# ПРАКТИЧЕСКАЯ РАБОТА № 12

## 12.1. ЧТО ТАКОЕ API

API (Application Programming Interface) — это программный интерфейс, обеспечивающий взаимодействие между приложениями и внешними сервисами. Веб-API представляет собой интерактивный набор команд, предоставляемых разработчиками службы, для доступа к определенным функциям и возможностям их программного обеспечения. Термин "интерфейс" указывает на то, что в хорошем API должны быть ясные и понятные команды, которые облегчают взаимодействие с ним для разработчиков.

## 12.2. КАК РАБОТАЕТ ВЕБ-API

Большинство API используют форматы данных, такие как XML или JSON, для обмена информацией. Эти языки позволяют нам эффективно отправлять и получать большие объемы данных в структурированном формате, используя объекты.

XML (Extensible Markup Language) — это язык разметки, который используется для описания структуры данных в текстовом формате.

```
<client>
<name>Иван</name>
<age>23</age>
</client>
```

JSON (JavaScript Object Notation) — это формат представления данных, который был изначально разработан для использования в JavaScript, но стал широко применяемым и в других языках программирования. JSON представляет собой компактный текстовый формат, который удобен для передачи данных по сети. Подобно XML или CSV-файлам, JSON может использоваться для передачи пар ключ-значение или атрибутов в структурированном формате. Он часто применяется в веб-разработке для обмена данными между клиентскими и серверными приложениями.

Однако в нашем случае синтаксис выглядит немного по-другому:

```
[{client: {“name”:”Иван”,”age”:23}}]
```

Да, в контексте JSON, "объекты данных" представляют собой концептуальные сущности (например, люди), которые могут быть описаны в виде пар ключ-значение. В разработке приложений для Android, JSON-данные часто преобразуются в объекты с использованием классов, чтобы обеспечить удобное взаимодействие с этими данными. Классы позволяют определить структуру и поведение объектов, основываясь на информации, полученной из JSON. Это позволяет эффективно работать с данными и выполнять различные операции и манипуляции в приложении.

## 12.3. RETROFIT

Retrofit — это REST клиент для Java и Android. Он позволяет легко получить и загрузить JSON (или другие структурированные данные) через веб-сервис на основе REST. REST (Representational State Transfer) - это архитектурный стиль, используемый при разработке распределенных систем. Он определяет набор принципов и ограничений для проектирования веб-сервисов. REST основывается на концепции ресурсов, представленных через уникальные идентификаторы (URL), и взаимодействия с ними с помощью стандартных методов HTTP, таких как GET, POST, PUT и DELETE.

RESTful API (или просто REST API) - это веб-сервис, который следует принципам и ограничениям REST. Он предоставляет доступ к ресурсам и операциям над ними через стандартные HTTP-методы. RESTful API использует различные форматы данных для представления ресурсов, такие как JSON или XML. Он позволяет клиентским приложениям взаимодействовать с сервером, выполнять операции чтения, создания, обновления и удаления данных.

В Retrofit вы настраиваете, какой конвертер используется для сериализации данных. Обычно для JSON используется GSon, но вы можете добавлять собственные конвертеры для обработки XML или других протоколов. В Retrofit используется библиотека OkHttp для HTTP-запросов.

Во-первых, нам нужно добавить разрешение для доступа в Интернет в наш файл манифеста Android, чтобы убедиться, что нашему приложению разрешено выходить в Интернет. Для того чтобы включить разрешение на доступ в Интернет в файле манифеста Android, вам необходимо добавить следующую строку в блок <manifest>:

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

Нам также необходимо добавить зависимость, если мы собираемся заставить Retrofit 2 работать в нашем приложении. Для того чтобы добавить зависимость Retrofit 2 в ваш проект Android, вам нужно отредактировать файл build.gradle на уровне модуля. Необходимо добавить следующую строку в блок <dependencies>:

```
implementation 'com.squareup.retrofit2:retrofit:2.9.0'
```

Также нам понадобится библиотека Gson - это библиотека для работы с JSON в языке Java. Она предоставляет удобные методы для преобразования объектов Java в формат JSON и обратно. Gson позволяет сериализовать (преобразовывать объекты Java в JSON) и десериализовать (преобразовывать JSON в объекты Java) данные.

Gson автоматически выполняет преобразование между полями объектов Java и их эквивалентными JSON-представлениями. Он может обрабатывать различные типы данных, включая примитивы, массивы, коллекции, объекты и вложенные структуры данных. Gson также обладает гибкими настройками и возможностью адаптации для обработки особых случаев.

Благодаря Gson, разработчикам легче работать с JSON в Java-приложениях, особенно при обмене данными с веб-серверами или другими сервисами, которые используют JSON для представления информации.

Мы также добавляем зависимость в блок <dependencies>:

```
implementation 'com.squareup.retrofit2:converter-gson:2.9.0'
```

### 12.4. Преобразование JSON в объект Java

При работе с API принято употреблять термин «Route», он представляется URL-адресом и означает конечную точку API, по которой можно произвести запрос данных. Для примера рассмотрим такой URL (пример URL) для JSON запроса. При его изучении можно заметить наличие так называемого пути «/posts». Данный путь является одним из Route’ов нашего API.

Для рассмотрения примера перейдем по следующему URL: https://jsonplaceholder.typicode.com/posts. Там вы увидите большой объем данных в формате JSON. Это фиктивный текст, который имитирует внешний вид страницы, полной сообщений в социальных сетях. Это информация,

которую мы хотим получить из нашего приложения и затем отобразить на экране.

