

Программа Android-разработчик

Конспект

The smartphone screen shows a navigation bar with a menu icon, search icon, and three-dot menu icon. The main content area lists the following topics:

- Быстрый старт в Android-разработку
- Многопоточность и сетевое взаимодействие** Часть 3
- Архитектура Android-приложений
- Тестирование и работа с библиотеками
- Дизайн и анимации
- Облачные сервисы и периферия

Оглавление

1 НЕДЕЛЯ 3	3
1.1 Знакомство с REST API	3
1.1.1 Что такое сервер, Http и REST	3
1.1.2 Знакомство с OkHttp3	10
1.1.3 Выбор сервера с открытым API	14
1.1.4 Регистрация с помощью OkHttp3	14
1.1.5 Создание ApiUtils	17
1.1.6 Авторизация с помощью OkHttp3	19
1.1.7 Изменение логики показа данных пользователя	20
1.2 Retrofit	21
1.2.1 Знакомство с Retrofit2	21
1.2.2 Добавление и инициализация Retrofit2 в проект	24
1.2.3 Добавление Gson конвертера для Retrofit2	25
1.2.4 Создание интерфейса API в проекте	27
1.2.5 Регистрация с помощью Retrofit	29

1.2.6 Добавление в проект RecyclerView, Adapter, Holder и получение списка альбомов	30
1.2.7 Добавление экрана детального отображения альбома	31

Глава 1

НЕДЕЛЯ 3

1.1. Знакомство с REST API

1.1.1. Что такое сервер, Http и REST

Что такое Сервер (физический)?

Есть множество различных понятий слова сервер, но мы с вами рассмотрим два самых распространенных. Это физическое устройство и программа.

Сервер (физический) – аппаратное устройство, которое выполняет те или иные задачи, как удаленно (что бывает чаще всего), так и локально на месте. Это тот же самый компьютер, только в большинстве случаев мощнее. Сервер, как и компьютер, состоит из: процессора, материнской платы, оперативной памяти и жесткого диска. Обычно для крупных серверов используют специально предназначенные для этого комплектующие, но есть сервера из тех комплектующих, которые используются и в обычных ПК. Другими словами – это компьютер, на котором установлено специальное ПО для работы с клиентской программой.

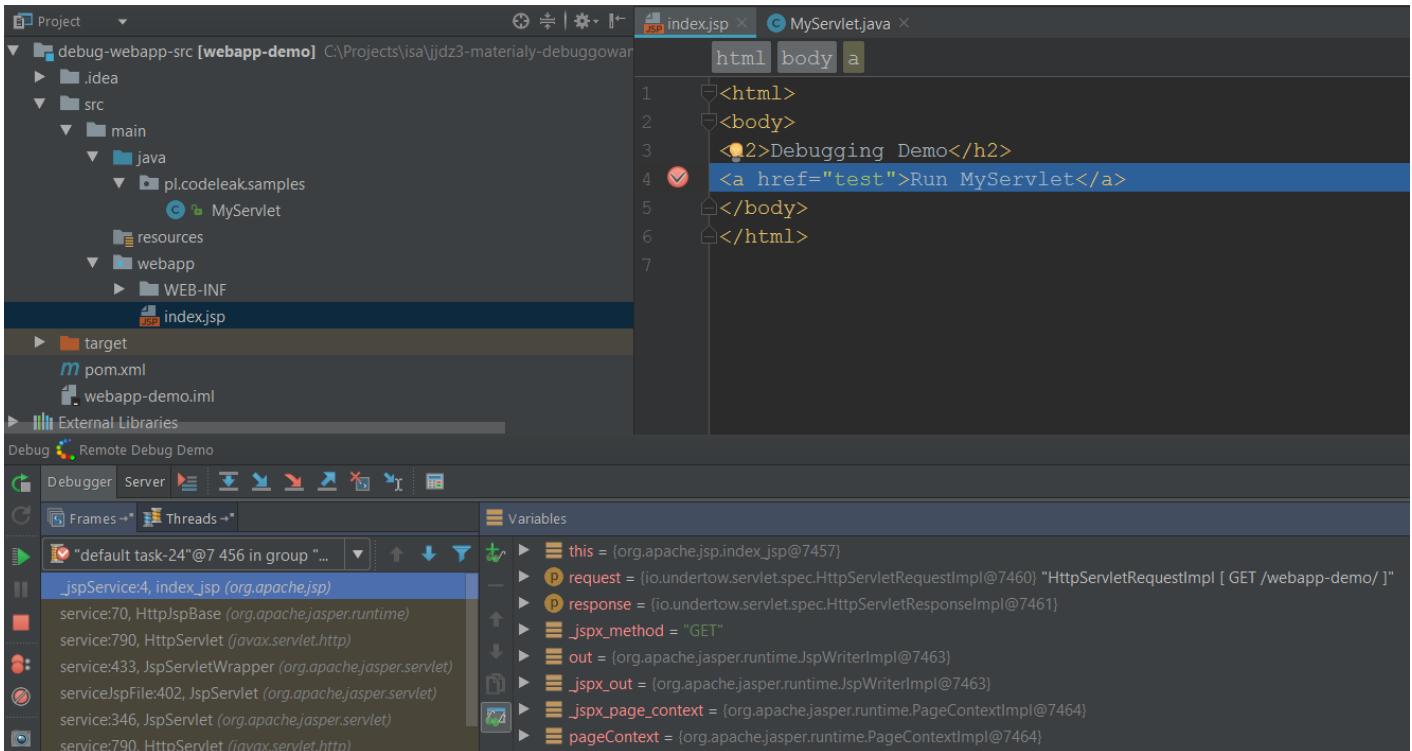
Управление большим количеством серверов, которые находятся в стойках, производится удаленно. То есть инженер, находясь как в соседнем кабинете, так и в другом городе – может на расстоянии управлять серверами. Установка же операционных систем на сервер производится на месте.

Пример того, как выглядит серверная:



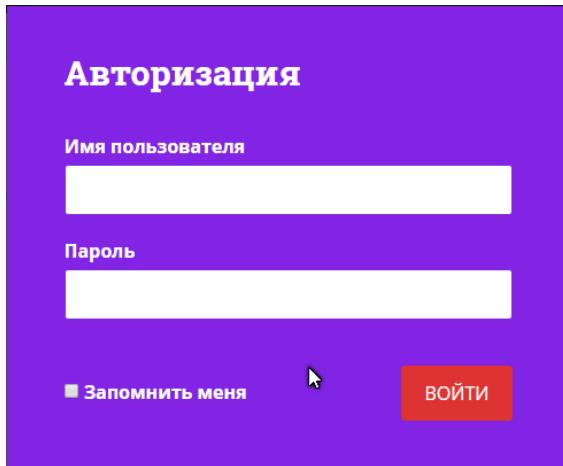
Что такое Сервер (программный)?

