|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |
| --- |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** |
| **Институт кибербезопасности и цифровых технологий** |

Кафедра КБ-14 «Цифровые технологии обработки данных»

**ОТЧЕТ**

**Практическая работа 6**

**по дисциплине «Разработка мобильных компонент анализа безопасности информационно-аналитических систем»**

Выполнил:

Студент группы БСБО-07-22

Сладкина Анастасия Андреевна

**Москва, 2025 г**

Для выполнения задания в res/layout/activity\_main.xmlи добавлено 3 EditText и Button (см. рис. 1).

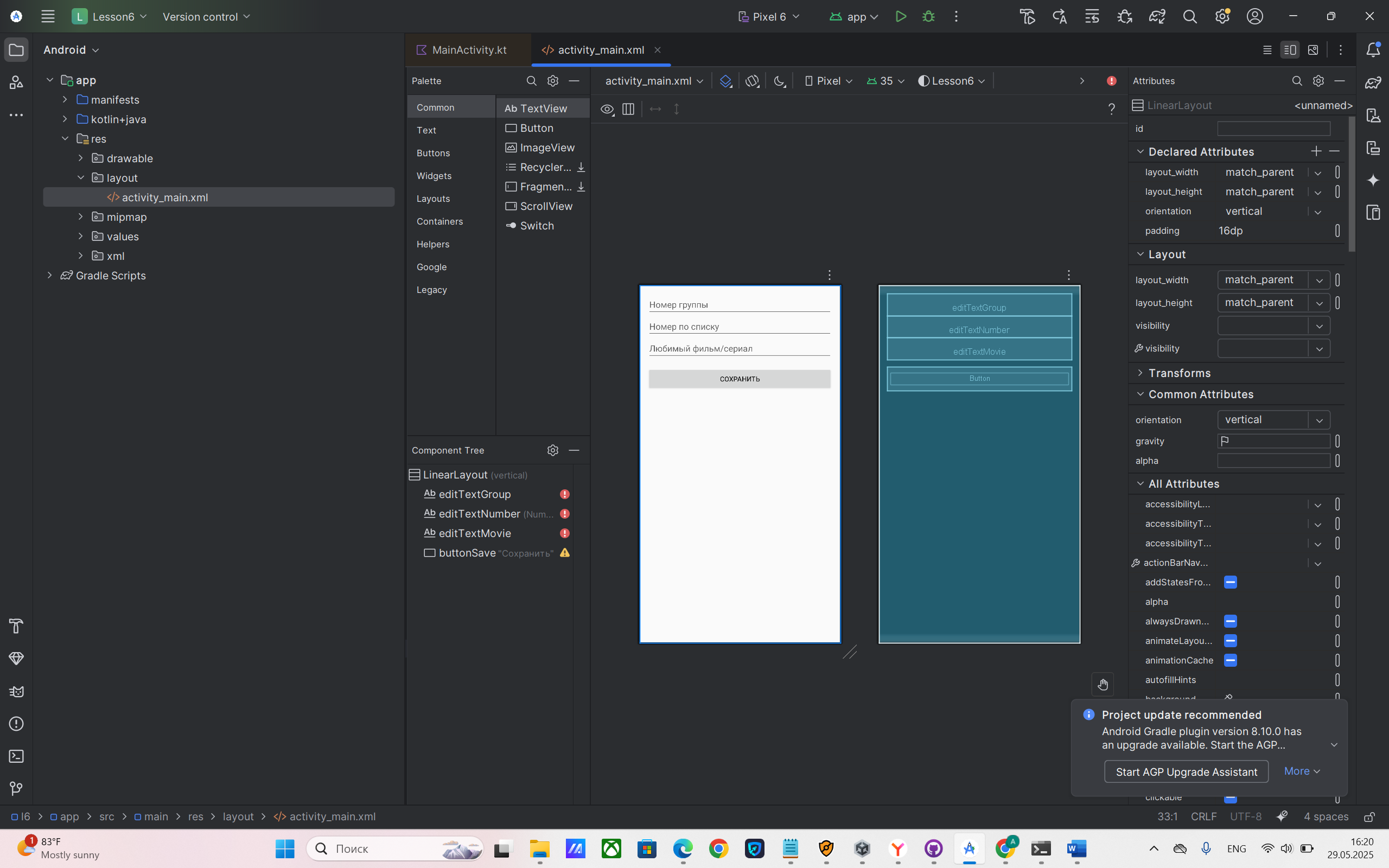


Рисунок 1. Дизайн экрана с 3 EditText и Button

Активность MainActivity предназначена для сохранения и отображения пользовательских данных через механизм SharedPreferences (см. рис. 2). При запуске активности инициализируются три текстовых поля (editTextGroup, editTextNumber, editTextMovie) для ввода информации о группе, номере и любимом фильме соответственно, а также кнопка сохранения buttonSave. Система автоматически загружает ранее сохраненные значения из SharedPreferences (файл настроек "mirea\_settings") и отображает их в соответствующих полях ввода при каждом запуске приложения. При нажатии на кнопку сохранения текущие значения из текстовых полей записываются в SharedPreferences через объект Editor с использованием методов putString() для каждого параметра, после чего изменения применяются асинхронно с помощью apply().

