Практическая работа 12 "Paбота с REST API. RETROFIT"



http://mskko2021.mad.hakta.pro/api - начальный URL к API

Продолжите работу с приложением Cinema.

Для корректной работы требуется разрешить приложению доступ к интернету. Добавьте указанную строку кода в файл манифеста.

<uses-permission android:name="android.permission.INTERNET" /> и очистку траффика (строки 4 и 6)

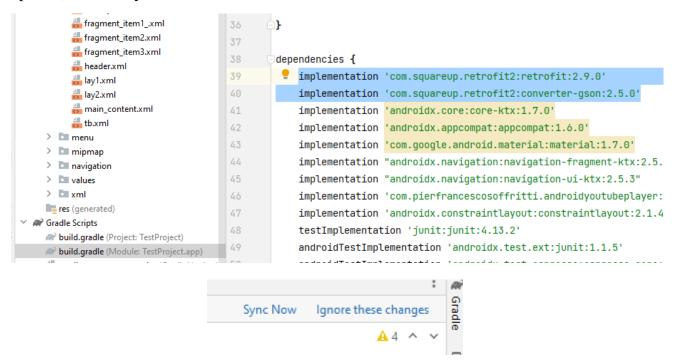
```
📭 арр
                                 1
                                        <?xml version="1.0" encoding="utf-8"?>
  sampledata
> manifests
                                 2
                                        <manifest xmlns:android="http://schemas.android.com/apk/res/android"</pre>
                                 3
                                            xmlns:tools="http://schemas.android.com/tools">
   com.example.testproject
                                 4
                                            <uses-permission android:name="android.permission.INTERNET"/>
       RetrofitServieces
                                 5
     model
                                            <application
       data_movie.kt
                                 6
                                                 android:usesCleartextTraffic="true"
      🥋 adapterh
      🛖 adapterv
                                                 android:allowBackup="true"
      # Datah.kt
```

Для работы с библиотекой Retrofit необходимо подключить в проект следующие зависимости:

implementation 'com.squareup.retrofit2:retrofit:2.9.0' - подключение библиотеки

implementation 'com.squareup.retrofit2:converter-gson:2.5.0' - Конвертер JSON в РОЛО

Добавьте указанные зависимости в файл build.gradle и пересоберите проект, нажав Sync Now:



Библиотека **GSON** была разработана программистами Google и позволяет конвертировать объекты JSON в Java-объекты и наоборот. POJO — это plain old Java object, простой Java-объект, не ограниченный какими-либо запретами, специфичными для того или иного фреймворка (за исключением спецификации самой Java, разумеется) и пригодный для использования в любой среде.

POJO используются для универсальной и наглядной сериализации и десериализации данных. Основное их назначение — это обработка Json в теле запросов и ответов.

Они позволяют достаточно гибко настраивать содержимое Json-объектов, отправляемых в теле запросов — например можно использовать один POJO-класс для составления Json-объектов с разным набором значений, избежав при этом многократного повторения одного и того же кода, или легко создавать Json-объекты с многоуровневыми вложениями, сохраняя при этом простую и наглядную структуру.

При обработке входящих ответов POJO позволяют извлекать любые необходимые значения для дальнейшего использования, а также дают (как уже было сказано выше – простой и наглядный) доступ к вложенным значениям. https://ga.crtweb.ru/docs/testing/auto/api/pojo/

Парсинг (parsing) — это сбор информации из сторонних источников для использования полученных данных в различных целях, от аналитики до копирования. Парсинг - это процесс автоматического сбора данных и их структурирования. Парсинг обычно применяют, когда нужно быстро собрать большой объем данных. Его выполняют с помощью специальных сервисов — парсеров

Для работы с Retrofit понадобятся три класса.

- 1. POJO (Plain Old Java Object) или Model Class json-ответ от сервера нужно реализовать как модель. **Model** это логика, которая связана с данными приложения. Другими словами это POJO, классы работы с API, базой данных.
- 2. Interface интерфейс для управления адресом, используя команды GET, POST и т.д.
- 3. Retrofit класс для обработки результатов. Ему нужно указать базовый адрес в методе **baseUrl()**

По запросу к серверу мы будем получать изображения и цитаты.

Необходимо создать дата класс, с помощью которых будем получать данные с сервера (идентификатор, цитату и изображение). Для этого необходимо посмотреть на структуру ответа в документации API в виде JSON (или других форматов) и создать на его основе дата класс.

