, так и на серверной. Это включает в себя:

На клиенте: Отображение пользователю уведомлений о проблемах с сетью или ошибках, полученных от сервера.

На сервере: Логирование ошибок и возврат соответствующих кодов состояния HTTP (например, 404 для не найденного ресурса или 500 для внутренней ошибки сервера).

## **2.3 Реализация функционала**

### **2.3.1. Авторизация и регистрация**

RegisterActivity — это класс на платформе Android, который отвечает за процесс регистрации нового пользователя в приложении. Он реализует интерфейс, позволяющий пользователю вводить свои данные для регистрации, такие как логин и пароль. Вот как работает этот класс:

**Основные этапы работы RegisterActivity**

* + 1. **Инициализация элементов интерфейса:**

В методе onCreate происходит установка содержимого активности через setContentView, что загружает интерфейс из XML-файла.

Затем события привязываются к полям ввода и кнопке: пользователю предоставляются три поля для ввода — логина, пароля и подтверждения пароля. Кнопка регистрации, при нажатии, соберет введенные данные.

* + 1. **Валидация ввода:**

Когда пользователь нажимает кнопку регистрации, срабатывает обработчик события.

Сначала проверяется, что все поля заполнены. Если какое-либо из полей пустое, пользователю будет показано сообщение с просьбой заполнить все поля.

Далее проверяется совпадение паролей. Если введенные пароли не совпадают, пользователю снова показывается сообщение с указанием на ошибку.

* + 1. **Регистрация пользователя:**

Если все проверки пройдены, начинается процесс регистрации пользователя. Для этого создает новый поток, чтобы не блокировать основной пользовательский интерфейс.

В этом потоке выполняется HTTP-запрос к серверу (в данном случае к register.php), используя метод GET. Логин и пароль передаются через URL.

После выполнения запроса происходит проверка кода ответа. Если код ответа успешен (HTTP 200), происходит чтение ответа от сервера, который может подтвердить успешность регистрации.

* + 1. **Обработка результатов запроса:**

В случае успеха — пользователю показывается диалоговое окно с сообщением об успешной регистрации. Диалог имеет кнопку для подтверждения, после нажатия на которую пользователь будет перенаправлен на экран входа в систему (LoginActivity).

В случае ошибки регистрируется соответствующее сообщение в логах, что поможет в диагностике проблемы.

* + 1. **Управление потоками:**

Для обработки сетевых операций используется HttpURLConnection, и все сетевые запросы обрабатываются в отдельном потоке, чтобы избежать блокировки основного потока, что важно для обеспечения хорошего пользовательского опыта.

PHP-скрипт для регистрации пользователя:

<?php

require 'db\_connect.php';

$con = connect\_to\_db(); // Используем функцию подключения

if ($\_SERVER["REQUEST\_METHOD"] == "GET") {

    // Проверка наличия данных в GET запросе

    if (isset($\_GET['login']) && isset($\_GET['password'])) {

        // Получение данных из GET запроса

        $login = $\_GET['login'];

        $password = $\_GET['password'];

        // Хранение пароля без хеширования (НЕ БЕЗОПАСНО!)

        $stmt = $con->prepare("INSERT INTO users (login, password) VALUES (?, ?)");

        $stmt->bind\_param("ss", $login, $password);

        // Выполнение запроса

        if ($stmt->execute()) {

            echo "Регистрация успешна!";

        } else {

            echo "Ошибка: " . $stmt->error;

        }

        // Закрытие подготовленного выражения

        $stmt->close();

    } else {

        echo "Пожалуйста, заполните все поля.";

    }

} else {

    echo "Недопустимый запрос.";

}

close\_db\_connection($con); // Закрываем соединение

?>

Фрагмент кода RegisterActivity:

private void registerUser(final String login, final String password) {

new Thread(new Runnable() {

@Override

public void run() {

HttpURLConnection urlConnection = null;

try {

URL url = new URL("http://10.0.2.2/register.php?login=" + login + "&password=" + password);

urlConnection = (HttpURLConnection) url.openConnection();

urlConnection.setRequestMethod("GET");

int responseCode = urlConnection.getResponseCode();

if (responseCode == HttpURLConnection.HTTP\_OK) {

BufferedReader in = new BufferedReader(new InputStreamReader(urlConnection.getInputStream()));

StringBuilder response = new StringBuilder();

String inputLine;

while ((inputLine = in.readLine()) != null) {

response.append(inputLine);

}

in.close();

// Показать успех в главном потоке

mainHandler.post(new Runnable() {

@Override

public void run() {

// Показать сообщение об успехе

new AlertDialog.Builder(RegisterActivity.this)

.setTitle("Регистрация")

.setMessage("Регистрация прошла успешно!")

