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.

Содержание

Цель работы 4

Постановка задачи 4

Ход работы 4

1. Лабораторная работа 1. Определение требований и постановка задач 4

1.1. Составить список основных функциональных требований. 4

1.2. Разработать диаграмму сценариев использования. 5

1.3. Составить список нефункциональных требований. 6

1.4. Разработать спецификацию требований. 8

2. Лабораторная работа 2. Проектирование архитектуры приложения 9

2.1. Определить основные модули и компоненты приложения. 9

2.2. Разработать диаграмму архитектуры приложения. 10

2.3. Определить используемые технологии и инструменты. 10

2.4. Описать архитектуру системы с учетом будущего масштабирования. 12

3. Лабораторная работа 3. Проектирование пользовательского интерфейса 12

3.1. Разработать макеты интерфейса для основных экранов приложения.. 12

3.2. Создать интерактивный прототип. 13

3.3. Провести тестирование прототипа на целевой аудитории и собрать обратную связь. 14

4. Лабораторная работа 4. Настройка инфраструктуры и CI/CD 14

4.1. Настроить репозиторий кода. 14

4.2. Настроить среду разработки и тестирования. 15

4.3. Настроить инструменты для автоматического тестирования и развертывания. 17

5. Лабораторная работа 5. Разработка бэкенд-части приложения 18

5.1. Разработать API для работы с клиентской частью. 18

5.2. Реализовать основные функциональные модули и бизнес-логику..... 21
5.3. Настроить взаимодействие с базой данных. 21
6. Лабораторная работа 6. Разработка фронтенд-части приложения 24
6.1. Реализовать пользовательский интерфейс на основе разработанных макетов 24
6.2. Реализация функциональных модулей: 27
6.3. Интегрировать фронтэнд с бэкэндом через API. 31
7. Лабораторная работа 7. Обеспечение безопасности приложения..... 32
7.1. Настроить аутентификацию и авторизацию пользователей. 32
7.2. Реализация защиты от распространённых атак: 33
7.3. Настроить шифрование данных 33
7.4. Провести тестирование безопасности..... 34
8. Лабораторная работа 8. Тестирование, отладка и развертывание SaaS-приложения 34
8.1. Провести модульное тестирование (Unit testing) всех компонентов.... 34
8.2. Провести интеграционное тестирование (Integration testing) для проверки взаимодействия компонентов 36
8.3. Провести нагрузочное тестирование (Load testing) для оценки производительности..... 42
8.4. Провести тестирование пользовательского интерфейса (UI testing).... 43
8.5. Развернуть приложение, настроить доменное имя и SSL-сертификат. 50
8.6. Настроить мониторинг работы приложения..... 51
8.7. Настроить процессы обновления и резервного копирования данных. 51
Вывод..... 52
Приложение А 53

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 7

Ход работы

1. Лабораторная работа 1. Определение требований и постановка задач

1.1. 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.

1.2. Разработать диаграмму сценариев использования.

Сценарий использования приложения To Do List показан на рисунке 2.

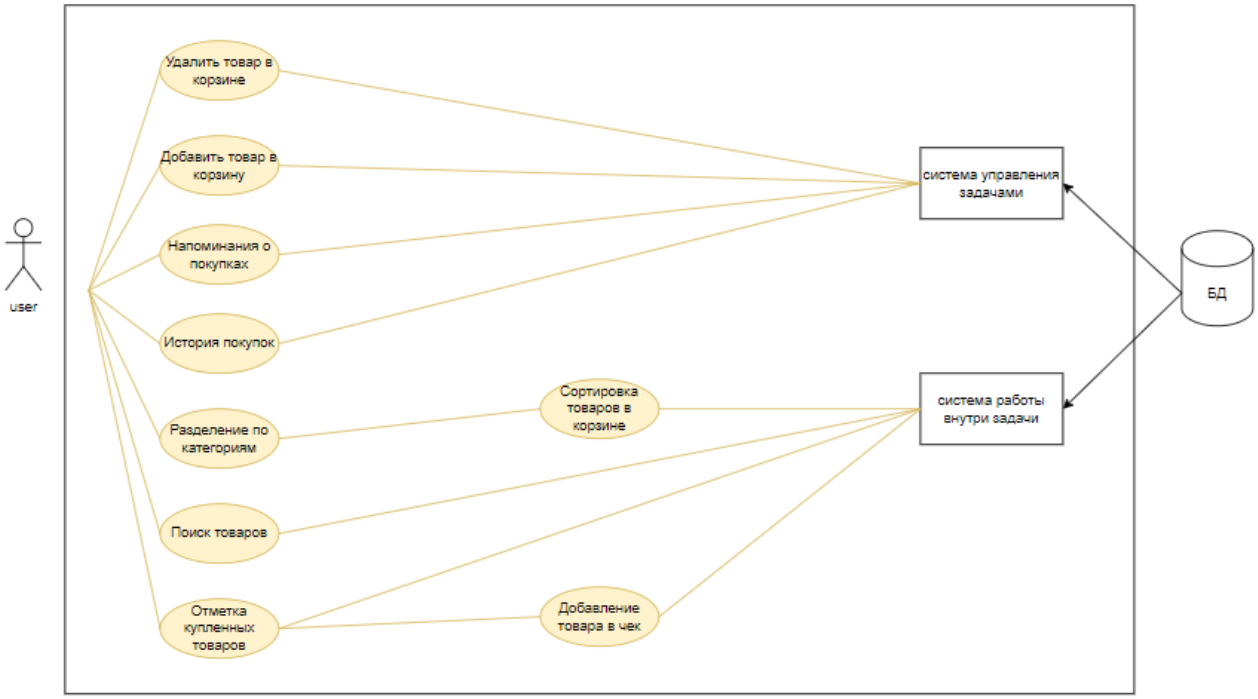

Рисунок 2 - Диаграмма сценариев использования

1.3. 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.

1.4. 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.

2. Лабораторная работа 2. Проектирование архитектуры приложения

2.1. Определить основные модули и компоненты приложения.

Существуют такие функциональные модули:

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

Определим не функциональные 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.

2.2. Разработать диаграмму архитектуры приложения.

Диаграмма архитектуры показана на рисунке 3.

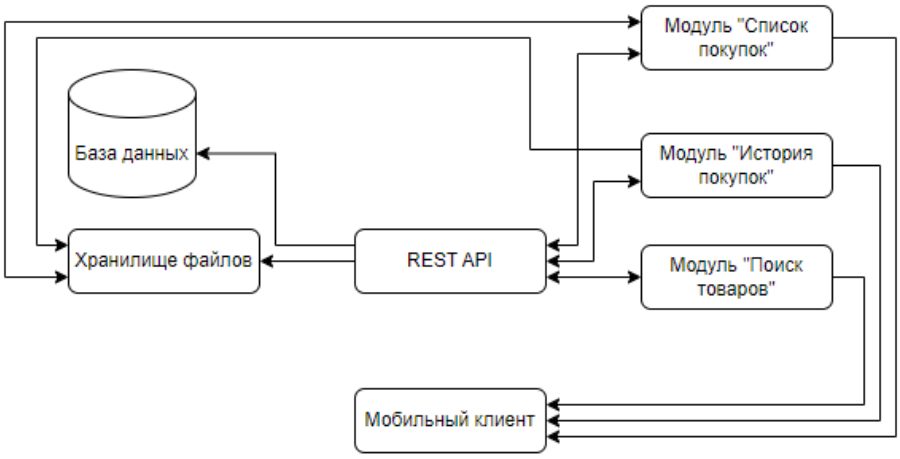

Рисунок 3 – Диаграмма архитектуры

2.3. 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:

2.4. 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 сбоев отдельных компонентов.

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

3. Лабораторная работа 3. Проектирование пользовательского интерфейса

3.1. 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.

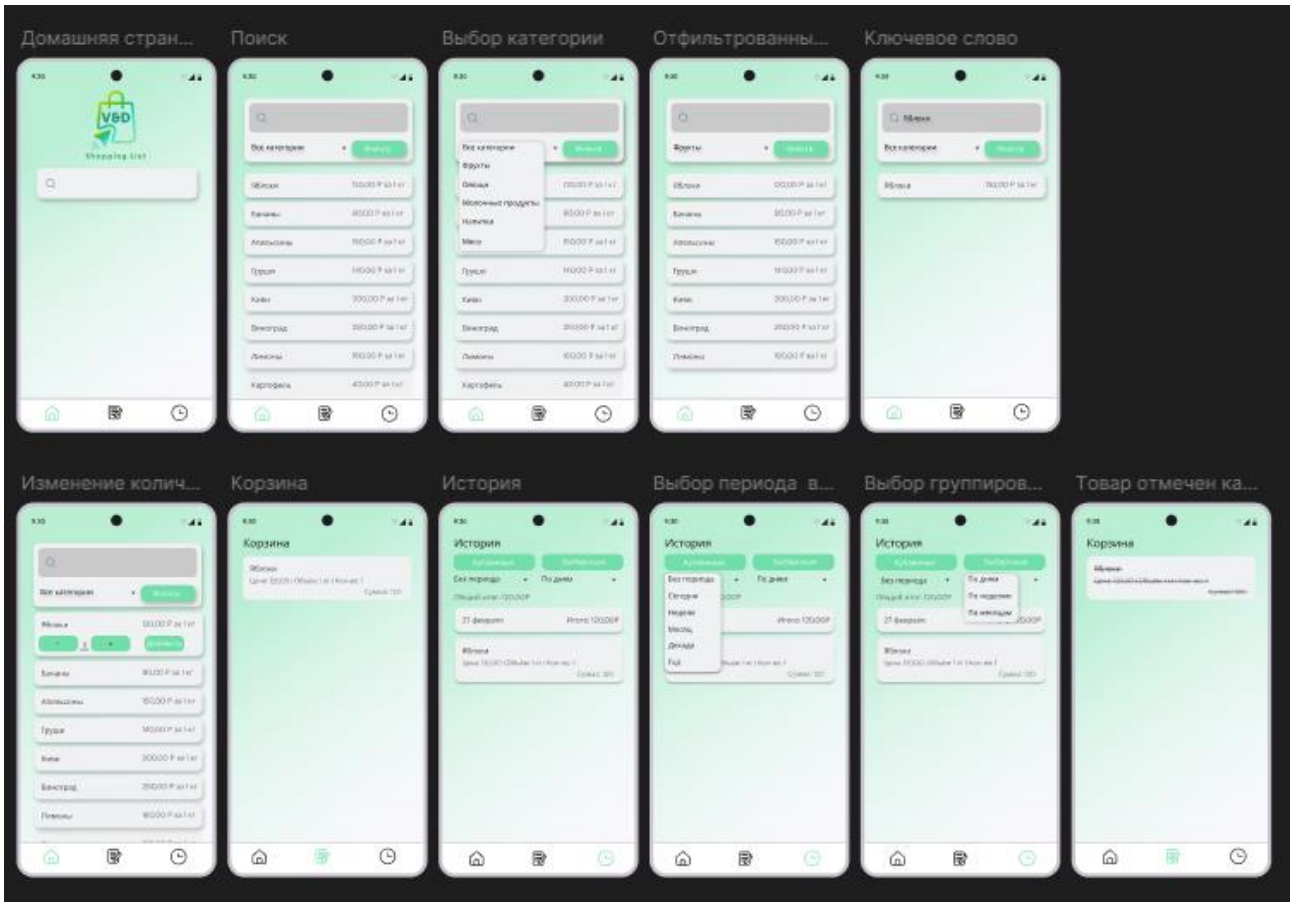

Рисунок 4 – Полный дизайн всего приложения

3.2. Создать интерактивный прототип.