Создайте папку и назовите ee model, внутри папки создайте data class, назовите его data_movie. В круглых скобках укажите поля класса, они должны по

названию и типу соответствовать объектам JSON в спецификации:

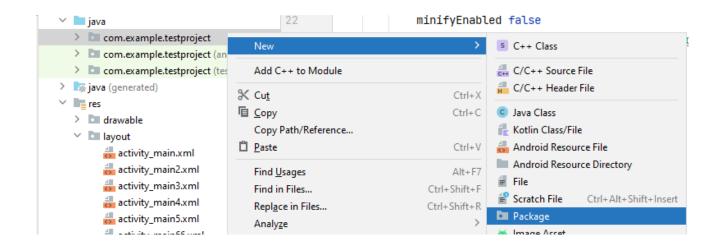
API Url: http://mskko2021.mad.hakta.pro/api API Default Construction:

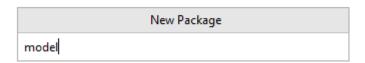
```
{
"success": boolean,
"data": JSONArray/JSONObject,
"debug" : JSONObject
}
```

Method URL Params Description Result GET /ping GET /quotes string \$endpointReturns array of quotes GET /feelings string \$endpointReturns array of quotes POST /user

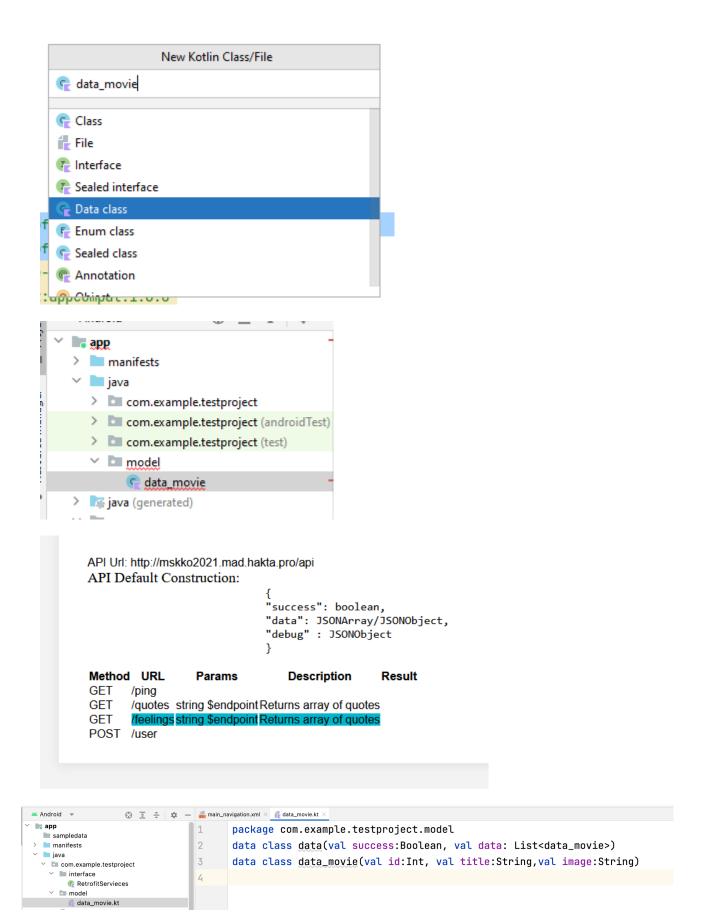
```
Необработанные данные
                                Заголовки
Сохранить Скопировать Свернуть все Развернуть все 

▼ Поиск в JSON
 success:
                     true
▼ 0:
      id:
      title:
                    "Мудрость"
    ▼ image:
                    "http://mskko2021.mad.hakta.pro/upLoads/files/quote 1.png"
      description: "Когда сидишь - ты совсем не лежишь, а сидишь"
  ▼ 1:
      id:
      title:
                     "О вечном"
    ▼ image:
                    "http://mskko2021.mad.hakta.pro/uploads/files/quote_2.png"
      description: "Когда ты думаешь, то время идёт быстрее"
  ▼ 2:
      id:
      title:
                     "Camoe-camoe"
    ▼ image:
                     "http://mskko2021.mad.hakta.pro/uploads/files/quote_2.png"
    ▼ description: "Чем скорее ты закончишь - тем скорее пойдешь поесть"
```





- 🗡 📑 app
 - sampledata
 - > manifests
 - 🗡 📄 java
 - com.example.testproject
 - interface
 - RetrofitServieces
 - ✓ Immodel
 - data_movie.kt
 - 🛖 adapterh
 - 🛖 adapterv
 - Datah.kt
 - 🕝 item1_Fragment
 - 🕝 item2_Fragment



Создайте Арі интерфейс и заполните его запросами.