.setPositiveButton("ОК", (dialog, which) -> {

// Переход на LoginActivity

Intent intent = new Intent(RegisterActivity.this, LoginActivity.class);

startActivity(intent);

finish(); // Завершить текущее Activity, чтобы его не было в стеке

})

.show();

}

});

Log.i("RegisterUser", "Response: " + response.toString());

} else {

Log.e("RegisterUser", "GET request failed, Response Code: " + responseCode);

}

} catch (Exception e) {

Log.e("RegisterUser", "Exception occurred: " + e.getMessage(), e);

} finally {

if (urlConnection != null) {

urlConnection.disconnect();

}

}

}

}).start();

}

}

Ниже представлено изображение интерфейса регистрации

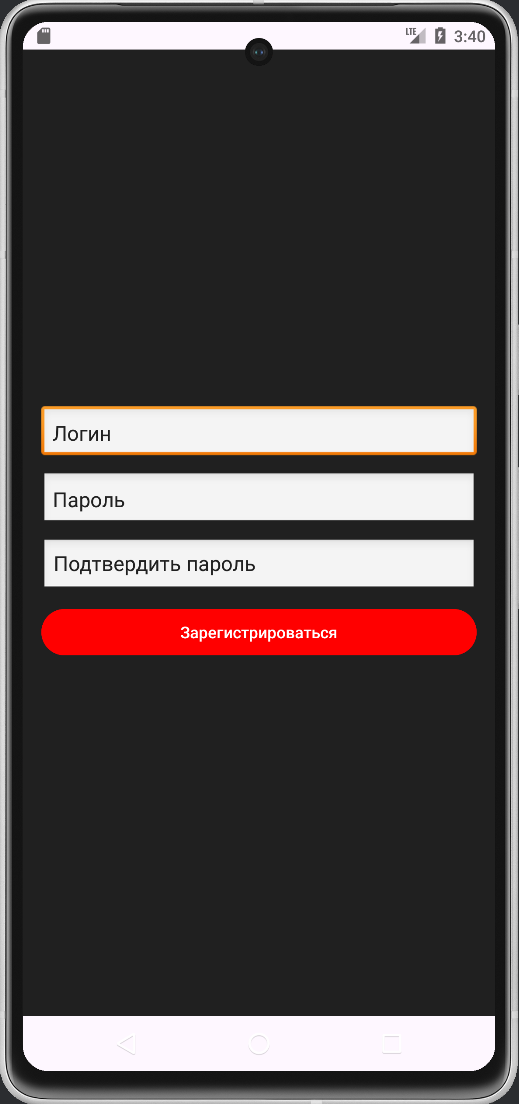


Рис.4 – Интерфейс регистрации

LoginActivity — это класс для Android-приложений, обеспечивающий интерфейс и логику для входа пользователя в приложение. Он управляет вводом логина и пароля, выполняет валидацию этих данных и отправляет запрос на сервер для проверки учетных данных пользователя.

**Основные компоненты и этапы работы LoginActivity:**

* + 1. **Инициализация интерфейса:**

При создании активности (в методе onCreate) происходит установка пользовательского интерфейса из XML-файла через метод setContentView, где размещены поля для ввода логина и пароля, а также кнопка для входа.

Элементы интерфейса, такие как EditText для логина и пароля, а также Button для входа, инициализируются с помощью метода findViewById.

* + 1. **Слушатель нажатий:**

Кнопка "Войти" (buttonLogin) имеет обработчик нажатий, который активируется, когда пользователь нажимает на нее.

Внутри этого обработчика происходит получение введенных логина и пароля, а также проверка их на заполненность. Если хотя бы одно из полей пустое, пользователю отображается сообщение с просьбой заполнить все данные.

* + 1. **Асинхронный вход:**

Если оба поля заполнены, запускается асинхронная задача типа AsyncTask (класс LoginUser), которая отвечает за выполнение сетевого запроса. Это позволяет избежать блокировки основного пользовательского интерфейса во время ожидания ответа от сервера.

В методе doInBackground класса LoginUser формируется URL запроса и открывается HTTP-соединение. Данные (логин и пароль) передаются через запрос с методом GET.

Сервер обрабатывает запрос и возвращает ответ, который затем считывается и возвращается в метод onPostExecute.

* + 1. **Обработка результата**:

Как только ответ от сервера получен, он обрабатывается в методе onPostExecute.

Ответ, возвращенный сервером, ожидается в формате JSON. Приложение проверяет поле "success" в ответе для определения успешности аутентификации.

Если вход успешен, приложение проверяет, является ли пользователь администратором (логин и пароль совпадают с "admin"). В зависимости от этого осуществляется переход либо к AdminActivity, либо к CatalogActivity.

Если вход не удался, пользователю показывается соответствующее сообщение об ошибке.

* + 1. **Завершение активности:**

В конце активности, если пользователь успешно вошел в систему, текущая активность завершается методом finish, что позволяет закрыть экран входа.

PHP-скрипт для авторизации пользователя:

<?php

require 'db\_connect.php';

$con = connect\_to\_db(); // Используем функцию подключения

// Проверяем, были ли переданы параметры

if (!isset($\_GET['login']) || !isset($\_GET['password'])) {

echo json\_encode(["success" => false, "message" => "Не все поля заполнены."]);

exit();

}

$login = $\_GET['login'];

$password = $\_GET['password'];

$sql = "SELECT password FROM users WHERE login = ?";

$stmt = $con->prepare($sql);

if (!$stmt) {

echo json\_encode(["success" => false, "message" => "Ошибка в запросе к базе данных."]);

exit();

}

$stmt->bind\_param("s", $login);

$stmt->execute();

$stmt->store\_result();

if ($stmt->num\_rows > 0) {

$stmt->bind\_result($stored\_password);

$stmt->fetch();

// Сравниваем введённый пароль с сохранённым в базе. Рекомендуется использовать password\_verify

if ($password === $stored\_password) {

echo json\_encode(["success" => true, "message" => "Вход успешен."]);

} else {

echo json\_encode(["success" => false, "message" => "Неверный пароль."]);

