



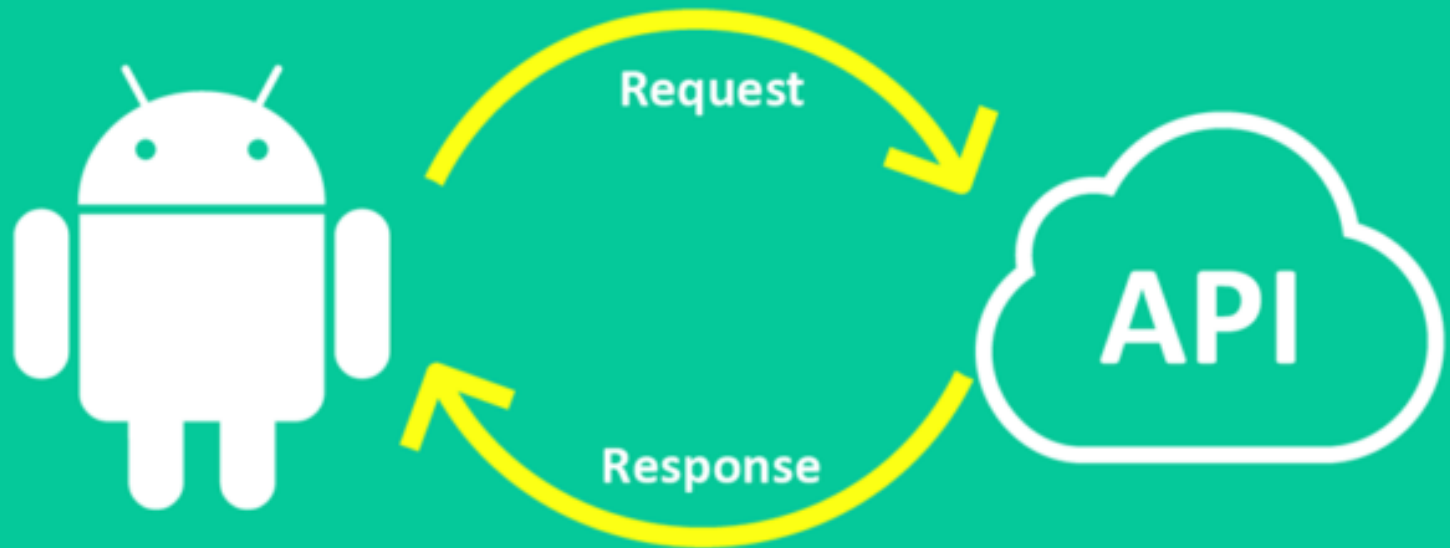
PROGRAMOWANIE URZĄDZEŃ MOBILNYCH

WYKŁAD 12

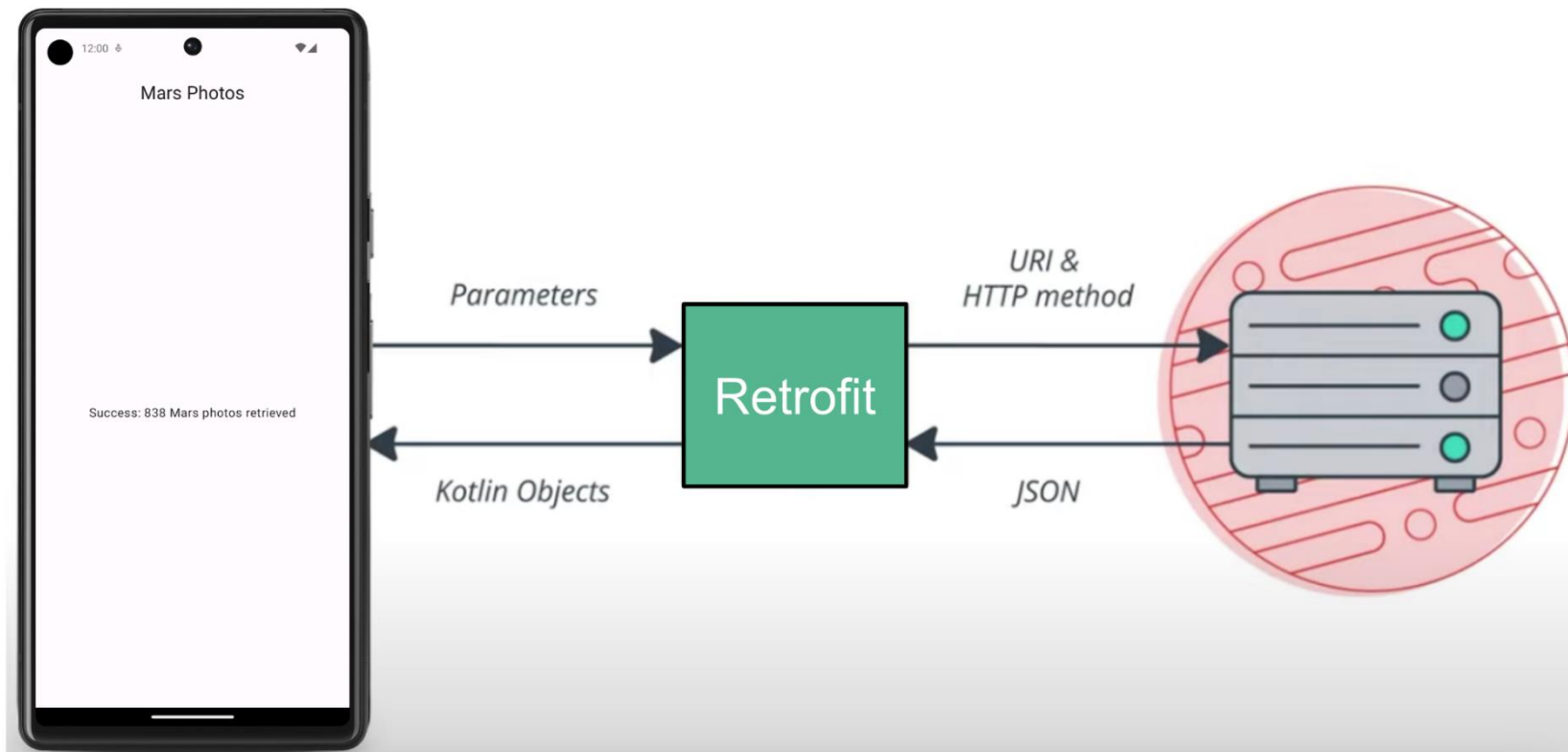
Praca z zewnętrznymi źródłami danych

- Retrofit
- Wzorzec ResourceBound

Retrofit Library in Android



Retrofit



<https://www.google.com/url?sa=i&url=https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-getting-data-internet>

Retrofit jest jedną z najpopularniejszych bibliotek wykorzystywana do wykonywania zapytań HTTP:

- **Współpracuje z biblioteką OkHttp:** Retrofit bazuje na OkHttp
- **Definiowanie interfejsu API:** Głównym zadaniem Retrofit jest ułatwienie tworzenia interfejsów do zdalnych API. Retrofit (podobnie jak ROOM) generuje implementację interfejsu automatycznie
- **Serializacja i deserializacja:** domyślnie obsługuje przekształcenia danych między formatem JSON a obiektami Javy/Kotlina
- **Obsługa zapytań HTTP:** obsługa GET, POST, PUT, DELETE, PATCH
- Obsługuje różne mechanizmy autentykacji: OAuth, Basic ...

- **GsonConverter** : To jeden z najczęściej używanych konwerterów. Wykorzystuje bibliotekę **Gson** do przekształcania danych JSON na obiekty Javy/Kotlin i vice versa. Wymaga dodatkowej zależności Gson w projekcie.

```