Интерактивным прототипом можно воспользоваться по ссылке:

https://www.figma.com/design/6I0GcwOI24wIKgSIlEMncl/Untitled?node-id=0-1&t=bGN22Tp55nsYI0Q0-1

3.3. 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.

4. Лабораторная работа 4. Настройка инфраструктуры и CI/CD

4.1. 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.

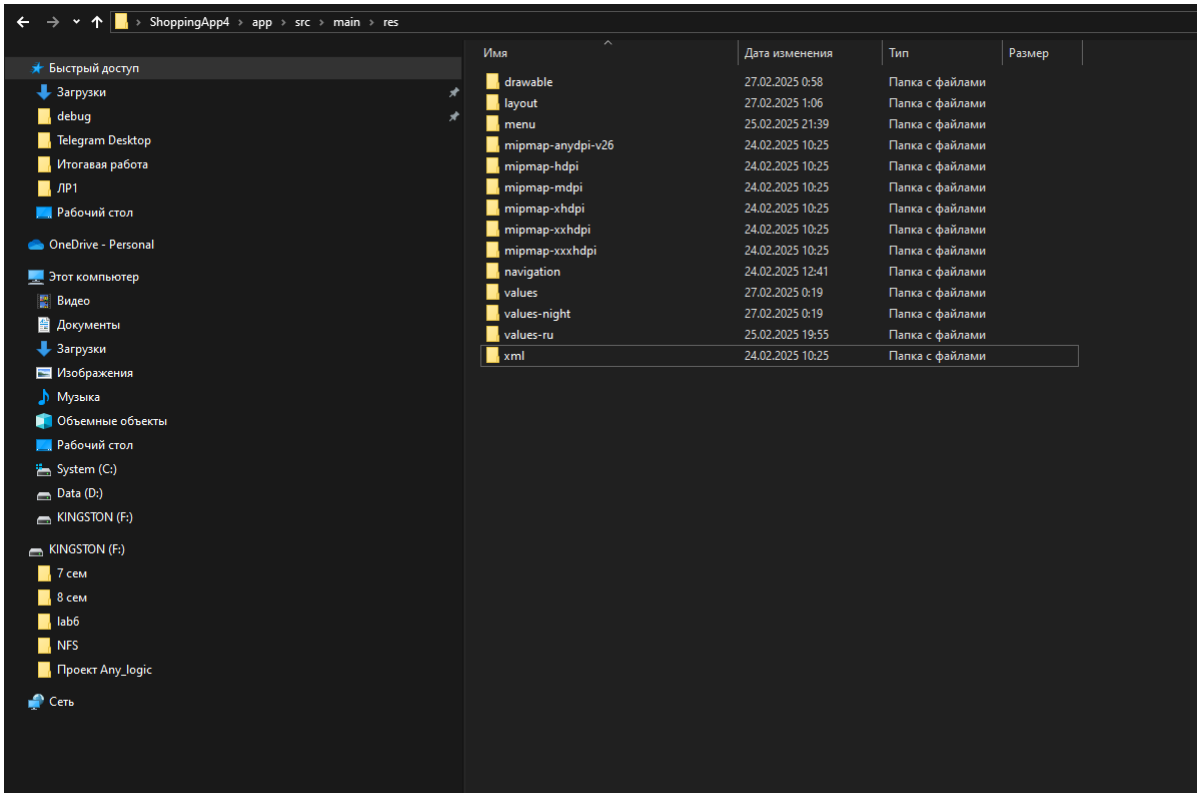

Рисунок 5 – Репозиторий кода

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 файла проекта через интернет не существует

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

4.2. 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).

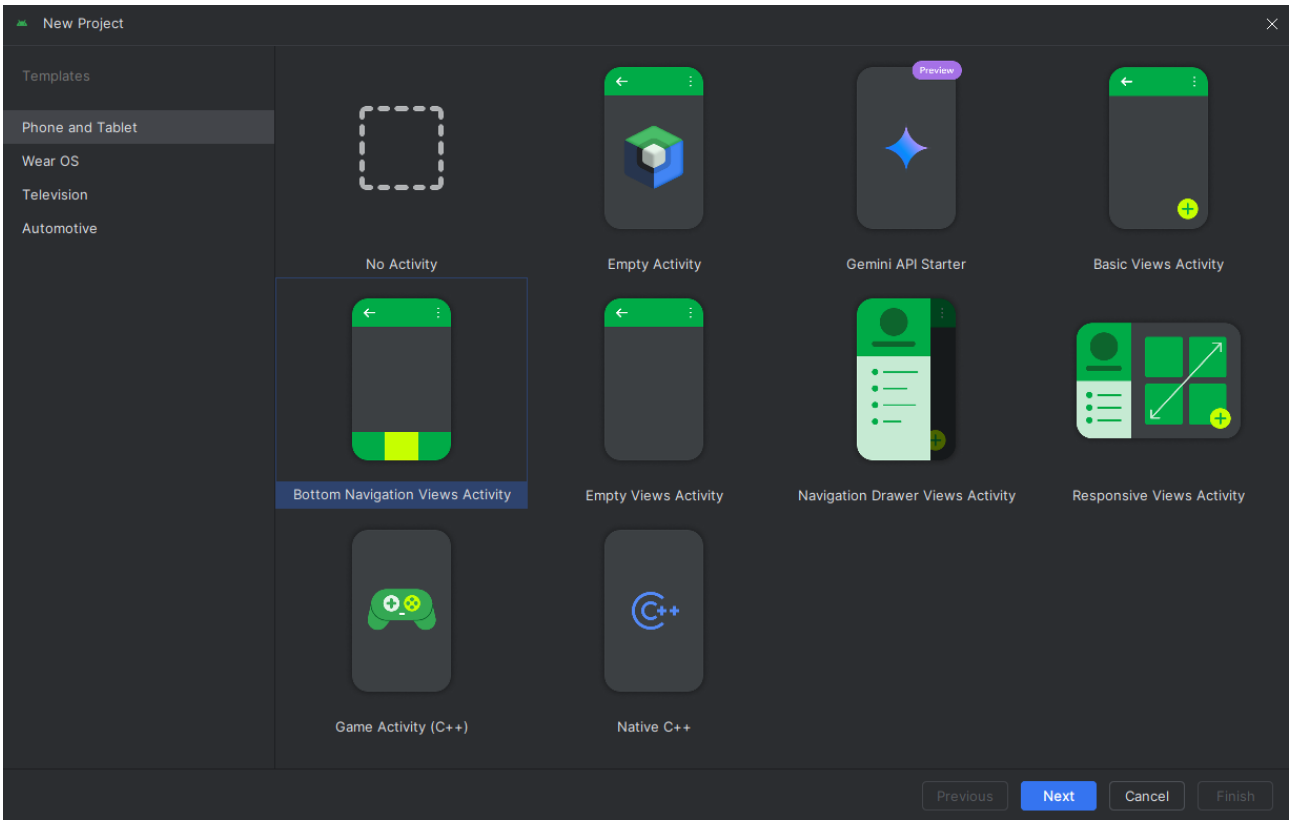

Рисунок 6 – Выбор пресета для приложения.

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) на 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.

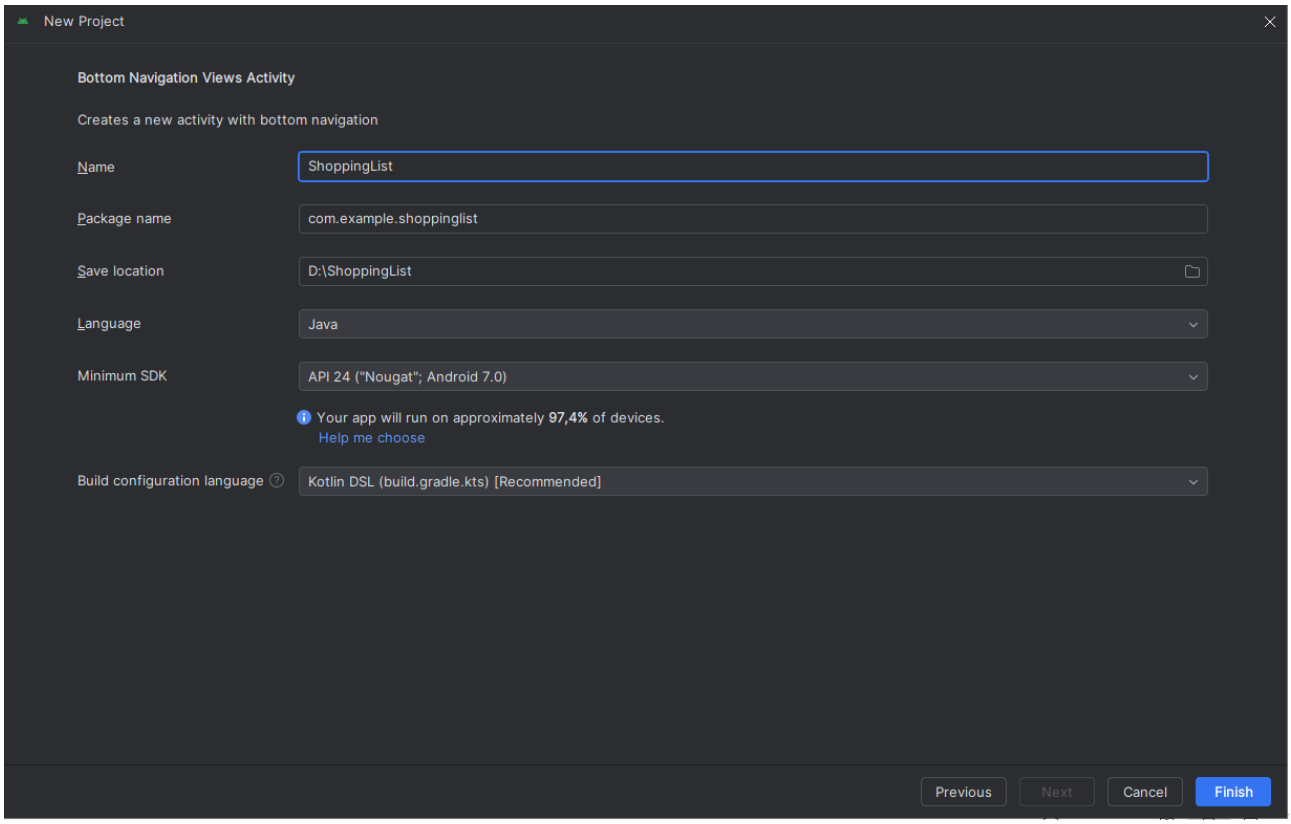

Рисунок 7 - Дополнительные настройки

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.