}

} else {

echo json\_encode(["success" => false, "message" => "Пользователь не найден."]);

}

$stmt->close();

$con->close(); // Закрываем соединение

?>

Фрагмент кода LoginActivity:

private class LoginUser extends AsyncTask<String, Void, String> {

@Override

protected String doInBackground(String... params) {

String login = params[0];

String password = params[1];

try {

String urlString = LOGIN\_URL + "?login=" + login + "&password=" + password; // Используем GET

URL url = new URL(urlString);

HttpURLConnection conn = (HttpURLConnection) url.openConnection();

conn.setReadTimeout(10000);

conn.setConnectTimeout(15000);

conn.setRequestMethod("GET"); // Устанавливаем метод GET

conn.setDoInput(true);

conn.connect(); // Подключаемся

BufferedReader in = new BufferedReader(new InputStreamReader(conn.getInputStream()));

StringBuilder sb = new StringBuilder();

String line;

while ((line = in.readLine()) != null) {

sb.append(line);

}

in.close();

return sb.toString();

} catch (Exception e) {

e.printStackTrace();

return "Exception: " + e.getMessage();

}

}

@Override

protected void onPostExecute(String result) {

try {

JSONObject res = new JSONObject(result);

if (res.getBoolean("success")) {

Toast.makeText(LoginActivity.this, res.getString("message"), Toast.LENGTH\_LONG).show();

// Проверяем пользователя на admin

String login = editTextLogin.getText().toString().trim();

String password = editTextPassword.getText().toString().trim();

if ("admin".equals(login) && "admin".equals(password)) {

// Переход к AdminActivity

Intent intent = new Intent(LoginActivity.this, AdminActivity.class);

startActivity(intent);

} else {

// Переход к CatalogActivity

Intent intent = new Intent(LoginActivity.this, CatalogActivity.class);

startActivity(intent);

}

finish();

} else {

Toast.makeText(LoginActivity.this, res.getString("message"), Toast.LENGTH\_LONG).show();

}

} catch (Exception e) {

e.printStackTrace();

Toast.makeText(LoginActivity.this, "Ошибка обработки ответа.", Toast.LENGTH\_LONG).show();

}

}

}

}

Ниже представлено изображение интерфейса авторизации

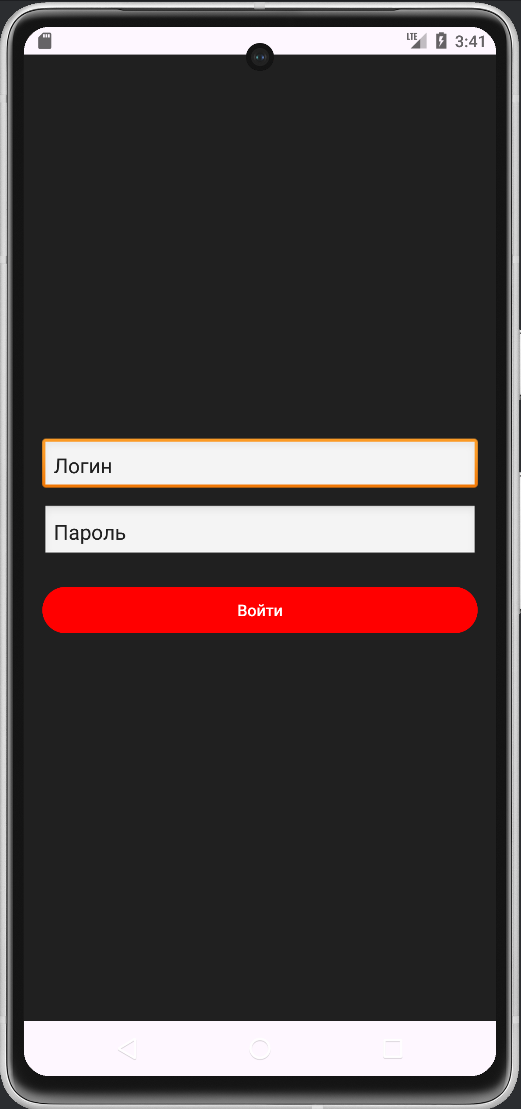


Рис.5 – Интерфейс авторизации

### **2.3.2. Каталог товаров**

PHP-скрипт для получение товаров из БД и отображения их в интерфейсе пользователя:

<?php

require 'db\_connect.php';

$con = connect\_to\_db();

if ($con->connect\_error) {

die("Connection failed: " . $con->connect\_error);

}

$sql = "SELECT id, name, description, price, image\_url FROM products";

$result = $con->query($sql);

$products = array();

if ($result && $result->num\_rows > 0) {

while ($row = $result->fetch\_assoc()) {

$products[] = $row;

}

} else {

// Если запрос вернул пустое значение, выводим пустой массив

echo json\_encode([]);

$con->close();

exit();

}

echo json\_encode($products);

$con->close();

?>

Класс Product, реализующий интерфейс Parcelable, предназначен для не только представления данных о товаре в вашем приложении, но и для упрощения передачи этих данных между компонентами Android, например, между активностями.

**Основные аспекты работы класса Product:**

Свойства класса: Product обычно содержит поля, которые описывают товар, такие как его идентификатор, название, описание, цена и возможно, URL изображения. Эти характеристики позволяют хранить все необходимые данные о продукте в одном объекте.