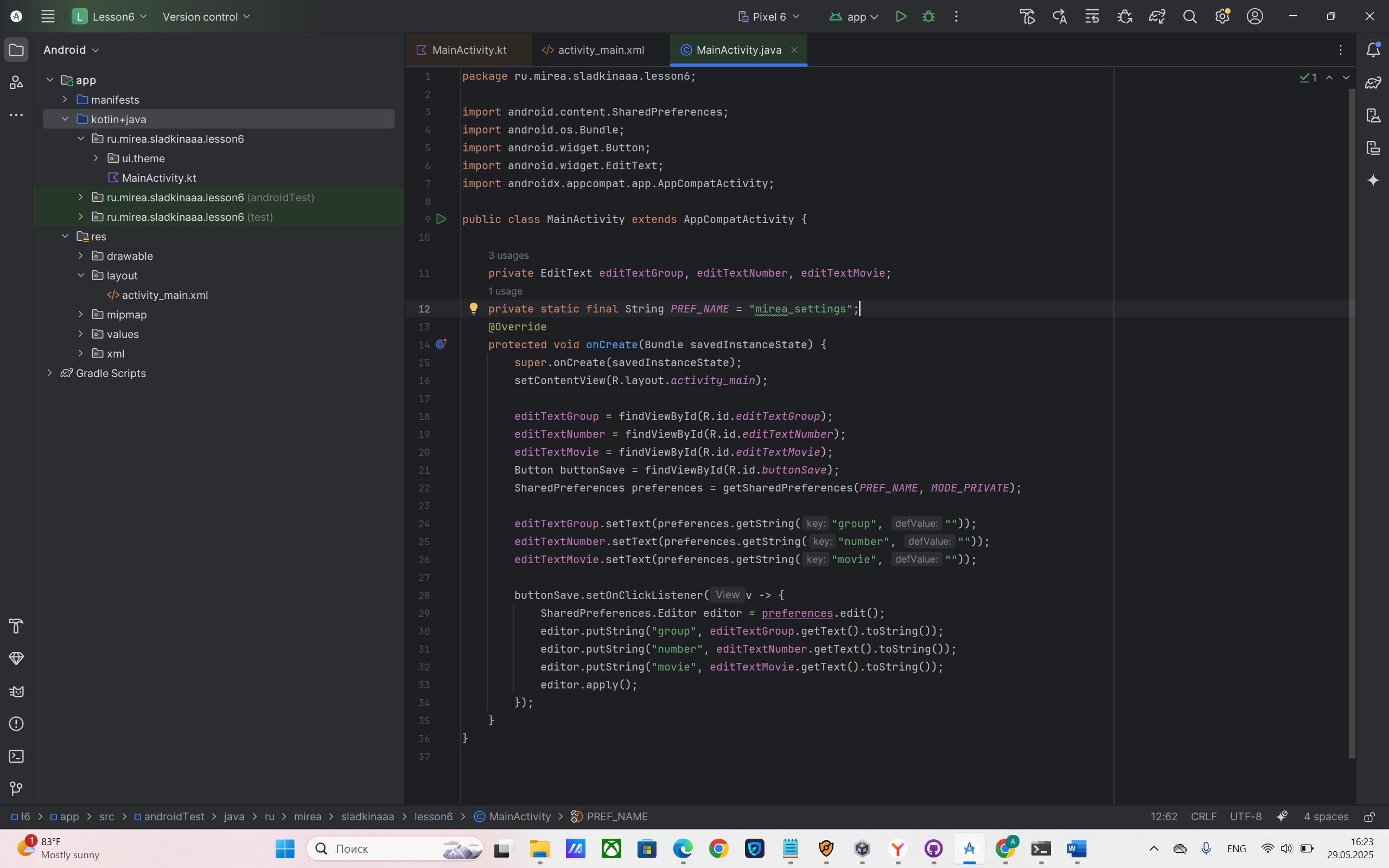


Рисунок 2. Сохранения и отображения пользовательских данных

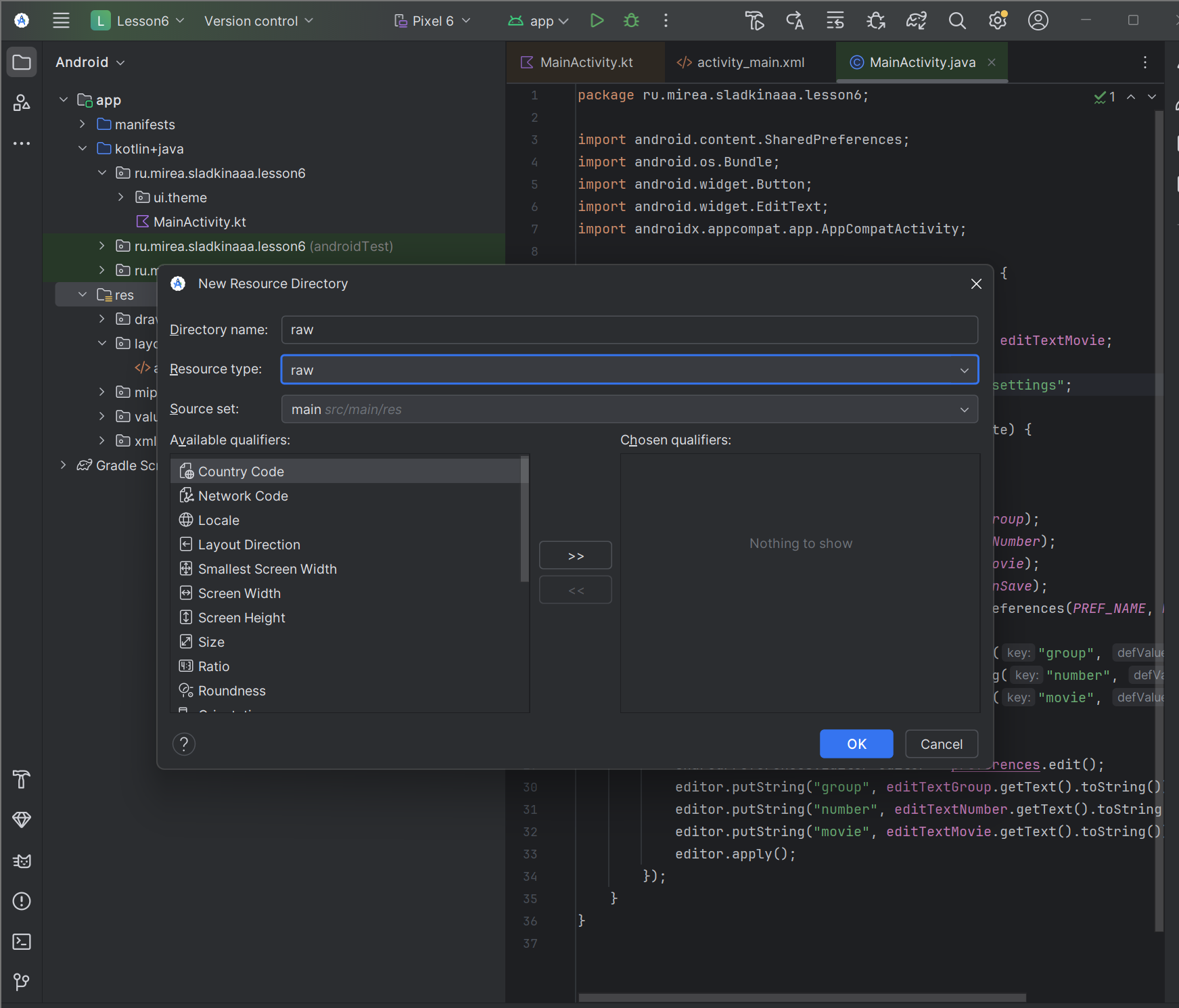


Рисунок 3. Создание директории raw

Создадим экран работы с файлами настроек (см. рис. 4).

