МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт Информационных технологий

Кафедра Математического обеспечения и стандартизации информационных технологий

Отчет по практической работе №12

по дисциплине «Проектирование и разработка мобильных приложений»

Выполнил:
Студент группы ИКБО-21-23 Лисовский И.В.

Проверил:
Старший преподаватель кафедры МОСИТ Шешуков Л.С.

Москва 2025 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В данной работе мы сосредоточились на реализации двух важных аспектов взаимодействия и обработки данных в Android-приложениях. Во-первых, мы настроили передачу данных между приложениями с помощью провайдера контента (Content Provider) — стандартного механизма Android, позволяющего безопасно и структурировано предоставлять доступ к данным другим приложениям. Во-вторых, мы выполнили преобразование информации в формат JSON, а затем реализовали сохранение этой информации во внешний файл. Далее мы научились читать данные из JSON-файла и заполнять соответствующие поля в приложении. Эти задачи позволили нам укрепить навыки межприложного взаимодействия и обработки структурированных данных.

1 ТЕОРЕТИЧЕСКОЕ ВВЕДЕНИЕ

1.1 Работа с JSON

JSON (JavaScript Object Notation) — это компактный и удобный формат представления данных, основанный на подмножестве синтаксиса JavaScript. Мы используем его для обмена структурированной информацией между клиентом и сервером. JSON широко применяется в веб-разработке, мобильных приложениях и других системах, где важно быстро и понятно передавать данные.

Формат JSON представляет данные в виде пар "ключ-значение", где ключи, как правило, являются строками, а значения могут быть строками, числами, булевыми значениями, массивами, объектами или null. Благодаря своей читаемости и простоте, JSON хорошо воспринимается как человеком, так и машиной, что делает его удобным стандартом для передачи данных.

В процессе Android-разработки мы часто используем JSON для обмена данными между приложением и сервером, а также для локального хранения информации.

Хотя в Android нет встроенных инструментов для работы с JSON, мы можем воспользоваться сторонними библиотеками. Одной из самых популярных является библиотека Gson из пакета com.google.code.gson.

Чтобы подключить её к проекту и упростить работу с JSON-объектами, необходимо добавить зависимость Gson в файл build.gradle нашего проекта.

На рисунке 1 показано добавление необходимой зависимости в Gradle.

Рисунок 1 – Добавление зависимости в Gradle

Теперь рассмотрим несколько примеров работы с JSON и классом User, который содержит три поля: имя, возраст и электронную почту. Создаём объект из JSON-строки (десериализация) и покажем это на рисунке 2.

```
public void parseJsonUsingGson() {
    String jsonStr = "{\"name\":\"John\", \"age\":30, \"email\":\"john@example.com\"}";
    Gson gson = new Gson();
    User user = gson.fromJson(jsonStr, User.class);

    System.out.println("Name: " + user.name);
    System.out.println("Age: " + user.age);
    System.out.println("Email: " + user.email);
}
```

Рисунок 2 – Десериализация

В этом примере (рисунок 2) мы создаём объект Gson, после чего вызываем метод fromJson(), чтобы преобразовать JSON-строку в объект класса User. Мы передаём в метод строку с сериализованными данными и указываем класс, в который нужно выполнить преобразование. Таким образом, мы легко получаем доступ к значениям из JSON.

Далее преобразуем объект в JSON-строку (сериализация) и покажем это на рисунке 3.

```
public String createJsonUsingGson() {
    User user = new User();
    user.name = "Alice";
    user.age = 25;
    user.email = "alice@example.com";

    Gson gson = new Gson();
    return gson.toJson(user);
}
```

Рисунок 3 – Сериализация

Здесь (рисунок 3) мы вручную создаём объект User и заполняем его данными. С помощью метода toJson() объекта Gson сериализуем его в строку

в формате JSON. Полученный результат можно сохранить, например, в файл или передать по сети.

