

RETROFIT. ПРИКЛАД ВИКОРИСТАННЯ ДЛЯ ANDROID

Детальніше:

<http://square.github.io/retrofit/>,

<http://square.github.io/okhttp/>,

<http://square.github.io/retrofit/2.x/retrofit/>

Retrofit — HTTP-клієнт з контролем типів для Android/Java.

За замовчуванням Retrofit може тільки десеріалізувати HTTP тіла у OkHttp тип `ResponseBody` і може приймати тільки його тип `RequestBody` для `@Body`.

Для підтримки інших типів можуть бути використані конвертери. Популярні бібліотеки для серіалізації адаптовано наступними спорідненими модулями:

- [Gson](#): `com.squareup.retrofit2:converter-gson`
- [Jackson](#): `com.squareup.retrofit2:converter-jackson`
- [Moshi](#): `com.squareup.retrofit2:converter-moshi`
- [Protobuf](#): `com.squareup.retrofit2:converter-protobuf`
- [Wire](#): `com.squareup.retrofit2:converter-wire`
- [Simple XML](#): `com.squareup.retrofit2:converter-simplexml`
- Scalars (primitives, boxed, and String): `com.squareup.retrofit2:converter-scalars`.

Retrofit адаптує інтерфейси Java до HTTP запитів за допомогою анотацій над оголошеними методами. Анотації визначають запити, які потрібно виконати.

Екземпляри створюють за допомогою [Builder](#), для генерування імплементації інтерфейси передають у `create(java.lang.Class<T>)`.

Виклик `baseUrl` повинен бути здійснений перед `build()`. Інші методи — опційні.

У наведеному нижче Android-прикладі використано клас `GsonConverterFactory` для генерування імплементації деякого інтерфейсу `RestApi`.

Додайте у `build.gradle` проекту у `dependencies` наступні рядки (або завантажте бібліотеки окремо і під'єднайте їх до проекту). Версії бібліотек — на момент написання цього матеріалу.

```
dependencies {  
    ...  
    compile 'com.google.code.gson:gson:2.6.1'  
    compile 'com.squareup.retrofit2:retrofit:2.0.0-beta4'  
    compile 'com.squareup.okhttp3:okhttp:3.1.2'  
    compile 'com.squareup.retrofit2:converter-gson:2.0.0-beta4'  
}
```

Опишемо за допомогою класу модель даних, які сервер віддаватиме у форматі JSON у відповідь на запит до нього. Для цього потрібно знати, як виглядає рядок у форматі JSON. Нехай це буде простий об'єкт з одним полем: `{"data":1}`. [Детальніше про JSON](#).

Клас можна написати уручну або скористатися проектом <https://github.com/joelittlejohn/jsonschema2pojo/>. Використаємо [онлайн конвертер](#) (рис. 1).

Вкажіть налаштування у відповідності до рисунку або задайте власні. У текстове поле зліва вставте рядок у форматі JSON, для якого потрібно створити модель. У полі `Package` задайте назву пакету, що міститиме клас моделі. У даному прикладі зроблено окремий пакет `POJO` (Plain Old Java Object). Коли кількість таких класів буде значною, зручно помістити їх в окремий пакет. Хоча це і не обов'язково. Назвемо клас `Model` і задамо це ім'я у полі конвертера `Class Name`. У випадковій ієрархії з кількох вкладених один в одного об'єктів у форматі JSON `Class Name` позначатиме ім'я класу верхнього рівня ієрархії.

Додайте до AndroidManifest.xml дозвіл на використання мережі:

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

jsonschema2pojo

Generate Plain Old Java Objects from JSON or JSON-Schema.

1 { "data":1 }

Package

Class name

Source type:
☐ JSON Schema ☒ JSON

Annotation style:
☐ Jackson 2.x ☐ Jackson 1.x
☒ Gson ☐ None

☐ Generate builder methods
☐ Use primitive types
☐ Use long integers
☒ Use double numbers
☐ Use Joda dates
☐ Use Commons-Lang3
☒ Include getters and setters
☐ Include constructors
☐ Include hashCode and equals
☐ Include toString
☐ Include JSR-303 annotations
☐ Allow additional properties

Property word delimiters:

Рисунок 1 — Онлайн конвертер jsonschema2pojo

Натисніть кнопку Preview і отримайте клас, який відповідає даному JSON (рис. 2). Скопіюйте його вміст і додайте до проекту у пакет POJO. Видаліть рядки

```
import javax.annotation.Generated; та @Generated("org.jsonschema2pojo")
```

При цьому файл Model.java матиме наступний вигляд.

```
package POJO;
```

```

import com.google.gson.annotations.Expose;
import com.google.gson.annotations.SerializedName;
public class Model {
    @SerializedName("data")
    @Expose
    private Integer data;

    public Integer getData() {
        return data;
    }

    public void setData(Integer data) {
        this.data = data;
    }
}