В интерфейсе задаются команды-запросы для сервера. Команда

комбинируется с базовым адресом сайта/сервера (baseUrl()) и получается полный путь к странице. Код может быть простым и сложным. Запросы размещаются в абстрактном классе Call с указанием желаемого типа. Interface - нужен для создания абстрактных классов.

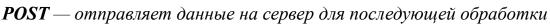
Интерфейсы схожи с абстрактными классами и предоставляют лишь методы без реализации. В интерфейсах можно записать методы, что должны реализовываться во всех классах, использующих интерфейс.

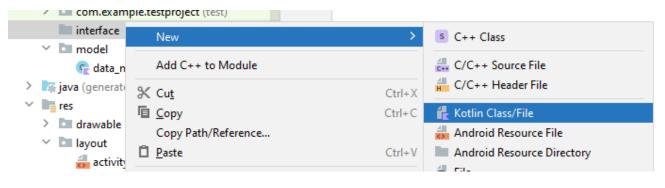
https://swiftbook.ru/post/tutorials/interfaces-and-abstract-classes-in-kotlin/ - про абстрактные классы

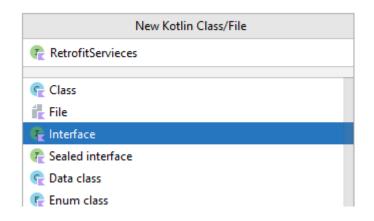
https://itproger.com/course/kotlin/8 - про интерфейсы

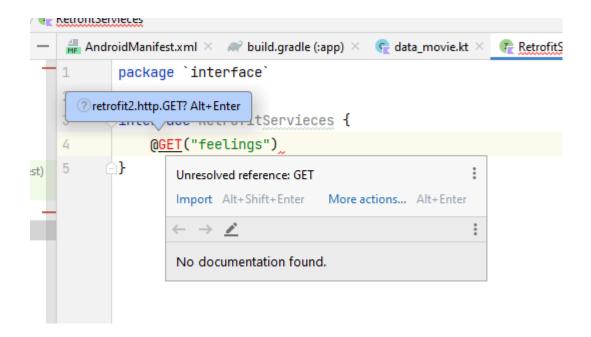
Создайте пакет interface в в нем создайте файл interface, назвав его RetrofitServieces. Пропишите @GET запрос в скобках напишите кавычки, а в кавычках укажите ветку, с которой будут парсить данные - feelings.

GET — запрашивает данные с определенного ресурса(сайта)









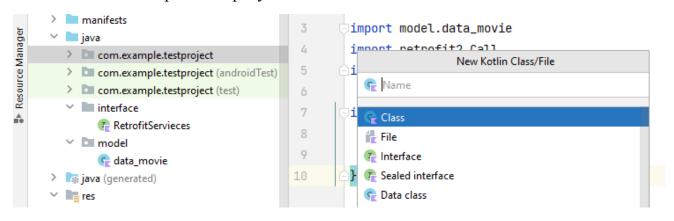
Пропишем функцию getImage, которая должна будет возвращать Call типа data movie (имя созданного ранее дата класса).

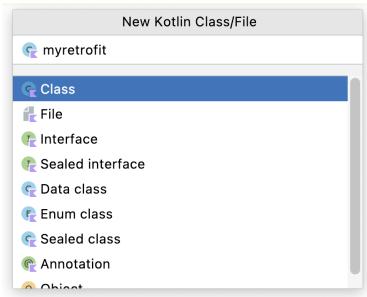
При создании функции обращения к REST API возвращаемым значением необходимо указать класс Call, который сформирует вызов. Обратите внимание при импорте библиотеки необходимо выбрать библиотеку Call (retrofit2) иначе у вас возникнут ошибки.

```
amain_navigation.xml × # data_movie.kt × # RetrofitServieces.kt ×
Android
app
                                            package com.example.testproject.`interface`
                                    1
  sampledata
                                     2
  manifests
  📄 java
                                            import com.example.testproject.model.data
                                     3
   com.example.testproject
   interface
                                            import com.example.testproject.model.data_movie
                                     4
       RetrofitServieces
                                            import retrofit2.Call
    model
                                     5
       # data_movie.kt
                                            import retrofit2.http.GET
                                     6
      adapterh
      🛖 adapterv
                                     7
      Datah.kt
      c item1_Fragment
                                            interface RetrofitServieces {
                                    8
      🛖 item2_Fragment
                                    9
                                                 @GET("feelings")
      🛖 item3_Fragment
      MainActivity
                                   10
                                                 fun getImage():Call<data>
      MainActivity2
      MainActivity3.kt
                                   11
                                            }
      MainActivity4
```