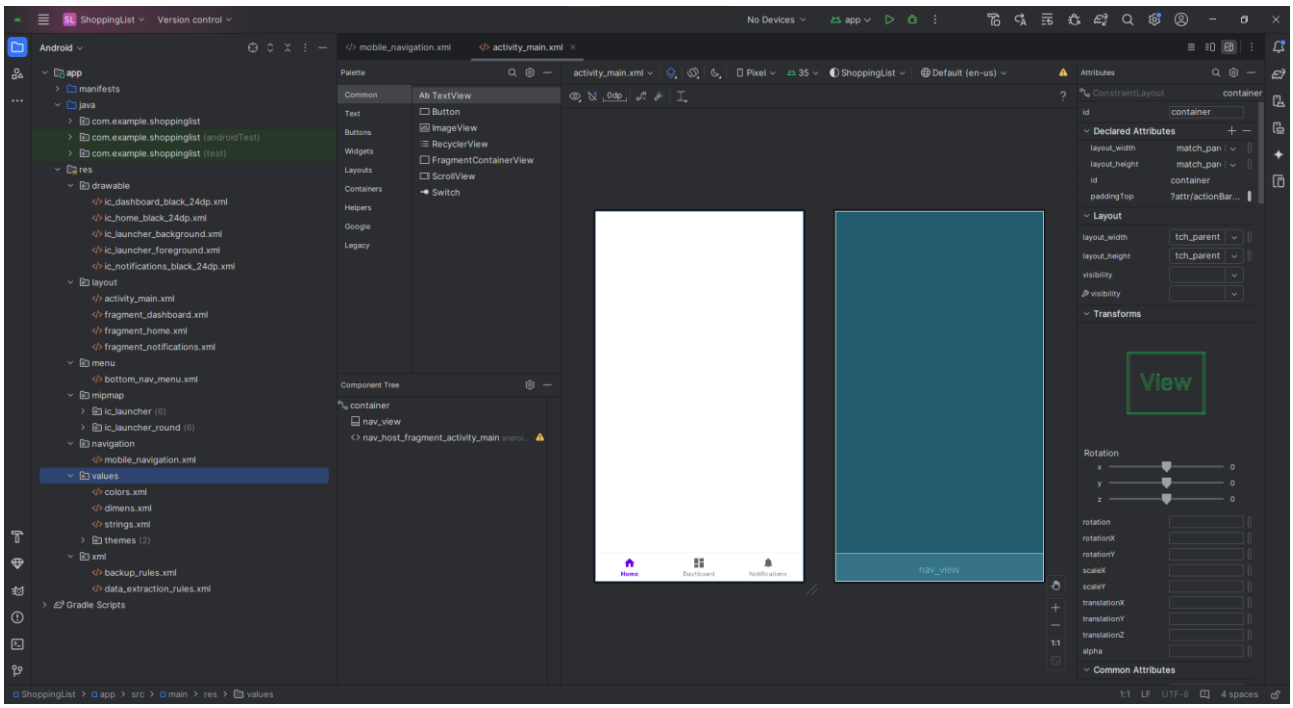

Рисунок 8 - Готовый для работы проект изображен

4.3. Настроить инструменты для автоматического тестирования и развертывания.

Для автоматического тестирования приложения можно воспользоваться сайтами, которы специализируются на данной таматике. Есть возможность подобрать необходимые настройки и протестировать конкретыне части и элементы приложения. Одним из подобных сайтов является lambdatest.com, котрый изображен на рисунке 9.

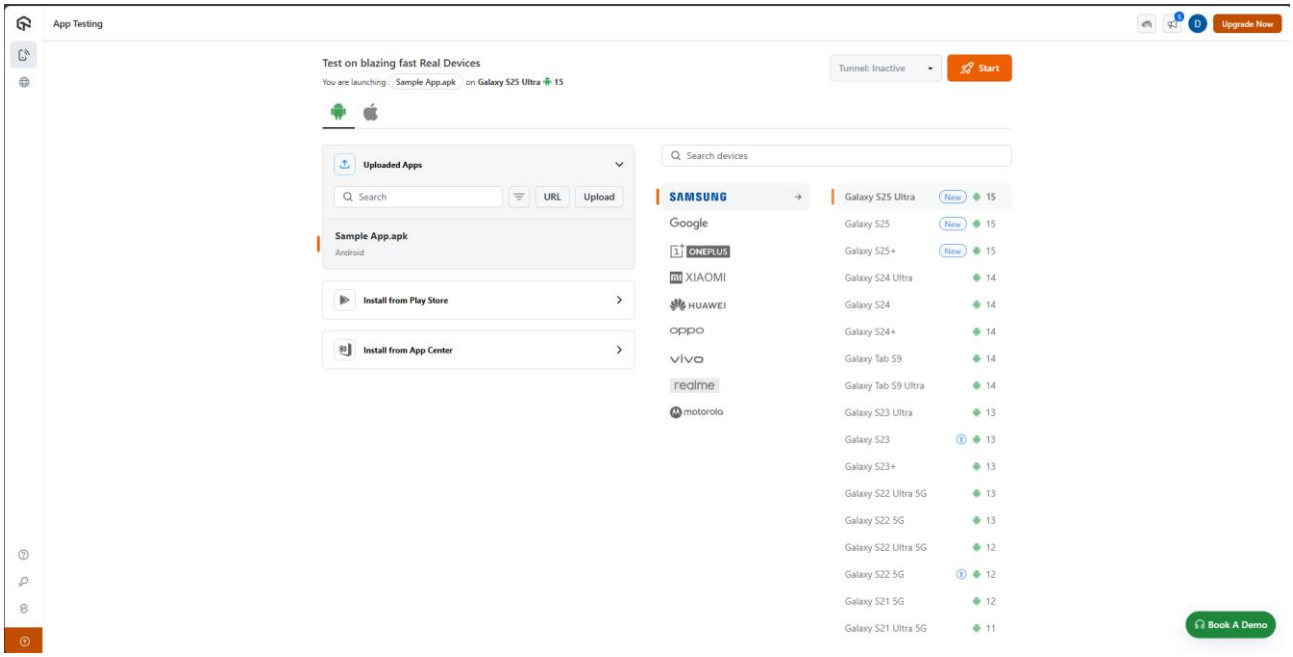

Рисунок 9 - Сайт для автоматического тестирования мобильного приложения

5. Лабораторная работа 5. Разработка бэкенд-части приложения

5.1. 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:

```python
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 10 – Код класса для получения всех данных товаров

```python
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 11 – Код класса для получения данных товаров с фильтрацией

```python
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 12 – Код класса для получения данных конкретного товара по его id

```python
# 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 13 – Код определения маршрутов эндпоинтов

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

5.2. 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.

5.3. Настроить взаимодействие с базой данных.

Необходимые базы данных:

SQLite – shopping.db – хранение списка товаров на стороне сервера, описание базы данных изображено на рисунке 14.

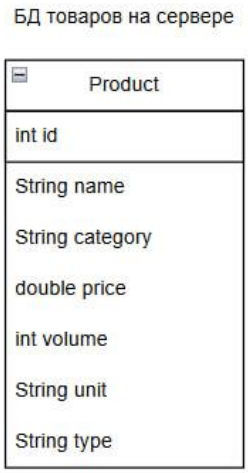

Рисунок 14 – Описание базы данных сервера

В данной таблице базы данных первичным ключом выступает поле int id – идентификатор записи в таблице базы данных, всегда уникален в пределах таблицы.

Методы подключения к базе данных и инициализации таблицы её таблицы приведены на рисунке 15.

```
def get_db_connection():
    conn = sqlite3.connect('shopping.db')
    conn.row_factory = sqlite3.Row
    return conn

def initialize_db():
    with get_db_connection() as db:
        db.execute('''
            CREATE TABLE IF NOT EXISTS products (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                name TEXT NOT NULL,
                price REAL NOT NULL,
                category TEXT NOT NULL,
                type TEXT NOT NULL,
                unit TEXT NOT NULL,
                volume UNSIGNED INTEGER
            )
        ''')
        db.commit()
    print("База данных инициализирована.")
```

Рисунок 15 – Реализации базы данных на сервере

Room – “app_database” – хранение таблиц для списка товаров, списка товаров, которые выбрал пользователь, а также история выбранных и купленных товаров. Эти 3 таблицы изображены на рисунке 16:

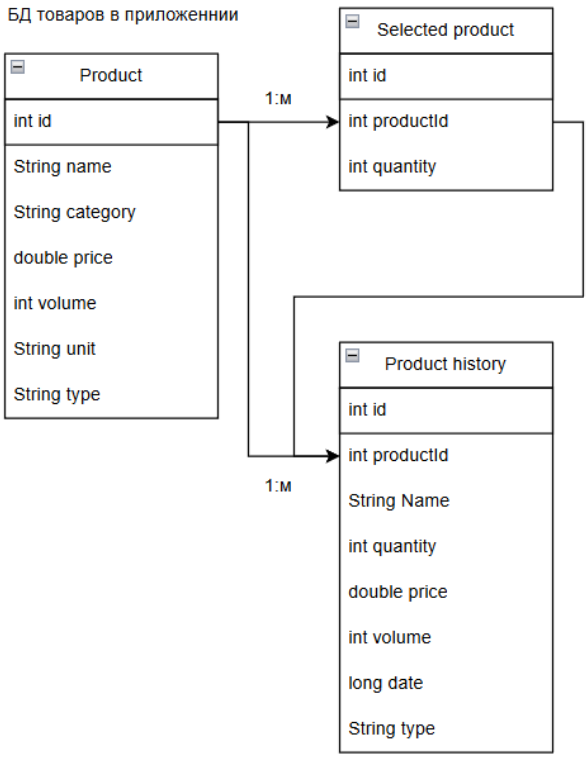

Рисунок 16 – Описание базы данных приложения

В данной базе данных структура сложнее, чем на серверной: productId служит внешним ключом для таблиц Selected product и Product history для связывания с полем id таблицы Product. Они тоже имеют первичный ключ id, уникальный для каждой из таблиц.