```

Preview

```

-----POJO.Model.java-----

package POJO;

import javax.annotation.Generated;
import com.google.gson.annotations.Expose;
import com.google.gson.annotations.SerializedName;

@Generated("org.jsonschema2pojo")
public class Model {
    @SerializedName("data")
    @Expose
    private Integer data;

    /**
     *
     * @return
     * The data
     */
    public Integer getData() {
        return data;
    }

    /**
     *
     * @param data
     * The data
     */
    public void setData(Integer data) {
        this.data = data;
    }
}

```

Рисунок 2 — Java-модель даних у форматі JSON

Відповідь сервера у даному прикладі генерує простий код на php (файл retrofit.php).

```

<?php
echo "{\"data\":123456789}";

```

Створіть файл з вказаним іменем у кореневій теці (DocumentRoot) сервера. Retrofit використовує для опису запитів до сервера анотації над методами інтерфейсів. Додайте до

проекту інтерфейс, назвіть його, наприклад, RestApi. Зверніть увагу на пакет проекту: com.example.student.retrofit — змініть його за потреби на власний. Анотація до методу інтерфейсу описуватиме запит до сервера.

```
package com.example.student.retrofit;
import POJO.Model;
import retrofit2.Call;
import retrofit2.http.GET;

public interface RestApi {
    @GET("retrofit.php")
    Call<Model> getData();
}
```

Дізнайтеся IP-адресу сервера за допомогою ipconfig (Windows) / ifconfig (Linux). Нехай ця адреса для даного прикладу буде 192.168.1.103.

Додайте до layout (content_main.xml) текстове поле:

```
<TextView
    android:id="@+id/data"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/tvText" />
```

та до /values/strings.xml текстовий рядок ("No data" відображатиметься за відсутності даних):

```
<resources>
    <string name="app_name">Retrofit</string>
    <string name="action_settings">Settings</string>
    <string name="tvText">No data</string>
</resources>
```

Додайте у класі MainActivity наступне:

```
import POJO.Model;
import retrofit2.Call;
import retrofit2.Callback;
import retrofit2.Response;
import retrofit2.Retrofit;
import retrofit2.converter.gson.GsonConverterFactory;
```

Поле класу, що відповідає текстовому полю layout та рядок для URL сервера:

```
String url = "http://192.168.1.103/";  
TextView textViewData;
```

У кінці onCreate() додайте наступне:

```
textViewData = (TextView)findViewById(R.id.data);  
getData();
```

Додайте до класу реалізацію методу getData(). У прикладі виконано асинхронний запит до сервера.

```
void getData(){  
    Retrofit retrofit = new Retrofit.Builder()  
        .baseUrl(url)  
        .addConverterFactory(GsonConverterFactory.create())  
        .build();  
    RestApi service = retrofit.create(RestApi.class);  
    Call<Model> call = service.getData();  
    call.enqueue(new Callback<Model>() {  
        @Override  
        public void onResponse(Call<Model> call, Response<Model> response) {  
            String data = String.valueOf(response.body().getData());  
            textViewData.setText("data : " + data);  
        }  
        @Override  
        public void onFailure(Call<Model> call, Throwable t) {  
        }  
    });  
}
```

Скомпілюйте проект. Результат наведено на рис. 3.

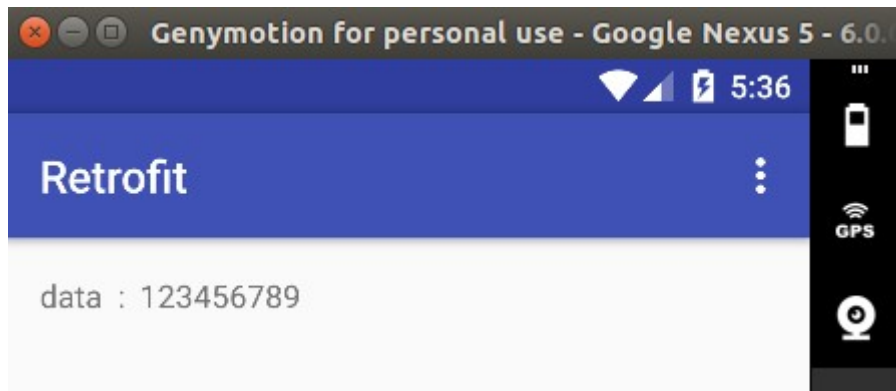


Рисунок 3 — Результат виконання запиту до сервера за допомогою бібліотеки Retrofit

Маніпулювання URL

URL запиту можна оновлювати динамічно, використовуючи блоки підстановок та параметри методів. Блок підстановки — буквочисловий рядок, обмежений дужками { та }. Відповідний параметр повинен бути анотований за допомогою @Path та того самого рядка.