Call<data>, где data - имя дата класса, который вы создаете для преобразования json в ројо

Создайте класс для ретрофита, который в дальнейшем будет использовать в активностях для обработки результатов.





```
RetrofitServieces (interface)
                             c Retrofit (retrofit2)
                             Retention (kotlin.annotation)
                             Returns (kotlin.contracts)
                             RetentionPolicy (java.lang.annotation)
                             RetryAndFollowUpInterceptor (okhttp3.internal.http)
d R<u>u</u>n <u>T</u>ools VC<u>S</u> <u>W</u>indow <u>H</u>elp
                             RetainInstanceUsageViolation (androidx.fragment.app.strictmode)
le > testproject > 👇 myretrofit > 📵 ge
                            ReturnsNotNull (kotlin.contracts)
                             Representation (kotlin.annotation)
 AndroidManifest.xml × R buil
                             RecursiveTask<V> (java.util.concurrent)
        package com.example
                             RecentTaskInfo (android.app.ActivityManager)
                                        ontInancformationMothod (android toxt mothod)
            fun getRetrofit():Ret
 5
```

```
Android

And
```

Объект для запроса к серверу создается в простейшем случае следующим образом

```
public static final String BASE_URL = "http://api.example.com/";

Retrofit retrofit = new Retrofit.Builder()
          .baseUrl(BASE_URL)
          .addConverterFactory(GsonConverterFactory.create())
          .build();
```

В итоге получился объект **Retrofit**, содержащий базовый URL и способность преобразовывать JSON-данные с помощью указанного конвертера Gson. Далее в его методе **create()** указывается класс интерфейса с запросами к серверу, например (в коде методички запись будет отличаться, но смысл останется).

```
UserService userService = retrofit.create(UserService.class);
```

После этого будет получен объект *Call* и вызван метод *enqueue()* (для асинхронного вызова) и создан для него *Callback*. Запрос будет выполнен в *omдельном потоке*, а результат придет в *Callback* в main-nomoke.

В результате библиотека **Retrofit** сделает запрос, получит ответ и производёт разбор ответа, раскладывая по полочкам данные. Вам остаётся только вызывать нужные методы класса-модели для извлечения данных.

Основная часть работы происходит в **onResponse()**, ошибки выводятся в **onFailure()** (неправильный адрес сервера, некорректные формат данных, неправильный формат класса-модели и т.п). HTTP-коды сервера (например, 404) не относятся к ошибкам.

Метод onResponse() вызывается всегда, даже если запрос был неуспешным. Класс Response имеет удобный метод isSuccessful() для успешной обработки запроса (коды 200хх).

Пропишите загрузку фотографий с сервера и цитат в ресайклер с описанием персонажей мультфильма.

Добавьте в проект зависимость для работы с библиотекой GLADE implementation 'com.github.bumptech.glide:glide:4.12.0'

Библиотека **Glide** (аналог библиотека <u>Picasso</u>) предназначена для асинхронной подгрузки изображений из сети, ресурсов или файловой системы, их кэширования и отображения.