Сервер (программный) – это программа, которая устанавливается на компьютер, с помощью которой вы можете описывать бизнес-логику вашего приложения и работать с ней. Она может быть написана на любом языке с использованием любых технологий. Это ПО, которое исполняет какие-либо функции в локальной или глобальной сети – почтовая служба, хранилище файлов, веб-сервер, система управления базой данных, обеспечение документооборота, хранилище данных пользователей сети (доменный контроллер) и т.д. Программные серверы обычно работают на аппаратных серверах. Обычно такую программу называют backend, а все программы, которые взаимодействуют с ней – frontend или client.



Простой пример взаимодействия с сервером

Представьте, что вам нужно авторизоваться на сайте или в Android/iOS приложении. Вы вводите свой логин, пароль. Но откуда программа на телефоне или компьютере узнает, верно ли вы ввели свои данные и существуете ли вы в базе данных? Вся эта информация как раз и хранится на сервере. Ваши введенные данные отправляются на сервер, там он делает некоторые операции, чтобы определить, успешно ли вы авторизовались, и затем от сервера приходит ответ с результатом авторизации и отображается вам на экране.



Что такое HTTP?

HTTP – широко распространенный протокол передачи данных, изначально предназначенный для передачи гипертекстовых документов (то есть документов, которые могут содержать ссылки, позволяющие организовать переход к другим документам).

Аббревиатура HTTP расшифровывается как HyperText Transfer Protocol, «протокол передачи гипертекста».

Этот протокол описывает взаимодействие между двумя компьютерами (клиентом и сервером). Это взаимодействие строится на двух типах сообщений:

- Запрос к серверу (Request);
- Ответ от сервера (Response).

Каждое сообщение состоит из трех частей:

1. Стартовая строка запроса. Из всех параметров обязательной является только она. Выглядит она так:

METHOD URI HTTP/VERSION

- METHOD – это метод HTTP-запроса;

- URI – идентификатор ресурса;
 - VERSION – версия протокола.
2. Заголовки (Headers) – это набор пар имя-значение, разделенных двоеточием. В заголовках передается различная служебная информация: кодировка сообщения, название и версия браузера, адрес, с которого пришел клиент (Referrer) и так далее.
 3. Тело сообщения (Body) – это передаваемые данные.

Методы HTTP-запроса:

- GET – получить;
- POST – создать;
- PUT – обновить (не переданные поля считаются null);
- DELETE – удалить;
- PATCH – обновить запись (только переданные поля).

Например:

- GET-запрос /rest/users – получение информации о всех пользователях;
- GET-запрос /rest/users/125 – получение информации о пользователе с id=125;
- POST-запрос /rest/users – добавление нового пользователя;
- PUT-запрос /rest/users/125 – изменение информации о пользователе с id=125;
- DELETE-запрос /rest/users/125 – удаление пользователя с id=125.

Помимо них есть и другие – CONNECT, HEAD, OPTIONS, TRACE.

Также в Response передается код ответа, чтобы лучше понимать результат ответа. Вот список основных:

Method	Method
200 OK	201 Created
202 Accepted	203 Not authorized
204 No content	205 Reset content
206 Partial content	
300 Multiple choice	301 Moved permanently
302 Found	303 See other
304 Not modified	306 (unused)
307 Temporary redirect	
400 Bad request	401 Unauthorized
402 Payment required	403 Forbidden
404 Not found	405 Method not allowed
406 Not acceptable	407 Proxy auth required
408 Timeout	409 Conflict
410 Gone	411 Length required
412 Preconditions failed	413 Request entity too large
414 Requested URI too long	415 Unsupported media
416 Bad request range	417 Expectation failed
500 Server error	501 Not implemented
502 Bad gateway	503 Service unavailable
504 Gateway timeout	505 Bad HTTP version

Более подробно с HTTP вы можете ознакомиться здесь:

1. <https://habrahabr.ru/post/215117/>
2. <https://ru.wikipedia.org/?oldid=91795909>

Что такое REST?

REST (Representational state transfer) – это стиль архитектуры программного обеспечения для распределенных систем, таких как World Wide Web, который, как правило, используется для построения веб-служб. Термин REST был введен в 2000 году Роем Филдингом, одним из авторов HTTP-протокола. Системы, поддерживающие REST, называются RESTful-системами.

Общение с сервером может быть представлено абсолютно различными способами, REST же описывает то, каким конкретно (в рамках http) оно будет. На самом деле помимо REST также существуют RPC, SOAP и другие, но REST является наиболее распространенным.

Особенность REST в том, что сервер не запоминает состояние пользователя между запросами – в каждом запросе передается информация, идентифицирующая пользователя (например, token, полученный через OAuth-авторизацию) и все параметры, необходимые для выполнения операции.

Примеры REST API:

1. <https://vk.com/dev/methods>
2. <https://core.telegram.org/bots/api>
3. <https://developer.twitter.com/en/docs>

Доп. чтение:

1. <https://habrahabr.ru/post/50147/>
2. <https://habrahabr.ru/post/158605/>
3. <https://habrahabr.ru/post/38730/>
4. <https://habrahabr.ru/post/265845/>
5. <https://ru.wikipedia.org/wiki/REST>

1.1.2. Знакомство с OkHttp3

Предыстория

Давным-давно, когда Android 5.0 был только в мечтах, существовало всего лишь 2 HTTP-клиента для Android:

- HttpURLConnection
- Apache HTTP Client

Apache HTTP client практически не содержит багов на Eclair и Froyo, поэтому он является лучшим выбором для этих версий. А для Gingerbread и младше лучше подходит HttpURLConnection. Простота API и небольшой вес хорошо подходят для Android. Прозрачное сжатие и кэширование ответов помогают увеличить скорость и сохранить батарею. Новые приложения должны использовать HttpURLConnection.

Это и являлось основной проблемой этих двух клиентов. Ребята из Square обратили на это внимание, когда создавали свой клиент, который называли OkHttp.

В итоге в июле 2015 Google официально признала AndroidHttpClient, основанный на Apache, устаревшим (deprecated).

OkHttp получил достаточно широкое распространение и признание за счет своего удобства и функционала.

Что такое OkHttp?

Как можно было понять из рассказа выше, OkHttp – проект с открытым исходным кодом, разработанный, чтобы быть эффективным HTTP-клиентом. Он поддерживает протокол SPDY. SPDY является основой для HTTP 2.0 и позволяет мультиплексировать несколько HTTP-запросов по одному socket-соединению. Помимо данных возможностей он содержит огромное количество других, не менее полезных, вот основные из них:

1. Пул соединений;
2. Gziping;

3. Кэширование;
4. Восстановление после сетевых ошибок;
5. Редиректы;
6. Повторы;
7. Поддержка синхронных и асинхронных вызовов;
8. Interceptors.