implementation 'com.squareup.retrofit2:converter-gson:latest_version'
```

- **MoshiConverter** : To inny popularny konwerter, który wykorzystuje bibliotekę **Moshi** do przekształcania danych JSON. Moshi jest lekką i wydajną biblioteką, która jest często wybierana przez deweloperów.

```
implementation 'com.squareup.retrofit2:converter-moshi:latest_version'
```

- **JacksonConverter** : Ten konwerter korzysta z biblioteki **Jackson** do obsługi serializacji i deserializacji danych JSON.

```
implementation 'com.squareup.retrofit2:converter-jackson:latest_version'
```

- **ScalarsConverter** : Pozwala na przekształcanie prostych typów danych, takich jak **String** , **Boolean** czy **Integer** , bezpośrednio z odpowiedzi HTTP.

```
implementation 'com.squareup.retrofit2:converter-scalars:latest_version'
```

- **SimpleXMLConverter** : Jeśli API korzysta z formatu XML, ten konwerter pozwala na przekształcanie danych XML na obiekty Javy/Kotlin.

```
implementation 'com.squareup.retrofit2:converter-simplexml:latest_version'
```

- **protobufConverter** : Ten konwerter jest przeznaczony do obsługi danych w formacie **Protocol Buffers** (**protobuf**).

```
implementation 'com.squareup.retrofit2:converter-protobuf:latest_version'
```

- **WireConverter** : **Wire** to inna biblioteka do obsługi formatu **Protocol Buffers** .

```
implementation 'com.squareup.retrofit2:converter-wire:latest_version'
```

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

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

```
{  
  "userId": 1,  
  "id": 1,  
  "title": "sunt aut ...",  
  "body": "quia et ..."  
}, uses-permission android:name="
```

```
data class Post (  
    val userId: Int,  
    val id: Int,  
    val title: String,  
  
    @SerializedName("body")  
    val content: String  
)
```

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

```
{  
  "userId": 1,  
  "id": 1,  
  "title": "sunt aut ...",  
  "body": "quia et ..."  
}, uses-permission android:name="
```

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

```
data class Post (  
    val userId: Int,  
    val id: Int,  
    val title: String,  
  
    @SerializedName("body")  
    val content: String  
)
```



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

```
{  
    "userId": 1,  
    "id": 1,  
    "title": "sunt aut ...",  
    "body": "quia et ..."  
}, uses-permission android:name="
```

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

```
data class Post (  
    val userId: Int,  
    val id: Int,  
    val title: String,  
  
    @SerializedName("body")  
    val content: String  
)
```

```
object RetrofitInstance {  
    val api: PlaceholderApi by lazy {  
        Retrofit.Builder()  
            .baseUrl("https://jsonplaceholder.typicode.com/")  
            .addConverterFactory(GsonConverterFactory.create())  
            .build()  
            .create(PlaceholderApi::class.java)  
    }  
}
```

```
class PostRepository {  
    private val api = RetrofitInstance.api  
  
    suspend fun getPosts(): List<Post>{  
        return api.posts()  
    }  
}
```

```
class PostViewModel : ViewModel() {  
    private val repository = PostRepository()  
    private val _posts = MutableStateFlow(emptyList<Post>())  
    val posts: StateFlow<List<Post>> = _posts  
  
    init {  
        getPosts()  
    }  
  
    private fun getPosts() {  
        viewModelScope.launch {  
            _posts.value = repository.getPosts()  
        }  
    }  
}
```

```
[
{
  "postId": 1,
  "id": 1,
  "name": "id labore ex et quam laborum",
  "email": "Eliseo@gardner.biz",
  "body": "laudantium enim quasi est quidem magnam voluptate ipsam eos\ntempora quo necessitatibus\ndolor quam autem quasi\nreiciendis et nam sapiente accusantium"
},
{
  "postId": 1,
  "id": 2,
  "name": "quo vero reiciendis velit similique earum",
  "email": "Jayne_Kuhic@sydney.com",
  "body": "est natus enim nihil est dolore omnis voluptatem numquam\net omnis occaecati quod ullam at\nvoluptatem error expedita pariatur\nnnihil sint nostrum voluptatem reiciendis et"
},
{
  "postId": 1,
  "id": 3,
  "name": "odio adipisci rerum aut animi",
  "email": "Nikita@garfield.biz",
  "body": "quia molestiae reprehenderit quasi aspernatur\naut expedita occaecati aliquam eveniet laudantium\nomnis quibusdam delectus saepe quia accusamus maiores nam est\ncum et ducimus et vero voluptates excepturi deleniti ratione"
},
]
```

```
data class CommentResponseItem(
    val body: String,
    val email: String,
    val id: Int,
    val name: String,
    val postId: Int
)
```

```
data class CommentResponseItem(  
    val body: String,  
    val email: String,  
    val id: Int,  
    val name: String,  
    val postId: Int  
)
```

```
interface PlaceholderApi {  
    @GET("comments")  
    suspend fun comments(): Response<List<CommentResponseItem>>  
}
```

- **Reprezentacja odpowiedzi HTTP:** Response zawiera informacje otrzymane od serwera w odpowiedzi na żądanie HTTP
- **Kod stanu HTTP:** Response zawiera kod stanu HTTP, który informuje o wyniku żądania. (404, 200 itd..)
- **Nagłówki HTTP:** Response zawiera nagłówki HTTP przekazane przez serwer
- **Przetwarzanie odpowiedzi:** Umożliwia przetwarzanie odpowiedzi w celu wyodrębnienia potrzebnych danych
- **Sprawdzenie odpowiedzi:** Testowanie poprawności i kompletności odpowiedzi

```
object RetrofitInstance {  
    val api: PlaceholderApi by lazy {  
        Retrofit.Builder()  
            .baseUrl("https://jsonplaceholder.typicode.com/")  
            .addConverterFactory(GsonConverterFactory.create())  
            .build()  
            .create(PlaceholderApi::class.java)  
    }  
}
```

```
class CommentRepository {  
    private val api = RetrofitInstance.api  
  
    suspend fun getComments() = api.comments()  
}
```

```
sealed class Resource<T> (  
    val data: T? = null,  
    val message: String? = null  
) {  
    class Success<T>(data: T) : Resource<T>(data)  
    class Error<T>(message: String, data: T? = null) : Resource<T>(data, message)  
    class Loading<T> : Resource<T>()  
}
```

- `sealed class Resource<T>` : Jest to deklaracja klasy `Resource`, która jest opakowaniem wyniku operacji. Parametr generyczny `T` oznacza typ danych, które będą przechowywane w obiekcie `Resource`.
- `val data: T? = null` : To jest pole `data`, które przechowuje wynik operacji. Może to być obiekt zawierający dane lub `null`, jeśli operacja nie zwróciła wyniku.
- `val message: String? = null` : To jest pole `message`, które może zawierać wiadomość lub komunikat związany z wynikiem operacji. Jest to przydatne do przechowywania informacji zwrotnych lub błędów.
- `class Success<T>(data: T) : Resource<T>(data)` : Jest to podklasa `Resource`, która reprezentuje sukces operacji. Przyjmuje dane (`data`) jako argument konstruktora.
- `class Error<T>(message: String, data: T? = null) : Resource<T>(data, message)` : Jest to podklasa `Resource`, która reprezentuje błąd operacji. Przyjmuje wiadomość błędu (`message`) i opcjonalnie dane związane z błędem (`data`) jako argumenty konstruktora.
- `class Loading<T> : Resource<T>()` : Jest to podklasa `Resource`, która reprezentuje stan ładowania. Nie przyjmuje żadnych danych. Jest używana do informowania interfejsu użytkownika, że operacja jest w trakcie wykonywania.

```
sealed class Resource<T> (  
    val data: T? = null,  
    val message: String? = null  
) {  
    class Success<T>(data: T) : Resource<T>(data)  
    class Error<T>(message: String, data: T? = null) : Resource<T>(data, message)  
    class Loading<T> : Resource<T>()  
}
```

```
class CommentsViewModel : ViewModel() {  
    private val repository = CommentRepository()  
    private var _comments: MutableStateFlow<Resource<List<CommentResponseItem>>> = MutableStateFlow(Resource.Loading())  
    val comments: StateFlow<Resource<List<CommentResponseItem>>> = _comments  
  
    init {  
        getCommentsList()  
    }  
  
    private fun getCommentsList() = viewModelScope.launch {  
        _comments.value = Resource.Loading()  
        val response = repository.getComments()  
        _comments.value = handleCommentsResponse(response)  
    }  
  
    private fun handleCommentsResponse(response: Response<List<CommentResponseItem>>)  
        : Resource<List<CommentResponseItem>> {  
        if (response.isSuccessful)  
            response.body()?.let { return Resource.Success(it) }  
        return Resource.Error(response.message())  
    }  
}
```


Wzorzec ResourceBound + Fragment

```
class CommentsFragment : Fragment() {  
  
    private lateinit var binding: FragmentCommentsBinding  
  
    private val viewModel: CommentsViewModel by viewModels()  
  
    override fun onCreateView(  
        inflater: LayoutInflater, container: ViewGroup?,  
        savedInstanceState: Bundle?  
    ): View {  
        binding = FragmentCommentsBinding.inflate(inflater)  
  
        val commentAdapter = CommentAdapter(CommentComparator())  
        viewLifecycleOwner.lifecycleScope.launch {  
            viewModel.comments.collectLatest { response ->  
                when (response) {  
                    is Resource.Success -> {  
                        response.data?.let { res ->  
                            commentAdapter.submitList(res)  
                        }  
                    }  
  
                    is Resource.Error -> {}  
                    is Resource.Loading -> {}  
                }  
            }  
        }  
  
        binding.recycler.apply{  
            adapter = commentAdapter  
            layoutManager = LinearLayoutManager(requireContext())  
        }  
  
        return binding.root  
    }  
}
```

Wzorzec ResourceBound + Compose

```
@Composable
fun CommentsScreen(){
    val viewModel: CommentsViewModel = viewModel()

    val response by viewModel.comments.collectAsStateWithLifecycle()

    when (response) {
        is Resource.Success -> { response.data?.let { ShowList(comments = it) } }
        is Resource.Error -> { }
        is Resource.Loading -> { }
    }
}
```

Wzorzec ResourceBound