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;
    }
}
```

6. Лабораторная работа 6. Разработка фронтенд-части приложения

6.1. Реализовать пользовательский интерфейс на основе разработанных макетов

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

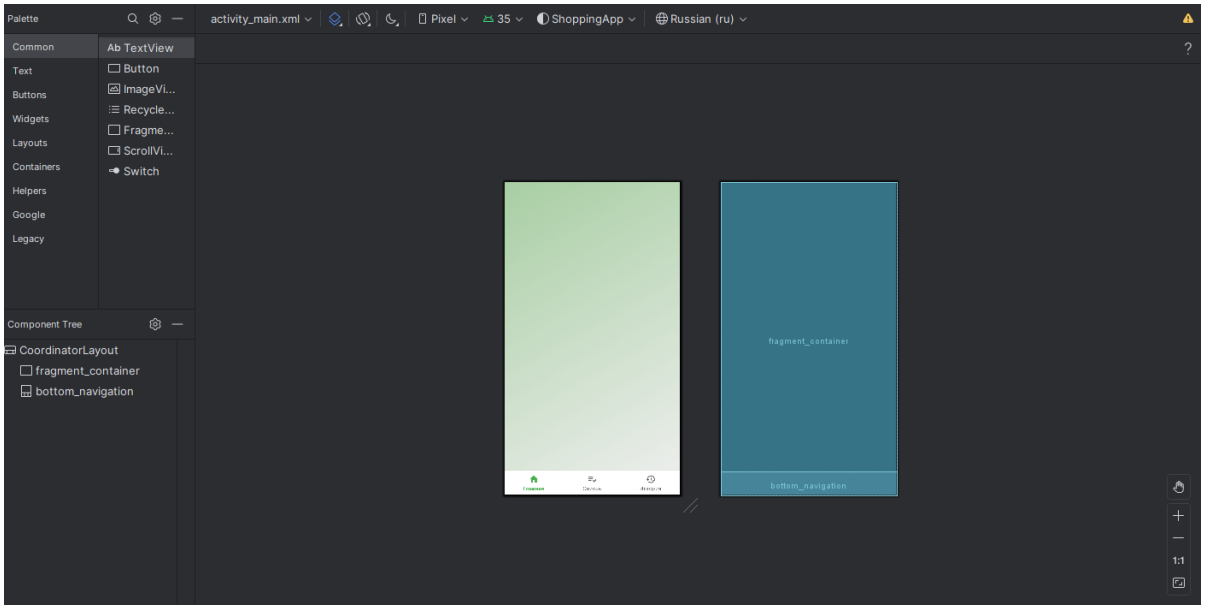

Рисунок 17 – Разметка activity_main.xml

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

Создадим разметку для главной страницы:

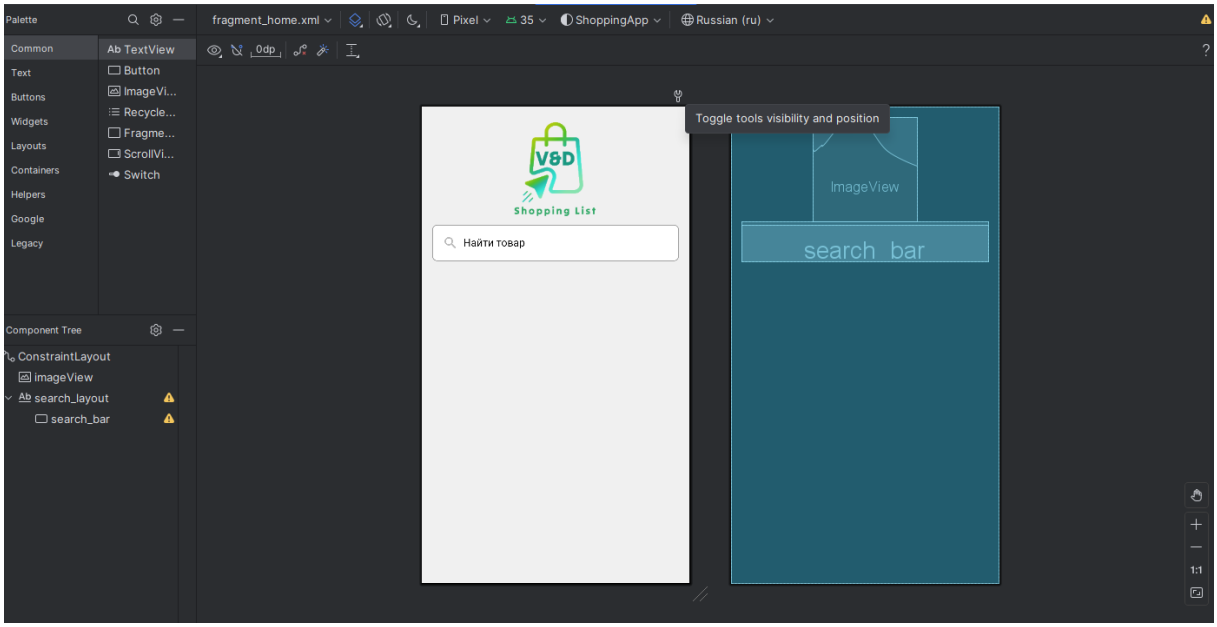

Рисунок 18 – Разметка fragment_home.xml

Создадим разметку для страницы со списком товаров:

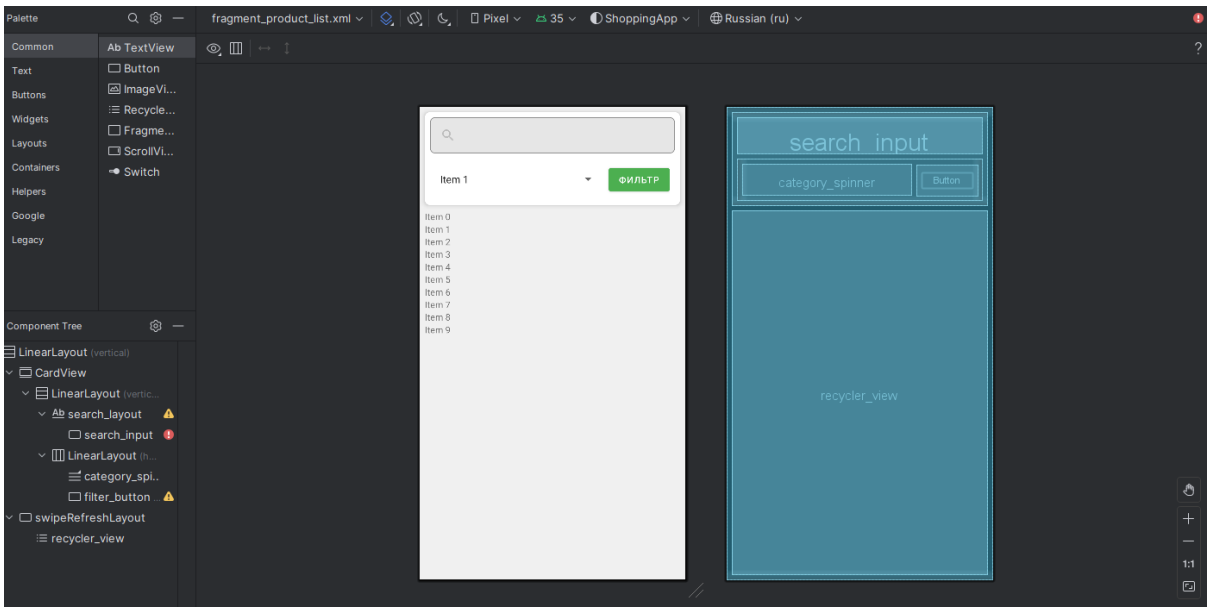

Рисунок 19 – Разметка fragment_product_list.xml

Создадим разметку для страницы со списком выбранных пользователем товаров:

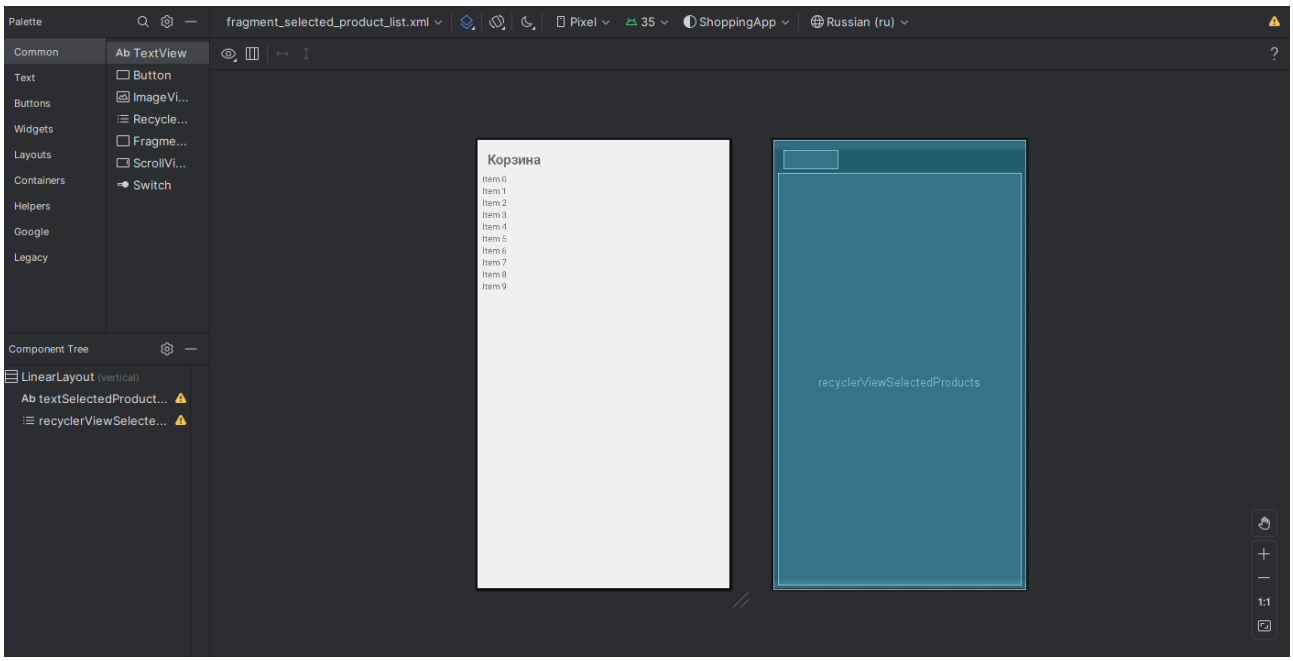

Рисунок 20 – Разметка fragment_selected_product_list.xml

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

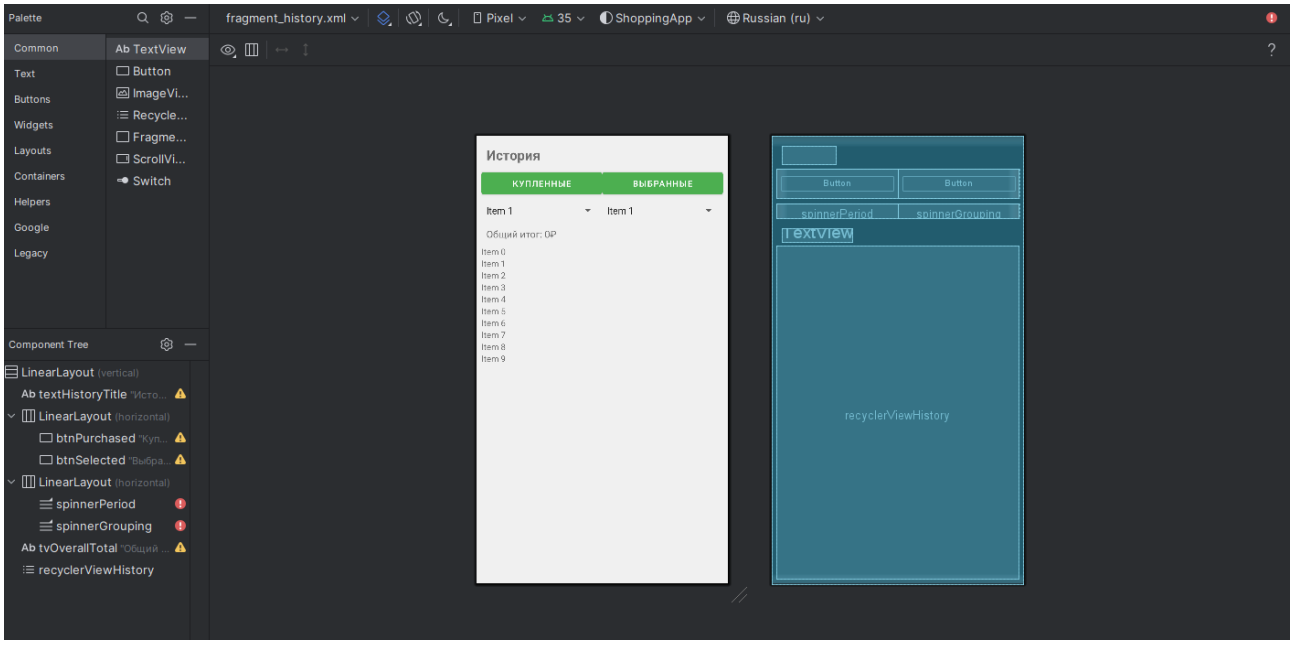

Рисунок 21 – Разметка fragment_history.xml

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

6.2. Реализация функциональных модулей:

Главный модуль приложения MainActivity.java он выполняет роль главного интерфейса приложения, при его создании также добавляется нижняя навигационная панель, необходимая для перемещения между страницами приложения. Реализация данного класса изображена на рисунке 22.

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        BottomNavigationView bottomNavigationView = findViewById(R.id.bottom_navigation);
        bottomNavigationView.setOnNavigationItemSelectedListener(item -> {
            Fragment selectedFragment = null;
            int id = item.getItemId();
            if (id == R.id.navigation_home) {
                selectedFragment = new HomeFragment();
            } else if (id == R.id.navigation_selected) {
                selectedFragment = new SelectedListFragment();
            } else if (id == R.id.navigation_history) {
                selectedFragment = new HistoryFragment();
            }
            if(selectedFragment != null) {
                getSupportFragmentManager().beginTransaction()
                        .replace(R.id.fragment_container, selectedFragment)
                        .commit();
            }
            return true;
        });

        if (savedInstanceState == null) {
            getSupportFragmentManager().beginTransaction()
                    .replace(R.id.fragment_container, new HomeFragment())
                    .commit();
        }
    }
}
```

Рисунок 22 – Класс MainActivity.java

При нажатии на кнопку нижней панели, откроется соответствующий фрагмент или же страница.