Створіть у DocumentRoot веб-сервера (чи за іншою довільною адресою) теки user1 та user2. Помістіть у них файли retrofit.php з наступним вмістом до теки user1

```
<?php
echo "{\"data\":\"1\"}";
```

та до теки user2:

```
<?php
echo "{\"data\":\"2\"}";
```

Завдання: отримати можливість динамічної зміни адреси за допомогою блоку підстановок. У даному випадкові це адреси /user1/retrofit.php та /user2/retrofit.php.

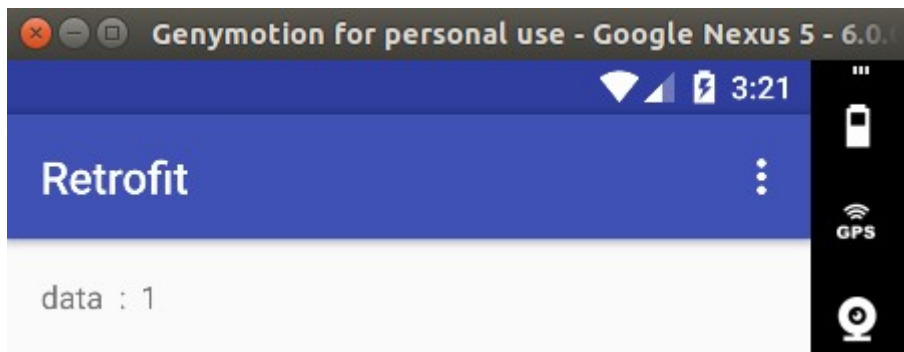
Модифікуємо метод інтерфейсу RestApi та анотацію над ним:

```
package com.example.student.retrofit;
import POJO.Model;
import retrofit2.Call;
import retrofit2.http.GET;
import retrofit2.http.Path;
public interface RestApi {
    @GET("{id}/retrofit.php")
    Call<Model> getData(@Path("id") String id);
}
```

У MainActivity.java модифікуйте рядок із записом запиту до сервера у call, скомпілюйте проект.

a) = service.getData("user1"); б) = service.getData("user2");

Результати наведено на рис 4.



а)



б)

Рисунок 4 — Динамічне модифікування URL

Query-параметри

Нехай, потрібно реалізувати GET-запит типу



Рисунок 5 — Параметри запиту

Параметри запиту також можна задати за допомогою анотації над методом інтерфейсу. Модифікуємо інтерфейс (RestApi.java):

```

package com.example.student.retrofit;
import POJO.Model;
import retrofit2.Call;
import retrofit2.http.GET;
import retrofit2.http.Path;
import retrofit2.http.Query;

public interface RestApi {
    @GET("{id}/retro.php")
    Call<Model> getData(@Path("id") String id, @Query("sort") String sort);
}

```

Параметр запиту реалізують за допомогою анотації `@Query()`. Модифікуємо роботу серверної частини (`/user1/retrofit.php`) так, щоб сервер повертав параметр, переданий у рядковій запиті:

```

<?php
    $sort = $_GET['sort'];
    echo '{"data":1, "sort":"'.$sort.'"}';

```

Також слід модифікувати клас моделі (`Model.java`), оскільки змінено формат рядка JSON:

```

package POJO;
import com.google.gson.annotations.Expose;
import com.google.gson.annotations.SerializedName;
public class Model {
    @SerializedName("data")
    @Expose
    private Integer data;
    @SerializedName("sort")
    @Expose
    private String sort;
    public Integer getData() { return data; }
    public void setData(Integer data) { this.data = data; }
    public String getSort() { return sort; }
    public void setSort(String sort) { this.sort = sort; }
}

```

Вносимо зміни до методу `getData()` класу `Activity`:

```

void getData(){

```



```

Retrofit retrofit = new Retrofit.Builder()
    .baseUrl(url)
    .addConverterFactory(GsonConverterFactory.create())
    .build();
RestApi service = retrofit.create(RestApi.class);
Call<Model> call = service.getData("user1", "asc");
call.enqueue(new Callback<Model>() {
    @Override
    public void onResponse(Call<Model> call, Response<Model> response) {
        String data = String.valueOf(response.body().getData());
        String sort = String.valueOf(response.body().getSort());
        textViewData.setText("data : " + data + ", " + sort);
    }
    @Override
    public void onFailure(Call<Model> call, Throwable t) {
    }
});
}

```

Таким чином передають параметр запиту за допомогою методу Get. Результат:

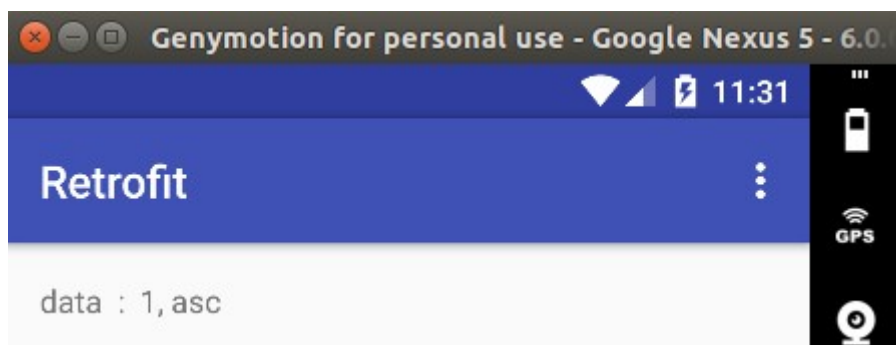


Рисунок 6 — Результат передачі параметра Get-запиту

Складені параметри запиту

Для складених параметрів запиту можна скористатися анотацією `@QueryMap`. Нехай потрібно реалізувати запит типу



Рисунок 7 — Параметри складеного запиту

У даному випадкові нехай тип параметру `limit` буде `String` (хоча він і представлятиме число). Змінимо анотацію над методом інтерфейсу:

```

package com.example.student.retrofit;
import java.util.Map;

import POJO.Model;
import retrofit2.Call;
import retrofit2.http.GET;
import retrofit2.http.Path;
import retrofit2.http.QueryMap;

public interface RestApi {
    @GET("{id}/retro.php")
    Call<Model> getData(@Path("id") String id, @QueryMap Map<String, String> options);
}

```

Замість явного задання параметрів запиту використано `@QueryMap Map<String, String> options`.

Модифікуємо модель, додавши кілька параметрів (direction, limit):

```

package POJO;
import com.google.gson.annotations.Expose;
import com.google.gson.annotations.SerializedName;

public class Model {
    @SerializedName("data")
    @Expose
    private Integer data;
    @SerializedName("sort")
    @Expose
    private String sort;
    @SerializedName("direction")
    @Expose
    private String direction;
    @SerializedName("limit")
    @Expose
    private String limit;
    public String getDirection() {
        return direction;
    }
    public void setDirection(String direction) {
        this.direction = direction;
    }
    public String getLimit() {
        return limit;
    }
    public void setLimit(String limit) {
        this.limit = limit;
    }
    public Integer getData() {

```

```

        return data;
    }
    public void setData(Integer data) {
        this.data = data;
    }
    public String getSort() {
        return sort;
    }
    public void setSort(String sort) {
        this.sort = sort;
    }
}

```

Також потрібно змінити серверну частину (/user1/retrofit.php):

```

<?php
echo '{"data":1';
foreach($_GET as $key => $value)
    echo ', "'.$key.'": "'.$value.'"';
echo '}';

```

MainActivity.java:

```

void getData(){
    Retrofit retrofit = new Retrofit.Builder()
        .baseUrl(url)
        .addConverterFactory(GsonConverterFactory.create())
        .build();
    RestApi service = retrofit.create(RestApi.class);
    Map map = new HashMap();
    map.put("sort", "id");
    map.put("direction", "asc");
    map.put("limit", "10");
    Call<Model> call = service.getData("user1", map);
    call.enqueue(new Callback<Model>() {
        @Override
        public void onResponse(Call<Model> call, Response<Model> response) {
            String data = String.valueOf(response.body().getData());
            String sort = String.valueOf(response.body().getSort());
            String direction = String.valueOf(response.body().getDirection());
            String limit = String.valueOf(response.body().getLimit());
            textViewData.setText("data : " + data + ", " + sort +
                ", " + direction + ", " + limit);
        }
    })
}

```

```
@Override
public void onFailure(Call<Model> call, Throwable t) {
}
});
}
```

Тіло запиту (HTTP Body)

Об'єкт може бути використано як тіло HTTP запиту за допомогою анотації `@Body`.

Створіть два класи: один — для запиту, інший — для відповіді сервера (обидва — у пакеті POJO, хоча це не принципово). Клас запиту: `Login.java`:

```
package POJO;
import com.google.gson.annotations.Expose;
import com.google.gson.annotations.SerializedName;
public class Login {
    @SerializedName("login")
    @Expose
    private String login;
    @SerializedName("pass")
    @Expose
    private String pass; // open password! ;)
    public Login(String login, String pass) {
        this.login = login;
        this.pass = pass;
    }
    public String getLogin() {
        return login;
    }
    public void setLogin(String login) {
        this.login = login;
    }
    public String getPass() {
        return pass;
    }
    public void setPass(String pass) {
        this.pass = pass;
    }
}
```

Клас відповіді (файл `HttpResponse.java`, пакет: POJO):

```
package POJO;
import com.google.gson.annotations.Expose;
import com.google.gson.annotations.SerializedName;
```

```

public class HttpResponse {
    @SerializedName("access")
    @Expose
    private String access;
    public String getAccess() {
        return access;
    }
    public void String(String access) {
        this.access = access;
    }
}

```

Створіть інтерфейс для запиту:

```

package com.example.student.retrofit;
import POJO.HttpResponse;
import POJO.Login;
import retrofit2.Call;
import retrofit2.http.Body;
import retrofit2.http.POST;
public interface RestApi {
    @POST("retrofit.php")
    Call<HttpResponse> sendData(@Body Login login);
}

```

Зверніть увагу на те, що як параметр анотованої функції використано об'єкт класу запиту. Додаткові параметри - відсутні. Об'єкт буде серіалізовано і сервер отримає запит у JSON-форматі.