Далее чтение JSON-массива объектов. Код покажем на рисунке 4.

```
public void parseJsonArrayUsingGson() {
    String jsonArrayStr = "[{\"name\":\"John\", \"age\":30, \"email\":\"john@example.com\"}," +
                          "{\"name\":\"Alice\", \"age\":25, \"email\":\"alice@example.com\"}]";

    Gson gson = new Gson();
    Type userListType = new TypeToken<List<User>>(){}.getType();
    List<User> users = gson.fromJson(jsonArrayStr, userListType);

    for (User user : users) {
        System.out.println("Name: " + user.name);
        System.out.println("Age: " + user.age);
        System.out.println("Email: " + user.email);
    }
}
```

Рисунок 4 – Чтение массива объектов

В этом случае (рисунок 4) мы работаем с JSON-массивом, содержащим несколько объектов User. Чтобы корректно его обработать, мы определяем тип List<User> с помощью TypeToken. Затем десериализуем строку и перебираем полученные объекты в цикле, выводя информацию о каждом пользователе.

Таким образом, используя библиотеку Gson, мы можем эффективно выполнять как сериализацию, так и десериализацию как одиночных объектов, так и коллекций, что делает работу с JSON в Android простой и удобной.

1.2 Провайдеры контента

Провайдеры контента (Content Providers) в Android — это компоненты, которые мы можем использовать для безопасного обмена данными между приложениями. С их помощью мы инкапсулируем данные внутри приложения и предоставляем унифицированный механизм доступа, соблюдая при этом контроль над тем, какие данные и кому открываются. Это особенно важно, когда несколько приложений нуждаются в доступе к одной и той же информации или когда одно приложение должно делиться данными с другим.

Провайдеры контента дают нам несколько ключевых преимуществ:

1) безопасность доступа: Мы можем предоставлять другим приложениям только нужные данные, скрывая внутреннюю структуру базы,

2) унифицированный интерфейс: Мы получаем стандартный механизм работы с данными, который упрощает интеграцию между разными приложениями,

3) удобное управление данными: С помощью методов query(), insert(), update(), delete() мы централизованно обрабатываем все операции,

4) интеграция с системой: Провайдеры легко взаимодействуют с такими компонентами, как загрузчики (Loaders) или система глобального поиска.

Допустим, у нас есть приложение, работающее с базой данных SQLite, в которой хранятся книги пользователя. Мы хотим разрешить другим приложениям обращаться к этим данным.

Шаг 1. Определим URI для доступа (рисунок 5).

```
public static final Uri CONTENT_URI = Uri.parse("content://com.example.app.provider/books");
```

Рисунок 5 – Определение URI

Шаг 2. Реализуем контент-провайдер (рисунок 6).

```
public class BookProvider extends ContentProvider {
    private SQLiteDatabase db;

    @Override
    public boolean onCreate() {
        DBHelper dbHelper = new DBHelper(getContext());
        db = dbHelper.getWritableDatabase();
        return (db != null);
    }

    @Override
    public Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder) {
        return db.query("books", projection, selection, selectionArgs, null, null, sortOrder);
    }

    @Override
    public Uri insert(Uri uri, ContentValues values) {
        long rowID = db.insert("books", "", values);
        Uri _uri = ContentUris.withAppendedId(CONTENT_URI, rowID);
        getContext().getContentResolver().notifyChange(_uri, null);
        return _uri;
    }

    @Override
    public int delete(Uri uri, String selection, String[] selectionArgs) {
        int count = db.delete("books", selection, selectionArgs);
        getContext().getContentResolver().notifyChange(uri, null);
        return count;
    }

    @Override
    public int update(Uri uri, ContentValues values, String selection, String[] selectionArgs) {
        int count = db.update("books", values, selection, selectionArgs);
        getContext().getContentResolver().notifyChange(uri, null);
        return count;
    }

    @Override
    public String getType(Uri uri) {
        return "vnd.android.cursor.dir/vnd.example.books";
    }
}
```