Со всеми вы можете ознакомиться здесь: <http://square.github.io/okhttp/>.

OkHttp. Interceptors

Interceptors – это мощный механизм, который может отслеживать, переписывать и повторять request'ы. Interceptors могут добавлять, удалять или заменять заголовки запросов (request header). Они также могут изменять тело (body) тех запросов, у которых он есть.

Пример простого interceptor:

```
1  class LoggingInterceptor implements Interceptor {  
2      @Override public Response intercept(Interceptor.Chain chain) throws IOException {  
3          Request request = chain.request();  
4          long t1 = System.nanoTime();  
5          logger.info(String.format("Sending request %s on %s%n%s",  
6              request.url(), chain.connection(), request.headers()));  
7  
8          Response response = chain.proceed(request);  
9  
10         long t2 = System.nanoTime();  
11         logger.info(String.format("Received response for %s in %.1fms%n%s",  
12             response.request().url(), (t2 - t1) / 1e6d, response.headers()));  
13  
14         return response;  
15     }  
16 }
```

Вызов chain.proceed() является важной частью реализации каждого interceptor, это то место, где происходит вся работа HTTP, создавая ответ для удовлетворения запроса.

Так можно подключить interceptor:

```
1 OkHttpClient client = new OkHttpClient.Builder()  
2     .addInterceptor(new LoggingInterceptor())  
3     .build();
```

OkHttp. Как работает?

Чтобы подключить OkHttp в свой проект в Gradle файл вашего проекта:

```
1 compile 'com.squareup.okhttp3:okhttp:3.9.1'
```

Добавьте разрешение на работу с Интернетом в AndroidManifest:

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

Чтобы создать объект запроса к серверу:

```
1 OkHttpClient client = new OkHttpClient();  
2 Request request = new Request.Builder().url("https://www.google.ru/").build();
```

Чтобы добавить query параметры:

```
1 HttpUrl.Builder urlBuilder = HttpUrl.parse("https://api.github.help").newBuilder();  
2 urlBuilder.addQueryParameter("version", "0.0.1");  
3 urlBuilder.addQueryParameter("user", "login");  
4 String url = urlBuilder.build().toString();  
5  
6 Request request = new Request.Builder().url(url).build();
```

Синхронный вызов сетевого запроса и получение response (результата запроса):

```
1 try {
2     Response response = client.newCall(request).execute();
3 } catch (IOException e) {
4     e.printStackTrace();
5 }
```

Асинхронный вызов сетевого запроса и получение response(результата запроса):

```
1 client.newCall(request).enqueue(
2     new Callback() {
3         @Override
4         public void onFailure(Call call, IOException e) {
5             e.printStackTrace();
6         }
7
8         @Override
9         public void onResponse(Call call, final Response response) throws IOException
10        {
11            if (!response.isSuccessful()) {
12                throw new IOException("Unexpected code " + response);
13            } else {
14                // do something with the result
15            }
16        }
17    });
18 );
```

Отличные туториалы:

1. <https://www.journaldev.com/13629/okhttp-android-example-tutorial>
2. https://github.com/codepath/android_guides/wiki/Using-OkHttp
3. <http://www.vogella.com/tutorials/JavaLibrary-OkHttp/article.html>

Документация:

1. <https://square.github.io/okhttp/>

2. <https://github.com/square/okhttp/wiki>

Дополнительная информация:

1. <https://habrahabr.ru/post/281965/>

1.1.3. Выбор сервера с открытым API

В качестве сервера, к которому мы будем обращаться, будет Android academy e-Legion с одноименным API (ссылка: <https://android.academy.e-legion.com/docs/#/>).

Мы будем использовать методы нескольких групп:

1. Auth – метод registration – POST-метод для отправки данных пользователя на регистрацию;
2. Albums – три GET-метода для получения информации об альбомах (albums), конкретном альбоме (albums с параметром id) или комментариях к этому альбому (albums с параметром id – comments);
3. Songs – два GET-метода для получения информации об песнях (songs) или конкретной песне (songs с параметром id);
4. Comments – два GET- и один POST-метод для получения списка комментариев (comments), конкретного комментария (comments с параметром id) или его создания (comments);
5. User – метод user – GET-метод для получения информации о текущем пользователе.

Более подробно ознакомиться с API можно по ссылке выше.

1.1.4. Регистрация с помощью OkHttp3

В данном занятии мы переделаем логику регистрации. Раньше она у нас работала с помощью shared preferences, то есть работала локально. Теперь же мы будем работать с сервером с помощью библиотеки OkHttp3. Также мы внесли несколько изменений в проект. Давайте посмотрим, что мы изменили.

Откроем сам наш проект, папка res/layout/fr.registration.xml. На макете мы увидим, что мы добавили поле Имя. Оно нам нужно, чтобы взаимодействовать с сервером, так как объект пользователя на сервере содержит поле Имя. Теперь можно переходить непосредственно к логике регистрации. Перейдем в Gradle Scripts/build.gradle(Module:app), зайдем в dependencies и добавим dependency для OkHttp3.

Я говорю сразу, что я не буду писать некоторые части кода, так как всего запомнить невозможно и в принципе это нормальная практика, то есть не обязательно полностью с нуля писать куски какого-то кода. Вы можете спокойно открыть свой старый проект, или пример, или док из интернета и скопировать код оттуда и его изменить под свои нужды. Это вполне нормальная практика.

Дальше мы добавили с вами dependency для OkHttp3, теперь давайте добавим build config field для OkHttp3 URL'a, то есть у нас будет какой-то базовый URL, который мы не будем постоянно переписывать, а который будет храниться у нас в build config field'e. Перейдем соответственно в defaultConfig.

buildConfigField "String", "SERVER_URL" так он будет называться. И допишем его. Поставим в конце /, чтобы быть уверенными, что все будет работать правильно. Теперь давайте перейдем в AndroidManifest. Здесь добавим Permission для работы с интернетом, ведь без этого Permission'a мы ничего не сможем сделать. И перейдем в RegistrationFragement, в mOnRegistrationClickListener.

Будем переписывать эту логику. Начнем с того, что удалим isAdded, но оставим User'a и сохраним его в переменную user. Как я уже говорил, мы добавили поле name и мы должны изменить объект для этого. Давайте перейдем и сделаем это. String email, String password. Этот конструктор нам больше не нужен. Нам нужен конструктор с вот таким содержанием. Чтобы он содержал email, name и пароль. Также я добавил аннотации SerializedName из библиотеки gson для того, чтобы можно сделать автоматический парсинг. Добавим поле name в конструктор:

```
1 mName.getText().toString();
```

Далее можно делать сам запрос к серверу. Мы забыли просинхронизироваться, давайте исправим это. Нажимаем «синхронизация». Чтобы мы получили доступ к запросу из OkHttp3-библиотеки:

```
1 Request request = new  
→ Request.Builder().url(BuildConfig.SERVER_URL.concat("registration/"));
```

Поскольку этот метод требует постзапрос, мы должны добавить к нему request body. Давайте сделаем это. Сделать это можно с помощью метода post и соответственно RequestBody.

```
1 .post(RequestBody.create());
```

Он потребует MediaType. В нашем случае MediaType'ом будет выступать JSON. Перейдем вверх RegistrationFragment'a. Добавим в него:

```
1 public static final MediaType JSON = MediaType.parse("application/json; charset=utf-8");
```

Можете просто скопировать ее, чтобы руками не писать эту строку. Далее добавим ее в метод создания create.

```
1 .post(RequestBody.create(JSON, new Gson().toJson(user)));
2 .build();
```

Запрос готов. Давайте теперь вызовем его. Для этого создадим OkHttp-клиент:

```
1 OkHttpClient client = new OkHttpClient();
2 client.newCall(request).enqueue(new Callback() {
3     ...
4 })
```

И переопределим методы onFailure и onResponse, вернее, заимплементим. Кстати говоря, чтобы мы имели доступ к UI, нам обязательно нужно делать это в main thread'e, иначе у нас будет ошибка. Так как мы выполняем наш запрос асинхронно, то все callback'и приходят к нам асинхронно. Для этого, чтобы получить доступ к UI мы будем использовать handler:

```
1 Handler handler = new Handler(getActivity().getMainLooper());
```

Далее оберачиваем всю работу внутри этих методов в handler.post. В методе run соответственно мы будем работать с UI, то есть в случае, если он failure, то у нас произошла ошибка. Скопирую-вставлю. Какая ошибка? Ошибка работы с request'ом, то есть request_error. Ctrl+C. В onResponse также мы работаем внутри Handler'a, внутри runnable, если точнее. Теперь внутри:

```
1 if (response.isSuccessful()) {  
2     showMessage(R.string.registration_success);  
3     getFragmentManager().popBackStack();  
4 } else {  
5     //todo детальная обработка ошибок  
6     showMessage(R.string.registration_error);  
7 }
```

Если запрос успешный, то показываем message о том, что регистрация прошла успешно. Иначе request_error, не совсем request_error, правда, скорее ошибка регистрации. Также добавим сюда todo. Этот todo поможет нам запомнить, что нам нужно добавить сюда детальную обработку ошибок, ведь мы не делаем детальную обработку ошибок, мы просто показываем какую-то ошибку, если запрос выполнился не успешно.

Теперь давайте посмотрим, как это работает на эмуляторе. Запустим наш эмулятор, нажмем кнопочку «зарегистрироваться», введем какой-нибудь email. Пусть будет te@te.te. Имя – te. Пароль обязательно вводим восьмизначный, так как сервер будет просить именно восьмизначный пароль, если мы введем пароль менее восьми знаков, то он покажет соответствующую ошибку, которую мы сможем распознать с помощью кода ответа. 12345678. 12345678. Нажмем «зарегистрироваться». Как мы видим, регистрация прошла успешно. В данном уроке мы научились использовать post-запрос с помощью библиотеки OkHttp3.

1.1.5. Создание ApiUtils

В этом занятии мы создадим класс ApiUtils, с помощью которого мы будем получать instance OkHttp-клиента. Как вы уже знаете, наш сервер содержит механизм basic auth. Чтобы каждый раз не возвращать новый instance OkHttp-клиента, и каждый раз его не создавать, и не передавать в него новый email и пароль, мы будем делать это только при аутентификации приложения, при авторизации, если точнее.

Давайте перейдем в сам проект. Зайдем в папочку Gradle Scripts, build.gradle и для начала добавим logging interceptor. Начнем синхронизироваться. Здесь мы ввели

```
1 implementation 'com/squareup.okhttp3:logginginterceptor:3.9.1'
```

Далее переходим в сам код, создаем ApiUtils, жмем Enter, далее OK. Это будет:

```
1 private static OkHttpClient okHttpClient;
```

и создадим:

```
1 public static OkHttpClient getBasicAuthClient(String email, String password, boolean
→ newInstance) {
2
3 }
```

Как я уже говорил, чтобы каждый раз не пересоздавать новую сущность, мы будем использовать переменную newInstance. Лейбл, если точнее. И newInstance true у нас будет только в том случае, когда нужно будет перезаписать наш email и наш пароль, то есть при входе в приложение, при авторизации. Далее:

```
1 public static OkHttpClient getBasicAuthClient(String email, String password, boolean
→ newInstance) {
2     if (newInstance || okHttpClient == null) {
3         OkHttpClient.Builder builder = new OkHttpClient.Builder();
4
5         builder.authenticator(new Authenticator() {
6             @Nullable
7             @Override
8             public Request authenticate(Route route, Response response)
→                 throws IOException {
9                 String credentials = Credentials.basic(email, password);
10                return response.request().newBuilder().header(
11                    "Authorization", credentials).build();
12                }
13            });
14        }
}
```

Отлично, с этим разобрались. Далее также давайте добавим log'ов, чтобы при каждом запросе и response нам приходили log'и и мы их отображали в log cat'e. Для этого делаем:

```
1 builder.addInterceptor(new HttpLoggingInterceptor ().Level(BODY));
```

Отлично, теперь делаем:

```
1 okHttpClient = builder.build();
```

Далее:

```
1 return okHttpClient;
```

Вот и все. В данном занятии мы научились добавлять аутентификатор в OkHttp-клиент и добавлять логирование в него.

1.1.6. Авторизация с помощью OkHttp3

В данном занятии мы будем имитировать логику авторизации. Именно имитировать, потому что на нашем сервере нет такой логики, так как он работает на механизме basic auth. Как вы знаете, basic auth нужен для того, чтобы проверять логин и пароль каждый раз, как мы обращаемся к какому-либо из методов сервера. Войдем в приложение, зайдем в AuthFragment, удалим SharedPreferencesHelper, так как она нам больше не пригодится, и mEmailedUsersAdapter, потому что он завязан на логике SharedPreferencesHelper. Перейдем в метод onCreateView, удалим SharedPreferencesHelper, удалим EmailedUsersAdapter и все, что с ним связано полностью. Далее можно переходить в mOnEnterClickListener. user пока что сотрем, так как он нам не пригодится, здесь эта логика должна работать, когда все прошло успешно, пока что оставим ее.

Теперь, чтобы было проще, скопируем логику из registrationFragment'a. Идем в registrationFragment в mOnRegistrationClickListener, выделим все, нажмем Ctrl+C, Ctrl+V, OK. POST-запроса у нас не будет теперь. Это не POST-запрос. Это не registrationMethod называется, а user. И это GET-запрос. Далее OkHttpClient'a мы берем из нашего ApiUtils, из mEmail:

```
1 OkHttpClient client = ApiUtils.getBasicAuthClient(
2     mEmail.getText().toString(),
3     mPassword.getText().toString(),
4     createNewInstance: true);
```

Обязательно передаем переменную true, так как ApiUtils должен создавать новую Instance, так как мы якобы авторизуемся. Далее post response is successful, если не is successful, то показывать auth, иначе эта логика у нас прописана вот здесь. Ctrl+X, Ctrl+V. Теперь это можно удалять. Здесь нам нужно создать объект User'a, который мы возьмем из:

```
1 User user = new Gson().fromJson(response.body().string(), User.class);
```

Он у нас на что-то ругается, давайте посмотрим, на что. А, здесь мы сделали хитрую штукку. Мы для начала преобразуем не в user'a, а в JsonObject, так как у нас есть небольшой нюанс в том плане, что метод user возвращает нам сам объект user'a внутри объекта data. Это немножко неправильно, но раз сервер так работает, мы ничего с этим поделать не можем и будем адаптироваться к этому. Для начала он нам вернет JsonObject, будет наш Json. Далее, наверное, создадим gson, чтобы несколько раз не создавать instance gson'a:

```
1 Gson gson = new Gson();
```

Здесь будем использовать именно его. gson и здесь тоже gson. Здесь у нас уже будет полноценный user, здесь у нас будет gson.fromJson и будем брать json.get("data"), но преобразовывать будем не в JsonObject, а в User.

Он нам говорит, что IOException у него будет. Наверное, чтобы было лучше, catch сделаем в конце, то есть если совсем ошибка, то вообще ничего не нужно делать. Ctrl+Alt+L. Далее json, это сотрем. Пишет, что где-то ошибка. A SharedPreferences – эта логика нам больше не нужна, потому что просто не нужна. Нажмем «Sync now», так как у нас что-то изменилось, и теперь давайте запустим эмулятор и посмотрим, как это работает. Введем какой-нибудь login, введем пароль. 12345678. Нажмем войти. Как вы видите, он правильно отображает наш email. Пароль он не отображает по той простой причине, что сервер не может его вернуть. Сервер возвращает только два поля: email и name. В данном занятии мы научились делать GET-запрос с помощью библиотеки OkHttp3.

1.1.7. Изменение логики показа данных пользователя

В данном занятии мы переделаем экран профиля так, чтобы на нем остались только два поля: email и name. Ровно так, как нам возвращает сервер. Откроем проект, перейдем в layout/ac_profile.xml. Удалим фото, так как нам фото в сервере не возвращает. В этот LinearLayout добавим margin слева:

```
1 android:layout_marginLeft="16dp"
```

Пароля у нас больше нет, вернее, он есть, но сервер нам его не возвращает. Здесь просто сделаем Shift+F6, Name. Label поменяем на name. Далее уберем пробел, нажмем сочетание клавиш Ctrl+Alt+L, чтобы отформатировать.

Теперь перейдем в ProfileActivity. Здесь password переименовываем в mName. Enter. Далее mPhoto удаляем, так как у нас его больше нет. SharedPreferencesHelper нам больше тоже не нужен. Всю логику, связанную с фото можно также удалить, то есть удаляем OnActivityResult.

В mName.setText должен быть не getPassword, а getName. mPhoto удаляем. Profile меню оставим, он нам нужен. onPhotoClickListener удаляем. Метод OpenGallery нам также не нужен. REQUEST_CODE_GET_PHOTO тоже.

Теперь давайте запустим эмулятор и посмотрим, как это работает. Введем логин. Введем пароль. Нажмем войти. Как вы видите, наш экран верно отображает и email, и name. В данном занятии мы с вами переделали отображение экрана профиля.

1.2. Retrofit

1.2.1. Знакомство с Retrofit2

Предисловие

В данной статье мы будем называть библиотеку Retrofit, а не Retrofit2. В подавляющем большинстве случаев в Интернете под Retrofit подразумевают именно Retrofit2.

Retrofit. Что это?

С OkHttp мы уже знакомы, но помимо него у Square есть еще одно детище и зовут его Retrofit. Retrofit – типобезопасный HTTP-клиент для Java (Android в том числе). Библиотеки лучше, чем Retrofit, для работы с API на данный момент не придумали. Но для чего он нам нужен? В чем существенные отличия между этими двумя HTTP-клиентами?

Retrofit. Для чего?

Он работает как обертка на, всеми любимым OkHttp и, соответственно, может делать все то, что умеет OkHttp и даже больше. Основная «фишка» Retrofit заключается в том, что он за счет своих возможностей убирает огромную часть boilerplate кода, который был бы при использовании OkHttp. Делается это с помощью интерфейсов API, конвертеров и CallAdapter, которые мы создаем внутри проекта. Эти интерфейсы описывают то, как должно проходить наше общение с сервером, т.е. описывают само API. Другими словами, Retrofit превращает HTTP API в Java-интерфейс. Также Retrofit является самым быстрым HTTP-клиентом и обладает киллер фичей – поддержка реактивности в лице RxJava.

Основные возможности Retrofit:

- Вся сила OkHttp – он умеет абсолютно все, что может OkHttp;
- API Interfaces – возможность создать удобный интерфейс для общения с API, который даже не нужно реализовывать, Retrofit сделает это за вас. Все это работает на аннотациях, с их помощью можно описать абсолютно любой HTTP-запрос;
- Call Adapters – адаптеры для изменения типа ответа сервера (<https://github.com/square/retrofit/wiki/Call-Adapters>);
- Converters – с помощью конвертеров мы можем указать, как нужно автоматически сериализовать запрос к серверу или десериализовать ответ от сервера (<https://github.com/square/retrofit/wiki/Converters>);
- Задание параметров URL и поддержка параметров запроса;
- Преобразование объектов в тело запроса (например, JSON);
- Многостраничный запрос и загрузка файлов;

Retrofit. Как работает?

Подключаем библиотеку:

```
1 implementation 'com.squareup.retrofit2:retrofit:2.3.0'
```

Если нужно, то converters и call adapters:

```
1 implementation 'com.squareup.retrofit2:converter-gson:2.3.0'  
2 implementation 'com.squareup.retrofit2:adapter-rxjava:2.3.0'
```

Кстати, когда мы подключаем Retrofit 2, то автоматически подключается библиотека OkHttp3 и ее не нужно добавлять отдельно.

Инициализируем Retrofit instance:

```
1 Retrofit retrofit = new Retrofit.Builder().baseUrl("https://api.github.com/")  
2   //.addConverterFactory(GsonConverterFactory.create(gson))  
3   //.addCallAdapterFactory(RxJavaCallAdapterFactory.create())  
4   .build();
```

Создаем интерфейс API с 1 методом:

```
1 public interface GitHubService {  
2     @GET("users/{user}/repos")  
3     Call<List<Repo>> listRepos(@Path("user") String user);  
4 }
```

Помимо аннотации @GET есть еще и другие типа @POST, @DELETE и т. д. Также мы можем описать абсолютно любой HTTP-запрос.

Создадим интерфейс API и вызовем метод HTTP-запроса:

```
1 GitHubService service = retrofit.create(GitHubService.class);  
2 Call<List<Repo>> repositories = service.listRepos("octocat");
```

Вот так просто работает Retrofit.

Для более глубокого изучения Retrofit рекомендуем ознакомиться с дополнительной информацией:

1. <http://square.github.io/retrofit/>
2. <https://github.com/square/retrofit/wiki/Retrofit-Tutorials>

3. <https://futurestud.io/tutorials/retrofit-getting-started-and-android-client>
4. <https://habrahabr.ru/post/314028/>
5. <http://javaway.info/ispolzovanie-retrofit-2-v-prilozheniyah-android/>

1.2.2. Добавление и инициализация Retrofit2 в проект

В данном занятии проинициализируем Retrofit Instance для того, чтобы использовать его в будущем, так как мы будем постепенно переезжать с библиотек OkHttp3 на Retrofit2, чтобы вы смогли прочувствовать всю разницу между этими библиотеками.

Теперь перейдем в наш проект, зайдем в Gradle Scripts, build.gradle(Module: app), Dependencies и добавим dependency для Retrofit'a 2. Сразу проинициализируемся. Перейдем в com.elegion.myfirst application/ApiUtils. Создадим Retrofit Instance.

```
1 private static Retrofit retrofit;
2 //И создадим метод для его получения
3 public static Retrofit getRetrofit() {
4     if (retrofit == null) {
5         retrofit = new Retrofit.Builder()
6             .baseUrl(BuildConfig.SERVER_URL)
7             //need for interceptors
8     }
9 }
```

На самом деле он будет использоваться не только для interceptor'ов, но и для логирования и также для basic auth'a, потому что мы будем использовать клиент OkHttp3. Соответственно, укажем клиент, который возьмем из getBasicAuthClient:

```
1 .client(getBasicAuthClient)(email "", password "", createNewInstance false)
```

Сюда передадим две пустые строки и зададим createNewInstance false. Нужно задать email и password пустыми, потому что он будет храниться в OkHttp-клиенте. Нам нужно этот OkHttp-клиент получить. И сделаем:

```
1 .build();
```

Также добавим:

```
1 return retrofit;
```

Вот и все. В данном занятии мы с вами успешно проинициализировали Retrofit Instance, который будем использовать в дальнейшем.

1.2.3. Добавление Gson конвертера для Retrofit2

В данном занятии мы с вами добавим конвертер-фабрику для нашего Retrofit Instance, который будет преобразовывать наши модели в Gson и Gson в наши модели. Также мы создадим модели для сервера.

Перейдем к делу. Зайдем в проект. Откроем наш любимый Gradle Scripts, build.gradle(Module: app). Добавим implementation для converter-gson, потому что мы используем библиотеку gson. Делаем синхронизацию. Далее перейдем в ApiUtilis. Создадим Instance для gson'a, чтобы он тоже не создавался несколько раз, и использоваться он будет только в Retrofit'e. Сюда добавим проверку на null:

```
1 if(gson == null) {  
2     gson = new Gson();  
3 }
```

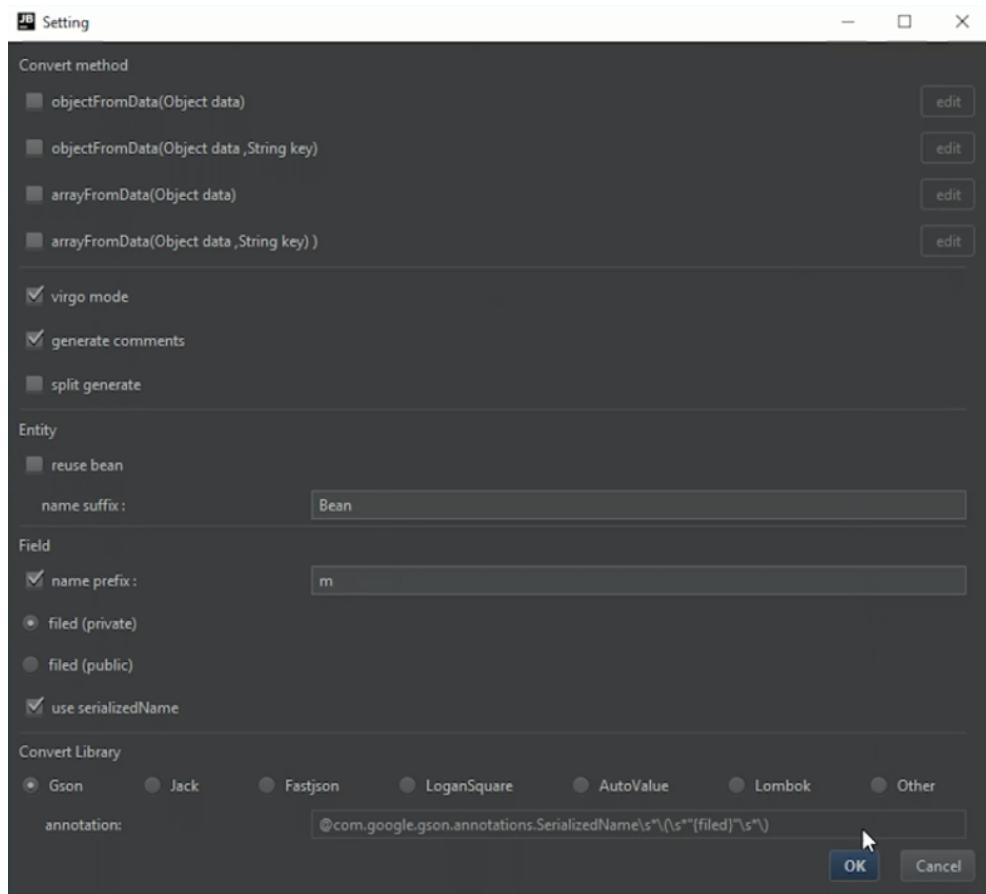
Сюда мы добавим:

```
1 .addConverterFactory(GsonConverterFactory.create(gson));
```

С этим мы разобрались. Теперь давайте перейдем в сваггер. Начнем с песен. Songs id – здесь просто копируем весь gson, который есть. Это будет наша модель песни. Создадим новый package, чтобы они не лежали хаотично со всеми классами, перенесем на него user'a в начале, перенесем user'a в model.

Здесь создадим New → Java Class. Назовем его Song, удалим подпись, она не обязательна. Также перейдем в файл в settings, в plugins и будем качать библиотеку Gson Format. У кого ее нет, просто нажмите на клавишу Browse repositories, введите Gson Format и скачайте ее. Поскольку у меня она скачана, у меня нет кнопки Install, у вас она будет. После этого перезагрузите Android Studio и она будет работать, то есть у вас будет работать плагин Gson Format.