Для отримання доступу до серіалізованих даних POST-запиту необхідно десеріалізувати їх та застосувати cast до Object, перетворивши його в array (файл retrofit.php):

```

<?php
if ($_SERVER['REQUEST_METHOD'] == 'POST')
{
    $data = json_decode(file_get_contents("php://input"));
    $data_decoded = (array)$data;
    echo '{"access":"","'.$data_decoded["login"]."' }';
}

```

Залишається модифікувати метод getData():

```

void getData(){
    Retrofit retrofit = new Retrofit.Builder()
        .baseUrl(url)

```

```

        .addConverterFactory(GsonConverterFactory.create())
        .build();
RestApi service = retrofit.create(RestApi.class);
Call<HttpResponse> call = service.sendData(new Login("Foo", "Bar"));
call.enqueue(new Callback<HttpResponse>() {
    @Override
    public void onResponse(Call<HttpResponse> call, Response<HttpResponse> response){
        String data = response.body().getAccess();
        textViewData.setText("data : " + data);
    }
    @Override
    public void onFailure(Call<HttpResponse> call, Throwable t) {
    }
});
}

```

Результат:



Рисунок 8 — Результат передачі POST-запиту із JSON-параметрами

Form Encoded

Також за допомогою Retrofit можна оголошувати методи для надсилання form-encoded та multipart даних. Form-encoded дані надсилаються, якщо метод анотовано за допомогою `@FormUrlEncoded`. Кожну пару ключ-значення анотують за допомогою `@Field`, що містить ім'я та об'єкт, котрий містить значення

Оголосіть інтерфейс

```

package com.example.student.retrofit;
import POJO.HttpResponse;
import retrofit2.Call;
import retrofit2.http.Field;
import retrofit2.http.FormUrlEncoded;
import retrofit2.http.POST;

```

```
public interface RestApi {
    @FormUrlEncoded
    @POST("retrofit.php")
    Call<HttpResponse> updateUser(@Field("first_name") String first,
                                @Field("last_name") String last);
}
```

Створіть клас для отримання відповіді від сервера.

```
package POJO;
import com.google.gson.annotations.Expose;
import com.google.gson.annotations.SerializedName;
public class HttpResponse {
    @SerializedName("access")
    @Expose
    private String access;
    public String getAccess() {
        return access;
    }
    public void String(String access) {
        this.access = access;
    }
}
```

Відповідь сервера:

```
<?php
    $last_name = $_POST['last_name'];
    echo '{"access":"","'.$last_name.'"}';
```

На відміну від попереднього прикладу тут параметри POST-запиту отримують аналогічно, як і у випадковій GET. Модифікуємо метод `getData()` для отримання відповіді сервера.

```
void getData(){
    Retrofit retrofit = new Retrofit.Builder()
        .baseUrl(url)
        .addConverterFactory(GsonConverterFactory.create())
        .build();
    RestApi service = retrofit.create(RestApi.class);
    Call<HttpResponse> call = service.updateUser("Foo", "Bar");
    call.enqueue(new Callback<HttpResponse>() {
        @Override
        public void onResponse(Call<HttpResponse> call, Response<HttpResponse> response)
```

```

    {
        String data = response.body().getAccess();
        textViewData.setText("data : " + data);
    }
    @Override
    public void onFailure(Call<HttpResponse> call, Throwable t) {
    }
    });
}

```

Результат:



Рисунок 9 — Результат передачі POST-запиту

FormUrlEncoded може бути визначено тільки для HTTP методів із тілом запиту (@POST).

Multipart

HTTP multipart запити — це HTTP запити, котрі HTTP клієнти конструюють для надсилання файлів та даних на HTTP-сервери. Зазвичай використовуються браузерами та HTTP-клієнтами для завантаження файлів на сервер.

Для здійснення multipart запиту використовують анотацію методів @Multipart. Частини (parts) оголошують за допомогою анотації @Part. Частини використовують для серіалізації Multipart один із конвертерів Retrofit або імплементують RequestBody для управління власною серіалізацією.

Завдання: надіслати на сервер за допомогою бібліотеки Retrofit довільний файл, наприклад, зображення jpg.