Рисунок 6 – Контент-провайдер

Шаг 3. Добавим провайдер в AndroidManifest.xml (рисунок 7).

```
<provider
    android:name=".BookProvider"
    android:authorities="com.example.app.provider"
    android:exported="true"
    android:permission="android.permission.READ_PROVIDER" />
```

Рисунок 7 – Добавление провайдера в манифест

Шаг 4. В другом приложении запрашиваем разрешение на чтение (рисунок 8).

```
<uses-permission android:name="android.permission.READ_PROVIDER" />
```

Рисунок 8 – Запрос на чтение

Шаг 5. Получаем данные о книгах из внешнего приложения (рисунок 9).

```
Uri contentUri = Uri.parse("content://com.example.app.provider/books");
ContentResolver resolver = getContentResolver();
Cursor cursor = resolver.query(contentUri, new String[] {"_id", "title", "author"}, null, null, "title ASC");

if (cursor != null) {
    try {
        int idColumn = cursor.getColumnIndex("_id");
        int titleColumn = cursor.getColumnIndex("title");
        int authorColumn = cursor.getColumnIndex("author");

        while (cursor.moveToNext()) {
            int id = cursor.getInt(idColumn);
            String title = cursor.getString(titleColumn);
            String author = cursor.getString(authorColumn);

            System.out.println("Book ID: " + id);
            System.out.println("Book Title: " + title);
            System.out.println("Author: " + author);
        }
    } finally {
        cursor.close(); // обязательно закрываем курсор
    }
}
```

Рисунок 9 – Получение данных о книгах

Таким образом, мы предоставили доступ к данным из SQLite-базы одного приложения другим приложениям через стандартизированный механизм Content Provider. Это позволяет централизованно управлять безопасностью, форматом данных и политикой доступа, не раскрывая внутреннюю реализацию базы.

2 ПРАКТИЧЕСКАЯ ЧАСТЬ

2.1 Реализация передачи данных в другое приложение через провайдер контента

На рисунке 10 показан код класса UserProvider.java.

```
public class UserProvider extends ContentProvider {
    static {
        uriMatcher.addURI(AUTHORITY,  path: "users", USERS);
    }

    4 usages
    private MatrixCursor cursor;

    @Override
    public boolean onCreate() {
        cursor = new MatrixCursor(new String[]{"id", "name"});
        cursor.addRow(new Object[]{1, "Alice"});
        cursor.addRow(new Object[]{2, "Bob"});
        return true;
    }

    @Nullable
    @Override
    public Cursor query(@NonNull Uri uri, @Nullable String[] projection,
                        @Nullable String selection, @Nullable String[] selectionArgs,
                        @Nullable String sortOrder) {
        if (uriMatcher.match(uri) == USERS) {
            return cursor;
        }
        throw new IllegalArgumentException("Unknown URI " + uri);
    }

    // Остальные методы не реализуем (insert/update/delete)
    @Nullable @Override public String getType(@NonNull Uri uri) { return null; }
    no usages
    @Nullable @Override public Uri insert(@NonNull Uri uri, @Nullable ContentValues values) { return null; }
    @Override public int delete(@NonNull Uri uri, @Nullable String selection, @Nullable String[] selectionArgs) { return 0; }
    no usages
    @Override public int update(@NonNull Uri uri, @Nullable ContentValues values, @Nullable String selection, @Nullable String[] selectionArgs) { return 0; }
}
```

Рисунок 10 – Класс UserProvider

На рисунке 11 показаны изменения в AndroidManifest.

```
<provider
    android:name=".UserProvider"
    android:authorities="com.example.provider.users"
    android:exported="true" />
```

