

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВАСТОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Кафедра «Информационные
технологии и компьютерные
системы»

ОТЧЕТ

о выполнении итоговой лабораторной работы
по дисциплине: «Технология разработки SaaS приложений»

Вариант 7

Выполнили:

ст.гр. ИВТ/б-21-2-о

Чубаров Д.Е.

Сеноженко И.С.

Проверил:

Малахов С.В.

Севастополь, 2025 г.

Цель работы:

Подготовить отчет по проделанной работе, состоящий из отчетов лабораторных работ

Постановка задачи

В отчете все рисунки должны быть с пояснениями. В отчете должна присутствовать ссылка на репозиторий. Оформление должно быть в едином стиле. Заголовки и подзаголовки должны соответствовать пунктам заданий из ЛР.

Вариант показан на рисунке 1.

7. **Список покупок.** Простое приложение для создания и управления списками покупок. Пользователи могут добавлять товары, разделять их на категории и отмечать уже купленные.

Рисунок 1 — Вариант 1

Ход работы:

1) Составить список основных функциональных требований.

- Добавление продукта:

Назначение: Позволяет пользователю добавлять товары в список.

Ожидаемое поведение: Пользователь вводит название и количество товара и нажимает кнопку "Добавить".

- Отметка купленных товаров:

Назначение: Позволяет пользователю отмечать купленные товары.

Ожидаемое поведение: Пользователь нажимает на товар, чтобы отметить его как купленный, изменяя его статус в списке.

- Удаление товаров:

Назначение: Удаляет ненужные товары из списка.

Ожидаемое поведение: Пользователь нажимает кнопку "Удалить" рядом с товаром.

- Сортировка и поиск товаров:

Назначение: Позволяет пользователю быстро находить и сортировать товары.

Ожидаемое поведение: Пользователь может использовать поля поиска и сортировки, чтобы увидеть только нужные товары.

- Разделение товаров на категории:

Назначение: Упрощает организацию списка.

Ожидаемое поведение: Пользователь может выбирать категорию товаров для упрощения поиска.

- История покупок:

Назначение: Позволяет пользователю отслеживать свои покупки.

Ожидаемое поведение: Пользователь может просматривать прошлые покупки и анализировать свои расходы.

2) Разработать диаграмму сценариев использования.

Сценарий использования приложения To Do List показан на рисунке 2.

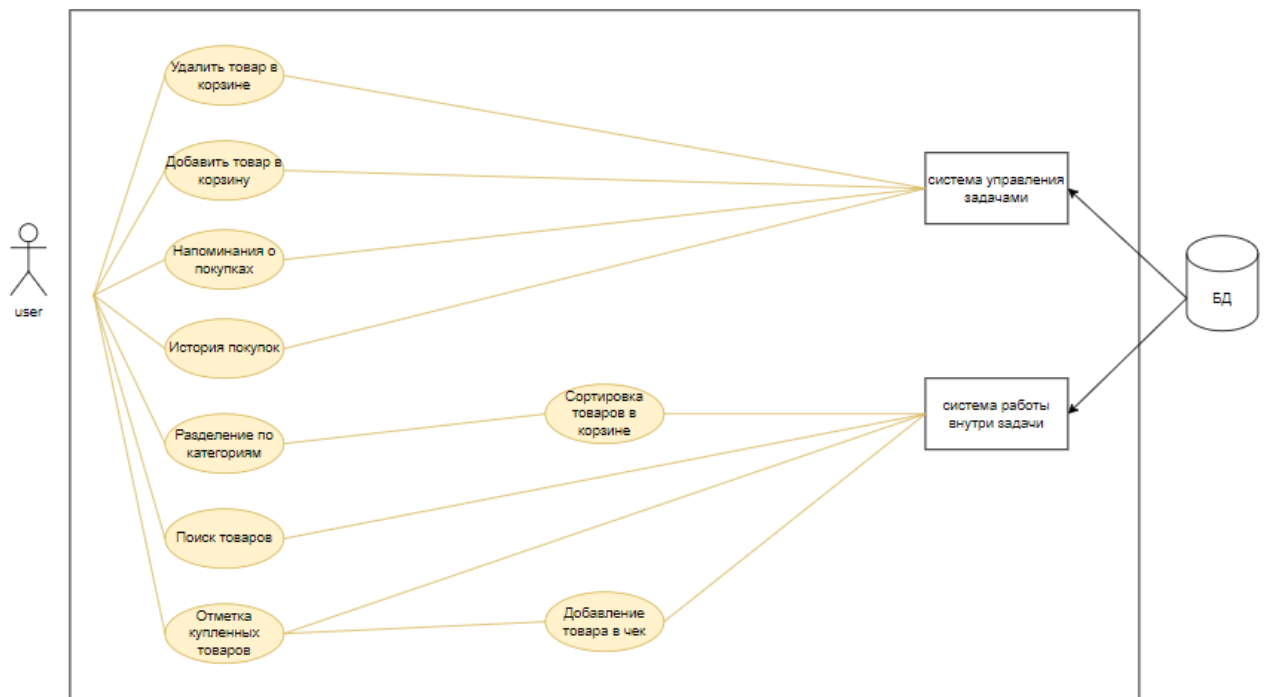


Рисунок 2 - Диаграмма сценариев использования

3) Составить список нефункциональных требований.

- Определение требований к производительности:

Максимальное время отклика на пользовательские запросы:

Приложение должно отвечать на действия пользователя не более чем за 1 секунду. Это важно для поддержания положительного пользовательского опыта, особенно на мобильных устройствах с низким уровнем мощности или при низкой скорости соединения.

Максимальная нагрузка, которую должно выдерживать приложение:

Приложение должно быть способно обрабатывать до 1,000 одновременных пользователей без значительного ухудшения производительности.

- Определение требований к безопасности:

Политики защиты данных пользователей:

Защита данных в данном типе приложения не требуется, так данные не являются конфиденциальными.

Требования к аутентификации и авторизации:

Аутентификация и авторизация не требуются.

Шифрование данных и коммуникаций:

Шифрование данных и коммуникаций не требуется

- Требования к масштабируемости:

Возможности горизонтального и вертикального масштабирования:

Приложение должно поддерживать горизонтальное масштабирование для увеличения количества пользователей. Также возможна поддержка вертикального масштабирования (увеличение мощности существующих серверов), если будет необходимость обработки больших объемов данных или более высоких требований к вычислениям.

- Требования к доступности и отказоустойчивости:

Минимально допустимое время простоя:

Приложение должно быть доступно для пользователей 99.9% времени, что допускает не более 8.76 часов простоя в год. Это требование обеспечивает высокую доступность, особенно для пользователей, которые используют приложение для ежедневных покупок.

Требования к резервному копированию и восстановлению данных:

Данные пользователей должны автоматически резервироваться ежедневно, а хранение резервных копий должно осуществляться минимум за последние 30 дней. В случае сбоя данные должны быть восстановлены в течение 1 часа, чтобы минимизировать потери для пользователей.

Интеграция с внешними сервисами и API:

Приложение должно поддерживать интеграцию с внешними API для добавления товаров из других приложений или веб-сайтов.

- Требования к удобству использования (usability):

Требования к пользовательскому интерфейсу:

Интерфейс должен быть простым и интуитивно понятным для пользователей разного уровня технической грамотности. Необходимо минимизировать количество шагов, необходимых для выполнения задач (например, добавление товара или создание новой категории). Пользовательский интерфейс должен адаптироваться к различным экранам и поддерживать мультизадачность.

4) Разработать спецификацию требований.

Спецификация требований:

- Разработка структурированного документа:

Документ должен содержать все собранные функциональные и нефункциональные требования, структурированные по секциям, чтобы обеспечить легкость восприятия и доступность для всех участников процесса. В документе необходимо указать, какие технологии будут использоваться для реализации требований.

- Определение зависимостей и ограничений:

Приложение может иметь ограничения, такие как лимиты на количество списков для бесплатных пользователей, зависимость от внешних API для работы с определенными товарами, или ограничения по памяти и вычислительным мощностям на низко производительных устройствах.

- Создание схем и диаграмм:

Создать блок-схемы, описывающие основные процессы: создание списка, добавление товара, пометка товара как купленного. Разработать диаграммы, показывающие архитектуру системы, включая серверную и клиентскую части, API-шлюзы и базы данных. Эти диаграммы помогут визуализировать, как разные компоненты приложения взаимодействуют друг с другом.

- Утверждение спецификации с заинтересованными сторонами:

После разработки документа необходимо провести встречи с командой разработки, менеджерами проектов и пользователями для обсуждения требований. Все правки, предложенные заинтересованными сторонами, должны быть учтены и отражены в финальной версии документа.

- Подготовка к передаче в разработку:

После всех правок и утверждений необходимо завершить документ, убедившись, что он включает все необходимые требования, схемы и диаграммы. Документ должен быть представлен команде разработчиков в удобной форме, чтобы они могли начать работу над проектом, опираясь на четкие и согласованные требования.

5) Определить основные модули и компоненты приложения.

Существуют такие функциональные модули:

- Модуль "Список покупок":
- Модуль "Поиск товаров":
- Модуль "История покупок":

Определим нефункциональные компоненты:

Главным компонентом будет являться "Автосохранение данных". Важнейший компонент, который позволяет пользователю не терять данные при неожиданном выходе из приложения или сбое. Все изменения в списке покупок

автоматически сохраняются, чтобы пользователь мог в любой момент вернуться к работе с актуальными данными.

6) Разработать диаграмму архитектуры приложения.

Диаграмма архитектуры показана на рисунке 3.

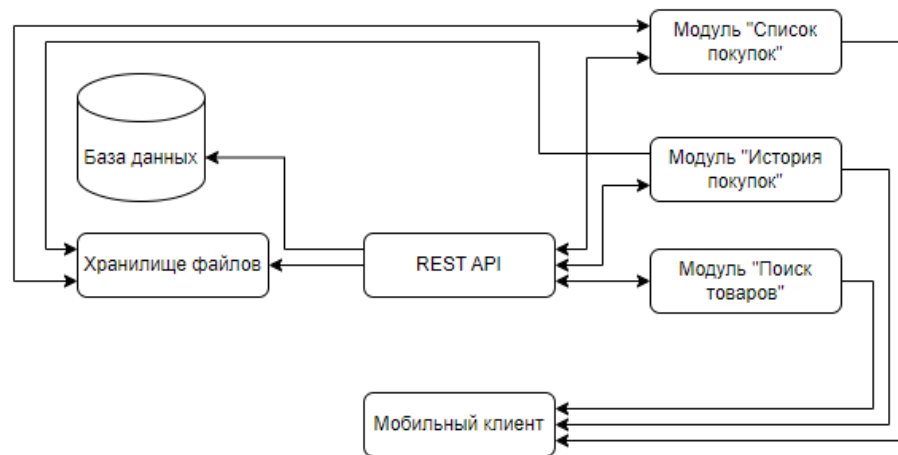


Рисунок 3 – Диаграмма архитектуры

7) Определить используемые технологии и инструменты.

- Выбор стека технологий:

Для разработки приложения по управлению списком покупок будет выбран Java как основной язык программирования, так как он идеально подходит для разработки под платформу Android. Для реализации интерфейса будет использоваться Android SDK, который предоставляет современный и гибкий подход к созданию UI в Android-приложениях. Он позволяет быстрее разрабатывать приложения, улучшает читаемость кода и упрощает взаимодействие с базой данных.

- Выбор базы данных:

Для данного приложения будет выбрана sqlite база данных, которая позволяет сохранять и извлекать данные о покупках, списках товаров и информации о пользователях. Она является легкой и быстрой для мобильных

приложений, при этом обеспечивает хорошую производительность при малых и средних объемах данных.

- Выбор средств для API и коммуникаций:

Для реализации API и обмена данными между приложением и сервером будет использована технология REST для работы с HTTP-запросами. Это идеальный выбор для мобильных приложений, так как он позволяет быстро и эффективно взаимодействовать с сервером, отправлять и получать данные о товарах, статусах покупок и скидках. Все данные о товарах, статусах покупок, уведомлениях и аналитике будут передаваться в формате JSON, что удобно для мобильных приложений и широко поддерживается как в Android, так и в серверах.

- Определение инструментов CI/CD:

Docker целесообразно применять для разработки программного обеспечения, включая настройку, создание и развёртывания контейнеров для быстрого и удобного масштабирования серверов приложения. Kubernetes подходит для операций с кластерами контейнеров, решая проблемы автоматизации развёртывания, работы в сети, масштабирования и мониторинга.

- Инструменты мониторинга и логирования:

Для мониторинга и логирования будет использоваться Logcat — встроенный инструмент в Android Studio, который позволяет отслеживать логи приложения в процессе разработки и тестирования. Это поможет в отладке и мониторинге поведения приложения.

- Выбор средств аутентификации и безопасности:

Так как в приложении отсутствуют конфиденциальные данные пользователей, средства аутентификации и безопасности использоваться не будут. Опишем архитектуру системы с учетом будущего масштабирования:

8) Описать архитектуру системы с учетом будущего масштабирования.

Для приложения по управлению списком покупок вертикальное масштабирование можно использовать на начальных этапах, когда нагрузка на систему ещё не слишком велика. По мере роста приложения горизонтальное масштабирование становится более приоритетным. Это добавление новых серверов для распределения нагрузки. Кроме того, будет внедрен механизм кэширования для ускорения доступа к часто запрашиваемым данным. Для повышения отказоустойчивости будут применена технология репликация базы данных. Это позволит системе оставаться в рабочем состоянии в случае сбоя отдельных компонентов.

Для мониторинга и автоматического масштабирования будем использовать функционал Kubernetes, который позволяет развертывать контейнеры и мониторить их состояние.

9) Разработать макеты интерфейса для основных экранов приложения.

На изображены экраны:

- Домашняя страница;
- Поиск товаров;
- Выбор категории;
- Страница с отфильтрованными продуктами;
- Страница с поиском продуктов по ключевым словам;
- Страница с изменением количества продукта;
- Страница с корзиной
- Страница с историей

- Страница с выбором периода в истории покупок
- Страница с выбором периода в истории покупок
- Страница товаром, отмеченным как купленный

Все меню, разработанное в приложении Figma показано на рисунке 4.

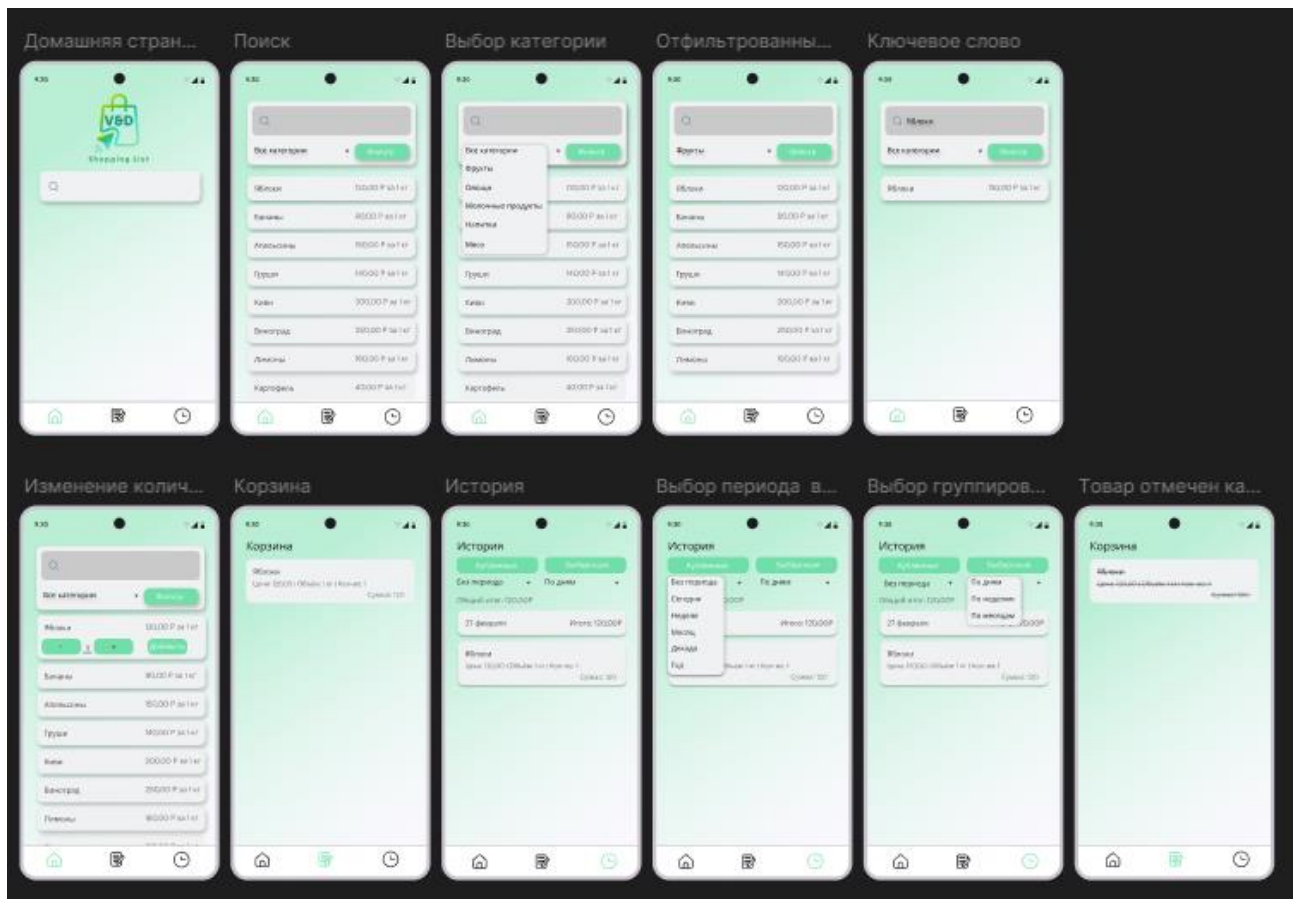


Рисунок 4 – Полный дизайн всего приложения

10) Создать интерактивный прототип.

Интерактивным прототипом можно воспользоваться по ссылке:

<https://www.figma.com/design/6I0GcwOI24wIKgSIEMncl/Untitled?node-id=0-1&t=bGN22Tp55nsYI0Q0-1>

11) Провести тестирование прототипа на целевой аудитории и собрать обратную связь.

Прототип приложения был протестирован среди 15 человек. В результате была получена положительная обратная связь, с отметкой незначительных ошибок, которые впоследствии были исправлены. Тестирование показало, что интерфейс приложения был понятен и лаконичен.

12) Настроить репозиторий кода.

Репозиторий программы хранится на персональном компьютере разработчика. Так как разработка ведется на платформе Android studio, то и репозиторий всего кода локальный, а именно находится в папке с проектом, как показано на рисунке 5.

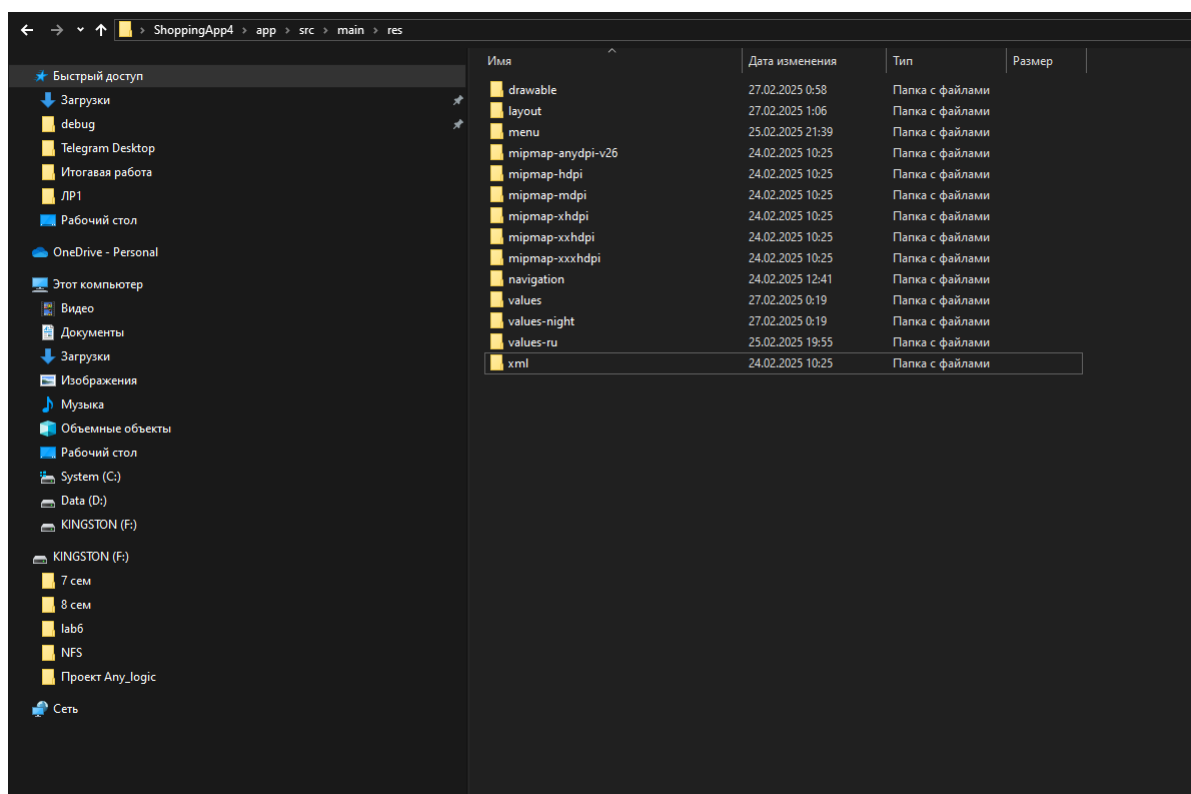


Рисунок 5 – Репозиторий кода

Метод локального сохранения был выбран по нескольким причинам:

- Автономная работа - возможность работать с репозиторием, без привязки к постоянному интернет-соединению;

- Безопасность изменений данных - при такой работе можно не беспокоиться об утечке кода, так как прямого доступа к корневым файлам проекта через интернет не существует

- Возможность работать с несколькими частями приложения, изменять их, при этом не изменяя и не затрагивая основную часть приложения.

13) Настроить среду разработки и тестирования.

Выполним настройку среды разработки тестирования. После установки Android Studio на компьютер создадим новый проект. Для этого выберем пресет для нашего приложения из предложенных (Рисунок 6).

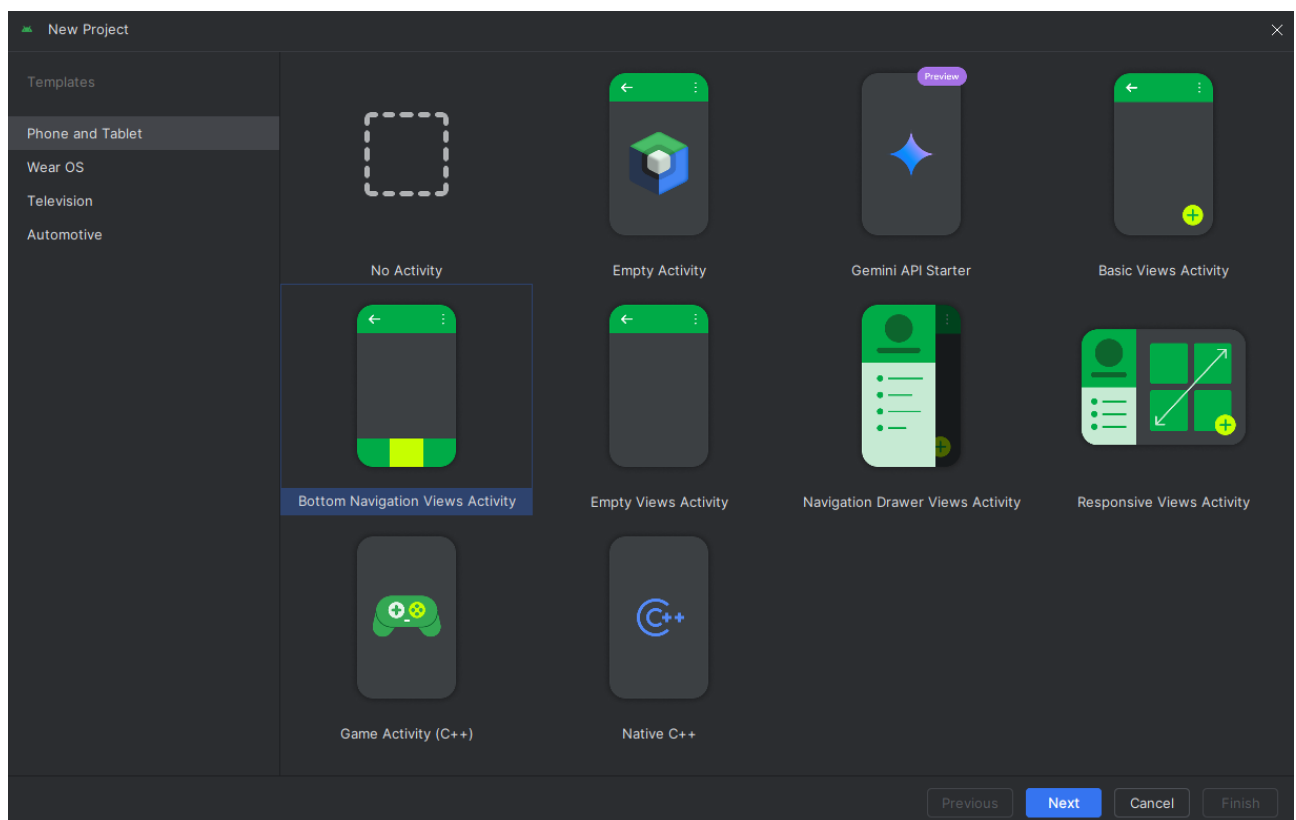


Рисунок 6 – Выбор пресета для приложения.

Из предложенных был выбран Bottom Navigation Views Activity. Он позволяет создать проект с базовыми настройками для Navigation Bar, то есть для перемещения между разными Activity приложения по нажатию кнопки в нижней части экрана. После чего жмем “Next” и переходим к следующей настройке проекта.

В данном окне (Рисунок 7) необходимо выбрать название нашего проекта, имя пакета, место сохранения проекта, язык программирования (можно выбрать Java или Kotlin), минимальную конфигурацию SDK, а также язык, на котором будет происходить сборка нашего проекта в APK файл.

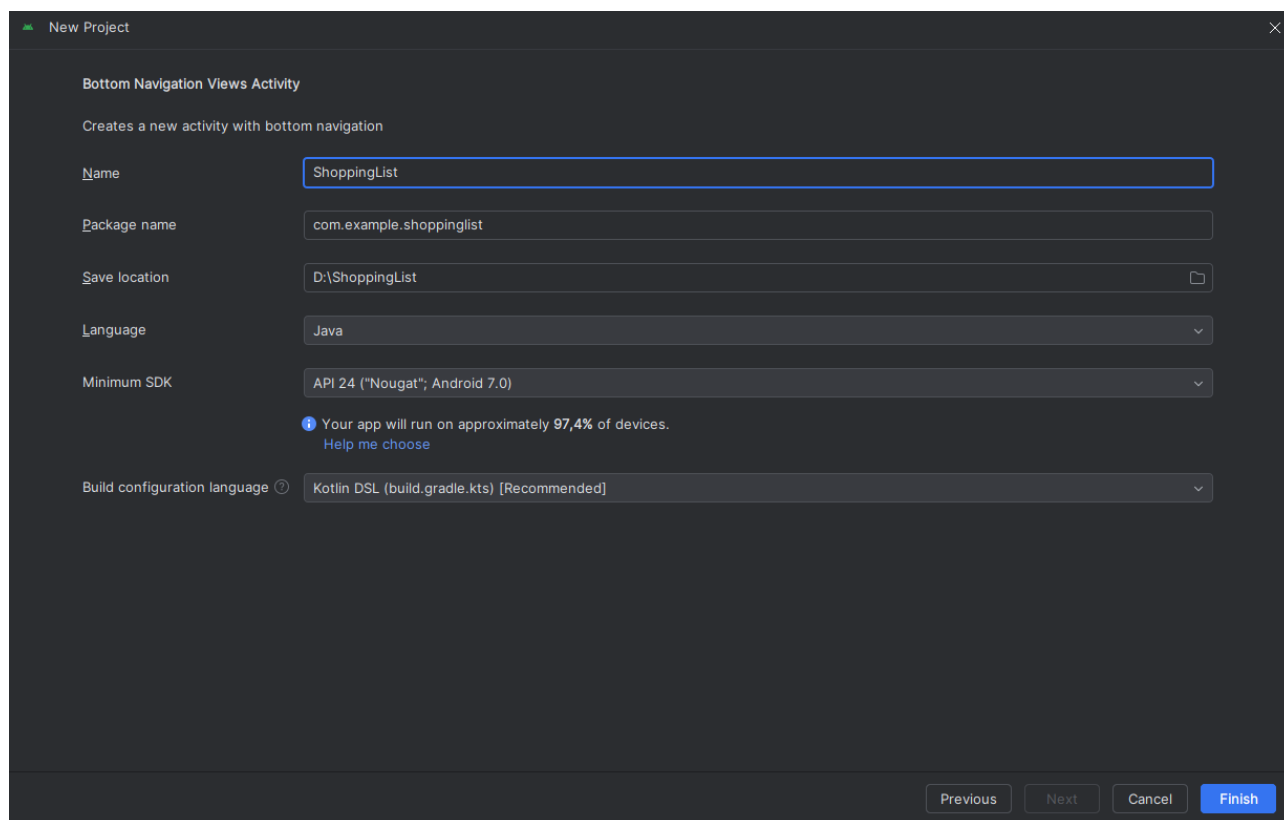


Рисунок 7 - Дополнительные настройки

После чего жмем на кнопку “Finish” и ждем пока Android Studio подгрузит все необходимые компоненты и настройки для корректной работы приложения. Должно появиться рабочее пространство, на котором уже есть Navigation Bar в базовой настройке. Готовый для работы проект изображен на рисунке 8.

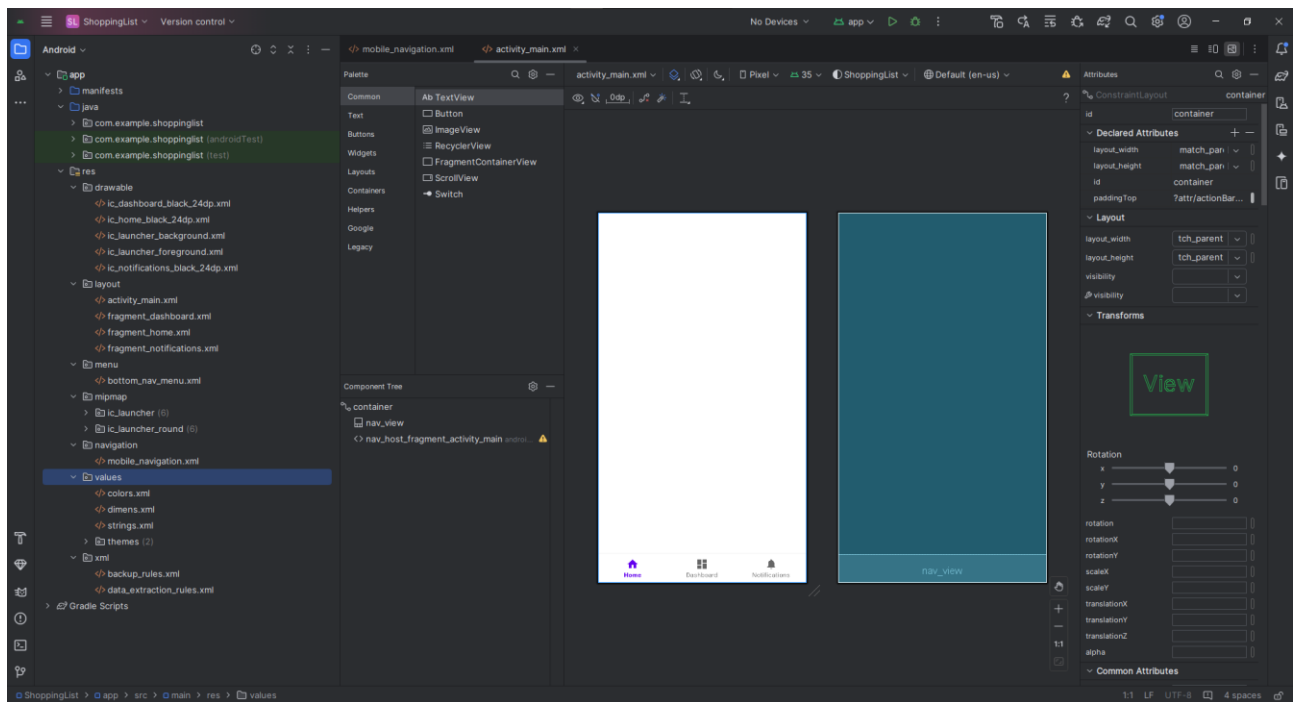


Рисунок 8 - Готовый для работы проект изображен

14) Настроить инструменты для автоматического тестирования и развертывания.

Для автоматического тестирования приложения можно воспользоваться сайтами, которые специализируются на данной тематике. Есть возможность подобрать необходимые настройки и протестировать конкретные части и элементы приложения. Одним из подобных сайтов является lambdatest.com, который изображен на рисунке 8.

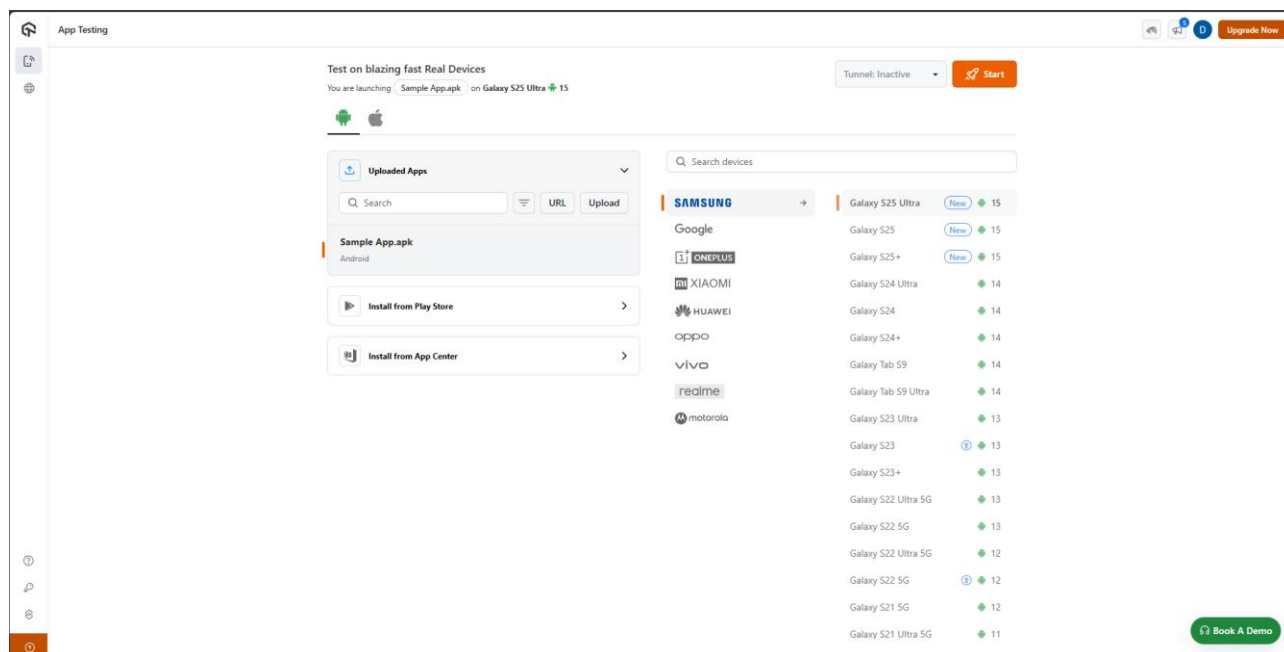


Рисунок 8 - Сайт для автоматического тестирования мобильного приложения

15) Разработать API для работы с клиентской частью.

Для взаимодействия с клиентской частью, необходимо создать API и определить маршруты эндпоинтов по которым можно будет обращаться к сайту и получать необходимые данные или функционал. В данном случае мы будем загружать данные с сайта в приложение.

Ниже приведён код API со стороны сайта:


```

class Product(Resource):
    """Работа с полным списком продуктов"""

    def get(self):
        """Получение всех товаров"""
        with get_db_connection() as db:
            products = db.execute('SELECT * FROM products').fetchall()
            return jsonify([dict(row) for row in products])

    def post(self):
        """Добавление нового товара"""
        data = request.get_json()
        name = data.get('name')
        price = data.get('price')
        category = data.get('category')
        product_type = data.get('type')
        unit = data.get('unit')
        volume = data.get('volume')

        if not all([name, price, category, product_type, unit]):
            return {"error": "Отсутствуют обязательные поля (name, price, category, type, unit)"}, 400

        with get_db_connection() as db:
            db.execute(
                'INSERT INTO products (name, price, category, type, unit, volume) VALUES (?, ?, ?, ?, ?, ?)',
                (name, price, category, product_type, unit, volume)
            )
            db.commit()

        return {"message": "Продукт успешно создан"}, 201

```

Рисунок 9 – Код класса для получения всех данных товаров

```

class ProductSearch(Resource):
    """Поиск товаров по названию и фильтрам"""

    def get(self):
        """Метод поиска товаров по части названия и фильтрам"""
        name = request.args.get('name', '')
        category = request.args.get('category', None)

        query = "SELECT * FROM products WHERE name LIKE ?"
        params = [f"%{name}%"]

        if category:
            query += " AND category = ?"
            params.append(category)

        with get_db_connection() as db:
            products = db.execute(query, params).fetchall()
            return jsonify([dict(row) for row in products])

```

Рисунок 10 – Код класса для получения данных товаров с фильтрацией

```

class SingleProduct(Resource):
    """Работа с отдельным товаром по ID"""

    def get(self, product_id):
        """Получение информации о товаре по ID"""
        with get_db_connection() as db:
            product = db.execute('SELECT * FROM products WHERE id = ?', (product_id,)).fetchone()
            if product:
                return jsonify(dict(product))
            return {"error": "Продукт не найден"}, 404

    def put(self, product_id):
        """Обновление информации о товаре"""
        data = request.get_json()
        name = data.get('name')
        price = data.get('price')
        category = data.get('category')
        product_type = data.get('type')
        unit = data.get('unit')
        volume = data.get('volume')

        if not all([name, price, category, product_type, unit]):
            return {"error": "Отсутствуют обязательные поля (name, price, category, type, unit)"}, 400

        with get_db_connection() as db:
            updated = db.execute('''
                UPDATE products
                SET name = ?, price = ?, category = ?, type = ?, unit = ?, volume = ?
                WHERE id = ?
            ''', (name, price, category, product_type, unit, volume, product_id)).rowcount
            db.commit()

        if updated:
            return {"message": "Продукт успешно обновлён"}, 200
        return {"error": "Продукт не найден"}, 404

    def delete(self, product_id):
        """Удаление товара по ID"""
        with get_db_connection() as db:
            deleted = db.execute('DELETE FROM products WHERE id = ?', (product_id,)).rowcount
            db.commit()

        if deleted:
            return {"message": "Продукт успешно удалён"}, 200
        return {"error": "Продукт не найден"}, 404

```

Рисунок 11 – Код класса для получения данных конкретного товара по его id

```

# Регистрация маршрутов API
api.add_resource(Product, "/api/Product")
api.add_resource(SingleProduct, "/api/SingleProduct/<int:product_id>")
api.add_resource(ProductSearch, "/api/ProductSearch")

```

Рисунок 11 – Код определения маршрутов эндпоинтов

Данный код позволяет перейти по ссылке и добавив в маршрут /api/ и соответствующий эндпоинт мы получим файл в формате JSON/

16) Реализовать основные функциональные модули и бизнес-логику.

Необходимо реализовать основную бизнес-логику приложения. Основным функционалом приложения для организации списка покупок является список товаров и место, где будут находиться выбранные товары.

`addProductToCart()` – добавление товара в список товаров, которые выбрал пользователь.

`bind(SelectedProductDetail item)` – переключение видимости карточки товара в 2 состояния: куплен ли он или доступен.

`showEditDialog()` – метод вызывающий диалог выбора из 2 команд, удаления или обновления количества в элементе списка выбранных товаров.

`DeleteSelectedProductTask(SelectedProductDetail item, int position)` – метод удаляющий ту запись для которой было вызвано диалоговое окно.

`UpdateSelectedProductTask(SelectedProductDetail item)` – метод позволяющий изменить количество которое выбрал пользователь для записи, для которой было вызвано диалоговое окно.

17) Настроить взаимодействие с базой данных.

Необходимые базы данных:

SQLite – `shopping.db` – хранение списка товаров на стороне сервера

Room – “`app_database`” – хранение списков для списка товаров, списка товаров которые выбрал пользователь, а также история выбранных и купленных товаров, эти 3 таблицы описаны в файле ниже:

Файл `main/db/AppDatabase.java`:

```
package com.example.shoppingapp.db;

import android.content.Context;
import androidx.room.Database;
import androidx.room.Room;
import androidx.room.RoomDatabase;
import com.example.shoppingapp.db.dao.ProductDao;
import com.example.shoppingapp.db.dao.SelectedProductDao;
```

```

import com.example.shoppingapp.db.dao.ProductHistoryDao;
import com.example.shoppingapp.db.models.Product;
import com.example.shoppingapp.db.models.SelectedProduct;
import com.example.shoppingapp.db.models.ProductHistory;

@Database(entities = {Product.class, SelectedProduct.class, ProductHistory.class}, version = 3)
public abstract class AppDatabase extends RoomDatabase {
    private static AppDatabase instance;

    public abstract ProductDao productDao();
    public abstract SelectedProductDao selectedProductDao();
    public abstract ProductHistoryDao productHistoryDao();

    public static synchronized AppDatabase getInstance(Context context) {
        if(instance == null) {
            instance = Room.databaseBuilder(context.getApplicationContext(),
                AppDatabase.class, "app_database")
                .fallbackToDestructiveMigration()
                .build();
        }
        return instance;
    }
}

```

18) Разработать пользовательский интерфейс на основе макетов

Создадим разметку для главной активности, которая будет запущена при старте приложения, в которой определяются фон и нижняя навигационная панель.

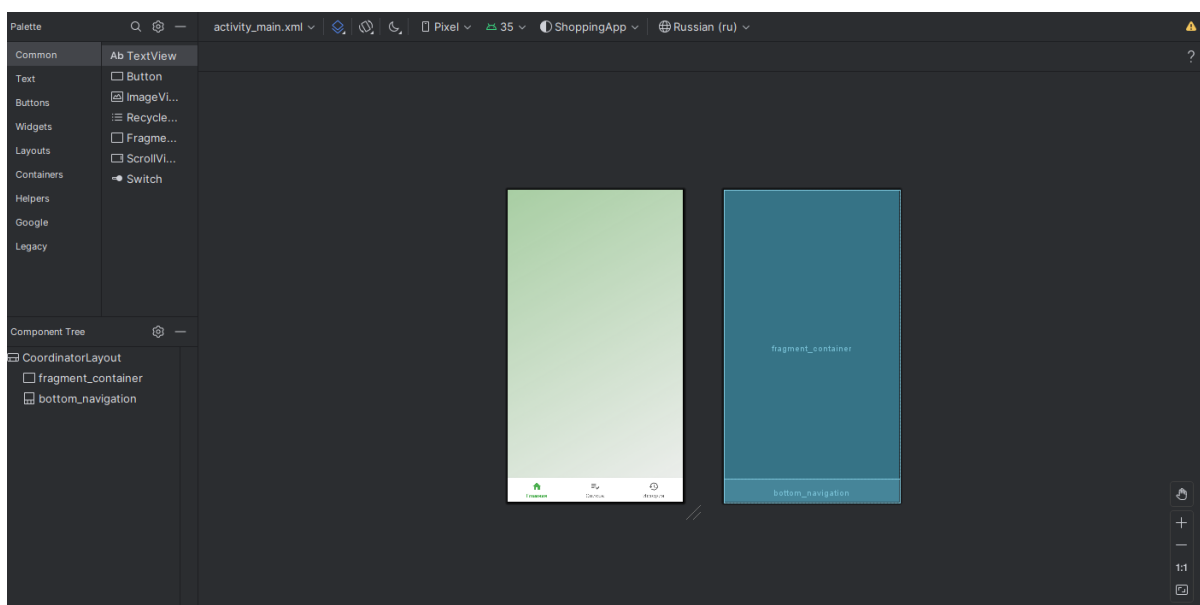


Рисунок 12 – Разметка activity_main.xml

Далее с помощью фрагментов будем выводить информацию на следующие файлы разметки:

Создадим разметку для главной страницы:

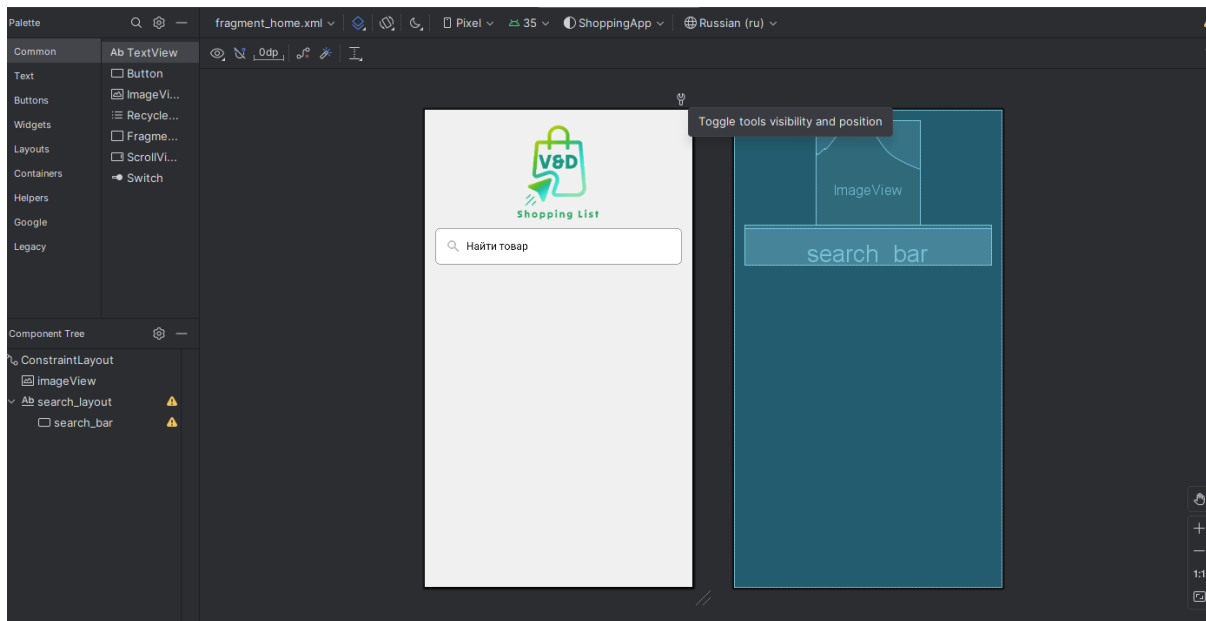


Рисунок 13 – Разметка fragment_home.xml

Создадим разметку для страницы со списком товаров:

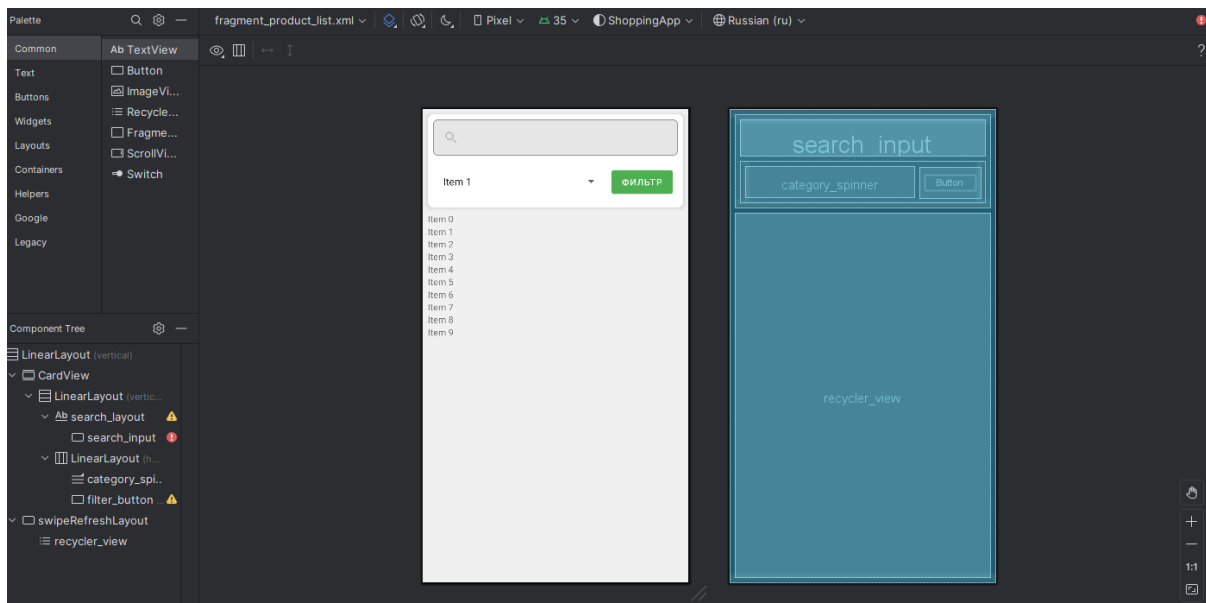


Рисунок 14 – Разметка fragment_product_list.xml

Создадим разметку для страницы со списком выбранных пользователем товаров:

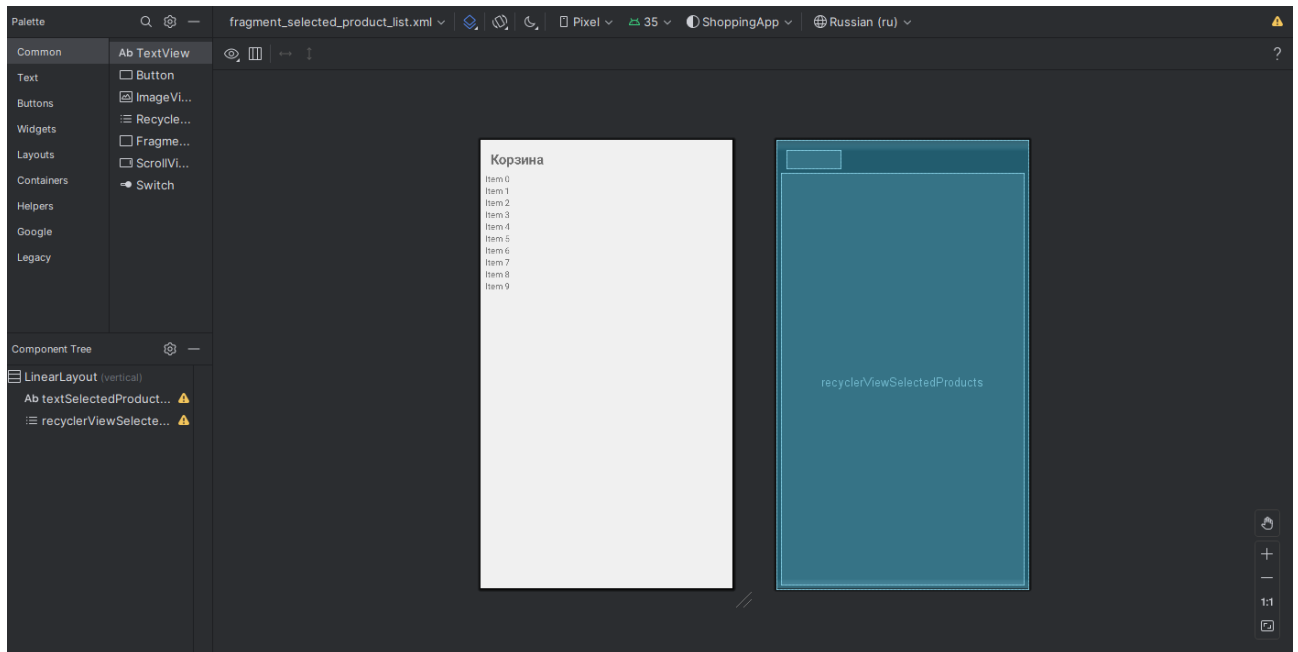


Рисунок 15 – Разметка fragment_selected_product_list.xml

Создадим разметку для страницы со списком истории добавления и покупки товаров:

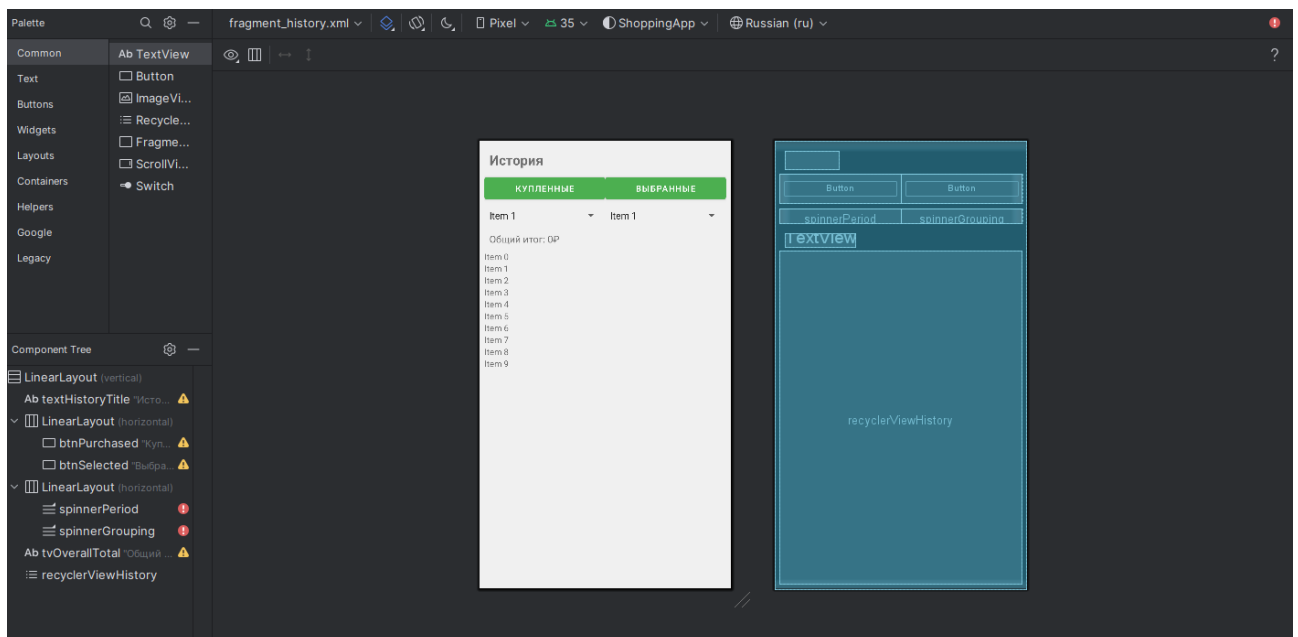


Рисунок 16 – Разметка fragment_history.xml

В каждом списке используются небольшие разметки с заголовком `item`, для отображения записей из таблицы базы данных, в представленном виде и стиле

19) Интегрировать фронтэнд с бэкэндом через API.

Ниже представлен код реализации API, который при вызове метода из этого класса обращается к эндпоинту базы данных и получает JSON файл, который преобразовывается в удобный формат для Room базы данных.

Для работы с API были добавлены следующие библиотеки:

```
implementation(libs.retrofit)
implementation(libs.converter.gson)
```

Файл `main/network/ApiService.java`:

```
package com.example.shoppingapp.network;

import com.example.shoppingapp.db.models.Product;

import java.util.List;

import retrofit2.Call;
import retrofit2.http.GET;
import retrofit2.http.Query;

public interface ApiService {
    @GET("/api/Product")
    Call<List<Product>> getAllProducts();

    @GET("/api/ProductSearch")
    Call<List<Product>> searchProducts(@Query("name") String name, @Query("category") String category);
}
```

Файл `main/network/RetrofitClient.java`:

```
package com.example.shoppingapp.network;

import retrofit2.Retrofit;
import retrofit2.converter.gson.GsonConverterFactory;

public class RetrofitClient {
    private static Retrofit retrofit;
```

```

private static final String BASE_URL = "https://fe6f-213-108-21-170.ngrok-free.app/api/";

public static Retrofit getInstance() {
    if (retrofit == null) {
        retrofit = new Retrofit.Builder()
            .baseUrl(BASE_URL)
            .addConverterFactory(GsonConverterFactory.create())
            .build();
    }
    return retrofit;
}

public static ApiService getApiService() {
    return getInstance().create(ApiService.class);
}
}

```

20) Настроить аутентификацию и авторизацию пользователей.

Так данные в приложении “Список покупок” не являются конфиденциальными, следовательно их защита не требуется. В данном приложении было решено убрать аутентификацию и авторизацию. Различия между пользователями состоит только в том, что они имеют разные устройства использования приложения, а следовательно разные списки и историю покупок продуктов.

21) Разберём эти случаи и предусмотрим защиту от них:

- 1) Загрузка вирусного кода через текстовое поле ввода. Поля доступные для пользователя имеют специальный формат, не позволяющий вводить не санкционированные значения;
- 2) Декомпиляция программы для поиска возможных дыр в безопасности или фишинга данных. Данная атака невозможна, так как Android studio тщательно скрывает сохранённые данные пользователя на его смартфоне.
- 3) Так как сервер непосредственно находится на ПК создателя, заполучить данные от аккаунта с сервером, невозможно. Данный метод обеспечивает безопасность данных пользователей от

раскрытия в результате получения административного доступа к серверу.

- 4) Шифрование данных реализовано на уровне движка проекта. При компиляции для смартфонов преобразует весь код в зашифрованный формат. Невозможно получить доступ к каким-либо исполняемым файлам приложения.
- 5) Была проведена атака в ручном режиме формата «SQL-инъекция», когда в поле ввода данных был загружен вредоносный SQL код.

22) Настроить шифрование данных

В данном проекте шифрование данных не требуется, однако оно реализуется в Android Studio на системном уровне с использованием встроенных механизмов безопасности, предоставляемых Android. Главным компонентом, который отвечает за это, является Android Keystore System. Этот механизм позволяет безопасно хранить криптографические ключи и выполнять операции шифрования без их экспорта из защищённого хранилища устройства.

23) Провести тестирование безопасности.

При декомпиляции проекта все файлы представлены в зашифрованном виде, что не даёт взломать код. Также база данных представлена в невозможном для чтения формате. Таким образом персональные данные пользователя невозможно украсть.

24) Провести модульное тестирование (Unit testing) всех компонентов.

Проведем проверку функции фильтрации продуктов (Рисунок 9). Для этого выберем одно из категорий и нажмем на кнопку “Фильтр”.



Рисунок 17 – Проверка функции фильтрации продуктов

Результат оправдал ожидания, продукты отсортировались по категории “Молочные продукты”

Проведем проверку поиска продукта по ключевому слову (Рисунок 10). Впишем слово “Бананы”, чтобы найти данный товар и нажмем кнопку “Фильтр”.

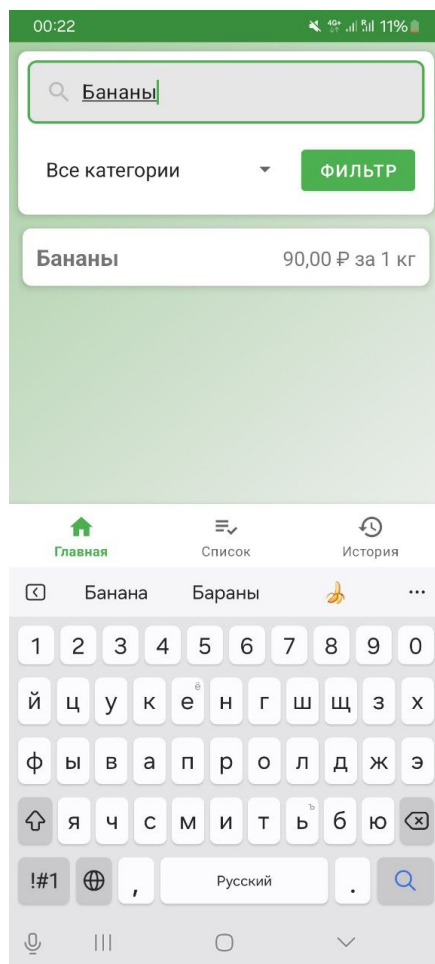


Рисунок 18 – Проверка поиска по ключевому слову

Результат оправдал ожидания, был получен ожидаемый товар по ключевому слову.

Проведем проверку функции изменения количества выбранного товара и добавления продукта в список покупок с подсчётом общей стоимости (Рисунки 11 и 12). Нажмем на товар и изменим количество, нажмем на кнопку “Добавить”.

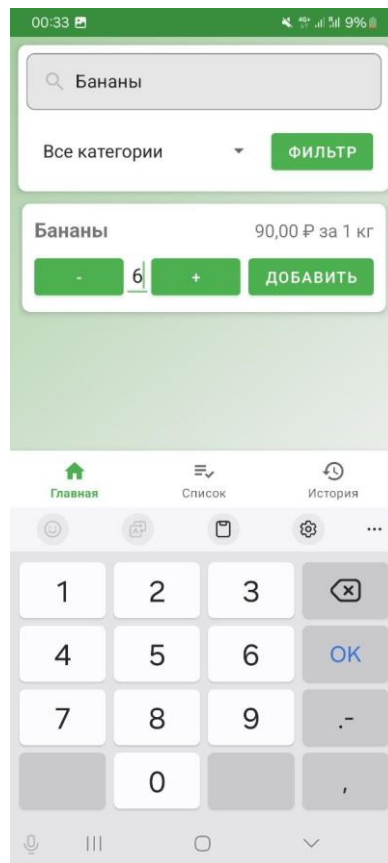


Рисунок 19 - Изменение количества товара



Рисунок 20 – Список покупок после выбора нужного товара

Результат оправдал ожидания, количество товара получилось изменить и добавить в список. Сумма продукта была посчитана верно.

Проведем проверку удаления продуктов из списка. Для этого удержим палец на одном из товаров и выберем “Удалить” (Рисунки 13, 14, 15).

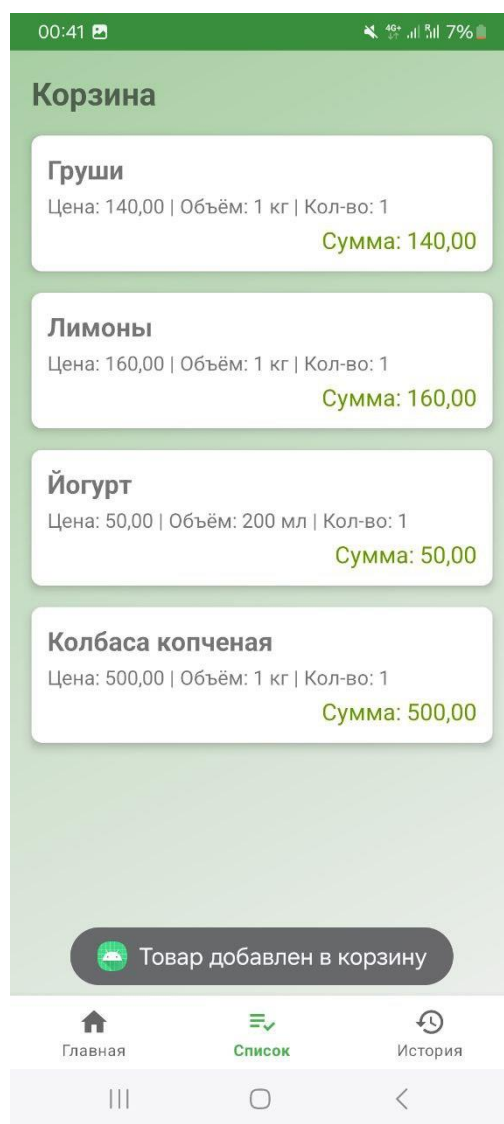


Рисунок 21 – Товары до удаления

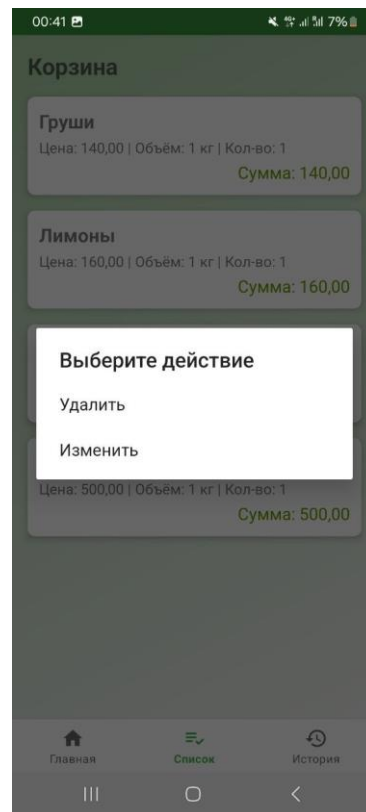


Рисунок 22 – Процесс удаления товара



Рисунок 23 – Список после удаления

Результат оправдал ожидания, товар “Колбаса копченая” был удален из списка.

Проведем проверку отметки товаров как купленных и достоверность занесения их в историю. Для этого часть товаров отметим, как купленные, а часть останутся как просто выбранные в списке. (Рисунки 16, 17, 18).

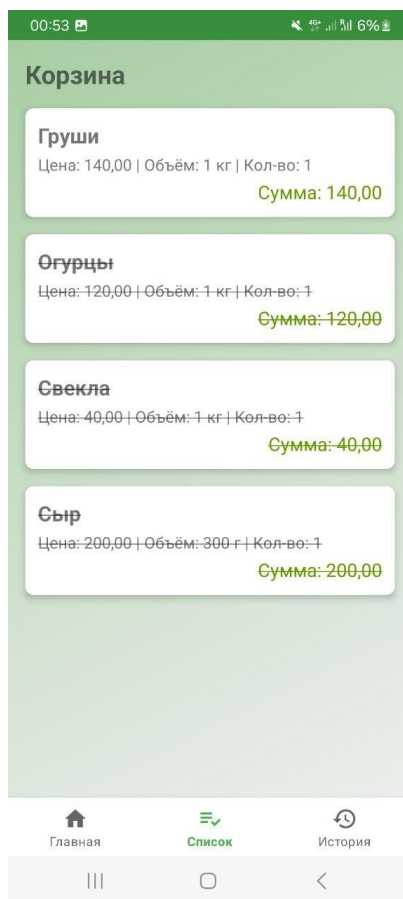


Рисунок 24 - Товары в списке



Рисунок 25- Отображение купленных товаров

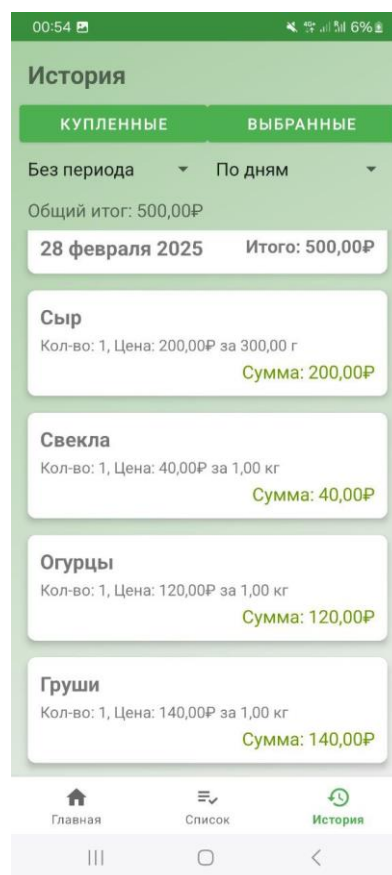


Рисунок 26 – Отображение всех выбранных в списке товаров

Результат оправдал ожидания. Все товары были правильно распределены на выбранные и купленные а также добавлены в историю покупок. Подсчет общей суммы также был произведен правильно

25) Провести нагрузочное тестирование (Load testing) для оценки производительности.

Для нагрузочного тестирования используем приложение JMeter от Apache, данная утилита предназначена для отладки и тестирования нагрузки на веб-страницы, а также для апи. В данном тесте симулировалась нагрузка на API когда к ней обращается одновременно 1000 пользователей по 5 раз.