Оголосіть інтерфейс з відповідною анотацією для надсилання запиту:

```

package com.example.student.retrofit;
import POJO.HttpResponse;
import okhttp3.RequestBody;
import retrofit2.Call;
import retrofit2.http.Multipart;
import retrofit2.http.POST;
import retrofit2.http.Part;

```



```
public interface RestApi {
    @Multipart
    @POST ("retrofit.php")
    Call<HttpResponse> uploadData (@Part("file\"; filename=\"image.jpg\" ") RequestBody
    file);
}
```

Зверніть увагу на дещо незвичний формат параметру file. [RFC2388](#) описує заголовок filename як параметр запиту multipart/form-data, що використовує Retrofit. Цей параметр вимагають деякі серверні конфігурації. [Детальніше](#). Отже, потрібен спосіб вказати ім'я файла, що передають. Проблема полягає у RequestBody, котрий є частиною OkHttp і не відображає імені файла при створенні на основі File.

Станом на момент виходу Retrofit v2.0.0-b4 немає можливості вказати серверові ім'я файла для передачі, використавши ім'я файла-носія даних. Деяким виходом із ситуації є використання ін'єкції параметра filename зі статичним значенням (тобто таким, котре не може бути змінено динамічно). При цьому усі передані серверові файли матимуть одне і те ж ім'я. Додатково вказати ім'я можна за допомогою ще однієї частини, наприклад:

```
@Part("fname") String fname
```

Таким чином, сам файл буде передано із іменем-заглушкою, проте а справжнє ім'я сервер отримає як окрему частину multipart-запиту.

Оголосіть клас для отримання відповіді сервера:

```
package POJO;
import com.google.gson.annotations.Expose;
import com.google.gson.annotations.SerializedName;
public class HttpResponse {
    @SerializedName("status")
    @Expose
    private String status;
    public String getStatus() {
        return status;
    }
    public void setStatus(String status) {
        this.status = status;
    }
}
```

Додайте до маніфесту дозвіл на доступ до носія даних:

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

<application....

Модифікуємо метод getData():

```
void getData() {
    Retrofit retrofit = new Retrofit.Builder()
        .baseUrl(url)
        .addConverterFactory(GsonConverterFactory.create())
        .build();
    RestApi service = retrofit.create(RestApi.class);
    File sdcard = Environment.getExternalStorageDirectory();
    File file = new File(sdcard + File.separator + "test.jpg");
    RequestBody fbody = RequestBody.create(MediaType.parse("image/jpg"), file);
    Call<HttpResponse> call = service.uploadData(fbody);
    call.enqueue(new Callback<HttpResponse>() {
        @Override
        public void onResponse(Call<HttpResponse> call, Response<HttpResponse> response)
        {
            String data = response.body().getStatus();
            textViewData.setText("data : " + data);
        }
        @Override
        public void onFailure(Call<HttpResponse> call, Throwable t) {
        }
    });
}
```

Серверна частина:

```
<?php
if(isset($_FILES['file']))
{
    $errors= array();
    $file_name = $_FILES['file']['name'];
    $file_size = $_FILES['file']['size'];
    $file_tmp = $_FILES['file']['tmp_name'];
    $file_type=$_FILES['file']['type'];
    $value = explode(".", $file_name);
    $file_ext = strtolower(array_pop($value));
    $extensions= array("jpeg","jpg","png","gif");
    if(in_array($file_ext,$extensions)== false)
    {
        $errors="Extension not allowed, please choose a JPEG or PNG file.";
    }
    if($file_size > 2097152)
    {
```

```

        $errors[]='File size must be exactly 2 MB';
    }
    if(empty($errors)==true)
    {
        move_uploaded_file($file_tmp,"images/".$file_name);
        //echo "Success";
        echo '{"status" : "ok"}';
    }else
    {
        //print_r($errors);
        echo '{"status" : "failed"}';
    }
}

```

Для того, щоб перевірити роботу сервера, можна додати до коду форму для надсилання multipart-запиту і поміняти місцями закоментовані та розкоментовані рядки виведення у кінці скрипта:

Увага! Обидва php-скрипти — навчальні і **не** надають потрібного рівня безпеки (наприклад, контролю MIME). Призначені тільки для демонстрації технології.

```

<?php
if(isset($_FILES['file']))
{
    $errors= array();
    $file_name = $_FILES['file']['name'];
    $file_size = $_FILES['file']['size'];
    $file_tmp = $_FILES['file']['tmp_name'];
    $file_type=$_FILES['file']['type'];
    $value = explode(".", $file_name);
    $file_ext = strtolower(array_pop($value));
    $extensions= array("jpeg","jpg","png","gif");
    if(in_array($file_ext,$extensions)== false)
    {
        $errors="Extension not allowed, please choose a JPEG or PNG file.";
    }
    if($file_size > 2097152)
    {
        $errors[]='File size must be exactly 2 MB';
    }
    if(empty($errors)==true)
    {
        move_uploaded_file($file_tmp,"images/".$file_name);
        echo "Success";
        //echo '{"status" : "ok"}';
    }else
    {
        print_r($errors);
    }
}

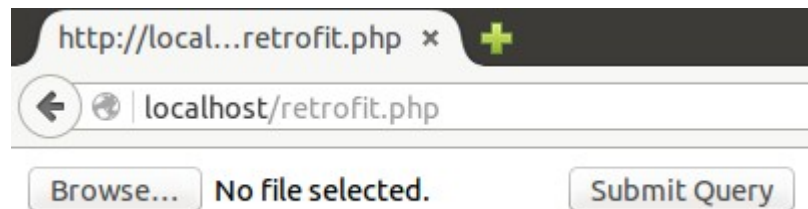
```

```

        //echo '{"status" : "failed"}';
    }
}
?>
<html>
  <body>
    <form action="" method="POST" enctype="multipart/form-data">
      <input type="file" name="file" />
      <input type="submit"/>
    </form>
  </body>
</html>

```

Створіть теку `images` там же, де і файл `retrofit.php`. Завантажені файли будуть скопійовані до неї.



Результат успішного завантаження файла повинен бути наступним:

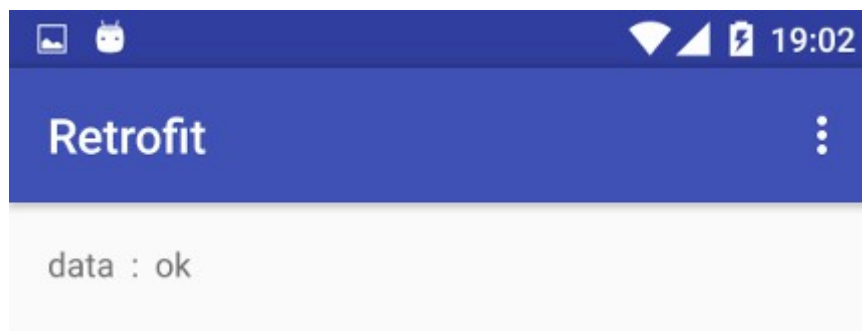


Рисунок 10 — Результат успішного завантаження файла

У теці `images` повинен з'явитися переданий за допомогою Retrofit файл.

Проблему передачі справжнього імені файла при завантаженні на сервер легко вирішити за допомогою анотації `@PartMap` над параметром типу `Map`. Змініть інтерфейс:

```

package com.example.student.retrofit;
import java.util.Map;
import POJO.HttpResponse;
import okhttp3.RequestBody;

```

```

import retrofit2.Call;
import retrofit2.http.Multipart;
import retrofit2.http.POST;
import retrofit2.http.PartMap;
public interface RestApi {
    @Multipart
    @POST ("retrofit.php")
    Call<HttpResponse> uploadData (@PartMap Map<String, RequestBody> pic);

```

Внесіть відповідні зміни у метод getData:

```

void getData() {
    Retrofit retrofit = new Retrofit.Builder()
        .baseUrl(url)
        .addConverterFactory(GsonConverterFactory.create())
        .build();
    RestApi service = retrofit.create(RestApi.class);
    File sdcard = Environment.getExternalStorageDirectory();
    File file = new File(sdcard + File.separator + "test.jpg");
    RequestBody fbody = RequestBody.create(MediaType.parse("image/jpg"), file);
    Map<String, RequestBody> picMap = new HashMap<>();
    picMap.put("file"; filename=\"" + file.getName(), fbody);
    Call<HttpResponse> call = service.uploadData(picMap);
    call.enqueue(new Callback<HttpResponse>() {
        @Override
        public void onResponse(Call<HttpResponse> call, Response<HttpResponse> response)
        {
            String data = response.body().getStatus();
            textViewData.setText("data : " + data);
        }
        @Override
        public void onFailure(Call<HttpResponse> call, Throwable t) {
        }
    });
}

```

У цьому випадкові передаються справжні імена файлів.

Додамо до запиту передостаннього прикладу окрему частину, що міститиме справжню назву файла. Модифікуйте інтерфейс:

```

package com.example.vitaliy.retrofit;
import POJO.HttpResponse;
import okhttp3.RequestBody;
import retrofit2.Call;
import retrofit2.http.Multipart;
import retrofit2.http.POST;

```

```
import retrofit2.http.Part;
public interface RestApi {
    @Multipart
    @POST ("retrofit.php")
    Call<HttpResponse> uploadData (@Part("file\"; filename=\"image.jpg\" ") RequestBody
file,
    @Part("truefilename") String truefilename);
}
```