Рисунок 11 – Изменения в AndroidManifest

На рисунке 12 показан код MainActivity из второго приложения.

```
public class MainActivity extends AppCompatActivity {

    4 usages
    TextView resultText;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        resultText = new TextView( context: this);
        setContentView(resultText);

        Uri uri = Uri.parse( uriString: "content://com.example.provider.users/users");
        Cursor cursor = getContentResolver().query(uri,  projection: null,  selection: null,  selectionArgs: null,  sortOrder: null);

        if (cursor != null) {
            StringBuilder builder = new StringBuilder();
            while (cursor.moveToNext()) {
                int id = cursor.getInt(cursor.getColumnIndexOrThrow( columnName: "id"));
                String name = cursor.getString(cursor.getColumnIndexOrThrow( columnName: "name"));
                builder.append("ID: ").append(id).append(" Имя: ").append(name).append("\n");
            }
            cursor.close();
            resultText.setText(builder.toString());
        } else {
            resultText.setText("Нет доступа к провайдеру");
        }
    }
}
```

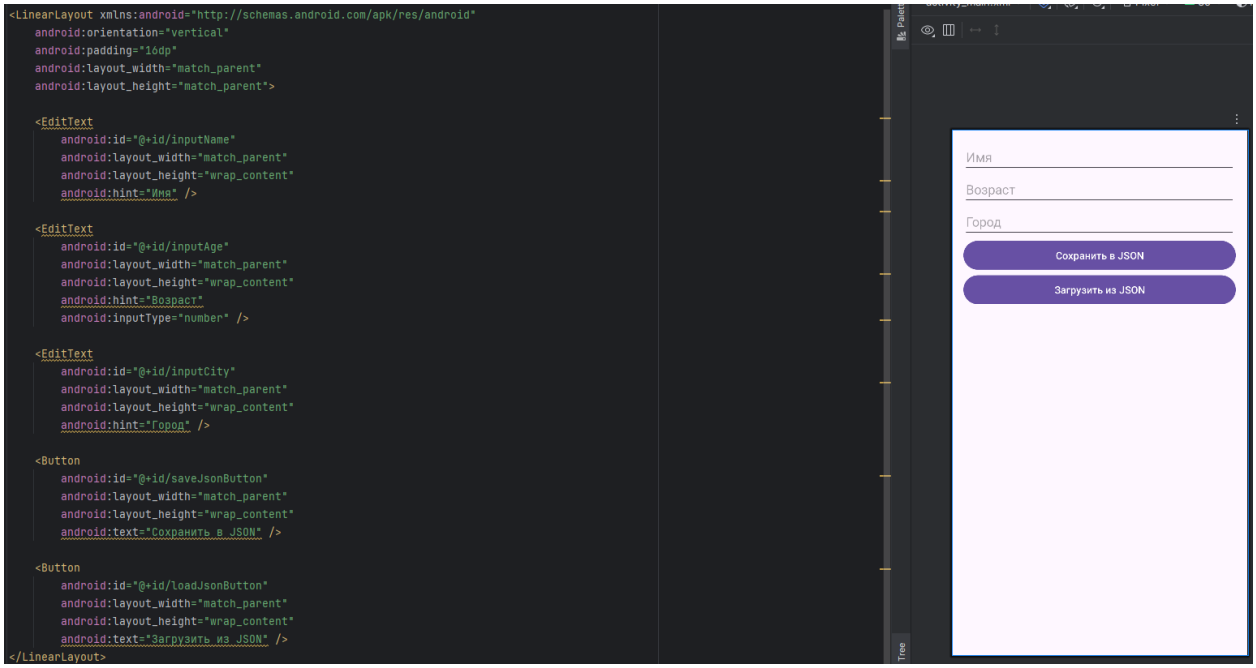
Рисунок 12 – MainActivity второго приложения

На рисунке 13 показан результат работы приложения.

