Практическая работа 14
“POST запросы”
https://java-help.ru/glide-getting-started/ - про библиотеку Glide
https://developer.alexanderklimov.ru/android/library/retrofit.php
https://habr.com/ru/articles/429058/

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

Продолжите работу с приложением Cinema.

Используя POST - запрос настройте проверку регистрации пользователя на сервере. Если пользователь был ранее зарегистрирован - будет осуществляться переход на следующий экран, если нет - появится сообщение об ошибке. Для работы с api используйте приведенный фрагмент документации:

Название метода	Описание	Принимаемые значения	Ответ
GET /quotes	Возвращает список цитат	-	{ "success": true, "data": [{ "id": 0, "title": "", "image": "", "description": "" },] }
GET /feelings	Возвращает список ощущений	-	{ "success": true, "data": [{ "id": 0, "title": "", "image": "", "position": 0 },] }
POST /user/login	Позволяет получить данные пользователя и token авторизации. Данные отправляются в POST Body. Content-Type: application/ json ***Для тестирования авторизации используйте одну из учетных записей:*** ***Login: junior@wsr.ru*** ***Password: junior*** ***Login: general@wsr.ru*** ***Password: general*** ***Login: wsr*** ***Password: wsr***	{ "email": "", "password":"" }	{ "id": "", "email": "", "nickName": "", "avatar": "", "token": "" }

Работа с POST-запросами аналогична работе с GET - запросами. Создайте необходимые классы для работы: дата класс (POST-запрос пришлет информацию о пользователе, поля дата класса перечислены на изображении), интерфейс, ретрофит (можно использовать тот, что был создан для 12 работы, если продолжаете работать с проектом Cinema).

```
1  package com.example.testproject.model
2
3  data class user_data(val id:String, val email:String, val nickName:String, val avatar:String, val token:String)
4
```

```kotlin
package com.example.testproject.`interface`

import com.example.testproject.model.user_data
import retrofit2.Call
import retrofit2.http.Body
import retrofit2.http.POST

interface UserInterface {
    @POST("user/login") //ветка указана в документации см.изображение
    fun getAuth(@Body hashMap: HashMap<String,String>): Call<user_data> // функция аутентификации пользователя
    // в параметрах функции прописывают, какие данные будут отправляться пост запросом см.документацию
    //@Body используется в POST-вызовах. Чтобы сформировать тело запроса для данного метода, используется аннотация @Body для передаваемого параметра.
    // Retrofit будет использовать Gson для конвертации @Body в JSON.
    //HashMap - карта. Карты очень полезны если нужно быстро найти значения с помощью определенного идентификатора.
// после двоеточия указывается, какой ответ вернет функция в случае успешной обработки запроса
}
```

*Чем карта отличается от массива? Из массива можно получить значение только по его индексу, который должен быть целым числом, и все индексы должны быть последовательными. В карте — **ключи** могут быть любого типа и, как правило, не расположены в определенном порядке.*

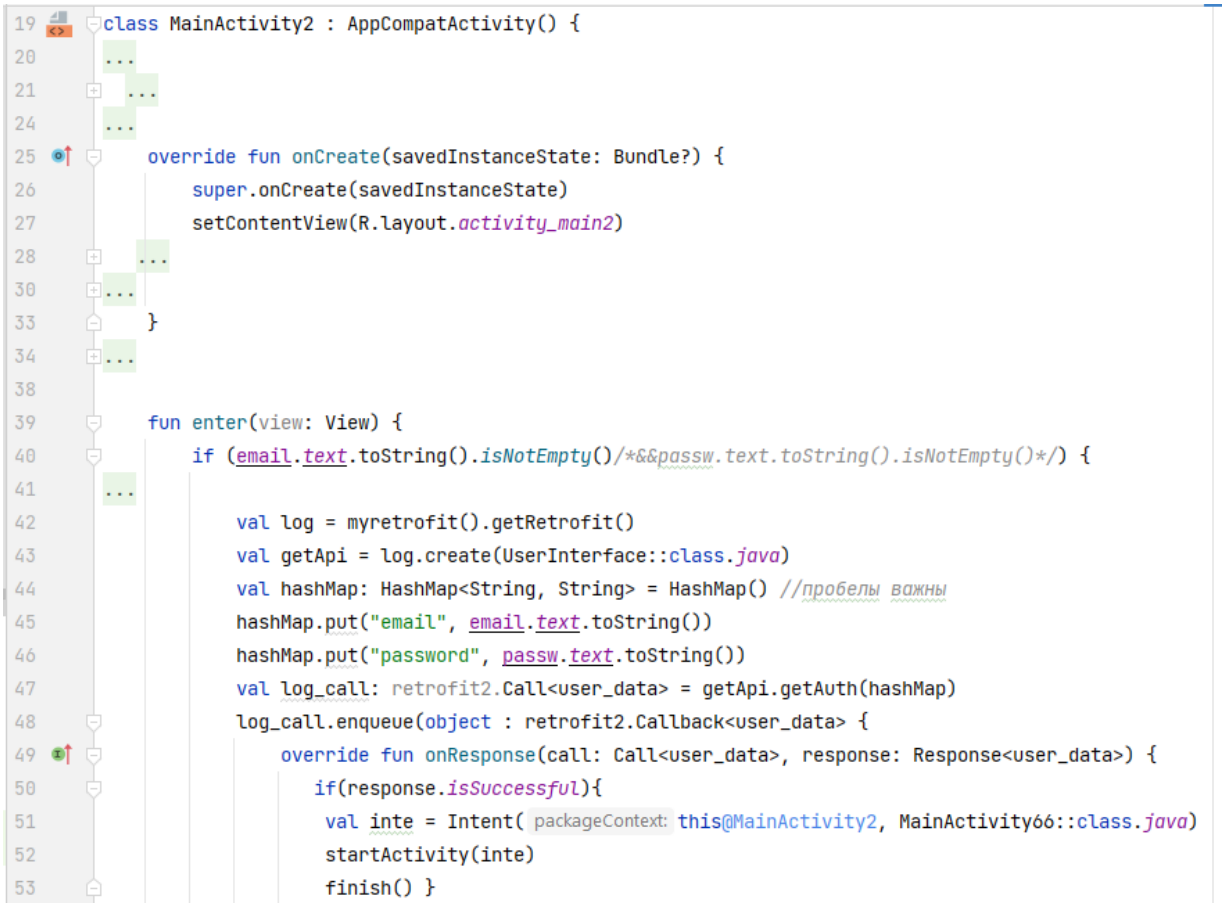
https://metanit.com/kotlin/tutorial/7.4.php