Реализация Parcelable: Для того чтобы передать объект Product между активностями, класс должен реализовать интерфейс Parcelable. Это включает в себя реализацию методов, которые обеспечивают упаковку и распаковку объекта:

Метод writeToParcel: Этот метод отвечает за запись данных объекта в Parcel, что позволяет упаковать объект для передачи.

Статический метод CREATOR: Этот метод позволяет создать объект Product из данных, хранящихся в Parcel, и используется для распаковки объекта, когда он передается в другой компонент.

Передача данных: Когда объект Product упаковывается в Parcel, он может быть вставлен в Intent (специальный объект для передачи данных между компонентами). Это позволяет отправить объект с одним набором данных прямо в другую активность. Получающая активность может затем извлечь объект, используя методы, предоставляемые интерфейсом Parcelable.

Использование объекта: После получения объекта Product в другой активности, вы можете обращаться ко всем его полям и использовать их для управления пользовательским интерфейсом, например, отображать информацию о товаре, запускать покупку или что-то еще, что соответствует логике вашего приложения.

Фрагмент кода класса Product:

public class Product implements Parcelable {

private int id;

private String name;

private String description;

private double price;

private String imageUrl;

public Product(int id, String name, String description, double price, String imageUrl) {

this.id = id;

this.name = name;

this.description = description;

this.price = price;

this.imageUrl = imageUrl;

}

protected Product(Parcel in) {

id = in.readInt();

name = in.readString();

description = in.readString();

price = in.readDouble();

imageUrl = in.readString();

}

@Override

public void writeToParcel(Parcel dest, int flags) {

dest.writeInt(id);

dest.writeString(name);

dest.writeString(description);

dest.writeDouble(price);

dest.writeString(imageUrl);

}

@Override

public int describeContents() {

return 0;

}

public static final Creator<Product> CREATOR = new Creator<Product>() {

@Override

public Product createFromParcel(Parcel in) {

return new Product(in);

}

@Override

public Product[] newArray(int size) {

return new Product[size];

}

};

// Геттеры:

public int getId() {

return id;

}

public String getName() {

return name;

}

public String getDescription() {

return description;

}

public double getPrice() {

return price;

}

public String getImageUrl() {

return imageUrl;

}

}

ProductAdapter — это класс-адаптер, который используется в Android для отображения списка товаров в RecyclerView. Он управляет представлением элементов списка и связывает данные о товарах с пользовательским интерфейсом. Вот основные аспекты его работы:

**Основные функции ProductAdapter**

Конструктор: Класс принимает два параметра: список товаров (productList) и контекст (context). Список содержит все объекты типа Product, которые нужно отобразить в интерфейсе пользователя, а контекст необходим для доступа к ресурсам приложения, таким как макеты представлений и изображения.

Создание представлений: Метод onCreateViewHolder отвечает за создание нового экземпляра класса ProductViewHolder, который представляет отдельный элемент списка. Здесь происходит инфляция макета элемента (обычно это XML-файл с описанием, как должен выглядеть элемент списка), и создается ViewHolder, который будет хранить ссылки на все UI-элементы в этом макете.

Привязка данных: Метод onBindViewHolder используется для связывания данных о товаре из списка с UI-элементами в соответствующем ViewHolder. В этом методе адаптер обновляет текстовые поля для названия, описания и цены товара, а также загружает изображение товара, используя библиотеку Glide. Этот процесс происходит для каждого элемента списка, который нужно отобразить.

Количество элементов: Метод getItemCount возвращает общее количество товаров в списке. Это важно для того, чтобы RecyclerView знал, сколько элементов ему необходимо отображать.

Обработка событий: Внутри метода onBindViewHolder также устанавливаются обработчики событий для интерактивных элементов, таких как кнопки. Например, при нажатии на кнопку "Добавить в корзину" вызывается метод addProductToCart в CatalogActivity, что позволяет добавить выбранный товар в корзину покупок.

ViewHolder: Вложенный статический класс ProductViewHolder упрощает код, храня ссылки на UI-элементы одного элемента списка. Это уменьшает количество вызовов метода findViewById, в результате чего повышается производительность.

Фрагемнт кода класса ProductAdapter:

@Override

public ProductViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {

View view = LayoutInflater.from(context).inflate(R.layout.product\_item, parent, false);

return new ProductViewHolder(view);

}

@Override

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

Product product = productList.get(position);

holder.productName.setText(product.getName());

holder.productDescription.setText(product.getDescription());

holder.productPrice.setText(String.valueOf(product.getPrice()));

Glide.with(context).load(product.getImageUrl()).into(holder.productImage);

// Обработчик нажатия кнопки "Добавить в корзину"

holder.addToCartButton.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

// Вызываем метод из CatalogActivity для добавления товара в корзину

if (context instanceof CatalogActivity) {

((CatalogActivity) context).addProductToCart(product);

}

}

});

}

@Override

public int getItemCount() {

return productList.size();

}

CatalogActivity — это класс активности в Android-приложении, отвечающий за отображение каталога продуктов и управление взаимодействием пользователя с элементами этого каталога. Вот основные аспекты его работы:

**Основные компоненты CatalogActivity**

* + 1. **Инициализация интерфейса:**

При создании активности (onCreate метод) устанавливается макет пользовательского интерфейса из XML-файла, который содержит различные UI-элементы, такие как RecyclerView для отображения списка продуктов и Toolbar для управления заголовком активности.

RecyclerView получает менеджер компоновки (LinearLayoutManager), который определяет вертикальное расположение элементов в списке.