https://bumptech.github.io/glide/

https://github.com/bumptech/glide

Удалите дата класс ресайклера (он больше не нужен, данные будут подгружаться по запросу). Измените класс адаптера для вашего ресайклера (обратите внимание, при автозавершении выбирайте варианты с retrofit2, обратите внимание на имена классов, которые используются):

```
package com.example.testproject
    sampledata
    manifests
                                            import android.content.Context
    🖿 java
     com.example.testproject
                                            import android.view.LayoutInflater
     ✓ interface

RetrofitServieces
                                            import android.view.View
                                            import android.view.ViewGroup
      ✓ 🛅 model
         data_movie.kt
                                            import android.widget.ImageView
         adapterh
                                      8
                                            import android.widget.TextView
       adapterv
                                            import androidx.recyclerview.widget.RecyclerView
       🕝 item1_Fragment
                                            import com.bumptech.glide.Glide
        c item2_Fragment
                                            import com.example.testproject.model.data
        tem3_Fragment
                                            import com.example.testproject.model.data_movie
       MainActivity
        MainActivity2
        MainActivity3.kt
                                            class adapterv(val con2: Context, val data: List<data_movie>): RecyclerView.Adapter<adapterv.link2>() {
       MainActivity4
                                                class link2 (vi: View): RecyclerView.ViewHolder(vi){
       MainActivity5
MainActivity66
                                                    val im_dii: ImageView =vi.findViewById(R.id.imageView3)
       MainActivityD
                                                    val ti_dii: TextView =vi.findViewById(R.id.textView4)
     myretrofit
com.example.testproject (androidTest)
                                     18
                                     19 🐠
                                                override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): link2 {
   > com.example.testproject (test)
                                                    val rri= LayoutInflater.from(con2).inflate(R.layout.lay2,parent, attachToRoot: false)
  ∨ 📭 res
                                                    return link2(rri)
     drawable
     menu menu
                                     23
                                                override fun onBindViewHolder(holder: link2, position: Int) {
       amain drawer.xml
                                                    Glide.with(con2).load(data[position].image).into(holder.im_dii)
                                                    holder.ti_dii.setText(data[position].title)
     navigation
26
                   override fun getItemCount(): Int {
27
    28
                          return data.size
29
                   }
30
            }
```

Подключите возможность обращения к ретрофиту в файле логики активити и подкорректируйте подключение адаптера:

```
Android
                      ⊕ 호 🛨 🜣 –
                                       🏭 main_navigation.xml 🗡 🦺 data_movie.kt 🗡 🥷 RetrofitServieces.kt 🗡 🎼 adapterv.kt 🗡 🕞 MainActivity.kt 🗡 🥷 myretrofit.kt
📭 арр
                                        1
                                               package com.example.testproject
   sampledata
                                               import android.content.SharedPreferences
> manifests
                                        3
                                               import androidx.appcompat.app.AppCompatActivity
java
  com.example.testproject
                                        4
                                               import android.os.Bundle
    interface
                                        5
                                               import android.view.View
        RetrofitServieces
                                        6
                                               import android.widget.TextView

✓ Immodel

         # data movie.kt
                                        7
                                               import android.widget.Toast
       ndapterh
                                        8
                                               import androidx.core.content.ContentProviderCompat.requireContext
       🛖 adapterv
                                        9
                                               import androidx.recyclerview.widget.RecyclerView
       Datah.kt
       🛖 item1_Fragment
                                       10
                                               import com.example.testproject.`interface`.RetrofitServieces
       ntem2_Fragment
                                               import com.example.testproject.model.data
       ragment 🚅
                                               import retrofit2.Call
       MainActivity
       MainActivity2
                                       13
                                               import retrofit2.Response
       MainActivity3.kt
                                       14
       MainActivity4
                                       15 🚓
                                               class MainActivity : AppCompatActivity() {
       MainActivity5
       MainActivity66
                                       16
       MainActivityD
                                       17
                                                   lateinit var rv22:RecyclerView
       myretrofit
                                       18
                                                   lateinit var res:TextView
 > com.example.testproject (androidTest)
                                       19 이
                                                   override fun onCreate(savedInstanceState: Bundle?) {
> com.example.testproject (test)
  🗽 java (generated)
                                       20
                                                        super.onCreate(savedInstanceState)
∨ 📭 res
                                                        setContentView(R.layout.activity_main)
  > 🖿 drawable
                                                        rv22=findViewById(R.id.rv2)
  > 🖿 layout
  Y 🖿 menu
                                       23
                                                        val retrofit=myretrofit().getRetrofit()
       amain_drawer.xml
                                       24
                                                        val api_ret=retrofit.create(RetrofitServieces::class.java)
  > 🖿 mipmap
                                       25
                                                        val retr_call:Call<data> = api_ret.getImage()
  navigation
```

```
25
               val retr_call:Call<data> = api_ret.getImage()
               retr_call.enqueue(object:retrofit2.Callback<data>
26
27
28
                   override fun onResponse(call: Call<data>, response: Response<data>) {
                       if(response.isSuccessful)
29
30
31
                           rv22.adapter=response.body()?.let { adapterv(applicationContext,it.data) }
32
                       }
33
                   }
34
                   override fun onFailure(call: Call<data>, t: Throwable) {
35
                       Toast.makeText(applicationContext, t.localizedMessage, Toast.LENGTH_SHORT).show()
36
37
               })
           }}
38
39
40
```

Запустите приложение. Если вы все сделали правильно, вместо кадров и описания персонажей вы увидите следующие изображения и текст:

