Uniwersytet Wrocławski

Rafał Lewandków

PROGRAMOWANIE URZĄDZEŃ MOBILNYCH

WYKŁAD 11

- Retrofit

Retrofit is a REST (Representational state transfer) Client for Java and Android

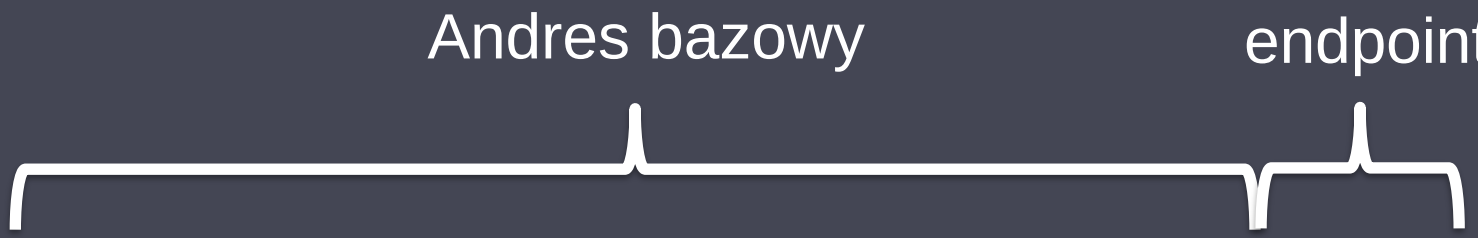
Retrofit2 pozwala na:

- dynamiczne zmienianie adresu URL serwera na podstawie danych z aplikacji
- automatyczne konwertowanie odpowiedzi z serwera na obiekty za pomocą konwerterów (np. GSON)
- wysyłanie zapytań z różnymi typami ciała (np. JSON, Form-data)
- obsługę asynchronicznego przetwarzania zapytań

Retrofit is a REST (Representational state transfer) Client for Java and Android

Retrofit2 jest biblioteką do komunikacji z serwerami HTTP, która umożliwia łatwe tworzenie interfejsów API. Dzięki niej możesz skonfigurować adres URL serwera, a następnie utworzyć interfejs, który opisuje pożądane zapytania HTTP (np. GET, POST, PUT itp.). Retrofit automatycznie mapuje odpowiedzi z serwera na obiekty w twojej aplikacji.

Retrofit

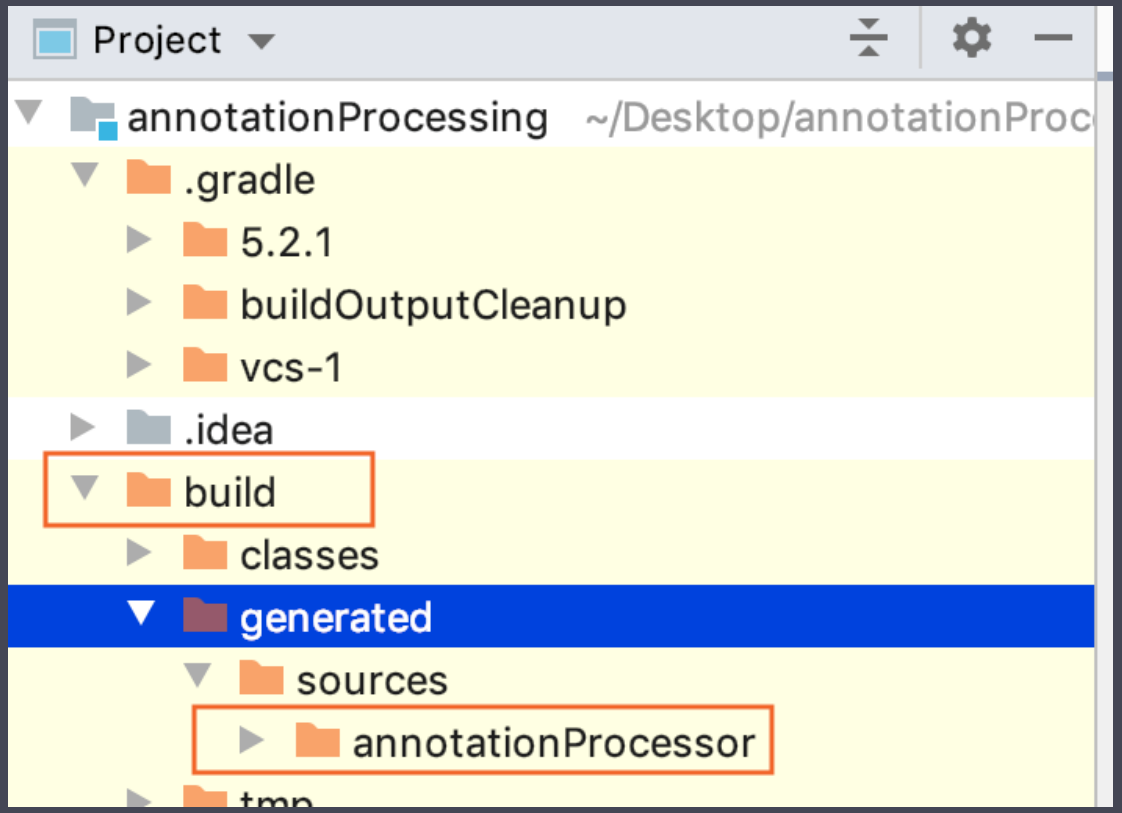



```
interface PlaceholderApi {
    @GET("posts")
    fun posts(): Call<List<Post>>
}
```

Retrofit

Annotation Processor umożliwia:

- generowanie kodu na podstawie adnotacji
- walidację elementów oznaczonych adnotacjami podczas kompilacji
- generowanie plików dodatkowych, takich jak pliki konfiguracyjne lub pliki XML



Retrofit

```
val retrofit = Retrofit.Builder()
    .baseUrl("https://jsonplaceholder.typicode.com/")
    .addConverterFactory(GsonConverterFactory.create())
    .build()
```

```
val api = retrofit.create(PlaceholderApi::class.java)
```

```
interface PlaceholderApi {
    @GET("posts")
    fun posts(): Call<List<Post>>
}
```

Retrofit

```
call.enqueue(object : Callback<List<Post>> {
    override fun onResponse(
        call: Call<List<Post>?>,
        response: Response<List<Post>?>) {
    }

    override fun onFailure(call: Call<List<Post>?>, t: Throwable) {
    }
})
```

Retrofit

GSON umożliwia:

- automatyczne mapowanie odpowiedzi JSON na obiekty Java
- konfigurację, aby ignorować pola, które nie powinny być mapowane
- mapowanie pola, które ma inną nazwę niż nazwa pola w klasie Java

```
data class Post (
    val userId: Int,
    val id: Int,
    val title: String,

    @SerializedName("body")
    val content: String
)
```

```
{
    "userId": 1,
    "id": 1,
    "title": "sunt aut ...",
    "body": "quia et ..."
},
```

Uniwersytet Wrocławski

OkHttp

OkHttp umożliwia:

- wysyłanie zapytań HTTP (np. GET, POST, PUT)
- obsługę różnych typów ciała zapytania (np. JSON, Form-data)
- automatyczne kompresowanie i deszyfrowanie danych
- obsługę certyfikatów SSL
- obsługę połączeń bezawaryjnych (np. przez proxy)

OkHttp jest bardzo popularną biblioteką w branży mobilnej i jest często używana w połączeniu z biblioteką Retrofit2, aby umożliwić łatwą i szybką komunikację z serwerem. OkHttp jest również używany przez inne popularne biblioteki jak np. Glide czy picasso.