* + 1. **Работа с данными:**

Создается список productList, который будет хранить объекты продукта. Для отображения списка создан адаптер (ProductAdapter), который связывает данные с визуальными элементами в RecyclerView.

Метод loadProducts выполняет запрос к серверу для получения данных о продуктах с использованием библиотеки Volley. Запрос отправляется в формате JSON, и в случае успешного ответа данные о продуктах извлекаются из JSON-массива.

* + 1. **Обработка данных:**

Полученный JSON-ответ обрабатывается в методе onResponse, где для каждого продукта извлекаются его атрибуты (идентификатор, название, описание, цена и адрес изображения). Создаются объекты Product, которые добавляются в productList.

После того как все продукты были загружены, адаптер уведомляется о необходимости обновления данных, обновляя отображение в RecyclerView.

* + 1. **Корзина покупок:**

CatalogActivity также управляет корзиной товаров. При нажатии на кнопку "Просмотр корзины" (viewCartButton) пользователь перенаправляется на CartActivity, где представлена корзина с товарами. Для передачи данных о товаре в корзину используется Intent, с помощью которого передается список товаров (cartItems).

* + 1. **Добавление товаров в корзину:**

В классе реализован метод addProductToCart, который принимает объект продукта в качестве аргумента и добавляет его в корзину. После добавления продукта, пользователю отображается уведомление о том, что продукт был успешно добавлен в корзину.

Фрагмент кода класса CatalogActivity

private void loadProducts() {

String url = "http://10.0.2.2/get\_products.php";

JsonArrayRequest jsonArrayRequest = new JsonArrayRequest(Request.Method.GET, url, null,

new Response.Listener<JSONArray>() {

@Override

public void onResponse(JSONArray response) {

for (int i = 0; i < response.length(); i++) {

try {

JSONObject productObject = response.getJSONObject(i);

int id = productObject.getInt("id");

String name = productObject.getString("name");

String description = productObject.getString("description");

double price = productObject.getDouble("price");

String imageUrl = productObject.getString("image\_url");

Product product = new Product(id, name, description, price, imageUrl);

productList.add(product);

} catch (JSONException e) {

e.printStackTrace();

}

}

productAdapter.notifyDataSetChanged();

}

}, error -> Log.e("CatalogActivity", "Error fetching products", error));

Volley.newRequestQueue(this).add(jsonArrayRequest);

}

Ниже представлен изображение интерфейса каталога товаров:

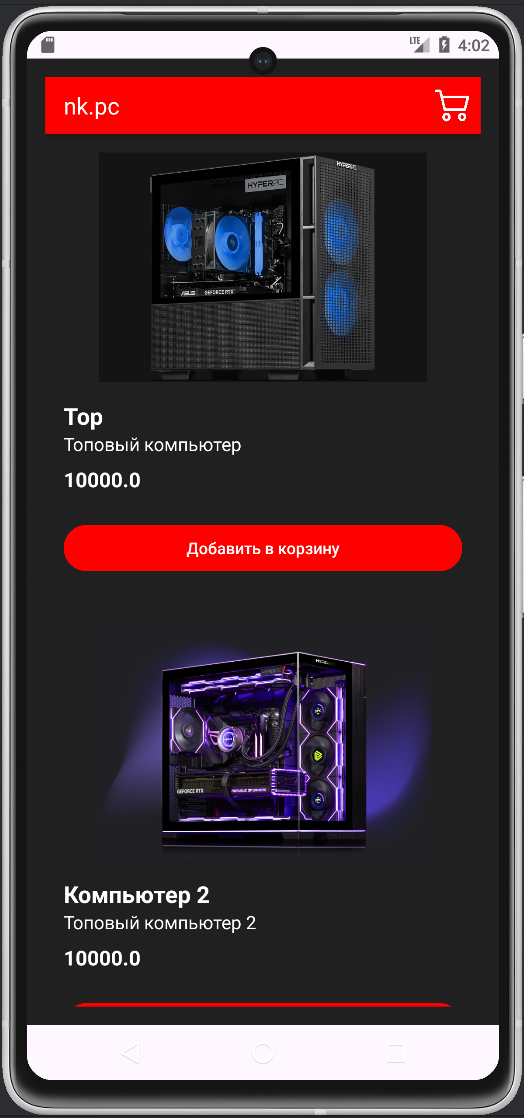


Рис.6 – Интерфейс каталога товаров

### **2.3.3. Административная панель**

Административная панель предназначена для управления каталогом товаров и заказами.

Ниже представлено изображение интерфейса административной панели:

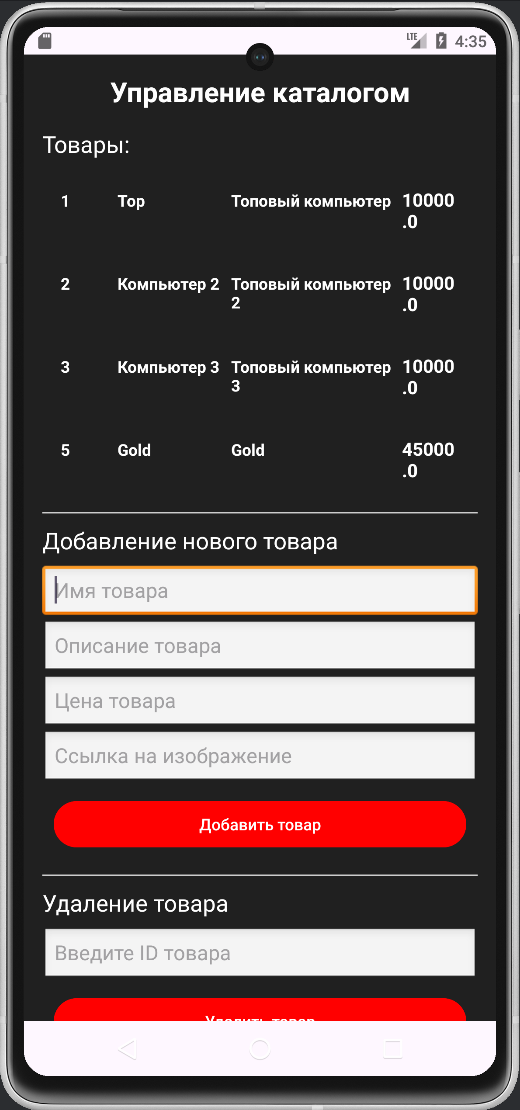


Рис.7 – Интерфейс административной панели

**Добавление товара**

PHP-скрипт для добавления нового товара:

<?php

require 'db\_connect.php';

$con = connect\_to\_db();

// Получение данных из запроса

$name = $\_POST['name'];

$description = $\_POST['description'];

$price = $\_POST['price'];

$image\_url = $\_POST['image\_url'];

// SQL запрос на добавление товара

$sql = "INSERT INTO products (name, description, price, image\_url) VALUES ('$name', '$description', '$price', '$image\_url')";

if ($con->query($sql) === TRUE) {

echo "Товар добавлен";

} else {

echo "Ошибка: " . $sql . "<br>" . $con->error;

}

$con->close();

?>

**Удаление товара**

PHP-скрипт для удаления товара:

<?php

require 'db\_connect.php';

$con = connect\_to\_db();

if ($con->connect\_error) {

die("Connection failed: " . $con->connect\_error);

}

// Получаем ID товара из запроса

if ($\_SERVER['REQUEST\_METHOD'] === 'POST') {

$product\_id = intval($\_POST['id']);

// Проверка, что ID действителен

if ($product\_id > 0) {

// SQL-запрос на удаление товара

$sql = "DELETE FROM products WHERE id = ?";

// Подготовка и выполнение

if ($stmt = $con->prepare($sql)) {

$stmt->bind\_param("i", $product\_id);

if ($stmt->execute()) {

echo json\_encode(["status" => "success", "message" => "Товар успешно удален."]);

} else {

echo json\_encode(["status" => "error", "message" => "Ошибка при удалении товара."]);

}

$stmt->close();

} else {

echo json\_encode(["status" => "error", "message" => "Ошибка подготовки запроса."]);

}

} else {

echo json\_encode(["status" => "error", "message" => "Некорректный ID товара."]);

}

} else {

// Если запрос не является методом POST

echo json\_encode(["status" => "error", "message" => "Неверный метод запроса. Используйте POST."]);

}

$con->close();

?>

**Редактирование товара**

PHP-скрипт для обновления информации о товаре:

<?php

require 'db\_connect.php';

$con = connect\_to\_db();

// Получаем данные

$id = $\_POST['id'];

$name = $\_POST['name'];

$description = $\_POST['description'];

$price = $\_POST['price'];

$image\_url = $\_POST['image\_url'];

// Обновление записи

$sql = "UPDATE products SET name='$name', description='$description', price='$price', image\_url='$image\_url' WHERE id=$id";

if ($con->query($sql) === TRUE) {

echo "Данные успешно обновлены";

} else {

echo "Ошибка обновление данных: " . $con->error;

}

$con->close();

?>

AdminActivity — это класс в приложении Android, отвечающий за управление административными функциями, связанными с товарами в каталоге. Основные функции AdminActivity можно описать следующим образом:

* + 1. **Инициализация интерфейса:** При создании активности загружается макет, где инициализируются поля для ввода информации о товарах (название, описание, цена, URL изображения) и кнопки для добавления, удаления и редактирования товаров.
    2. **Список товаров:** Используется RecyclerView для отображения списка товаров. Вся информация о товарах загружается из сервера и отображается с помощью адаптера AdminProductAdapter.
    3. **Добавление товара:** Когда администратор нажимает кнопку добавления, приложение проверяет, заполнены ли все необходимые поля. Если да, отправляется запрос на сервер для добавления нового товара. Если операция успешна, обновляется список товаров.
    4. **Удаление товара:** Администратор может ввести ID товара для его удаления. При этом после подтверждения удаляется товар с сервера, и обновляется список товаров.
    5. **Редактирование товара:** Администратор может редактировать существующий товар, заполнив необходимые поля и указав его ID. После редактирования товар обновляется на сервере, и список тоже обновляется.
    6. **Асинхронные операции:** Все сетевые запросы (добавление, удаление, редактирование и получение списка товаров) выполняются в фоновом потоке с помощью AsyncTask, что позволяет избежать блокировки пользовательского интерфейса.
    7. **Обработка ошибок:** В случае неудачных операций, таких как ошибки сети или пустые поля, администратору отображаются соответствующие сообщения.

Фрагмент кода класса AdminActivity:

private void getProducts() {

new GetProductsTask().execute();

}

private class GetProductsTask extends AsyncTask<Void, Void, List<Product>> {

@Override

protected List<Product> doInBackground(Void... voids) {

List<Product> products = new ArrayList<>();

try {

URL url = new URL(GET\_PRODUCTS\_URL);

HttpURLConnection connection = (HttpURLConnection) url.openConnection();

connection.setRequestMethod("GET");

InputStream inputStream = connection.getInputStream();

BufferedReader reader = new BufferedReader(new InputStreamReader(inputStream));

StringBuilder response = new StringBuilder();

String line;

while ((line = reader.readLine()) != null) {

response.append(line);

}

reader.close();

JSONArray jsonArray = new JSONArray(response.toString());

for (int i = 0; i < jsonArray.length(); i++) {

JSONObject jsonObject = jsonArray.getJSONObject(i);

int id = jsonObject.getInt("id");

String name = jsonObject.getString("name");

String description = jsonObject.getString("description");

double price = jsonObject.getDouble("price");

String imageUrl = jsonObject.getString("image\_url");

products.add(new Product(id, name, description, price, imageUrl));

}

} catch (Exception e) {

e.printStackTrace();

}

return products;

}

@Override

protected void onPostExecute(List<Product> products) {

super.onPostExecute(products);

// Обновляем адаптер с новыми данными

productList.clear(); // Очищаем существующий список

productList.addAll(products); // Добавляем новые продукты

productAdapter.notifyDataSetChanged(); // Уведомляем адаптер об изменениях

}

}

private class AddProductTask extends AsyncTask<String, Void, String> {

@Override

protected String doInBackground(String... params) {

String name = params[0];

String description = params[1];

String price = params[2];

String imageUrl = params[3];

try {

URL url = new URL(ADD\_PRODUCT\_URL);

HttpURLConnection connection = (HttpURLConnection) url.openConnection();

connection.setRequestMethod("POST");

connection.setDoOutput(true);

String postData = "name=" + URLEncoder.encode(name, "UTF-8") +

"&description=" + URLEncoder.encode(description, "UTF-8") +

"&price=" + URLEncoder.encode(price, "UTF-8") +

"&image\_url=" + URLEncoder.encode(imageUrl, "UTF-8");

OutputStream os = connection.getOutputStream();

os.write(postData.getBytes());

os.flush();

os.close();

return String.valueOf(connection.getResponseCode());

} catch (Exception e) {

e.printStackTrace();

return "Ошибка: " + e.getMessage();

}

}

@Override

protected void onPostExecute(String result) {

super.onPostExecute(result);

if (result.equals("200")) {

Toast.makeText(AdminActivity.this, "Товар добавлен успешно", Toast.LENGTH\_SHORT).show();

// Очистка полей ввода

editTextProductName.setText("");

editTextProductDescription.setText("");

editTextProductPrice.setText("");

editTextProductImageUrl.setText("");

// Обновляем список товаров после добавления

getProducts();

} else {

Toast.makeText(AdminActivity.this, "Ошибка при добавлении товара", Toast.LENGTH\_SHORT).show();

}

}

}

private class DeleteProductTask extends AsyncTask<String, Void, String> {

@Override

protected String doInBackground(String... params) {

String productId = params[0];

try {

URL url = new URL("http://10.0.2.2/delete\_product.php"); // Замените на ваш URL для удаления

HttpURLConnection connection = (HttpURLConnection) url.openConnection();

connection.setRequestMethod("POST");

connection.setDoOutput(true);

String postData = "id=" + URLEncoder.encode(productId, "UTF-8");

OutputStream os = connection.getOutputStream();

os.write(postData.getBytes());

os.flush();

os.close();

return String.valueOf(connection.getResponseCode());

} catch (Exception e) {

e.printStackTrace();

return "Ошибка: " + e.getMessage();

}

}

@Override

protected void onPostExecute(String result) {

super.onPostExecute(result);

if (result.equals("200")) {

Toast.makeText(AdminActivity.this, "Товар удалён успешно", Toast.LENGTH\_SHORT).show();

// Обновляем список товаров после удаления

getProducts();

} else {

Toast.makeText(AdminActivity.this, "Ошибка при удалении товара", Toast.LENGTH\_SHORT).show();

}

}

}

private class EditProductTask extends AsyncTask<String, Void, String> {

@Override

protected String doInBackground(String... params) {

String productId = params[0];

String newName = params[1];

String newDescription = params[2];

String newPrice = params[3];

String newImageUrl = params[4];

try {

URL url = new URL("http://10.0.2.2/edit\_product.php"); // Замените на ваш URL для редактирования

HttpURLConnection connection = (HttpURLConnection) url.openConnection();

connection.setRequestMethod("POST");

connection.setDoOutput(true);

// Обновляем данные, которые отправляем на сервер

String postData = "id=" + URLEncoder.encode(productId, "UTF-8") +

"&name=" + URLEncoder.encode(newName, "UTF-8") +

"&description=" + URLEncoder.encode(newDescription, "UTF-8") +

"&price=" + URLEncoder.encode(newPrice, "UTF-8") +

"&image\_url=" + URLEncoder.encode(newImageUrl, "UTF-8");

OutputStream os = connection.getOutputStream();

os.write(postData.getBytes());

os.flush();

os.close();

return String.valueOf(connection.getResponseCode());

} catch (Exception e) {

e.printStackTrace();

return "Ошибка: " + e.getMessage();