По умолчанию при старте приложения открывается главная страница, на которой находится логотип и строка поиска, при её активации пользователь переходит на страницу со списком всех товаров и может задать необходимые фильтры, чтобы найти подходящие ему товары, реализация представлена на рисунке 23.

Рисунок 23 – Основные методы ProductListFragment.java

Метод performSearch позволяет получить информацию и обработать её для последующего вызова метода получения товаров из локальной базы данных.

Метод loadProductsFromDatabase позволяет получить записи из базы данных по полученным в прошлом методе фильтрам

Метод updateUI вызывается после обновления списка товаров и далее в фоновом потоке обновляет список, выводимый адаптером. Дальше адаптер позволяет изменять содержание страницы в реальном времени, основные функции, которые предоставляет этот класс изображены на рисунке 24.

```
private void updateQuantity(int delta) {
    try {
        int qty = Integer.parseInt(quantityEdit.getText().toString());
        qty = Math.max(qty + delta, 1);
        quantityEdit.setText(String.valueOf(qty));
    } catch (NumberFormatException e) {
        quantityEdit.setText("1");
    }
}

1 usage
private void addProductToCart() {
    try {
        int qty = Integer.parseInt(quantityEdit.getText().toString());
        Product product = productList.get(getAdapterPosition());

        SelectedProductRepository.getInstance(itemView.getContext())
                .insertSelectedProduct(new SelectedProduct(product.getId(), qty));

        new Thread(() -> {
            ProductHistory history = new ProductHistory(
                    product.getName(),
                    qty,
                    product.getPrice(),
                    product.getVolume(),
                    product.getUnit(),
                    System.currentTimeMillis(),
                    1  // 1 - выбранный товар
            );
            AppDatabase.getInstance(itemView.getContext())
                    .productHistoryDao().insertHistory(history);
        }).start();

        Toast.makeText(itemView.getContext(),
                text: "Товар добавлен в корзину",
                Toast.LENGTH_SHORT).show();

    } catch (NumberFormatException e) {
        quantityEdit.setText("1");
    }
```

Рисунок 24 – Основные методы ProductAdapter.java

Метод updateQuantity позволяет изменять количество, которое пользователь хочет добавить в свой список.

Метод addProductToCart позволяет добавлять выбранное количество товара в список пользователя. Также заносит данные в историю выбранных товаров.

После этого пользователь может в нижней панели перейти на страницу его списка. На странице обновляется список пользователя, а в адаптере определена логика выполнения со списком.

```
public void bind(SelectedProductDetail item) {
    tvName.setText(item.getProductName());

    String details = String.format("Цена: %.2f | Объём: %d %s | Кол-во: %d",
            item.getProductPrice(),
            item.getProductVolume(),
            item.getProductUnit(),
            item.getQuantity());
    tvDetails.setText(details);
    tvTotal.setText(String.format("Сумма: %.2f", item.getTotalSum()));

    if (item.isPurchased()) {
        tvName.setPaintFlags(tvName.getPaintFlags() | Paint.STRIKE_THRU_TEXT_FLAG);
        tvDetails.setPaintFlags(tvDetails.getPaintFlags() | Paint.STRIKE_THRU_TEXT_FLAG);
        tvTotal.setPaintFlags(tvTotal.getPaintFlags() | Paint.STRIKE_THRU_TEXT_FLAG);
    } else {
        tvName.setPaintFlags(tvName.getPaintFlags() & (~Paint.STRIKE_THRU_TEXT_FLAG));
        tvDetails.setPaintFlags(tvDetails.getPaintFlags() & (~Paint.STRIKE_THRU_TEXT_FLAG));
        tvTotal.setPaintFlags(tvTotal.getPaintFlags() & (~Paint.STRIKE_THRU_TEXT_FLAG));
    }

    itemContainer.setOnClickListener(v -> {
        item.setPurchased(!item.isPurchased());
        new UpdateSelectedProductTask(item).execute();

        if (item.isPurchased()) {
            new Thread(() -> {
                ProductHistory history = new ProductHistory(
                        item.getProductName(),
                        item.getQuantity(),
                        item.getProductPrice(),
                        item.getProductVolume(),
                        item.getProductUnit(),
                        System.currentTimeMillis(),
                        0  // 0 - купленный товар
                );
                AppDatabase.getInstance(context)
                        .productHistoryDao().insertHistory(history);
            }).start();
        } else {
            new Thread(() -> {
                AppDatabase.getInstance(context)
                        .productHistoryDao().deleteHistoryByProductAndType(item.getProductName(), 0);
            }).start();
```

```
        }
        notifyItemChanged(getAdapterPosition());
    });

    itemContainer.setOnLongClickListener(v -> {
        showActionsDialog(item, getAdapterPosition());
        return true;
    });
}
```

Данный метод отвечает за отображение элементов списка и добавляет возможность отмечать каждую из них как купленный или отменить эту отметку. Каждый товар, отмеченный как купленный попадает в историю в соответствующий список.