Метод getData:

```
void getData() {
    Retrofit retrofit = new Retrofit.Builder()
        .baseUrl(url)
        .addConverterFactory(GsonConverterFactory.create())
        .build();
    RestApi service = retrofit.create(RestApi.class);
    File sdcard = Environment.getExternalStorageDirectory();
    File file = new File(sdcard + File.separator + "test2.jpg");
    RequestBody fbody = RequestBody.create(MediaType.parse("image/jpg"), file);
    Call<HttpResponse> call = service.uploadData(fbody, file.getName());
    call.enqueue(new Callback<HttpResponse>() {
        @Override
        public void onResponse(Call<HttpResponse> call, Response<HttpResponse> response)
        {
            String data = response.body().getStatus();
            textViewData.setText("data : " + data);
        }
        @Override
        public void onFailure(Call<HttpResponse> call, Throwable t) {
        }
    });
}
```

Серверный код:

```
<?php
if(isset($_FILES['file']))
{
    $errors= array();
    $true_file_name = $_POST['truefilename'];
    $file_name = $_FILES['file']['name'];
    $file_size = $_FILES['file']['size'];
    $file_tmp = $_FILES['file']['tmp_name'];
    $file_type=$_FILES['file']['type'];
    $value = explode(".", $file_name);
}
```

```

$file_ext = strtolower(array_pop($value));
$extensions= array("jpeg","jpg","png","gif");
if(in_array($file_ext,$extensions)== false)
{
    $errors="Extension not allowed, please choose a JPEG or PNG file.";
}
if($file_size > 2097152)
{
    $errors[]='File size must be excately 2 MB';
}
if(empty($errors)==true)
{
    move_uploaded_file($file_tmp,"images/".$true_file_name);
    //echo "Success";
    echo '{"status" : "ok"}';
}else
{
    //print_r($errors);
    echo '{"status" : "failed"}';
}
}

```

<http://httpbin.org/>

It echoes the data used in your request for any of these types:

ENDPOINTS

- [/](#) This page.
- [/ip](#) Returns Origin IP.
- [/user-agent](#) Returns user-agent.
- [/headers](#) Returns header dict.
- [/get](#) Returns GET data.
- [/post](#) Returns POST data.
- [/patch](#) Returns PATCH data.
- [/put](#) Returns PUT data.
- [/delete](#) Returns DELETE data
- [/encoding/utf8](#) Returns page containing UTF-8 data.
- [/gzip](#) Returns gzip-encoded data.
- [/deflate](#) Returns deflate-encoded data.
- [/status/:code](#) Returns given HTTP Status code.
- [/response-headers?key=val](#) Returns given response headers.
- [/redirect/:n](#) 302 Redirects *n* times.
- [/redirect-to?url=foo](#) 302 Redirects to the *foo* URL.

- [/relative-redirect/:n](#) 302 Relative redirects n times.
- [/absolute-redirect/:n](#) 302 Absolute redirects n times.
- [/cookies](#) Returns cookie data.
- [/cookies/set?name=value](#) Sets one or more simple cookies.
- [/cookies/delete?name](#) Deletes one or more simple cookies.
- [/basic-auth/:user/:passwd](#) Challenges HTTPBasic Auth.
- [/hidden-basic-auth/:user/:passwd](#) 404'd BasicAuth.
- [/digest-auth/:qop/:user/:passwd](#) Challenges HTTP Digest Auth.
- [/stream/:n](#) Streams $\min(n, 100)$ lines.
- [/delay/:n](#) Delays responding for $\min(n, 10)$ seconds.
- [/drip?numbytes=n&duration=s&delay=s&code=code](#) Drips data over a duration after an optional initial delay, then (optionally) returns with the given status code.
- [/range/1024?duration=s&chunk_size=code](#) Streams n bytes, and allows specifying a *Range* header to select a subset of the data. Accepts a *chunk_size* and request *duration* parameter.
- [/html](#) Renders an HTML Page.
- [/robots.txt](#) Returns some robots.txt rules.
- [/deny](#) Denied by robots.txt file.
- [/cache](#) Returns 200 unless an If-Modified-Since or If-None-Match header is provided, when it returns a 304.
- [/cache/:n](#) Sets a Cache-Control header for n seconds.
- [/bytes/:n](#) Generates n random bytes of binary data, accepts optional *seed* integer parameter.
- [/stream-bytes/:n](#) Streams n random bytes of binary data, accepts optional *seed* and *chunk_size* integer parameters.
- [/links/:n](#) Returns page containing n HTML links.
- [/image](#) Returns page containing an image based on sent Accept header.
- [/image/png](#) Returns page containing a PNG image.
- [/image/jpeg](#) Returns page containing a JPEG image.
- [/image/webp](#) Returns page containing a WEBP image.
- [/image/svg](#) Returns page containing a SVG image.
- [/forms/post](#) HTML form that submits to */post*
- [/xml](#) Returns some XML