Рисунок 13 – Работоспособность приложения

2.2 Преобразование данных в JSON и сохранение в отдельный файл, преобразование из JSON в различные поля

На рисунке 14 покажем разметку для activity_main.xml.




```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:padding="16dp"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <EditText
        android:id="@+id/inputName"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Имя" />

    <EditText
        android:id="@+id/inputAge"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Возраст"
        android:inputType="number" />

    <EditText
        android:id="@+id/inputCity"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Город" />

    <Button
        android:id="@+id/saveJsonButton"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Сохранить в JSON" />

    <Button
        android:id="@+id/loadJsonButton"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Загрузить из JSON" />
</LinearLayout>
```

Рисунок 14 – Разметка activity_main.xml

На рисунке 15 покажем первую часть MainActivity.java.

```java
public class MainActivity extends AppCompatActivity {
    3 usages
    EditText inputName, inputAge, inputCity;
    2 usages
    Button saveJsonButton, loadJsonButton;
    2 usages
    private static final String FILE_NAME = "userdata.json";

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

        inputName = findViewById(R.id.inputName);
        inputAge = findViewById(R.id.inputAge);
        inputCity = findViewById(R.id.inputCity);
        saveJsonButton = findViewById(R.id.saveJsonButton);
        loadJsonButton = findViewById(R.id.loadJsonButton);

        saveJsonButton.setOnClickListener( View v -> saveToJson());
        loadJsonButton.setOnClickListener( View v -> loadFromJson());
    }

    1 usage
    private void saveToJson() {
        try {
            JSONObject jsonObject = new JSONObject();
            jsonObject.put( name: "name", inputName.getText().toString());
            jsonObject.put( name: "age", inputAge.getText().toString());
            jsonObject.put( name: "city", inputCity.getText().toString());

            try (FileOutputStream fos = openFileOutput(FILE_NAME, MODE_PRIVATE)) {
                fos.write(jsonObject.toString().getBytes());
                Toast.makeText( context: this,  text: "Сохранено в JSON", Toast.LENGTH_SHORT).show();
            }
        } catch (Exception e) {
            Toast.makeText( context: this,  text: "Ошибка сохранения", Toast.LENGTH_SHORT).show();
            e.printStackTrace();
        }
    }
```

Рисунок 15 – MainActivity ч.1

На рисунке 16 покажем вторую часть класса MainActivity.

```
    1 usage
    private void loadFromJson() {
        try (FileInputStream fis = openFileInput(FILE_NAME)) {
            BufferedReader reader = new BufferedReader(new InputStreamReader(fis));
            StringBuilder result = new StringBuilder();
            String line;

            while ((line = reader.readLine()) != null) {
                result.append(line);
            }

            JSONObject jsonObject = new JSONObject(result.toString());

            inputName.setText(jsonObject.getString( name: "name"));
            inputAge.setText(jsonObject.getString( name: "age"));
            inputCity.setText(jsonObject.getString( name: "city"));

            Toast.makeText( context: this,  text: "Загружено из JSON", Toast.LENGTH_SHORT).show();
        } catch (Exception e) {
            Toast.makeText( context: this,  text: "Ошибка загрузки", Toast.LENGTH_SHORT).show();
            e.printStackTrace();
        }
    }
}
```

Рисунок 16 – MainActivity ч.2

На рисунке 17 покажем работоспособность приложения.