https://kotlins.org/maps

```kotlin
package com.example.testproject

import retrofit2.Retrofit
import retrofit2.converter.gson.GsonConverterFactory

class myretrofit {
    fun getRetrofit():Retrofit=Retrofit.Builder() //Builder в Retrofit – это экземпляр, который использует интерфейс
                                                  // и API Builder, чтобы задать определение конечной точки URL для операций HTTP
        .baseUrl("http://mskko2021.mad.hakta.pro/api/") //базовый адресс-ссылка с которой запрашиваются данные, но не прописывать последнюю ветку
        .addConverterFactory(GsonConverterFactory.create())//добавляем конвертер обработки ответа с сервер
        .build()
}
```

Пропишите логику экрана :

```kotlin
package com.example.testproject

import android.content.Intent
import android.content.SharedPreferences
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.view.View
import android.widget.EditText
import android.widget.TextView
import android.widget.Toast
import com.example.testproject.`interface`.UserInterface
import com.example.testproject.model.user_data
import retrofit2.Call
import retrofit2.Response
```



```
class MainActivity2 : AppCompatActivity() {
...
  ...
...
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main2)
    ...
...
    }
...

    fun enter(view: View) {
        if (email.text.toString().isNotEmpty()/*&&passw.text.toString().isNotEmpty()*/) {
    ...
            val log = myretrofit().getRetrofit()
            val getApi = log.create(UserInterface::class.java)
            val hashMap: HashMap<String, String> = HashMap() //пробелы важны
            hashMap.put("email", email.text.toString())
            hashMap.put("password", passw.text.toString())
            val log_call: retrofit2.Call<user_data> = getApi.getAuth(hashMap)
            log_call.enqueue(object : retrofit2.Callback<user_data> {
                override fun onResponse(call: Call<user_data>, response: Response<user_data>) {
                   if(response.isSuccessful){
                    val inte = Intent( packageContext: this@MainActivity2, MainActivityóó::class.java)
                    startActivity(inte)
                    finish() }
```

```
                else
                   {
                       Toast.makeText( context: this@MainActivity2,  text: "Такого пользователя в системе нет", Toast.LENGTH_SHORT).show()
                   }}
                override fun onFailure(call: Call<user_data>, t: Throwable) {
                    Toast.makeText( context: this@MainActivity2, t.message, Toast.LENGTH_SHORT).show()
                }})}
            else
                Toast.makeText( context: this, text: "Ошибка ввода",Toast.LENGTH_SHORT).show()
    }}
```

44 строка - создание объекта типа HashMap, в который будет передано 2 значение типа String (строки 45 и 46). Название параметра - ключа берется из описания запроса (см.изображение) https://habr.com/ru/articles/128017/

Использование интерфейса и ретрофита приводилось в 12 и 13 практических работах.

Запустите приложение, для теста используйте почту и пароль, указанные в документации (см.изображение). В используемом API отсутствует механизм для добавления новых пользователей, поэтому тестируем на предоставленных и уже сохраненных на сервере.

Настройте переход на новый экран, на котором будет отображаться

произвольное изображение, подгружаемое по ссылке (еще один вариант использования библиотеки Glide):

```
Glide.with( activity: this).load( string: "https://www.sostav.ru/app/public/images/news/2015/12/18/Screenshot_2.jpg?rand=0.2807936074677855").into(img)
//как отобразить изображние по ссылке .load указать ссылку конкретно на изображение .into указать id-контейнера куда поместить
```