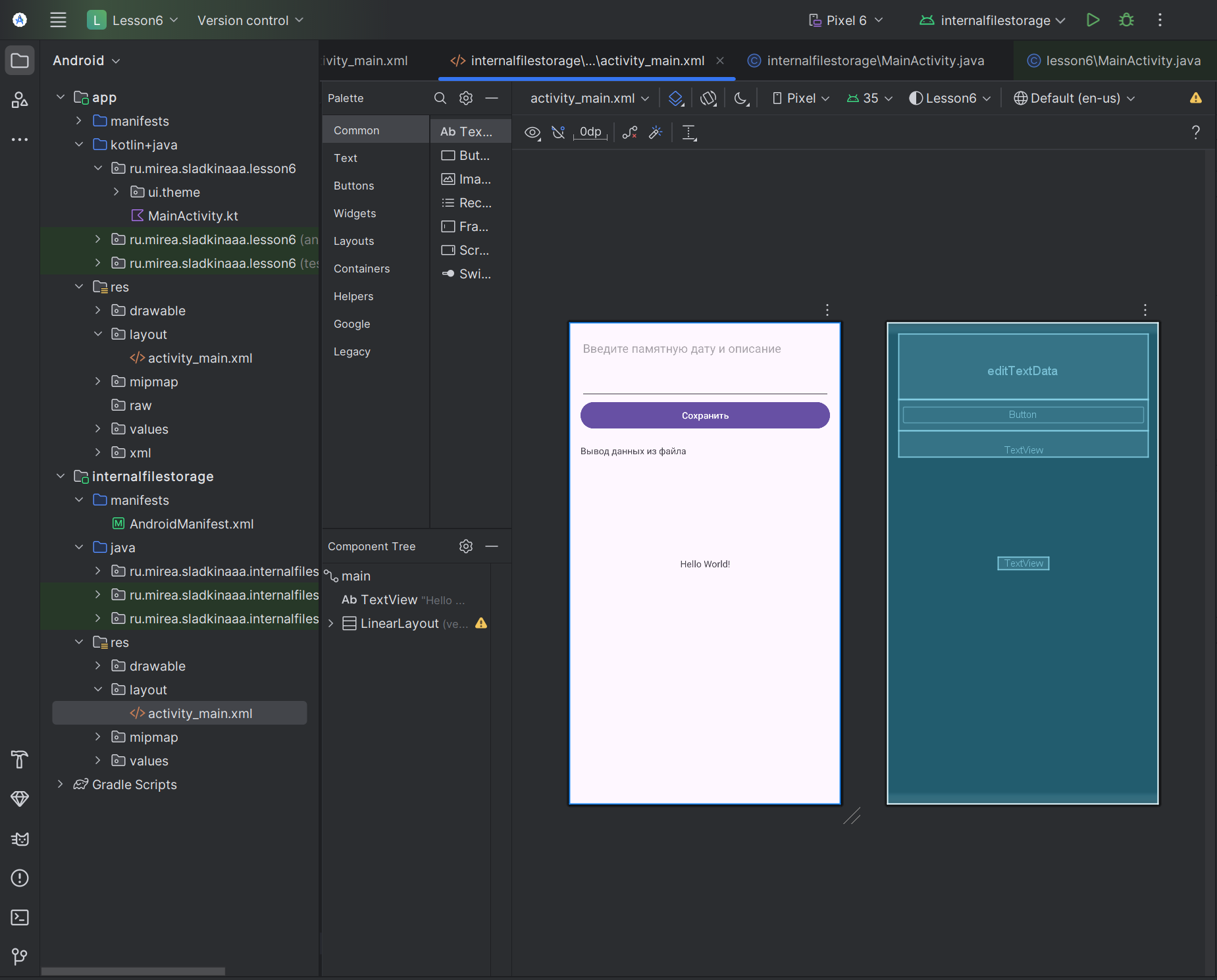


Рисунок 4. Дизайн экрана работы с данными

Рассмотрим MainActivity, которая обеспечивает ввод, сохранение и чтение текстовых данных с использованием внутреннего хранилища устройства (см. рис. 5). В методе onCreate происходит инициализация пользовательского интерфейса: поле ввода EditText, текстовое поле TextView для вывода информации и кнопка Button. При нажатии на кнопку вводимые пользователем данные из EditText сохраняются в файл russian\_history.txt во внутренней памяти с помощью метода writeToFile, использующего FileOutputStream. После записи запускается отдельный поток readFromFileInThread, в котором происходит считывание данных из того же файла через FileInputStream, обёрнутый в BufferedInputStream. Считанные байты преобразуются в символы и добавляются в StringBuilder, после чего результат отображается в TextView с помощью метода runOnUiThread, обеспечивая безопасное обновление пользовательского интерфейса из фонового потока. Такой подход позволяет корректно работать с файлами и не блокировать главный поток приложения.

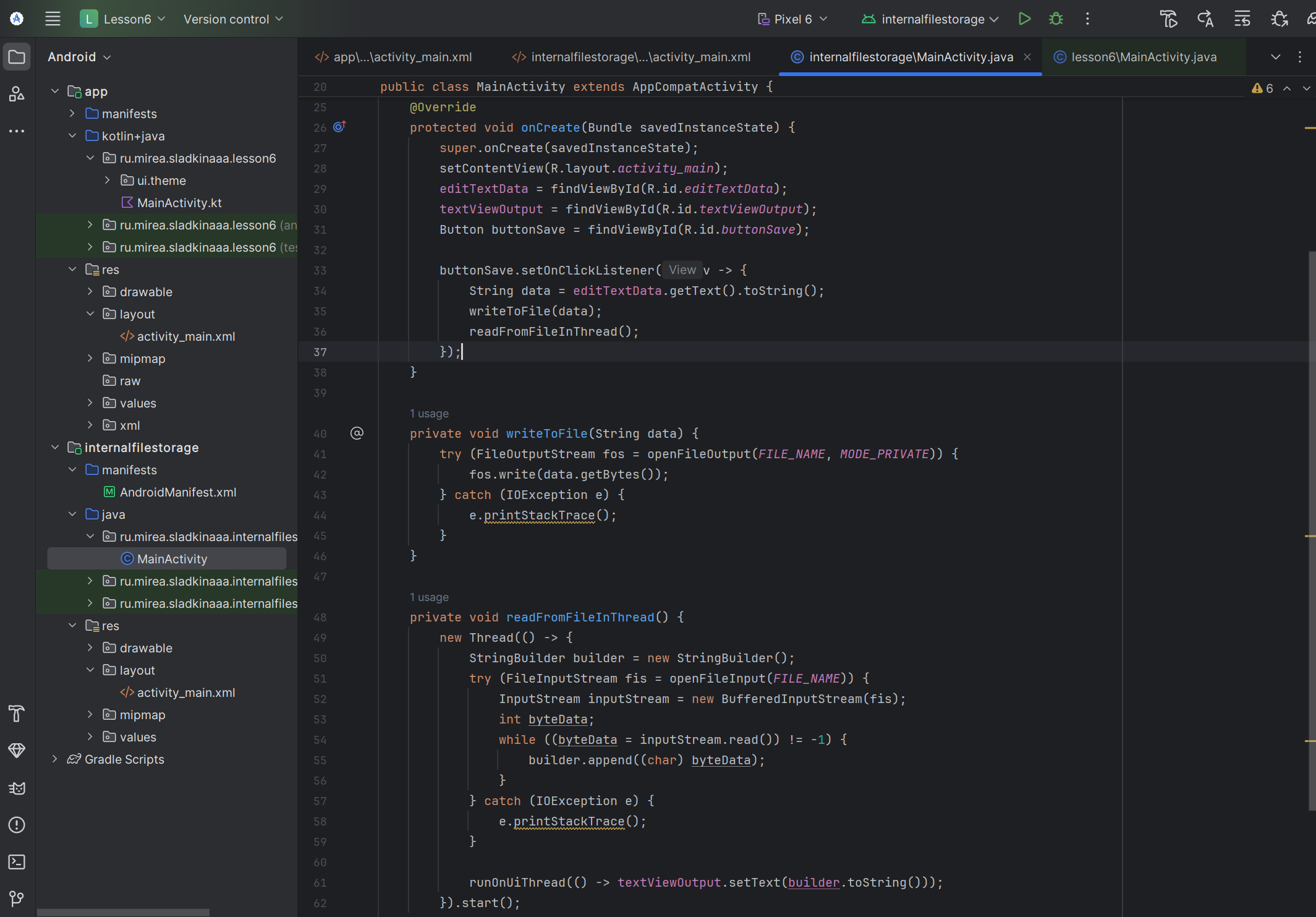


Рисунок 5. Работа с данными в MainActivity

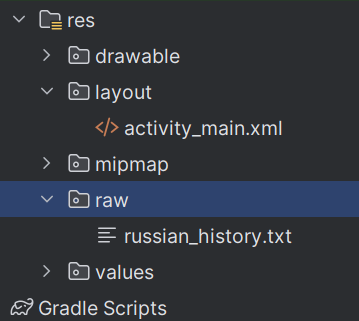


Рисунок 6. Перемещение созданного файла в raw

Для приложения «Блокнот» были добавлены на экран поля ввода «названия файла» и «цитата», кнопки «сохранить данные в файл» и «загрузить данные» (см. рис. 7).

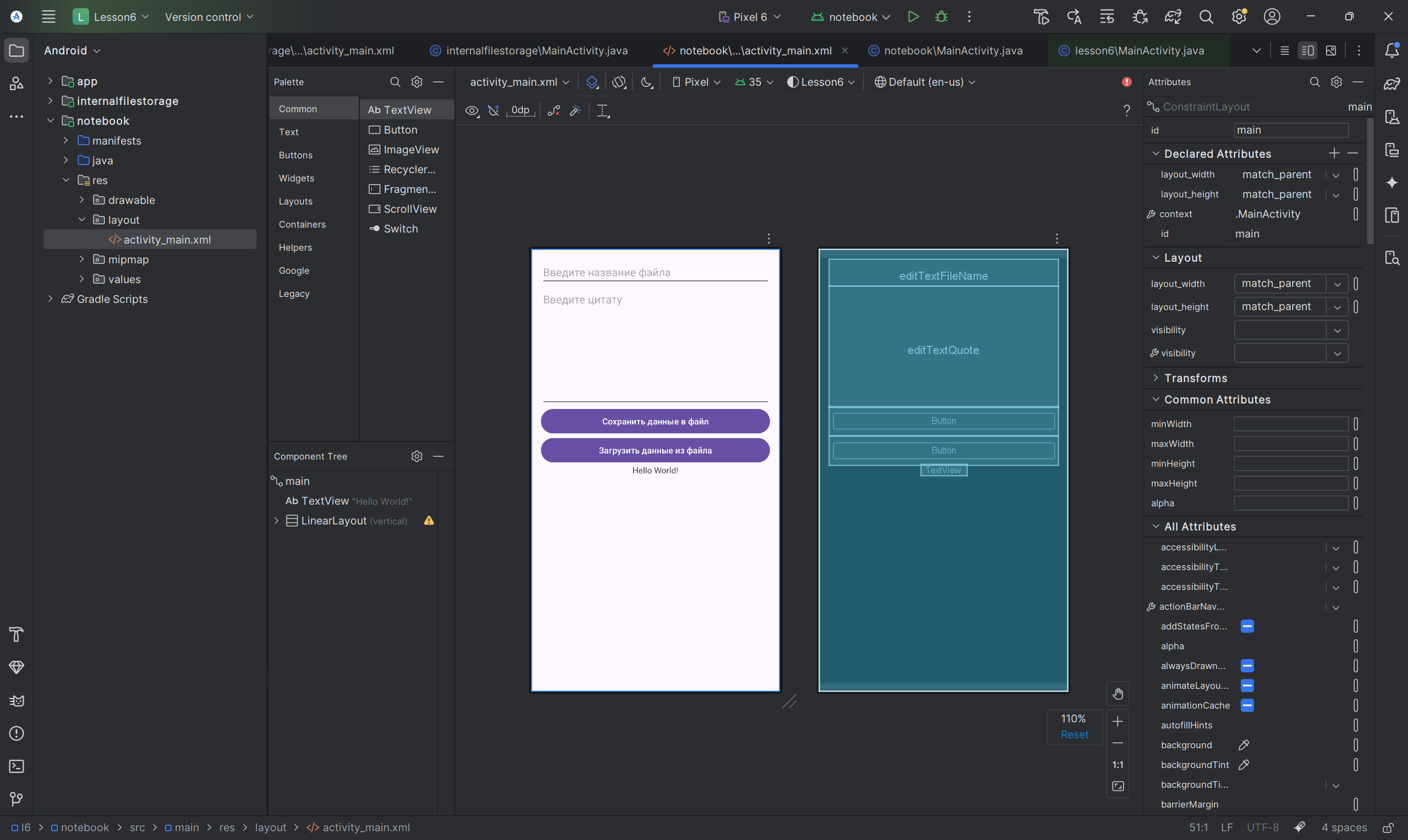


Рисунок 7. Экран приложения «Блокнот»

При нажатии на кнопку сохранения вызывается метод saveToFile(), который формирует путь к файлу, создаёт при необходимости директорию, и записывает в файл текст, введённый в поле цитаты. При успешном завершении действия выводится уведомление Toast. Кнопка загрузки запускает метод loadFromFile(), который по заданному имени файла пытается открыть его, прочитать содержимое построчно с помощью BufferedReader и отобразить его в поле цитаты. В случае ошибок при сохранении или чтении файла, пользователю выводится соответствующее сообщение об ошибке (см. рис. 8).



Рисунок 8. Методы saveToFile() и loadFromFile()

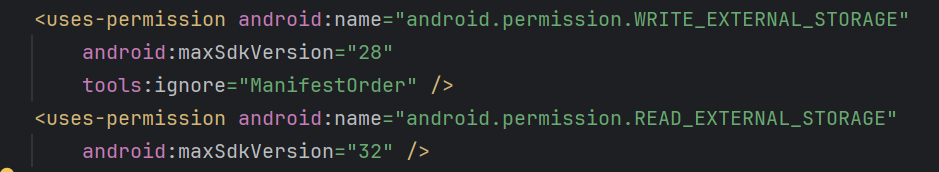


Рисунок 9. Добавление разрешений в манифест-файл

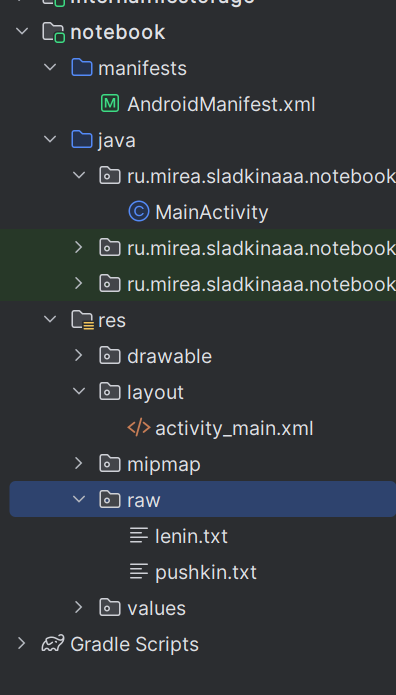


Рисунок 10. Перемещение созданных файлов в raw

Добавим зависимости Room (см. рис. 11).



Рисунок 11. Добавление зависимостей

Рассмотрим класс Employee в Java, который используется в Android-приложении с базой данных Room (см. рис. 12). Класс помечен аннотацией @Entity, что означает, что он соответствует таблице в базе данных. У класса есть четыре поля: id, name, power и powerLevel. Поле id обозначено как первичный ключ с автоинкрементом с помощью аннотации @PrimaryKey(autoGenerate = true), то есть значение этого поля будет автоматически увеличиваться при добавлении новых записей. Поля name, power и powerLevel хранят соответственно имя сотрудника, его суперспособность и уровень этой способности. Также в классе определён конструктор, который инициализирует значения этих трёх полей (кроме id, который присваивается автоматически Room).