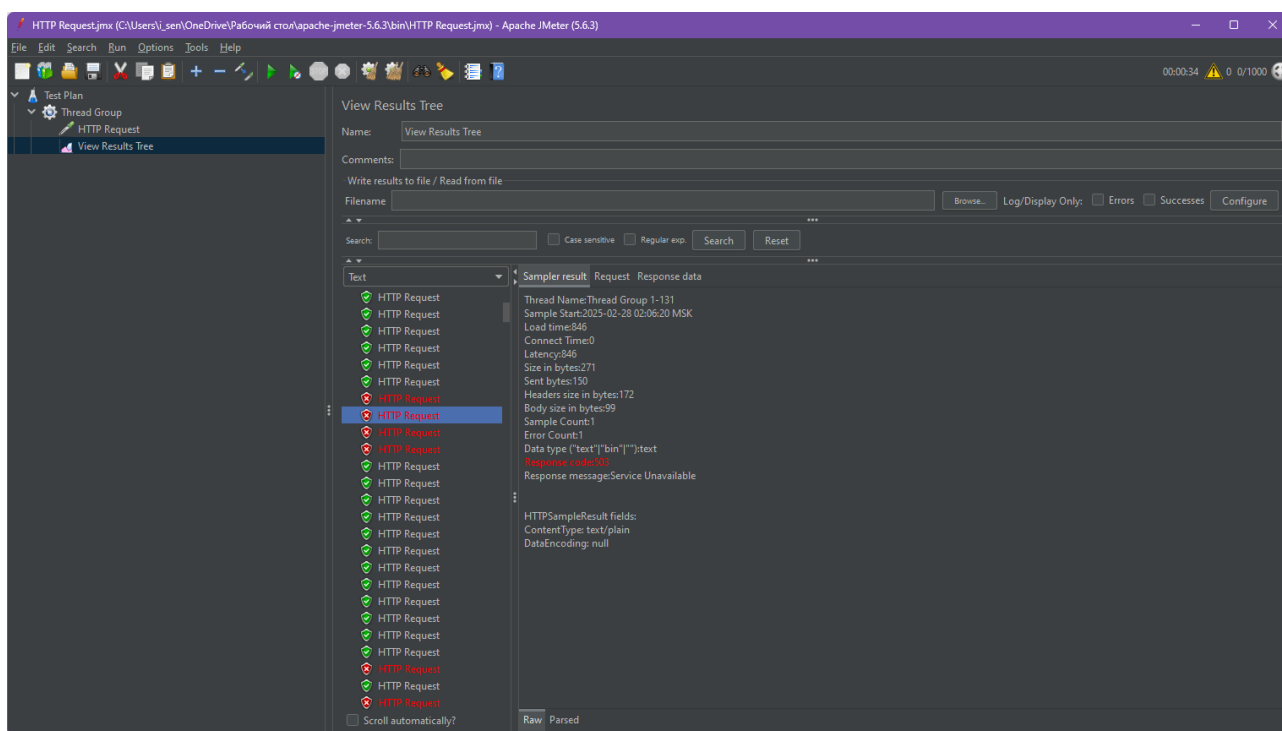


Рисунок 27 – Результаты тестирования сайта

Как можно заметить время от времени возникала ошибка 503 что означает что нагрузка на сервер слишком высокая и он не может ответить на запрос. При проведении теста 800 пользователей без проблем получили ответ на запрос, а дальше сайт начал не успевать обрабатывать запросы и время ответа значительно возросло.

26) Развернуть приложение, настроить доменное имя и SSL-сертификат.

Для разворачивания приложения требуется заhostить сервер самому либо через специальные сервисы. В данном случае создаётся локальный сервер в консоли с помощью команды `python main.py`, это команда запускает одноимённый файл в текущей папке.

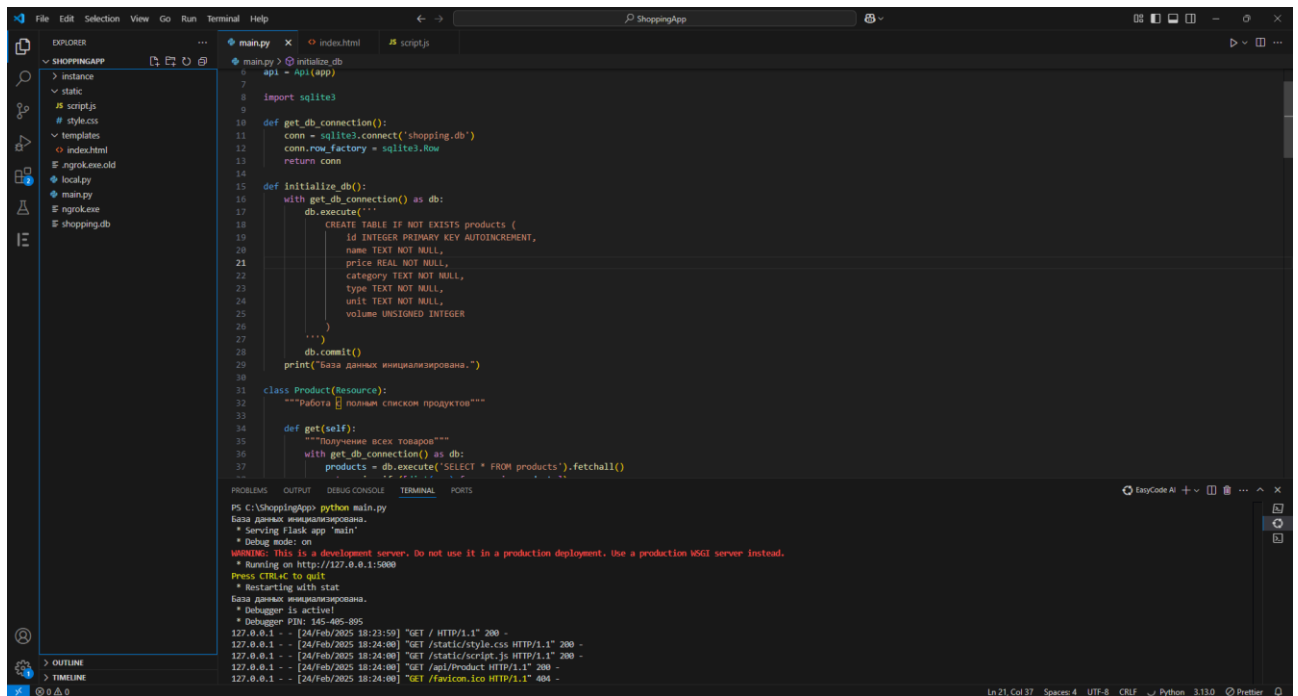
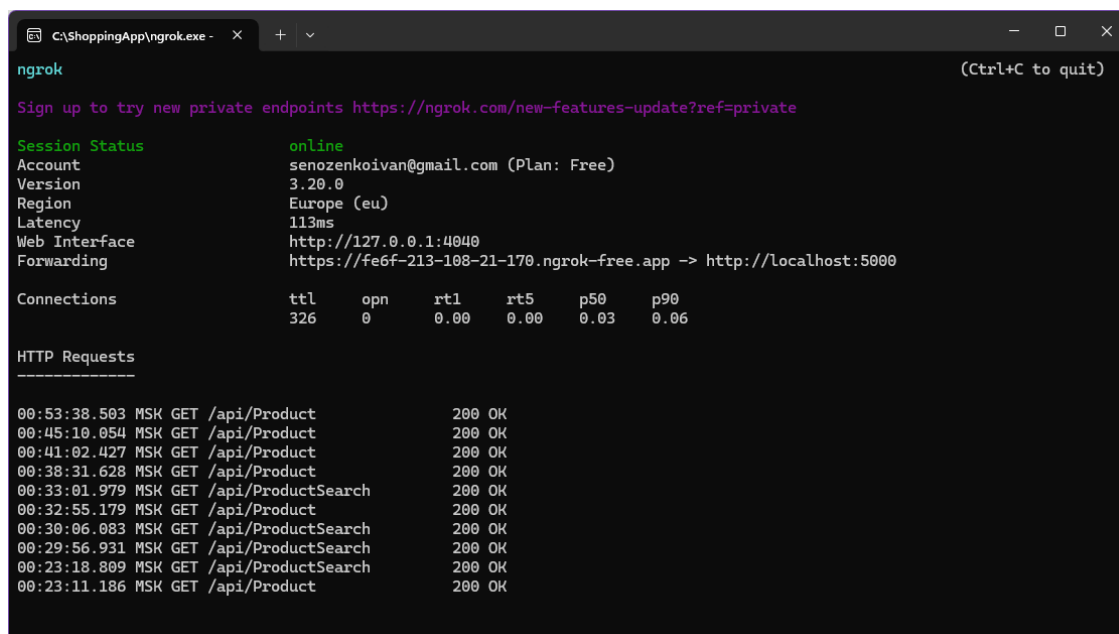


Рисунок 28 – Окно приложения Visual Studio code и запуск локального сервера

Далее необходимо открыть доступ локального сервера в сеть интернет, для этого одним из решений будет использование сервиса ngrok, который создаёт защищённое https соединение с SSL-сертификатом для указанного порта. Для этого используется команда `ngrok http 5000`, где 5000 – номер порта, на котором запущен локальный сервер.



```
C:\ShoppingApp\ngrok.exe - x + v - □ x
ngrok (Ctrl+C to quit)
Sign up to try new private endpoints https://ngrok.com/new-features-update?ref=private

Session Status      online
Account             senozenkoivan@gmail.com (Plan: Free)
Version             3.20.0
Region              Europe (eu)
Latency              113ms
Web Interface       http://127.0.0.1:4040
Forwarding           https://fe6f-213-108-21-170.ngrok-free.app -> http://localhost:5000

Connections          ttl    opn    rt1    rt5    p50    p90
                   326    0      0.00   0.00   0.03   0.06

HTTP Requests
-----
00:53:38.503 MSK GET /api/Product          200 OK
00:45:10.054 MSK GET /api/Product          200 OK
00:41:02.427 MSK GET /api/Product          200 OK
00:38:31.628 MSK GET /api/Product          200 OK
00:33:01.979 MSK GET /api/ProductSearch    200 OK
00:32:55.179 MSK GET /api/Product          200 OK
00:30:06.083 MSK GET /api/ProductSearch    200 OK
00:29:56.931 MSK GET /api/ProductSearch    200 OK
00:23:18.809 MSK GET /api/ProductSearch    200 OK
00:23:11.186 MSK GET /api/Product          200 OK
```

Рисунок 29 – Консоль сервиса ngrok сервера

27) Настроить мониторинг работы приложения.

Для мониторинга работы приложения можно использовать инструменты из прошлого пункта, поскольку они предоставляют логи обращений к сайту, а также в консоли ngrok можно увидеть задержку ответа от сайта, его различные подключения и также запросы к сайту.

28) Настроить процессы обновления и резервного копирования данных.

Для обновления приложения достаточно получить новую версию из магазина или скачать арк файл более новой версии. Обновления вне скачивания последней версии не предусмотрены, так как большая часть функций выполняется на клиентской части. Резервное копирование данных не предусмотрено.

Вывод:

В ходе выполнения данной лабораторной работы были изложены основные моменты из всех прошлых лабораторных и практических работ, для понимания общей картины устройства и функционирования приложения.