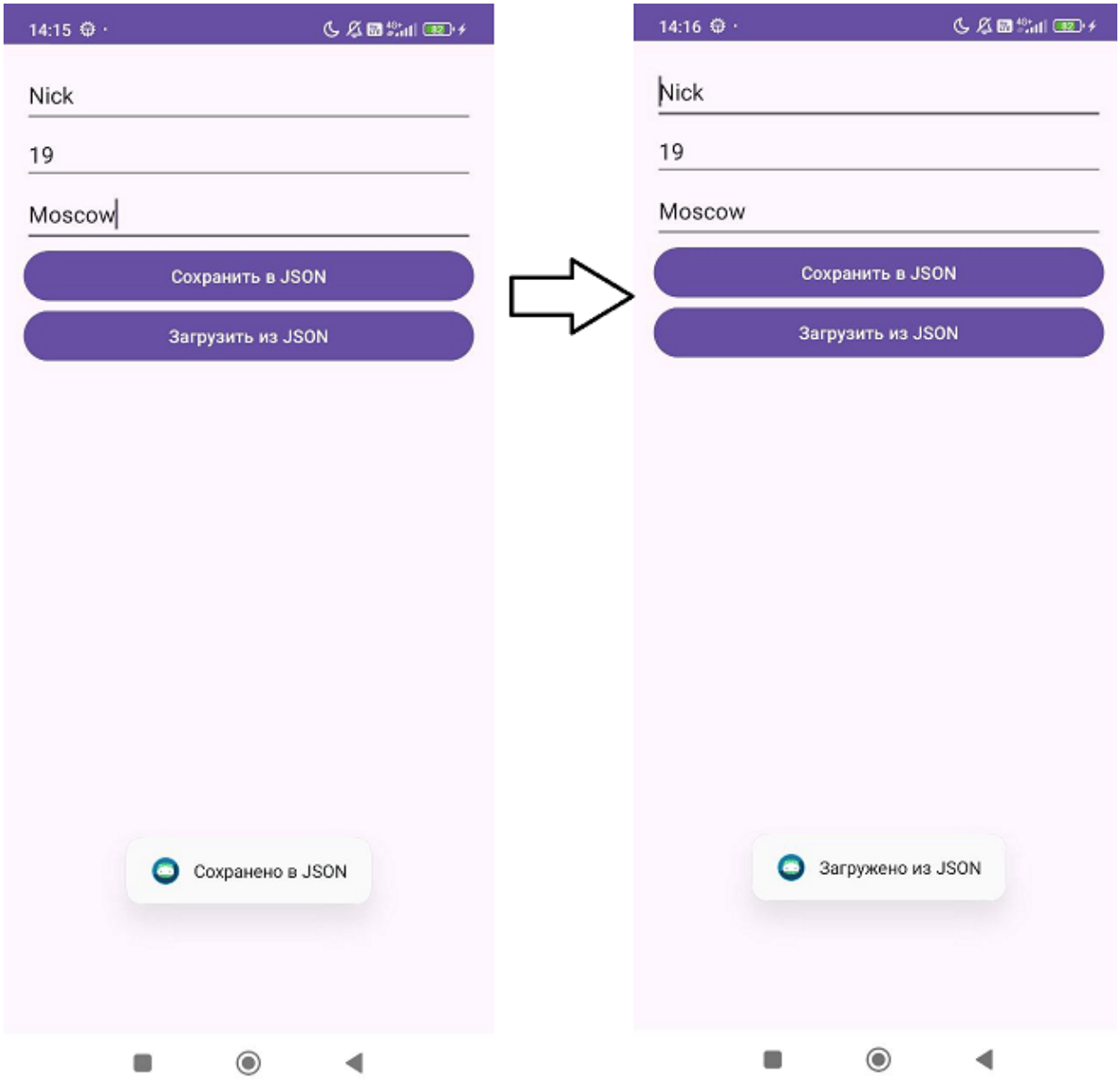


Рисунок 17 – Работоспособность приложения

ЗАКЛЮЧЕНИЕ

В результате выполнения работы мы реализовали полноценный обмен данными между приложениями на Android с помощью контент-провайдера. Мы научились создавать URI, управлять запросами и настраивать безопасный доступ к базе данных. Помимо этого, мы освоили сериализацию объектов в формат JSON с использованием библиотеки Gson, сохранили данные во внешний файл, а затем успешно выполнили обратную операцию — десериализацию данных и отображение их в пользовательском интерфейсе. Таким образом, мы закрепили знания, необходимые для построения надёжных, гибких и взаимодействующих между собой Android-приложений.