Далее мы создали класс Song. Alt+S, либо Alt+Insert и тут вот будет Gson Format. Разницы нет, кому как удобнее. Нажимаем Format, можно зайти в settings и настроить. Можете посмотреть, как это сделано у меня. Нажимаем OK.



Здесь тоже нажимаем OK. И тут. Как вы видите, плагин автоматически сгенерировал наш класс. Объект Song создан.

Давайте теперь создадим объект Songs. Здесь нажмем Ctrl+C, Ctrl+V. Будет объект Songs. Перейдем в Songs. Скопируем весь gson, удалим все что есть. Сочетанием Alt+S откроем Gson Format. Нажмем Ctrl+V, Format. Удаляем meta и links, они нам не пригодятся. Но вы можете оставить их, если хотите. Нажмем OK. Удалим комментарии.

Теперь можно приступать к созданию объекта Album. Идем в сваггер, Albums id, копируем все, что тут есть, удаляем, нажимаем Alt+S, Ctrl+V, Format. Здесь ничего удалять не нужно – все стоит правильно. Нажимаем OK.

SongsBean нам не нужен, можем удалить. Можем изменить сразу же data type, так как нам будет возвращаться список Song, изменим это. Здесь у нас будет ListSong. Нажимаем OK. Какая-то ошибка. Давайте попробуем сконвертировать еще раз. Alt+S, закроем ошибки. Alt+S, Ctrl+V. Иногда такое бывает, ничего страшного, ведь эти плагины тоже писали программисты. Жмем OK, здесь все равно нажмем OK на всякий случай, удалим SongsBean, потому что он нам не нужен.

Далее List у нас здесь будет просто Song, скопируем, чтобы было проще, здесь тоже будет Song, то есть get List<Song>, set List<Song>. Удаляем комментарии. Здесь тоже все удаляем. Объект Album готов, теперь нужно сделать объект Albums.

Перейдем в сваггер, Albums, скопируем все, что тут есть. Это мы поторопились. Сначала создадим объект albums. Здесь все удалим, скопируем тут, сделаем Ctrl+Shift, нажмем Alt+S, Ctrl+V, Format, удалим meta и links, они нам не нужны. Нажмем OK. Далее жмем также OK. Удаляем комментарии, он нам больше не нужны.

Вот и все, в данном занятии мы с вами добавили конвертер-фабрику для gson'а и создали модели, которые соответствуют моделям сервера.

1.2.4. Создание интерфейса API в проекте

В данном занятии мы создадим интерфейс API, который будет использоваться через Retrofit для того, чтобы взаимодействовать с сервером. Давайте перейдем к делу.

Зайдем в проект, создадим New → Java Class, назовем его AcademyApi, сделаем его интерфейсом, нажмем OK. Теперь перейдем в фраггер. Видим, что у нас здесь есть пять методов. Registration, albums, albums id, songs, songs id. И есть еще раздел Comments. Раздел Comments останется вам на домашнее задание, то есть вы будете с ним работать сами. Видим, что метод

registration – POST. Соответственно в интерфейсе пишем:

```
1 @POST  
2 Call<Void> registration(@Body User user)
```

Мы в этом можем убедиться, когда посмотрим на то, что он принимает в request body. Находится он по методу registration. Давайте добавим.

```
1 @POST("registration")
```

Скопируем его. Следующим идет метод get albums. И все остальные в точно таком же порядке:

```
1 @GET("albums")  
2 Call<Albums> getAlbums();
```

Следующий метод у нас будет albums id:

```
1 @GET("albums/{id}")  
2 Call<Album> getAlbum(@Path("id") int id);
```

Далее скопируем эти два метода. Здесь все по аналогии:

```
1 @GET("songs")  
2 Call<Songs> getSongs();  
3 @GET("songs/{id}")  
4 Call<Song> getSong(@Path("id") int id);
```

Интерфейс API мы создали. Теперь давайте создадим его внутри ApiUtils. Перейдем в ApiUtils. Сделаем ему отдельную переменную, назовем ее

```
1 private static AcademyApi api;
```

Здесь добавим метод, который будет возвращать нам api:

```
1 public static AcademyApi getApi() {
2     if(api == null) {
3         api = getRetrofit().create(AcademyApi.class);
4     }
5     return api;
6 }
```

Вот и все. В данном занятии мы создали интерфейс для работы с нашим серверным апи.

1.2.5. Регистрация с помощью Retrofit

В данном занятии мы перепишем регистрацию с OkHttp3 на Retrofit2. Откроем проект. app/java/com.elegion.myfirstapplication/RegistrationFragment. Перейдем в mOnRegistrationClickListener. Сотрем OkHttp-клиент, он нам теперь не нужен, request тоже нам не пригодится. client переделаем в

```
1 ApiUtils.getApiService().
2 ApiUtils.getApiService().registration(user).enqueue(new Callback() {
3     ...
4 })
```

Как вы видите, количество кода явно сократилось, соответственно он стал лучше. Этот код закомментируем, потому что он нам еще пригодится. Это делается сочетанием клавиш Ctrl, Shift и слэш. Здесь сделаем new retrofit2.Callback<Void>, переместим его. OnResponse копируем тот, который был до этого, то есть в принципе вся логика осталась та же, даже взаимодействие тоже. onFailure тоже скопируем, Ctrl+V. Handler MainHandler тоже скопируем, даже вместе с комментариями, и удалим этот комментарий, чтобы он нам больше не мешался. Добавим его сюда. Alt+Enter, android.os. Response сделаем final. Вот и все.

Теперь давайте проверим, как это все работает в эмуляторе. Перейдем в эмулятор. Зайдем в «зарегистрироваться», введем какой-нибудь email, fgh@f.f пусть будет. Имя – fgh. Введем пароль. 12345678. Зарегистрироваться. Теперь введем его в поле логин. fgh@f.f. 12345678. Войти.

В данном занятии мы успешно переделали логику регистрации с OkHttp3 на Retrofit2.

1.2.6. Добавление в проект RecyclerView, Adapter, Holder и получение списка альбомов

В данном занятии мы будем выводить список альбомов на экран с помощью нашего интерфейса api. Мы уже заранее проинициализировали некоторый UI, и некоторый код я взял у Азрета. Вы его видели, когда проходили RecyclerView. Давайте посмотрим, что я изменил, и что у нас есть теперь.

Перейдем в AuthFragment, в mOnEnterClickListener. В onResponse, и если response у нас успешный, то, как вы видите, я закомментировал код startProfileIntent. Он нам больше не нужен, потому что теперь, после успешной авторизации у нас будет открываться AlbumsActivity.