6.3. 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);
    }
}
```

7. Лабораторная работа 7. Обеспечение безопасности приложения

7.1. Настроить аутентификацию и авторизацию пользователей.

Так как 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.

7.2. Реализация защиты от распространённых атак:

Приложение не хранит персональные данные пользователя, поэтому для обеспечения защиты приложения и данных, нужен минимальные требования к защите, рассмотрим несколько распространённых атак:

1) Загрузка вирусного кода через текстовое поле ввода. Пользователь приложения не имеет доступа к публикации какой-либо информации на сервер, что защищает данные и других пользователей от получения вредоносной информации;
2) 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.
3) Шифрование данных реализовано на системном уровне в Android studio. При компиляции для смартфонов преобразует весь код в зашифрованный формат, а для дешифрации используются ключи, которые надёжно хранятся с помощью утилиты Android Keystore System. Поэтому невозможно получить доступ к каким-либо исполняемым файлам приложения или хранилищу приложения.

7.3. 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.

7.4. Провести тестирование безопасности.

При декомпиляции проекта все файлы представлены в зашифрованном виде, что делает файлы приложения нечитаемыми и не позволяет редактировать код. База данных также зашифрована из-за чего она находится невозможном для чтения формате. Таким образом данные, находящиеся в базе данных, невозможно изменить или украсть.

8. Лабораторная работа 8. Тестирование, отладка и развертывание SaaS-приложения

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

Для этого отобразим, что каждый модуль работает исправно и отображает необходимую информацию:

1) Модуль "Список покупок" изображен на рисунке 25:

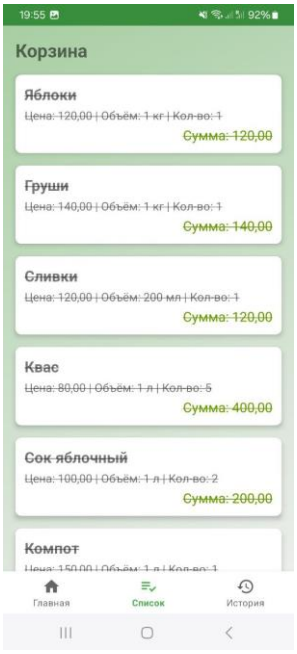

Рисунок 25 - Модуль списка покупок

Как и планировалось, модуль "Список покупок" отображает выбранные продукты из базы данных с указанием стоимость за определенны объем.

2) Модуль “Поиск товаров” изображен на рисунке 26:

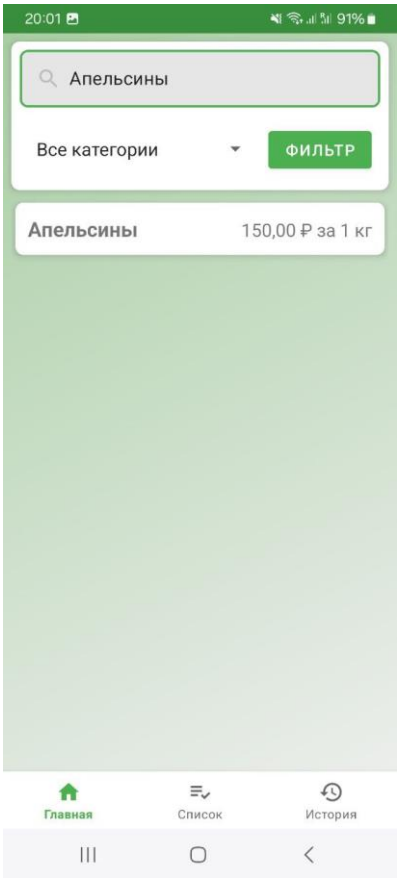

Рисунок 26 - Модуль “Поиск товаров”

Как и планировалось, модуль "Поиск товаров" выполняет поиск продукта по ключевому слову.

3) Модуль “История покупок” изображен на рисунке 27:

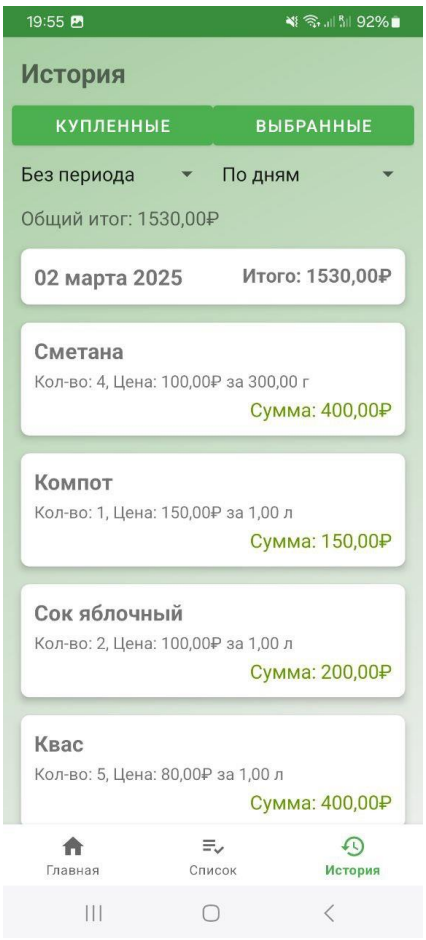

Рисунок 27 - Модуль “История покупок”

Как и планировалось, модуль "История покупок" показывает товары, которые пользователь отметил как купленные.

8.2. Провести интеграционное тестирование (Integration testing) для проверки взаимодействия компонентов

Для выполнения интеграционного тестирования необходимо проверить как разные модули взаимодействуют между собой. В приложении есть 3 основных модуля:

1) Модуль "Список покупок" позволяет пользователю добавлять товары в список, редактировать их или помечать как купленные. Этот модуль взаимодействует с базой данных для хранения всех изменений. Основная функция — поддержка актуальности списка, отображение товаров, которые ещё

не куплены, и тех, что уже находятся в корзине. Для проверки его добавим в бузу данных тестовый продукт, после чего добавим его уже в список покупок.

Для начала добавим тестовый товар (Рисунок 28):

65	Квас	Напитки	80 руб. за 1 л	Удалить
66	Пиво	Напитки	100 руб. за 500 мл	Удалить
67	Лимонад	Напитки	70 руб. за 400 мл	Удалить
68	Молочный коктейль	Напитки	120 руб. за 350 мл	Удалить
69	Компот	Напитки	150 руб. за 1 л	Удалить
71	Тест	Фрукты	999 руб. за 1 шт.	Удалить

Рисунок 28 – Добавление тестового товара в базу данных продуктов

Список продуктов до добавления тестового товара (Рисунок 29):

Рисунок 29 - Список продуктов до добавления тестового товара

Список продуктов после добавления тестового товара (Рисунок 30):

Рисунок 30 - Список продуктов после добавления тестового товара

После чего добавим данный товар в список покупок и отметим, как купленный, при этом поменяв количество (Рисунки 31 – 32)

Рисунок 31 – Добавление товара в список

Рисунок 32 - Товар добавлен в список и отмечен как купленный

Результат оправдал ожидания, товар, как и планировалось записался в нашу базу, а после обновления списка всех продуктов (свайпом вниз) внутри приложения отобразился вместе с остальным списком товаров. Далее тестовый продукт был добавлен в список покупок с изменённым количеством и отмечен как купленный. В результате товар был отображен в списке в выставленном количестве, как и планировалось.

2) Модуль “Поиск товаров” - Он позволяет искать товары по названиям. Взаимодействует с модулем "Список покупок", чтобы быстро добавлять найденные товары в список покупок. Ключевая функция — упрощение процесса добавления товаров в список, минимизируя время на поисковые операции. Для его проверки попробуем найти 2 товара. Первый, который точно есть в нашей базе данных, а второй, которого там точно нет (Рисунки 33 и 34).

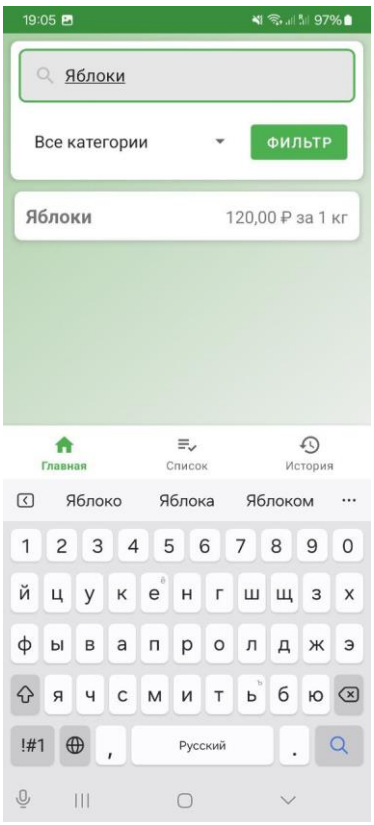

Рисунок 33 – Найденный товар из базы данных

Рисунок 34 - Не найденный товар в базе данных

В результате модуль поиска работает как планировалось, отображает только те товары, которые есть в базе данных.

3) Модуль “История покупок” - позволяет сохранять данные об уже купленных пользователем товарах. Взаимодействует с модулем "Список покупок", для определения товаров, которые были куплены ранее. Ключевая функция – доступ к информации об уже завершенных покупках, для дальнейшего планирования расходов. Для его проверки перейдем на вкладку “История” и проверим, что наш товар там отображается (Рисунок 35).

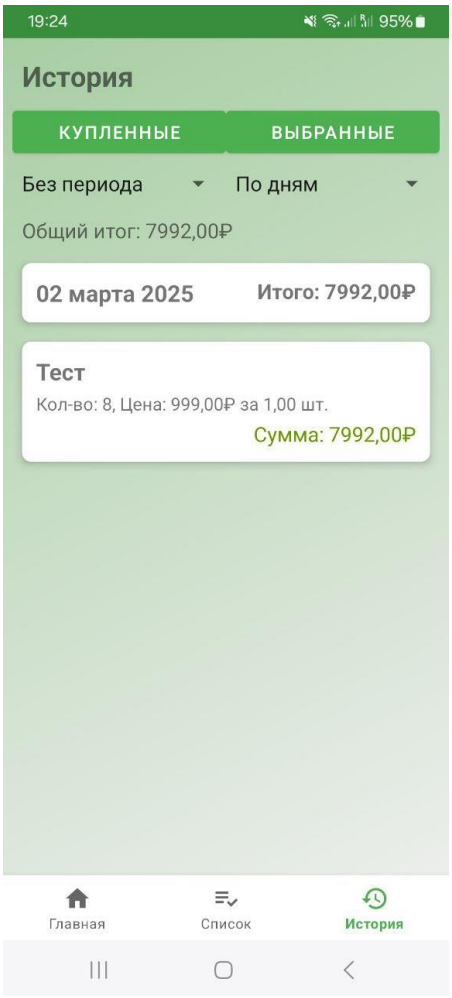

Рисунок 35 – Товар на вкладке истории

Результат оправдал ожидания, после отметки товара как купленного, он отобразился на вкладке “История” с указанием даты, количества цены и общей стоимости продуктов за этот день.

8.3. 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 api. В данном тесте симулировалась нагрузка на API когда к ней обращается одновременно 1000 пользователей по 5 раз.

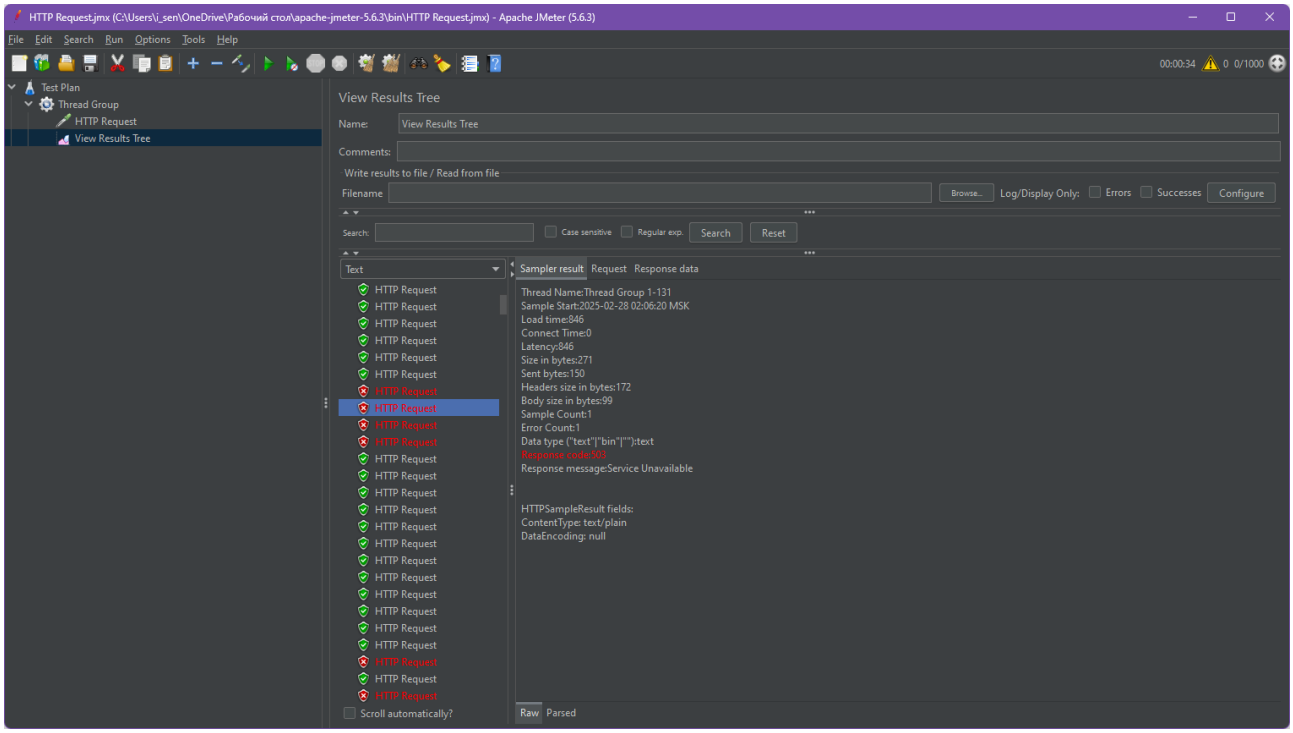

Рисунок 36 – Результаты тестирования сайта

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.

8.4. Провести тестирование пользовательского интерфейса (UI testing).

Проведем проверку функции фильтрации продуктов (Рисунок 37). Для этого выберем одно из категорий и нажмем на кнопку “Фильтр”.

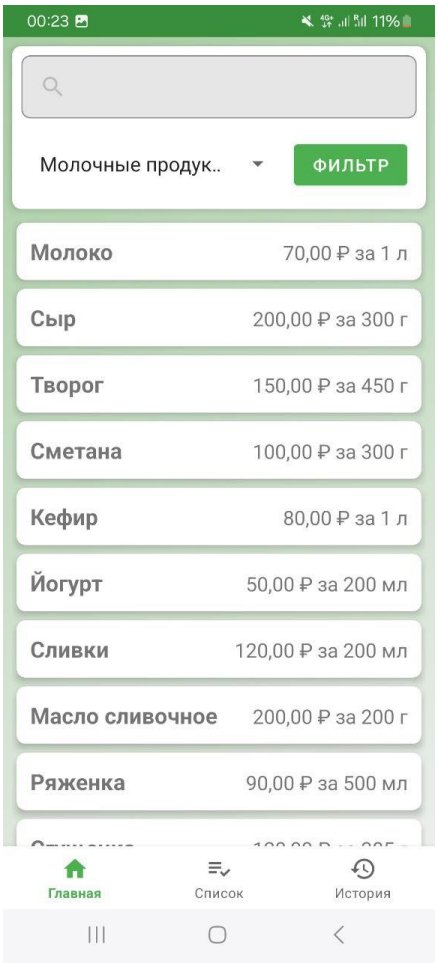

Рисунок 37 – Проверка функции фильтрации продуктов

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

Проведем проверку поиска продукта по ключевому слову (Рисунок 38). Впишем слово “Бананы”, чтобы найти данный товар и нажмем кнопку “Фильтр”.

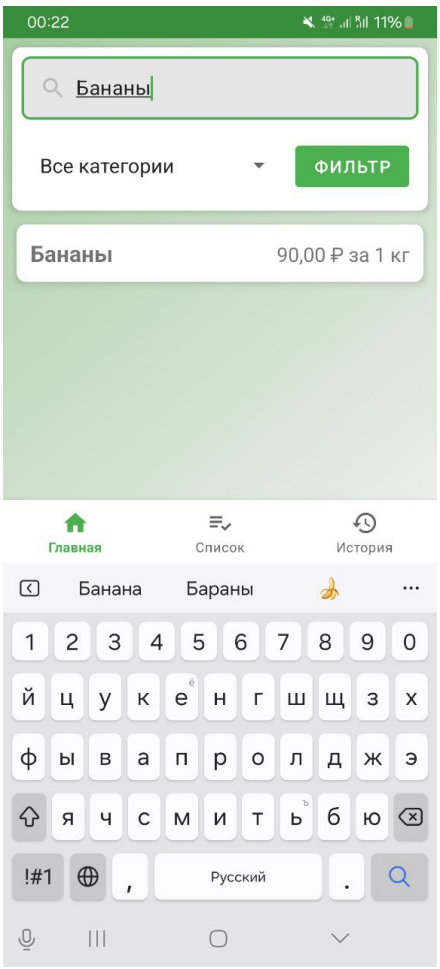

Рисунок 38 – Проверка поиска по ключевому слову

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 39 и 40). Нажмем на товар и изменим количество, нажмем на кнопку “Добавить”.

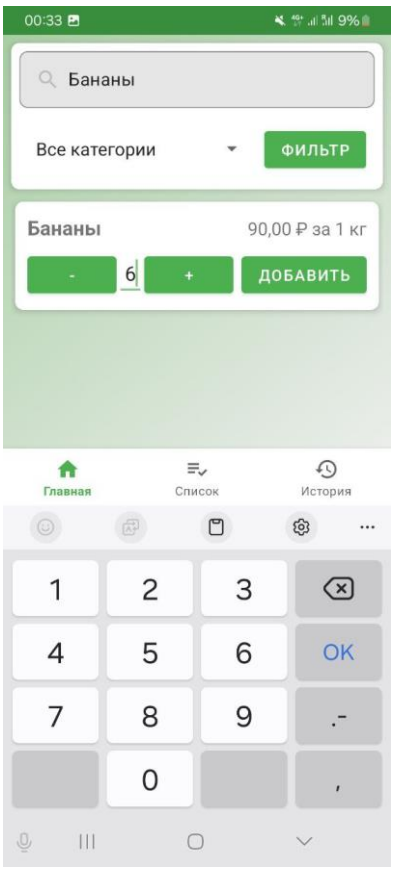

Рисунок 39 - Изменение количества товара

Рисунок 40 – 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 41, 42, 43).

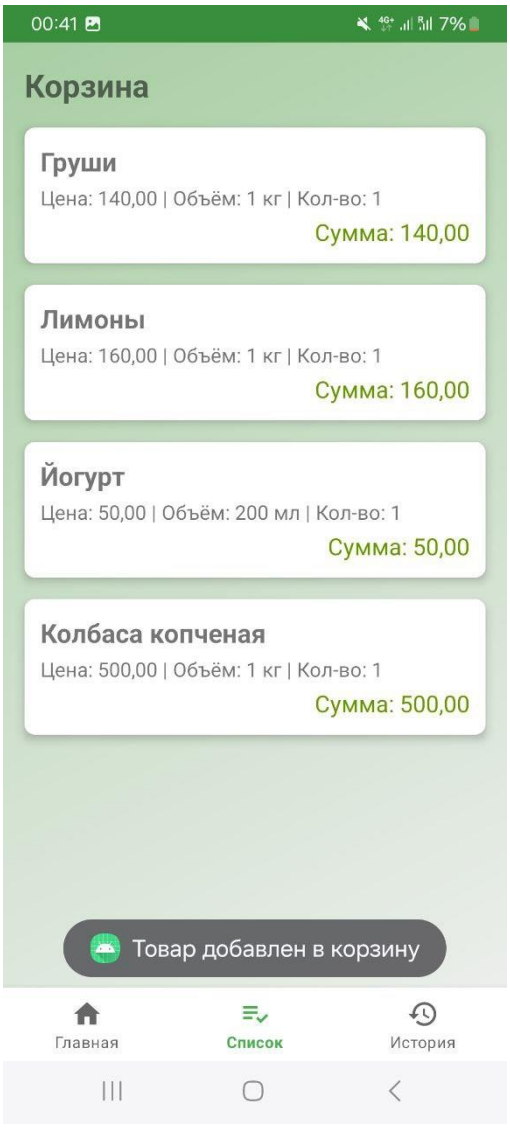

Рисунок 41 – Товары до удаления

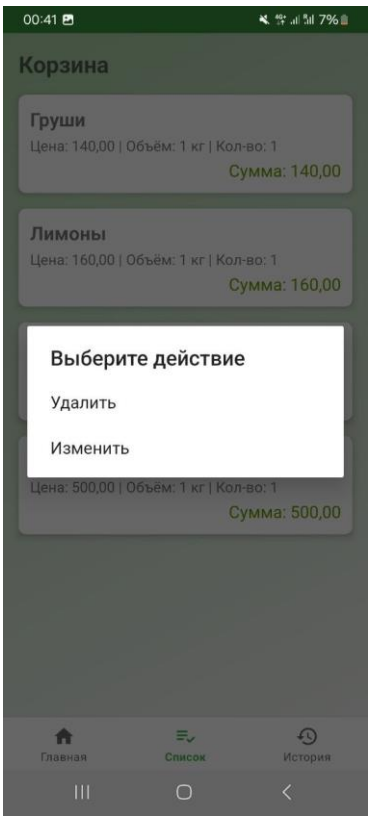

Рисунок 42 – Процесс удаления товара

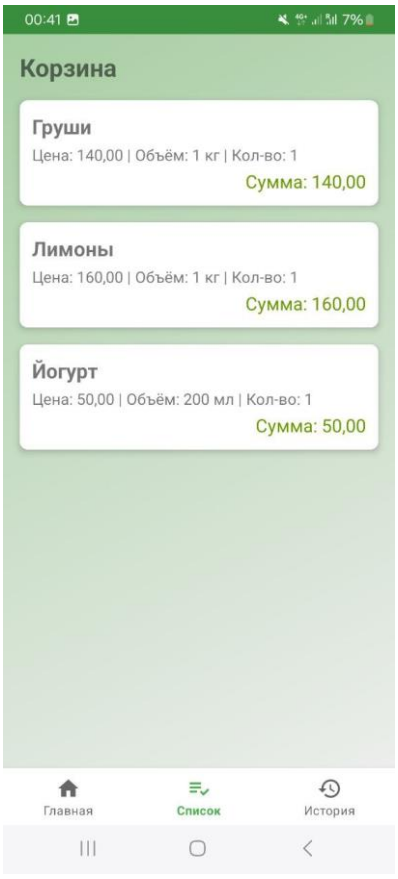

Рисунок 43 – Список после удаления

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 44, 45, 46).

Рисунок 44 - Товары в списке

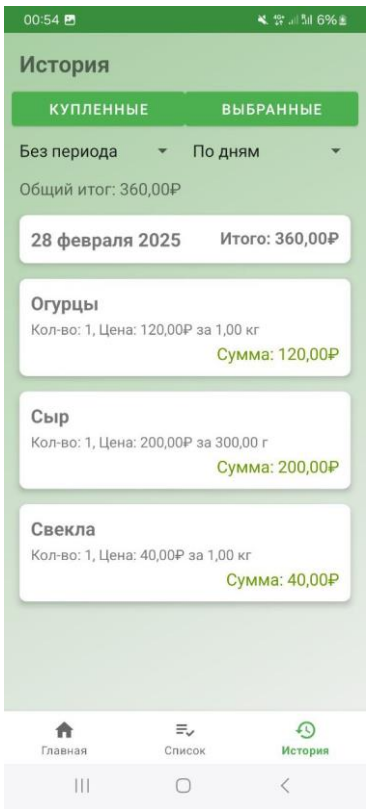

Рисунок 45 - Отображение купленных товаров

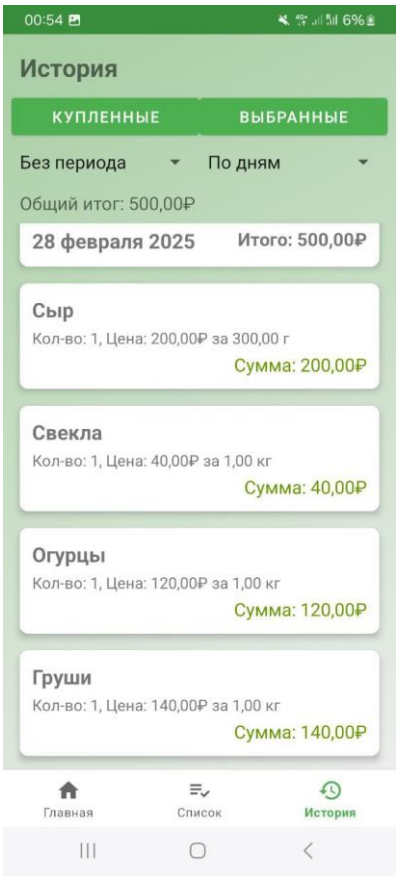

Рисунок 46 – Отображение всех выбранных в списке товаров

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

8.5. Развернуть приложение, настроить доменное имя и SSL-сертификат.

Для разворачивания приложения требуется захостить 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.

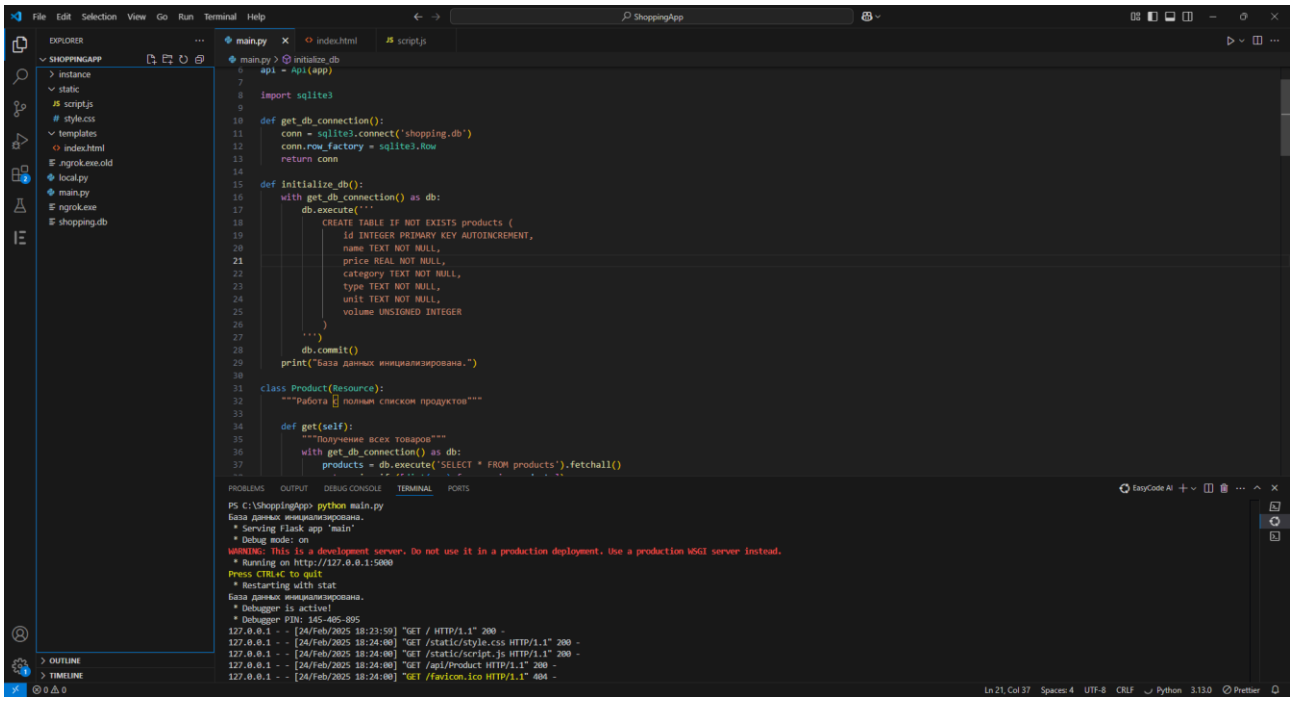

Рисунок 47 – Окно приложения Visual Studio code и запуск локального сервера

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.

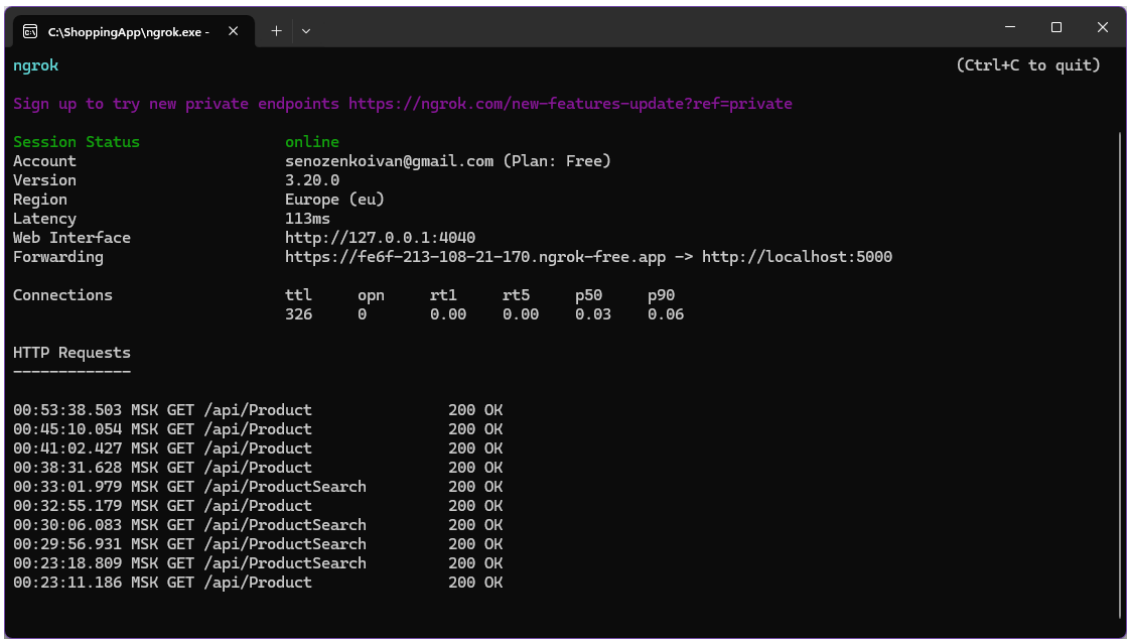

Рисунок 48 – Консоль сервиса ngrok сервера

8.6. 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.

8.7. Настроить процессы обновления и резервного копирования данных.

Для обновления приложения достаточно получить новую версию из магазина или скачать apk 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.

Ссылка на гит: https://github.com/Butearbroad/ShoppingAppSaaS

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.

Приложение А

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        BottomNavigationView bottomNavigationView = findViewById(R.id.bottom_navigation);
        bottomNavigationView.setOnNavigationItemSelectedListener(item -> {
            Fragment selectedFragment = null;
            int id = item.getItemId();
            if (id == R.id.navigation_home) {
                selectedFragment = new HomeFragment();
            } else if (id == R.id.navigation_selected) {
                selectedFragment = new SelectedListFragment();
            } else if (id == R.id.navigation_history) {
                selectedFragment = new HistoryFragment();
            }
            if(selectedFragment != null) {
                getSupportFragmentManager().beginTransaction()
                        .replace(R.id.fragment_container, selectedFragment)
                        .commit();
            }
            return true;
        });

        if (savedInstanceState == null) {
            getSupportFragmentManager().beginTransaction()
                    .replace(R.id.fragment_container, new HomeFragment())
                    .commit();
        }
    }
}


public class HomeFragment extends Fragment {
    @Nullable
    @Override
    public View onCreateView(@NonNull LayoutInflater inflater, @Nullable ViewGroup container,
                             @Nullable Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_home, container, false);
        EditText searchBar = view.findViewById(R.id.search_bar);
        searchBar.setFocusable(false); // чтобы не открывалась клавиатура сразу

        searchBar.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Bundle bundle = new Bundle();
                bundle.putBoolean("activateSearch", true);
```

```
            ProductListFragment productListFragment = new ProductListFragment();
            productListFragment.setArguments(bundle);
            getActivity().getSupportFragmentManager().beginTransaction()
                    .replace(R.id.fragment_container, productListFragment)
                    .addToBackStack(null)
                    .commit();
        }
    });
    return view;
  }
}


public class ProductListFragment extends Fragment {
  private ProductRepository productRepository;
  private RecyclerView recyclerView;
  private ProductAdapter productAdapter;
  private List<Product> productList = new ArrayList<>();
  private EditText searchInput;
  private Spinner categorySpinner;
  private Button filterButton;
  private SwipeRefreshLayout swipeRefreshLayout;

  private final List<String> categories = Arrays.asList(
      "Все категории", "Фрукты", "Овощи",
      "Молочные продукты", "Мясо", "Напитки"
  );

  @SuppressLint("MissingInflatedId")
  @Nullable
  @Override
  public View onCreateView(@NonNull LayoutInflater inflater,
                           @Nullable ViewGroup container,
                           @Nullable Bundle savedInstanceState) {
    View view = inflater.inflate(R.layout.fragment_product_list, container, false);

    searchInput = view.findViewById(R.id.search_input);
    categorySpinner = view.findViewById(R.id.category_spinner);
    filterButton = view.findViewById(R.id.filter_button);
    recyclerView = view.findViewById(R.id.recycler_view);
    swipeRefreshLayout = view.findViewById(R.id.swipeRefreshLayout);
    productRepository = ProductRepository.getInstance(requireContext());

    recyclerView.setLayoutManager(new LinearLayoutManager(getContext()));
    productAdapter = new ProductAdapter(productList);
    recyclerView.setAdapter(productAdapter);

    ArrayAdapter<String> spinnerAdapter = new ArrayAdapter<>(
        requireContext(),
        android.R.layout.simple_spinner_item,
```

```
                categories
        );

spinnerAdapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
        categorySpinner.setAdapter(spinnerAdapter);
        categorySpinner.setSelection(0);

        filterButton.setOnClickListener(v -> performSearch());

        swipeRefreshLayout.setOnRefreshListener(() -> {
            String query = searchInput.getText().toString().trim();
            String category = categorySpinner.getSelectedItem().toString();
            if (category.equals("Все категории")) category = "";

            String finalCategory = category;
            productRepository.syncProductsWithApi(query, category, () -> {
                loadProductsFromDatabase(query, finalCategory);
                swipeRefreshLayout.setRefreshing(false);
            });
        });

        productRepository.syncProductsWithApi("", "", () ->
                loadProductsFromDatabase("", "")
        );

        return view;
    }

    private void performSearch() {
        String query = searchInput.getText().toString().trim();
        String category = categorySpinner.getSelectedItem().toString();
        if (category.equals("Все категории")) category = "";

        String finalCategory = category;
        productRepository.syncProductsWithApi(query, category, () ->
                loadProductsFromDatabase(query, finalCategory)
        );
    }

    private void loadProductsFromDatabase(String query, String category) {
        productRepository.searchProducts(query, category, this::updateUI);
    }

    private void updateUI(List<Product> data) {
        requireActivity().runOnUiThread(() -> {
            productList.clear();
            productList.addAll(data);
            productAdapter.resetSelection();
            productAdapter.notifyDataSetChanged();