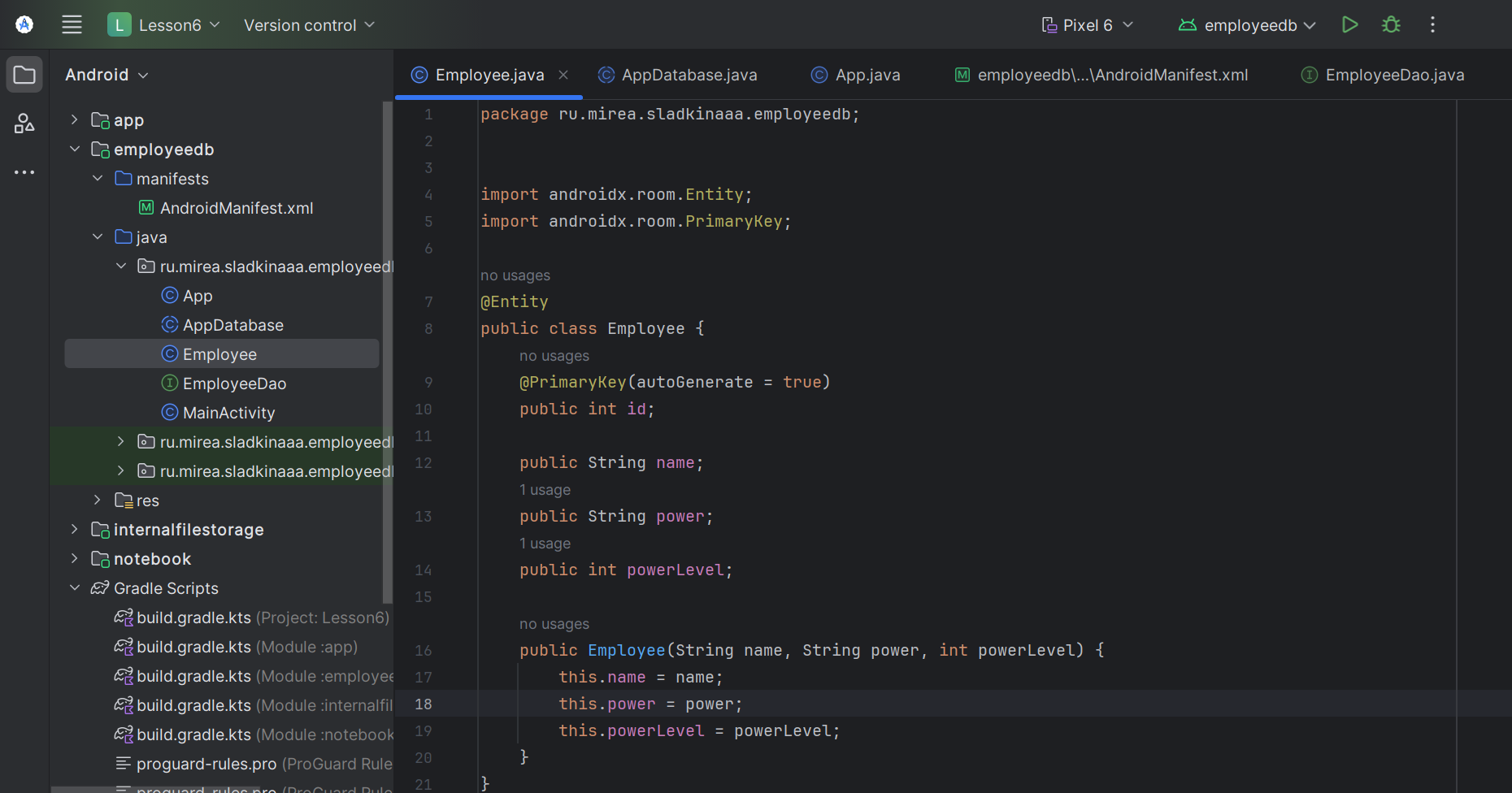


Рисунок 12. Класс Employee

Интерфейс EmployeeDao используется для доступа к данным таблицы Employee. Интерфейс помечен аннотацией @Dao, что означает "Data Access Object" — объект доступа к данным. Внутри определены методы, которые позволяют выполнять различные операции с данными. Метод getAll() возвращает список всех сотрудников из таблицы. Метод getById(int id) получает одного сотрудника по его идентификатору. Метод insert(Employee employee) вставляет одного сотрудника в базу данных. Метод insertAll(Employee... employees) позволяет вставить сразу несколько сотрудников, и если при этом возникает конфликт (например, повторяющийся ID), то запись будет заменена благодаря стратегии REPLACE. Метод update(Employee employee) обновляет существующую запись в базе, а метод delete(Employee employee) удаляет сотрудника.

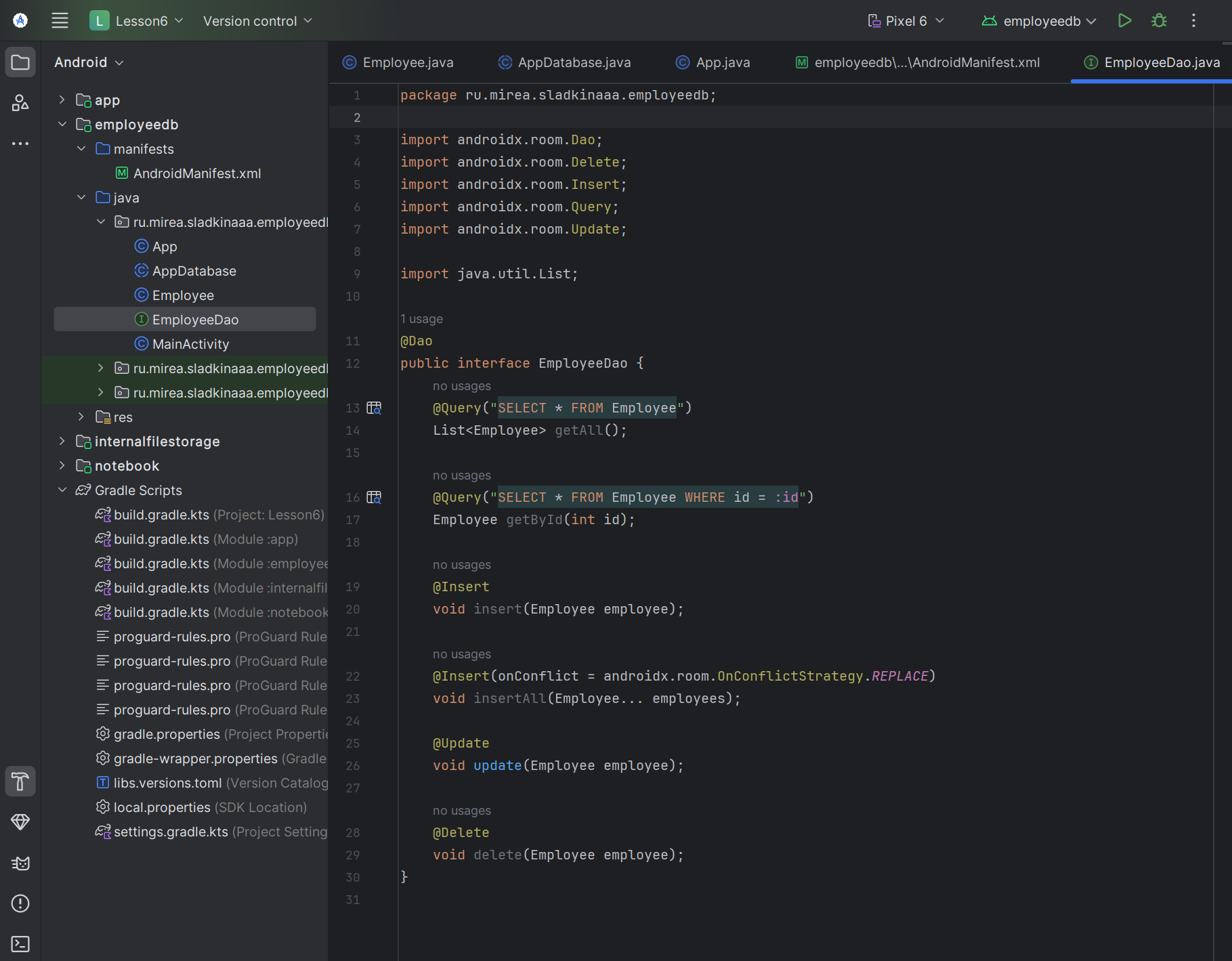


Рисунок 13. Интерфейс EmployeeDao

Абстрактный класс AppDatabase представляет собой основную базу данных Room в Android-приложении (см. рис. 14). Класс помечен аннотацией @Database, где указано, что в базе содержится одна сущность — класс Employee, а версия базы данных — 1. Класс AppDatabase наследуется от RoomDatabase, что делает его центральной точкой доступа к данным. Внутри определён абстрактный метод employeeDao(), который предоставляет объект EmployeeDao — интерфейс для выполнения операций с таблицей сотрудников. Этот класс используется для создания и управления экземпляром базы данных и доступа к DAO в приложении.

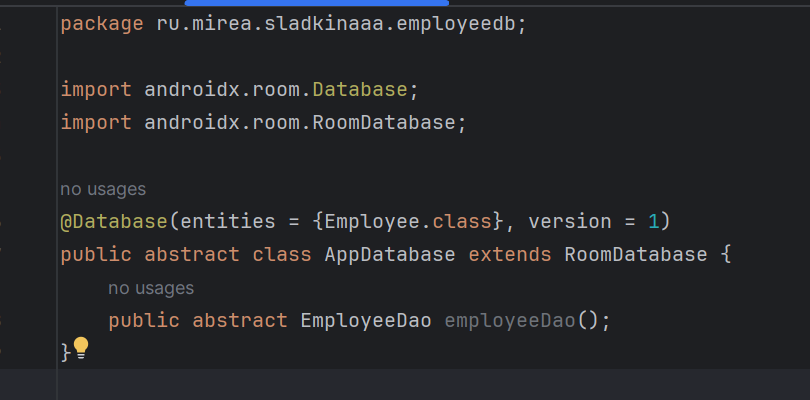


Рисунок 14. Абстрактный класс AppDatabase

Класс App расширяет Application и служит точкой инициализации базы данных Room при запуске Android-приложения (см. рис. 15). В методе onCreate(), который вызывается при старте приложения, создаётся объект базы данных AppDatabase с помощью Room.databaseBuilder(). В качестве контекста используется getApplicationContext(), указывается класс базы данных (AppDatabase.class) и задаётся имя базы — "superhero\_db". Полученный экземпляр сохраняется в статическом поле database, чтобы к нему можно было легко получить доступ из других частей приложения. Для этого также определён статический метод getDatabase(), который возвращает этот экземпляр.

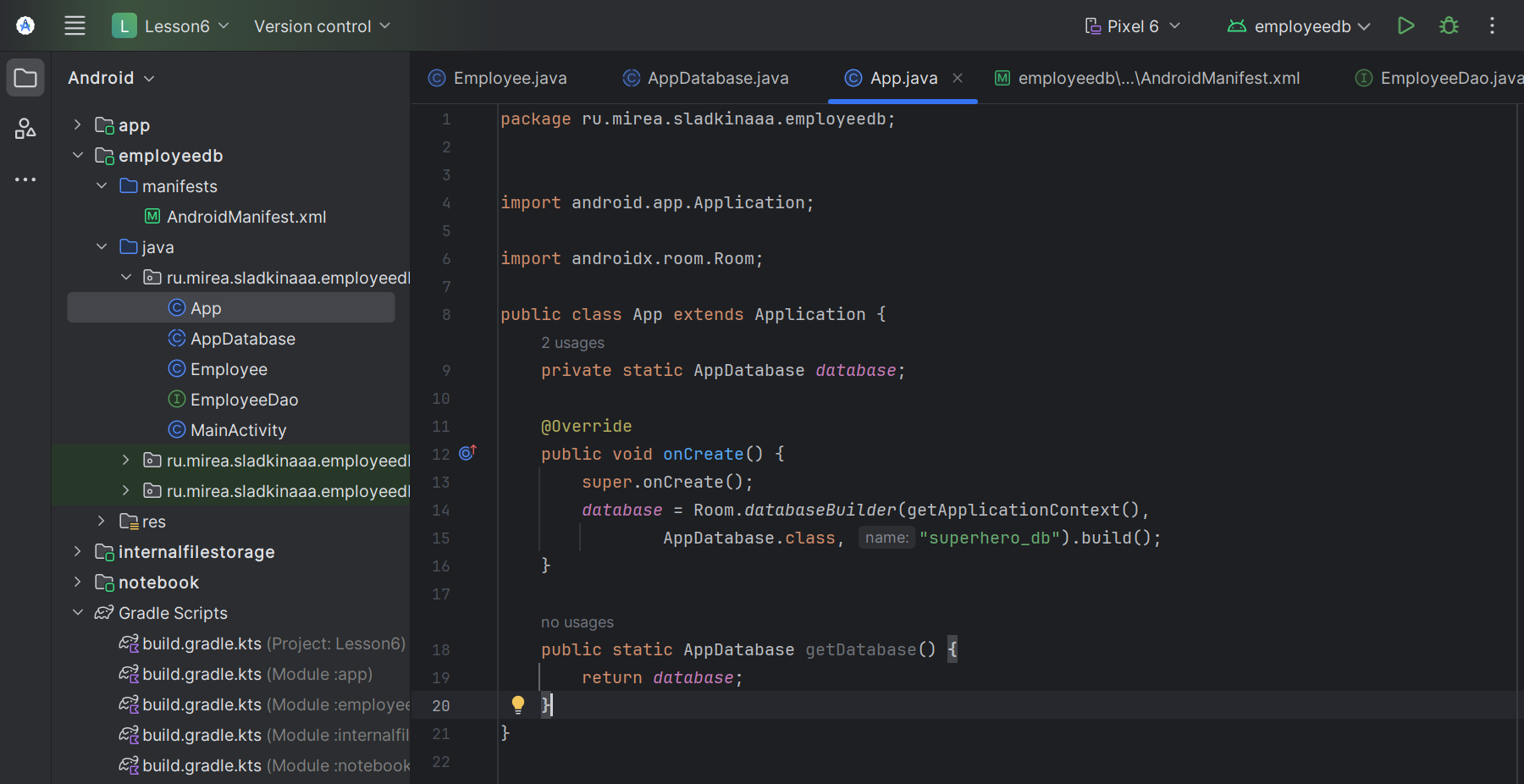


Рисунок 15. Класс App

Фрагмент **«Профиль»**, в котором пользователь указывает имя, возраст и уровень суперсилы (см. рис. 16). Все данные сохраняются в SharedPreferences и загружаются при открытии фрагмента.

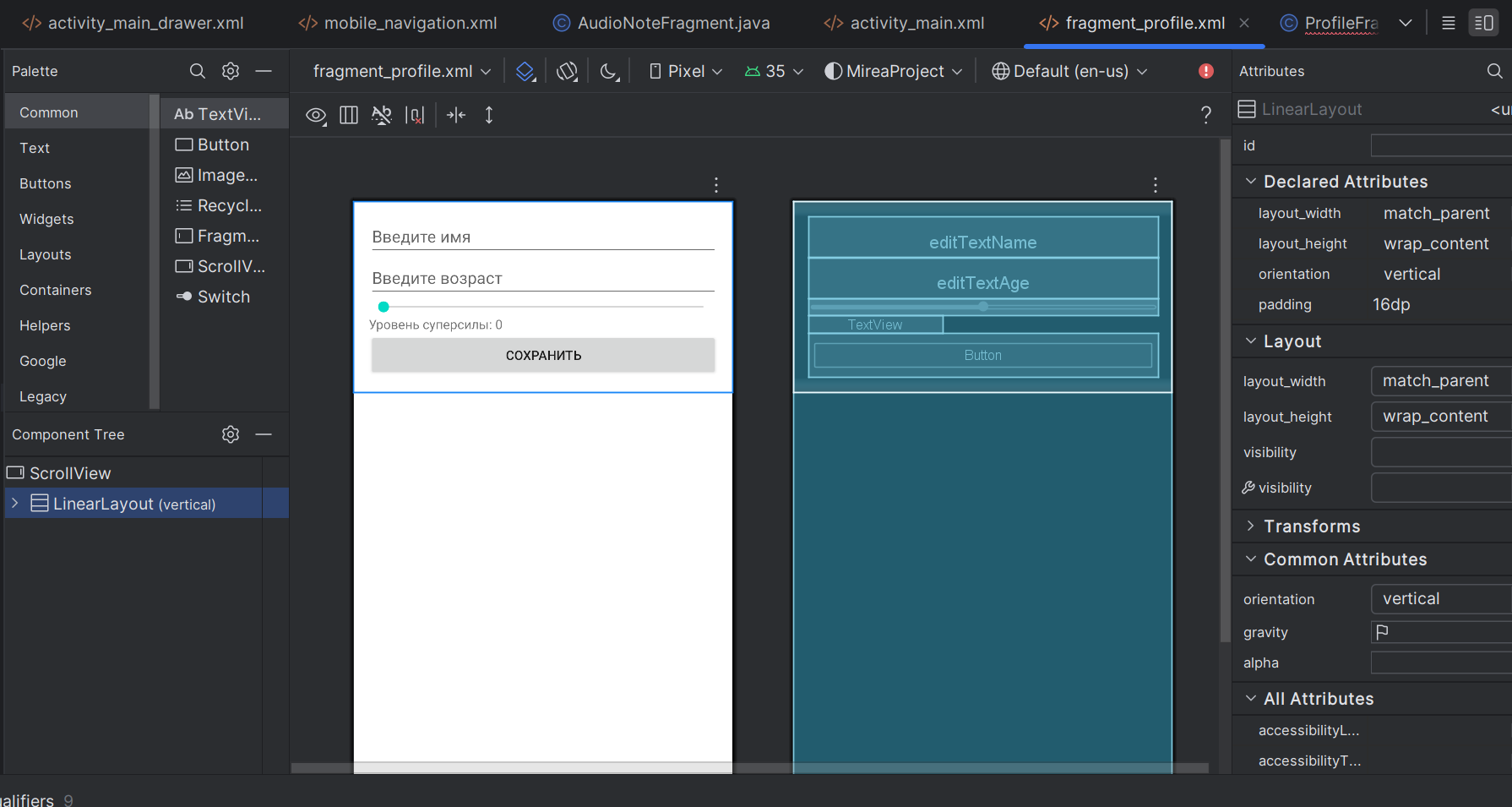


Рисунок 16. Дизайн экрана фрагмента «Профиль»

Фрагмент ProfileFragment, в котором пользователь может ввести своё имя, возраст и выбрать уровень суперсилы с помощью ползунка, после чего сохранить эти данные в SharedPreferences. При открытии фрагмента данные автоматически загружаются и отображаются в соответствующих полях. Ползунок (SeekBar) обновляет текст с текущим значением уровня суперсилы, а при нажатии на кнопку «Сохранить» введённые значения записываются в локальное хранилище устройства и выводится уведомление об успешном сохранении (см. рис. 17).

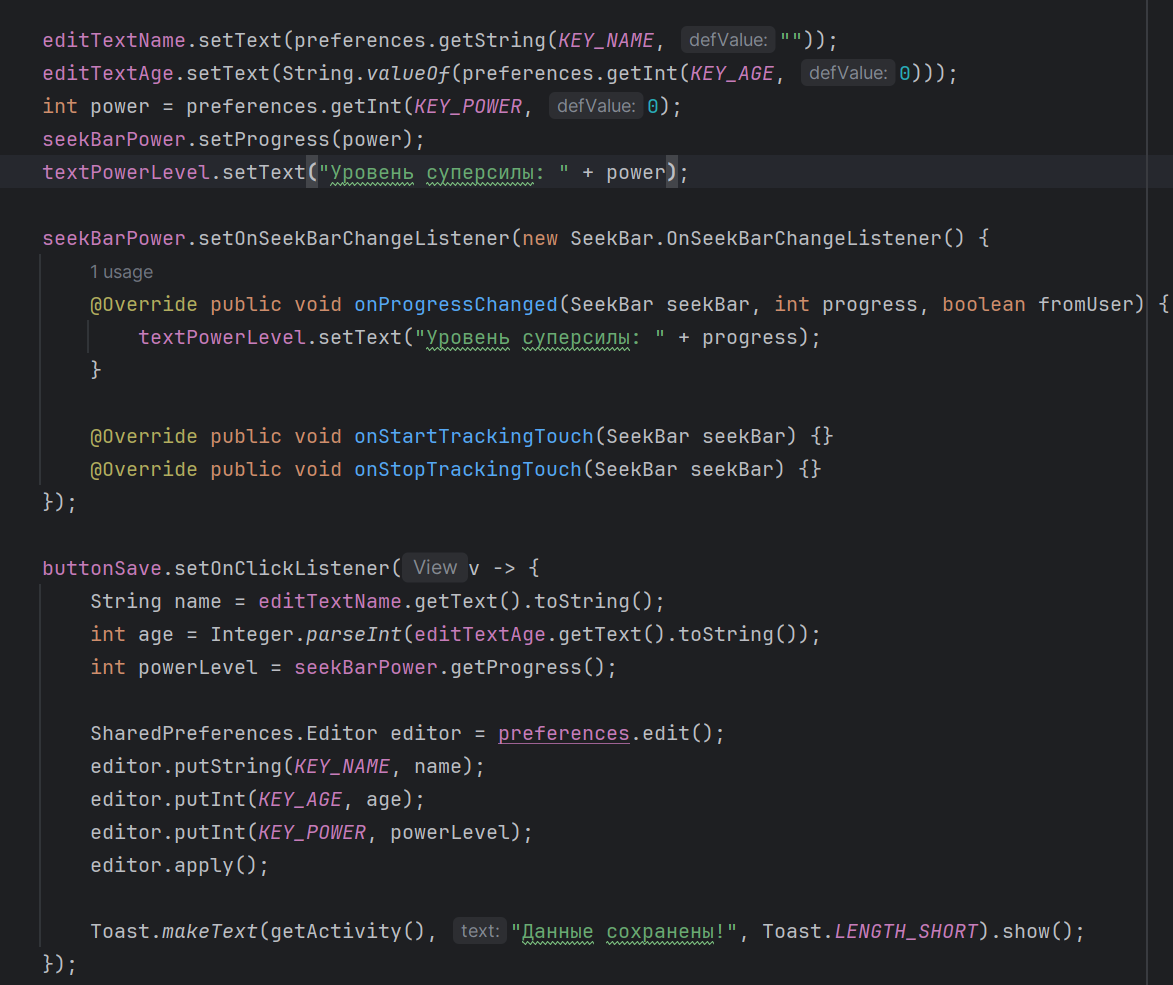


Рисунок 17. Реализация фрагмента ProfileFragment

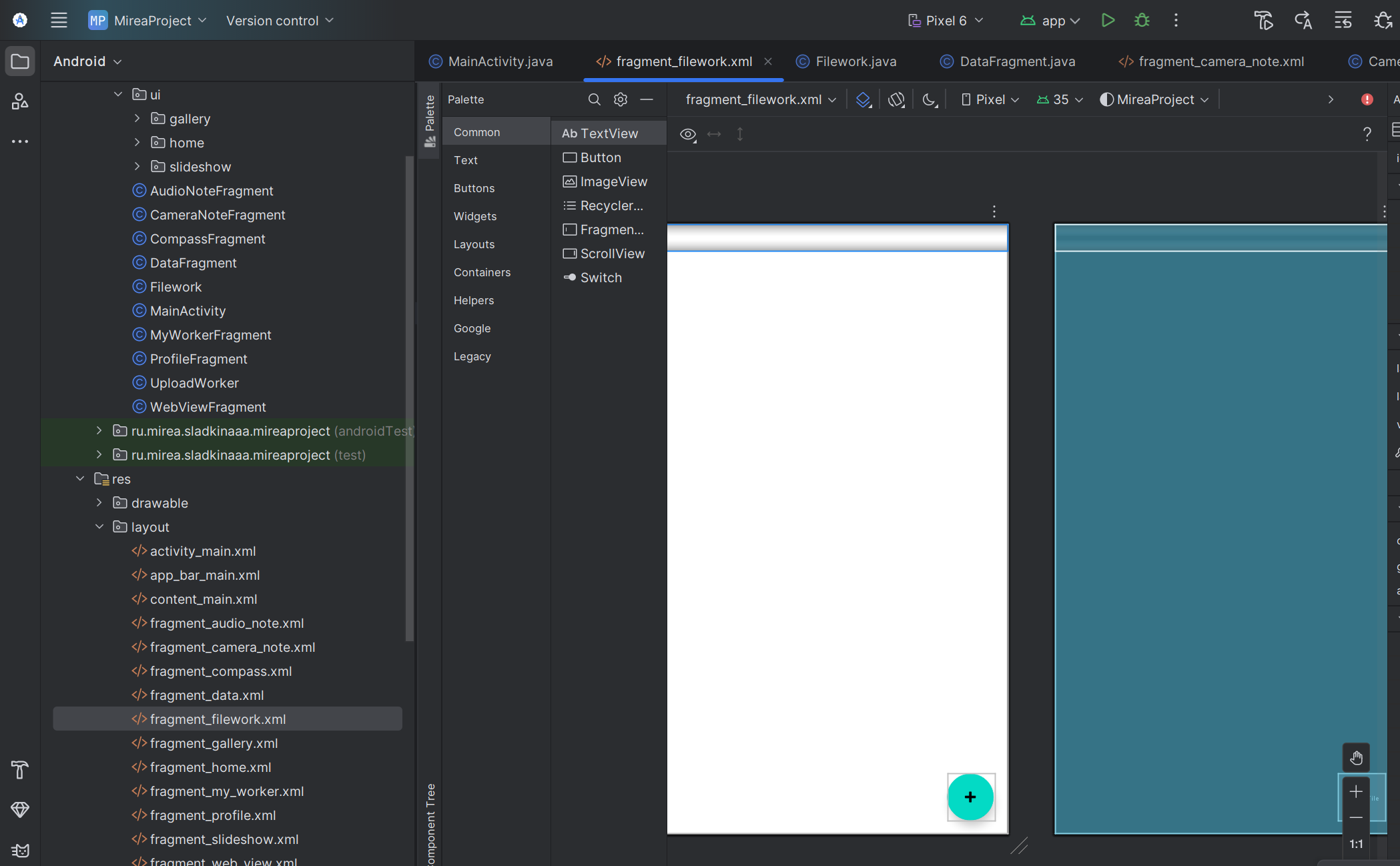


Рисунок 18. Дизайн фрагмента ProfileFragment

Фрагмент FileWorkFragment предоставляет простой интерфейс для работы с текстовыми файлами, закодированными в формате Base64 (см. рис. 19). При создании фрагмента происходит инициализация контейнера LinearLayout, в который динамически добавляются TextView для каждого файла, находящегося во внутреннем хранилище приложения. Эти текстовые поля отображают имена файлов, а при нажатии на них — содержимое файлов, расшифрованное из Base64. Кнопка FloatingActionButton вызывает диалоговое окно FileDialogFragment, где пользователь может ввести текст и имя файла; после подтверждения введённый текст шифруется с помощью Base64 и сохраняется во внутреннюю память в файл с заданным именем. Метод saveEncryptedFile отвечает за кодирование текста и запись в файл, а readDecryptedFile — за чтение содержимого файла и его расшифровку. Каждый раз после сохранения нового файла вызывается loadFiles, чтобы обновить список отображаемых файлов на экране.



Рисунок 19. Реализация фрагмента FileWorkFragment

Напоследок рассмотрим реализацию диалогового окна FileDialogFragment, которое используется для создания новой текстовой записи и последующего её шифрования и сохранения в файл (см. рис. 20). Диалог наследуется от DialogFragment и при создании отображает два поля ввода: одно для имени файла, другое для текста, который нужно сохранить. Эти поля добавляются в вертикальный LinearLayout, который затем передаётся в качестве содержимого диалога. Когда пользователь нажимает кнопку «Сохранить», код проверяет, что оба поля не пусты, и передаёт введённые данные через интерфейс OnFileSavedListener в вызывающий фрагмент. Далее эти данные будут зашифрованы и записаны в файл. Таким образом, этот диалог служит промежуточным интерфейсом для сбора информации от пользователя перед сохранением данных.



Рисунок 20. Диалогового окна FileDialogFragment