AlbumsActivity выглядит так, то есть она наследуется от SingleFragmentActivity и открывает AlbumsFragment. AlbumsFragment практически точно такой же, какой делал Азрет. В нем ничего нового нет, то есть есть RecyclerView, refresher и mErrorView. Даже подпись стоит, что его делал Азрет. Далее, инициализация view осталась такая же, немножечко переделана логика mRefresher. Нам для того, чтобы показать наши альбомы, останется только в этом методе получать альбомы и выводить их в адаптер.

Далее, AlbumsAdapter выглядит вот так. Он в себе хранит List<Albums.DataBean> и выводит их. AlbumsHolder выглядит вот так. mTitle и mReleaseDate, то есть также два поля. Далее AlbumsFragment. Перейдем в него и будем писать метод для получения самих альбомов:

```

1 private void getAlbums() {
2     ApiUtils.getApiService().getAlbums().enqueue(new Callback<Albums>(){
3         });
4     Ctrl-Alt. onResponse и onFailure. Здесь в onResponse
5     if (response.isSuccessful()){
6
7     } else {
8
9 }

```

todo можно стереть, она нам больше не пригодится. Здесь добавим метод getAlbums и все будет работать. Далее:

```

1 if (response.isSuccessful()){
2     mRecyclerView.setVisibility(View.VISIBLE);
3     mErrorView.setVisibility(View.GONE);

```

```
4
5 } else {
6     mRecyclerView.setVisibility(View.GONE);
7     mErrorView.setVisibility(View.VISIBLE);
8 }
```

Копируем это и вставляем в метод onFailure, так как там должно быть точно такое же поведение. Также в конце каждого метода нам нужно скрывать наш refresher, то есть в конце методов onResponse и onFailure. Делается это следующим образом:

```
1 mRefresher.setRefreshing(false);
```

И добавим эту же строчку кода в onFailure. Теперь, если запрос у нас выполнен успешно, то мы берем и добавляем все данные в адаптер, addData. Здесь делаем response.body().getData(). Как вы видите, body у нас возвращает наши альбомы и нам нужные его data bin'ы. Далее здесь пишем true, так как нам нужно полностью перезаписать весь адаптер.

Теперь давайте посмотрим как это выглядит на эмуляторе. Введем логин. fgh@f.f. Введем пароль. Жмем «войти». Как вы видите, у нас отобразилась загрузка. Давайте попробуем обновить наш список. Список успешно обновился. Не удивляйтесь, что наши данные отличаются от тех, что будут у вас в эмуляторе. Наши данные тестовые.

В данном занятии мы успешно получили список альбомов сервера и отобразили их.

1.2.7. Добавление экрана детального отображения альбома

В данном занятии мы с вами доработаем экран альбома, который мы уже создали для того, чтобы отображать список песен. Но для начала посмотрим, что мы уже изменили. Перейдем в проект, зайдем в AlbumsHolder и увидим, что мы добавили onItemClickListener, который вызывается при нажатии на itemView в списке альбомов. В адаптере он выглядит вот так, то есть onClickListener передается в holder.bind. Мы немного изменили AlbumFragment, давайте в него перейдем. Добавили в onActivityCreated title, чтобы у нас в заголовке отображалось «альбомы». Далее в создании инстанса AlbumsAdapter мы добавили следующий, который будет открывать DetailAlbumFragment, который уже готов. Также мы создали SongsHolder и SongsAdapter. По факту это те же аналоги из списка альбомов, просто они отображают песни. SongsHolder содержит

жит два поля: title и duration.

SongsAdapter выглядит следующим образом, отображает песни. Теперь давайте перейдем в DetailAlbumFragment и изменим его, чтобы мы могли отображать список песен. Добавим new instance, нажмем Enter. Сюда будем передавать Albums.DataBean album, нажмем Enter:

```
1 args/putSerializable();
```

Добавим key. Ключом у нас будет

```
1 public static final String ALBUM_KEY = "ALBUM_KEY";
```

Далее в путь serializable добавим наш ключ и добавим album. Зайдем в albums DataBean и сделаем implements Serializable. Перейдем в DetailAlbumFragment. Далее в onActivityCreated добавим заголовок:

```
1 getActivity().setTitle();
```

Но перед этим добавим:

```
1 private Albums.DataBean mAlbum;
```

И перед установкой заголовка сделаем:

```
1 mAlbum = getArguments().getSerializable(ALBUM_KEY);
```

Далее в альбоме сделаем каст к Albums.DataBean. И в set title добавим:

```
1 getActivity().setTitle(mAlbum.getName());
```

Это нужно, чтобы при отображении экрана альбома в его заголовке устанавливалось название альбома. Теперь перейдем в метод getAlbums. Раскомментируем код, который был. Это точно такой же код, который был у нас в AlbumsFragment. Мы сейчас должны его переделать так, чтобы он получал сущность альбома.

Здесь поменяем `getAlbums` на `getAlbum`. Передадим в него `mAlbum.getId()`, как он просит. В качестве callback'а он будет получать `Album`. Изменим это везде. `AlbumAdapter` у нас теперь `mSongsAdapter`.

```
1 response.body().getData().getSongs();
```

Вот и все. Теперь давайте перейдем в эмулятор и посмотрим, как это работает. Введем логин, введем пароль. Нажмем «войти». Нажмем на какой-нибудь из альбомов, пусть будет Beatles. Как вы видите, абсолютно все песни отображаются.

В данном занятии мы доработали экран альбома так, чтобы он отображал список песен.

О проекте

Академия e-Legion – это образовательная платформа для повышения квалификации в мобильной разработке. Слушайте лекции топовых разработчиков, выполняйте практические задания и прокачивайте свои скиллы. Получите высокооплачиваемую профессию – разработчик мобильных приложений.

Программа “Многопоточность и сетевое взаимодействие”

Блок 1. Обзор средств для обеспечения многопоточности

- Знакомство с курсом
- Многопоточность и параллельное программирование
- Обзор инструментов для обеспечения многопоточности в Java (Thread, Runnable, Callable, Future, Executors)
- Обзор инструментов для обеспечения многопоточности в Android (IntentService + BroadcastReceiver, HaMeR, AsyncTask, Loaders)
- Материалы для самостоятельного изучения
- Задание на программирование. Многопоточность.

Блок 2. Service + BroadcastReceiver

- Знакомство с Service, IntentService
- Создание Service
- Бродкастресивер, знакомство
- Создание BroadcastReceiver
- Связка Activity-Service-BroadcastReceiver-Activity
- Материалы для самостоятельного изучения
- Задание на программирование. Многопоточность.

Блок 3. Многопоточность в Android

- AsyncTask, знакомство
- AsyncTask, работа
- HaMeR
- Пример работы HaMeR
- Loader, знакомство

- ContentProvider, знакомство
- Материалы для самостоятельного изучения
- Задание на программирование. Многопоточность.