}

}

@Override

protected void onPostExecute(String result) {

super.onPostExecute(result);

if (result.equals("200")) {

Toast.makeText(AdminActivity.this, "Товар успешно отредактирован", Toast.LENGTH\_SHORT).show();

// Очищаем поля ввода

editTextEditProductId.setText("");

editTextEditProductName.setText("");

editTextProductDescription.setText("");

editTextProductPrice.setText("");

editTextProductImageUrl.setText("");

// Обновляем список товаров после редактирования

getProducts();

} else {

Toast.makeText(AdminActivity.this, "Ошибка при редактировании товара", Toast.LENGTH\_SHORT).show();

}

}

}

}

**2.3.4. Корзина**

CartActivity в приложении Android представляет собой экран, на котором пользователи могут просматривать и управлять товарами, добавленными в корзину. Основные функции и работа этого класса описываются следующим образом:

1. **Инициализация компонентов:** При создании активности загружается макет, и инициализируются элементы пользовательского интерфейса, такие как ListView для отображения товаров в корзине и кнопка "Оформить заказ".
2. **Получение данных корзины:** Товары, добавленные в корзину, передаются в CartActivity через Intent. Эти товары хранятся в списке cartItems.
3. **Отображение товаров:** Метод displayCartItems() отвечает за формирование списка с деталями о товарах (названия и цены) и отображение его с помощью адаптера. Пользователь может кликнуть на элемент в списке для его удаления из корзины, после чего обновляется интерфейс.
4. **Добавление товара:** Метод addProductToCart(Product product) позволяет добавлять новый товар в корзину, а также выводить уведомление о добавлении.
5. **Оформление заказа:** При нажатии кнопки "Оформить заказ" запускается метод placeOrder(), который отправляет информацию о заказанных товарах на сервер с помощью библиотеки Volley. Для каждого товара из корзины создается запрос, в котором передается идентификатор пользователя, идентификатор товара и количество.
6. **Обработка ответов:** В случае успешного выполнения запроса пользователю выводится уведомление с ответом сервера, а в случае ошибки — сообщение об ошибке.

Ниже представлен интерфейс корзины:



Рис.8 – Интерфейс корзины

Фрагмент кода класса CartActivity:

public void addProductToCart(Product product) {

cartItems.add(product); // Добавляем продукт в корзину

Toast.makeText(this, product.getName() + " добавлен в корзину", Toast.LENGTH\_SHORT).show(); // Уведомление

}

private void displayCartItems() {

ArrayList<String> itemDetails = new ArrayList<>();

for (Product product : cartItems) {

itemDetails.add(product.getName() + " - " + product.getPrice() + "₽");

}

ArrayAdapter<String> adapter = new ArrayAdapter<String>(this, R.layout.list\_item, itemDetails) {

@Override

public View getView(int position, View convertView, ViewGroup parent) {

View view = super.getView(position, convertView, parent);

TextView textView = view.findViewById(R.id.item\_text);

textView.setText(itemDetails.get(position));

return view;

}

};

listView.setOnItemClickListener((parent, view, position, id) -> {

// Удаляем товар из списка cartItems

Product productToRemove = cartItems.get(position);

cartItems.remove(position);

// Обновляем отображение списка

displayCartItems();

// Если необходимо, можно добавить уведомление о том, что товар удалён

Toast.makeText(CartActivity.this, productToRemove.getName() + " был удалён из корзины", Toast.LENGTH\_SHORT).show();

});

listView.setAdapter(adapter);

# **Заключение**

В процессе разработки мобильного приложения для магазина по продаже компьютеров была выполнена полная реализация проекта, начиная с постановки задач и анализа требований, заканчивая созданием работающего программного продукта. Приложение включает в себя все ключевые функции, необходимые для удобного взаимодействия пользователей с каталогом товаров и оформления заказов.

Были достигнуты следующие результаты:

1. **Анализ предметной области и существующих решений**. На основе исследования современных тенденций мобильной коммерции были выделены основные требования к приложению. Учет актуальных трендов позволил разработать продукт, который соответствует ожиданиям пользователей.
2. **Проектирование и реализация приложения**. Разработаны интерфейсы для работы с каталогом, корзиной, системой авторизации и регистрации пользователей. В приложение также интегрирована административная панель для управления товарами и заказами.
3. **Функциональные особенности**. Пользователи могут просматривать каталог, добавлять товары в корзину, оформлять заказы. Для владельцев бизнеса реализована возможность управления ассортиментом через удобный интерфейс.
4. **Практическая значимость**. Разработанное приложение может быть использовано как готовое решение для небольших и средних магазинов специализирующихся на продаже компьютеров. Это позволит оптимизировать процессы торговли, улучшить пользовательский опыт и повысить уровень продаж.

# **Список литературы**

1. Шимкин, Д. А. Основы программирования на Java для Android. — М.: БХВ-Петербург, 2021. — 280 с.
2. Фрэнк, М. Мобильные приложения для Android: от идеи до реализации. — М.: Основание, 2020. — 256 с.
3. Android Developers. Документация по Android [Электронный ресурс]. — Режим доступа: https://developer.android.com/docs. — Дата обращения: 15.11.2023.
4. Закутин, В. Б. Проектирование пользовательского интерфейса в мобильных приложениях. — М.: Наука, 2021. — 215 с.
5. Кузнецов, А. В. Лучшие практики разработки Android-приложений. — М.: Альпина Паблишер, 2023. — 320 с.