        });
```

```
    }
}


public class SelectedListFragment extends Fragment {

    private RecyclerView recyclerView;
    private SelectedProductAdapter adapter;
    private final List<SelectedProductDetail> selectedProductDetailList = new ArrayList<>();

    public SelectedListFragment() {
        // Пустой конструктор
    }

    @Override
    public View onCreateView(@NonNull LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_selected_product_list, container, false);
        recyclerView = view.findViewById(R.id.recyclerViewSelectedProducts);
        recyclerView.setLayoutManager(new LinearLayoutManager(getContext()));
        adapter = new SelectedProductAdapter(getContext(), selectedProductDetailList);
        recyclerView.setAdapter(adapter);


        new LoadSelectedProductsTask().execute();

        return view;
    }

    private class LoadSelectedProductsTask extends AsyncTask<Void, Void,
List<SelectedProductDetail>> {
        @Override
        protected List<SelectedProductDetail> doInBackground(Void... voids) {
            SelectedProductDao selectedDao =
AppDatabase.getInstance(getContext()).selectedProductDao();
            List<SelectedProduct> selectedProducts = selectedDao.getAllSelectedProducts();
            List<SelectedProductDetail> details = new ArrayList<>();
            for (SelectedProduct sp : selectedProducts) {
                Product product =
AppDatabase.getInstance(getContext()).productDao().getProductById(sp.getProductId());
                if (product != null) {
                    details.add(new SelectedProductDetail(
                            sp.getId(),
                            product.getId(),
                            product.getName(),
                            product.getPrice(),
                            product.getUnit(),
                            product.getVolume(),
                            sp.getQuantity(),
```

```
                        sp.isPurchased()
                ));
            }
        }
        return details;
    }

    @Override
    protected void onPostExecute(List<SelectedProductDetail> details) {
        selectedProductDetailList.clear();
        selectedProductDetailList.addAll(details);
        adapter.notifyDataSetChanged();
    }
  }
}


public class ProductAdapter extends RecyclerView.Adapter<ProductAdapter.ProductViewHolder>
{
  private List<Product> productList;
  private int selectedPosition = -1;

  public ProductAdapter(List<Product> productList) {
    this.productList = productList;
  }

  public void resetSelection() {
    int previous = selectedPosition;
    selectedPosition = -1;
    if (previous != -1) {
      notifyItemChanged(previous);
    }
  }

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

  @Override
  public void onBindViewHolder(@NonNull ProductViewHolder holder, int position) {
    Product product = productList.get(position);
    holder.bind(product);

    if (position == selectedPosition) {
      holder.quantityLayout.setVisibility(View.VISIBLE);
      holder.quantityEdit.setText("1");
```

```
    } else {
        holder.quantityLayout.setVisibility(View.GONE);
    }
}

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

public class ProductViewHolder extends RecyclerView.ViewHolder {
    TextView productName, productPrice;
    LinearLayout quantityLayout;
    Button addButton, minusButton, plusButton;
    EditText quantityEdit;

    public ProductViewHolder(@NonNull View itemView) {
        super(itemView);
        productName = itemView.findViewById(R.id.product_name);
        productPrice = itemView.findViewById(R.id.product_price);
        quantityLayout = itemView.findViewById(R.id.quantity_layout);
        addButton = itemView.findViewById(R.id.add_button);
        minusButton = itemView.findViewById(R.id.minus_button);
        plusButton = itemView.findViewById(R.id.plus_button);
        quantityEdit = itemView.findViewById(R.id.quantity_edit);

        itemView.setOnClickListener(view -> {
            int position = getAdapterPosition();
            if (position == RecyclerView.NO_POSITION) return;

            int previous = selectedPosition;
            selectedPosition = (position == selectedPosition) ? -1 : position;

            if (previous != -1) notifyItemChanged(previous);
            if (selectedPosition != -1) notifyItemChanged(selectedPosition);
        });

        minusButton.setOnClickListener(v -> updateQuantity(-1));
        plusButton.setOnClickListener(v -> updateQuantity(1));

        addButton.setOnClickListener(v -> {
            int position = getAdapterPosition();
            if (position != RecyclerView.NO_POSITION) {
                addProductToCart();
                resetSelection();
            }
        });
    }

    private void updateQuantity(int delta) {
```

```
            try {
                int qty = Integer.parseInt(quantityEdit.getText().toString());
                qty = Math.max(qty + delta, 1);
                quantityEdit.setText(String.valueOf(qty));
            } catch (NumberFormatException e) {
                quantityEdit.setText("1");
            }
        }

        private void addProductToCart() {
            try {
                int qty = Integer.parseInt(quantityEdit.getText().toString());
                Product product = productList.get(getAdapterPosition());

                SelectedProductRepository.getInstance(itemView.getContext())
                        .insertSelectedProduct(new SelectedProduct(product.getId(), qty));

                new Thread(() -> {
                    ProductHistory history = new ProductHistory(
                            product.getName(),
                            qty,
                            product.getPrice(),
                            product.getVolume(),
                            product.getUnit(),
                            System.currentTimeMillis(),
                            1  // 1 - выбранный товар
                    );
                    AppDatabase.getInstance(itemView.getContext())
                            .productHistoryDao().insertHistory(history);
                }).start();

                Toast.makeText(itemView.getContext(),
                        "Товар добавлен в корзину",
                        Toast.LENGTH_SHORT).show();

            } catch (NumberFormatException e) {
                quantityEdit.setText("1");
            }
        }

        public void bind(Product product) {
            productName.setText(product.getName());
            productPrice.setText(String.format("%.2f ₽ за %d %s", product.getPrice(),
product.getVolume(), product.getUnit()));
        }
    }
}

@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;
    }
}

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);
}
```

```
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://88aa-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);
    }
}
```