Uniwersytet Wrocławski

LogingInterceptor

- Logowanie adresu URL połączenia,
- Logowanie nagłówków połączenia,
- Logowanie ciała zapytania,
- Logowanie odpowiedzi z serwera,
- Ustawienie poziomu logowania (np. tylko błędy, wszystko).
- Ustawienie formatera logów, który pozwala na przekształcenie logów do odpowiedniego formatu.

OkHttp

```
val interceptor = HttpLoggingInterceptor()
```

```
interceptor.setLevel(HttpLoggingInterceptor.Level.BODY)
```

```
val client = OkHttpClient.Builder()
    .addInterceptor(interceptor)
    .build()
```

```
Retrofit.Builder()
    .baseUrl(url)
    .addConverterFactory(GsonConverterFactory.create())
    .client(client)
    .build().create(PlaceholderService::class.java)
```

Uniwersytet Wrocławski

OkHttp

```
I/okhttp.OkHttpClient: --> PUT https://jsonplaceholder.typicode.com/posts/101
I/okhttp.OkHttpClient: Content-Type: application/json; charset=UTF-8
I/okhttp.OkHttpClient: Content-Length: 29
I/okhttp.OkHttpClient: {"body":"content","userId":1}
I/okhttp.OkHttpClient: --> END PUT (29-byte body)

I/okhttp.OkHttpClient: <-- 500 https://jsonplaceholder.typicode.com/posts/101 (20666ms)
...
I/okhttp.OkHttpClient: <-- END HTTP (819-byte body)
```

Podstawowe adnotacje

```
interface PlaceholderApi {
    @GET("posts")
    fun posts(): Call<List<Post>>
}
```

```
@GET("posts/{id}/comments")
fun getComments(@Path("id") postId: Int): Call<List<Comment>>
```

Uniwersytet Wrocławski

Podstawowe adnotacje

```
interface PlaceholderApi {
    @GET("posts")
    fun posts(): Call<List<Post>>
}
```

- GET /posts
- GET /posts/1
- GET /posts/1/comments
- GET /comments?postId=1
- POST /posts
- PUT /posts/1
- PATCH /posts/1
- DELETE /posts/1

Uniwersytet Wrocławski

Podstawowe adnotacje

- GET /posts
- GET /posts/1
- GET /posts/1/comments
- GET /comments?postId=1
- POST /posts
- PUT /posts/1
- PATCH /posts/1
- DELETE /posts/1

```
@GET("posts/{id}/comments")
fun getComments(@Path("id") postId: Int): Call<List<Comment>>
```

Uniwersytet Wrocławski

Podstawowe adnotacje

- GET /posts
- GET /posts/1
- GET /posts/1/comments
- GET /comments?postId=1
- POST /posts
- PUT /posts/1
- PATCH /posts/1
- DELETE /posts/1

```
@GET("comments")
fun getCommentsFromQuery(@Query("postId") postId: Int)
```

Podstawowe adnotacje

/comments?postId=1&_sort=id&_order=desc

```
@GET("comments")
fun getSortedComments(
    @Query("postId") postId: Int,
    @Query("_sort") sort: String,
    @Query("_order") order: String
): Call<List<Comment>>
```

Podstawowe adnotacje

/comments?postId=1&_sort=id&_order=desc

```
@GET
fun getComments(
    @Url url: String
): Call<List<Comment>>
```

Uniwersytet Wrocławski

Retrofit

- Umożliwia łatwe tworzenie interfejsów API
- Pozwala na dynamiczne zmienianie adresu URL serwera na podstawie danych z aplikacji
- Automatycznie mapuje odpowiedzi z serwera na obiekty w aplikacji
- Pozwala na wysyłanie zapytań z różnymi typami ciała (np. JSON, Form-data)
- Obsługa asynchronicznego przetwarzania zapytań
- Wsparcie dla wielu konwerterów (np. GSON, Moshi, Jackson)
- Obsługa adnotacji do oznaczania parametrów zapytania, nagłówków, ciała zapytania itp.