Чтобы обработать эту информацию, нам потребуется создать класс, который будет создавать объекты на основе десериализованных данных. Для этого добавьте новый класс в ваш проект с названием "PlaceholderPost". В этом классе необходимо объявить переменные, соответствующие данным, получаемым со страницы /posts, такие как "body" и "ID". Поскольку мы получаем эту информацию из веб-API, для каждой переменной нам потребуется создать геттер.

```
public class PlaceholderPost {

    private int userID;
    private int id;
    private String title;
    private String body;

    public int getUserId() {
        return userID;
    }
    public int getId() {
        return id;
    }
    public String getTitle() {
        return title;
    }
    public String getBody() {
        return body;
    }

}
```

Здесь мы можем получать информацию абсолютно с любого сайта, необходимо только задать нужные поля для получения данных.

## 12.5. ФАЙЛЫ ИНТЕРФЕЙСА

Затем нам потребуется создать новый файл интерфейса. Чтобы создать его, в окне проекта вы выбираете имя вашего пакета, щелкаете правой кнопкой мыши и выбираете «New > Class». Однако, вместо класса вам нужно выбрать «Interface» в списке опций, которые появятся при вводе имени.

В этом интерфейсе нам нужен только один метод для извлечения всех данных из "/Post". Если вы еще раз взглянете на этот JSON, вы заметите, что фигурные скобки находятся внутри квадратных скобок. Это означает, что у нас есть массив объектов, поэтому мы хотим создать список для них. В этом списке мы будем хранить экземпляры нашего только что созданного "PlaceholderPost" объекта.

```
import java.util.List;
import retrofit2.Call;
import retrofit2.http.GET;

public interface PlaceholderAPI {

    @GET("posts")
    Call<List> getPosts();

}
```

### 12.6. Показ содержимого

Вместо того, чтобы создавать сложный макет для отображения всех этих данных, на первый раз рекомендуется оставить макет без изменений, чтобы сохранить его простоту и эстетический вид.

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

public class RetrofitFactory {

    private static Retrofit retrofit = null;

    private RetrofitFactory () {} // class should not be
initialized

    public static Retrofit getRetrofit(String baseUrl) {
        if(retrofit == null) {
            retrofit = new Retrofit.Builder()
                    .baseUrl(baseUrl)

.addConverterFactory(GsonConverterFactory.create())
                    .build();
        }
```

```
        return retrofit;
    }
}
```

Как можно заметить, мы передаем в этом случае оставшуюся часть URL-адреса. После этого мы сможем использовать наш интерфейс для взаимодействия с API.

```
Call&lt;List&gt; call = placeholderAPI.getPosts();
```

Теперь нам осталось только вызвать метод в классе MainActivity! Однако, Android не позволяет нам делать это в основном потоке, так как это может привести к зависанию приложения, если процесс займет слишком много времени. Это особенно важно при использовании веб-API. Хотя это имеет свою логику, оно не очень удобно, когда мы просто хотим создать учебный пример. Но к счастью, нам не нужно создавать второй поток вручную, поскольку Retrofit на самом деле выполняет все это за нас.

Теперь мы получим обратные вызовы `onResponse` и `onFailure`. Метод `onFailure` предназначен для обработки любых ошибок, которые могут возникнуть.

Однако метод `onResponse` не гарантирует, что все прошло гладко. Он просто указывает на наличие ответа и на то, что веб-сайт существует. Например, даже если мы получим ответ с кодом 404, это всё равно будет считаться "ответом". Поэтому нам необходимо дополнительно проверить, прошел ли процесс успешно, с помощью метода `isSuccessful()`, который проверяет, является ли код HTTP ошибкой.

Для простоты планируется отобразить только один фрагмент данных от одного из полученных объектов. Для этого переименуем `textView` в файле макета, присвоив ему идентификатор "текст". Вы можете самостоятельно провести эксперименты с этим. Ниже представлен полный код:

```
import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.util.Log;
import android.widget.TextView;

import java.util.List;
```

```
import retrofit2.Call;
import retrofit2.Callback;
import retrofit2.Response;
import retrofit2.Retrofit;

public class MainActivity extends AppCompatActivity {

    public final String URL_API =
"https://jsonplaceholder.typicode.com/";

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

     private void getDataFromApi () {
        Retrofit retrofit =
RetrofitFactory.getRetrofit(URL_API);
        PlaceholderAPI placeholderAPI =
retrofit.create(PlaceholderAPI.class);
        Call&lt;List&gt; call = placeholderAPI.getPosts();

         call.enqueue(new Callback&lt;List&gt;() {
            @Override
            public void onResponse(Call&lt;List&gt; call,
Response&lt;List&gt; response) {
                if (response.isSuccessful()) {
                    List posts = response.body();
                    Log.d("Success",
posts.get(3).getBody().toString());
                    TextView textView =
findViewById(R.id.text);
          textView.setText(posts.get(3).getBody().toString());
                } else {
                    Log.d("Ей", "Bay!");
                    return;
                }
            }
            @Override
            public void onFailure(Call&lt;List&gt; call,
Throwable t) {
```

```
                Log.d("Ей", "Ошибка!!");
            }
        });
        Log.d("Эй","Привет!");
    }
}
```

**Задание:**

1. Реализовать, как минимум три запроса данных с сайта согласно предметной области. Запросы должны включать как минимум один GET и один POST.