Uniwersytet Wrocławski

Rafał Lewandków

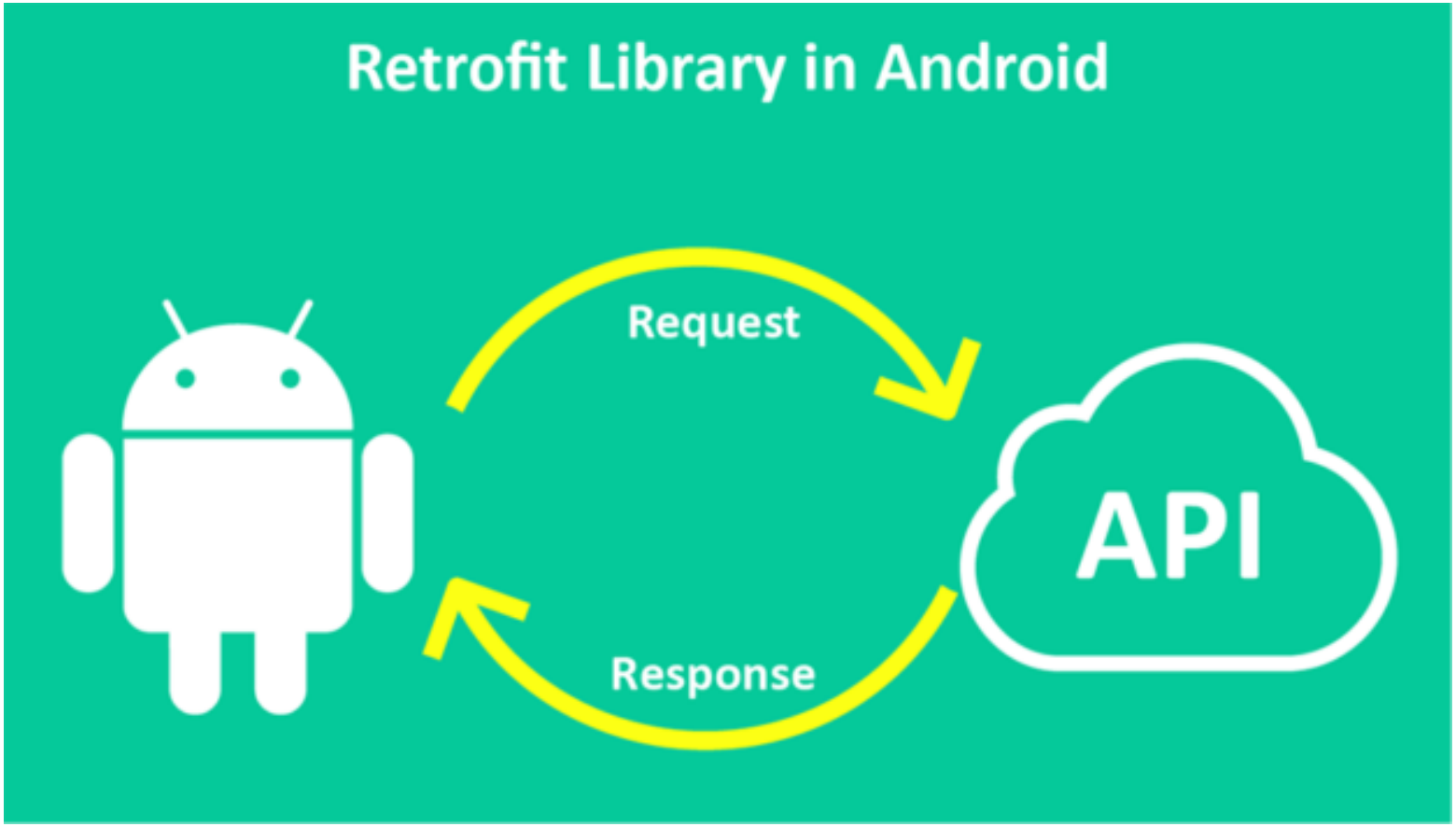
PROGRAMOWANIE URZĄDZEŃ MOBILNYCH

WYKŁAD 12

Praca z zewnętrznymi źródłami danych

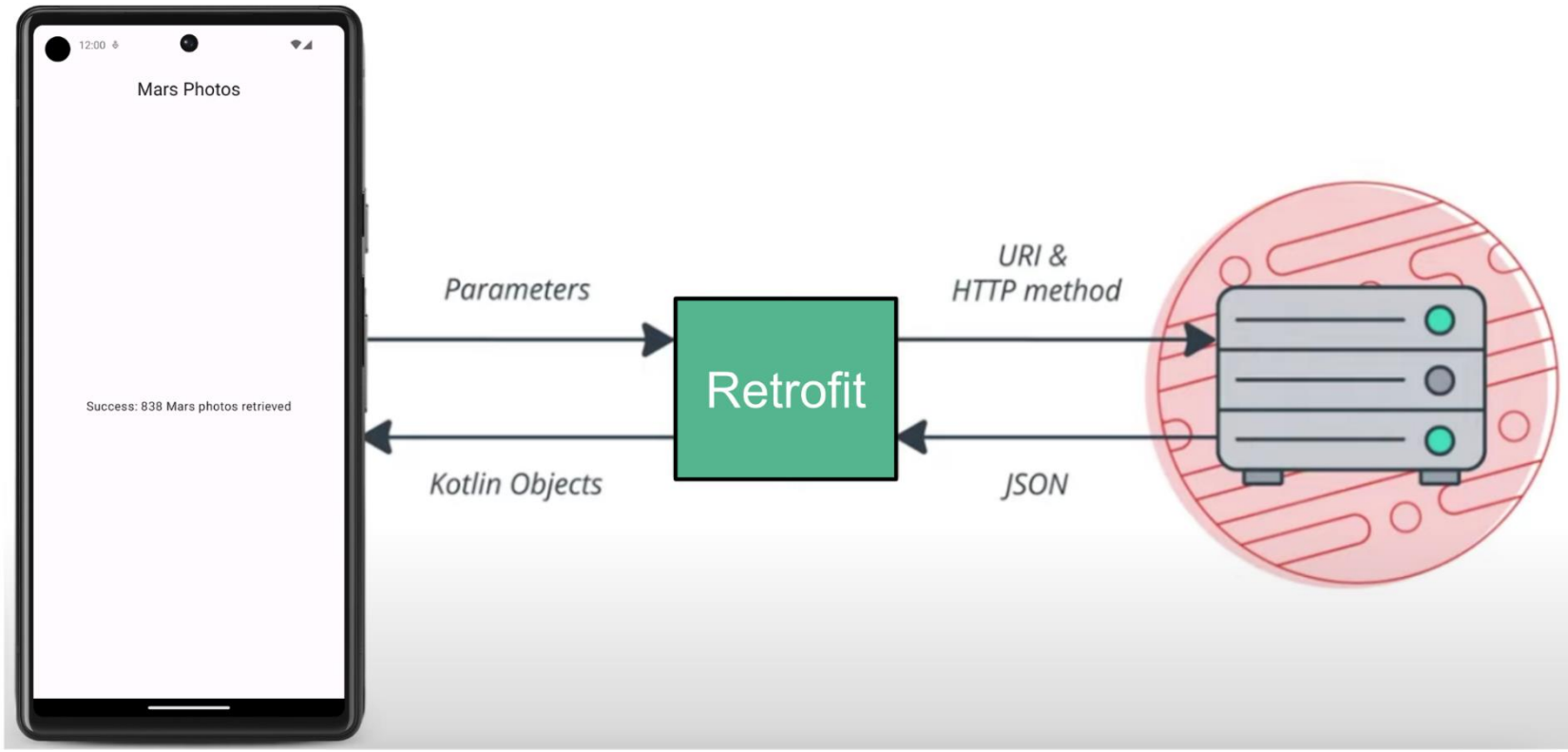
- Retrofit
- Wzorzec ResourceBound

Retrofit



https://www.google.com/url?sa=i&url=https%3A%2F%2Finnovationm.co%2Fretrofit-library-in-android%2F

Retrofit



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

Uniwersytet Wrocławski

Retrofit

`Retrofit` to popularna biblioteka do tworzenia interfejsów API w aplikacjach. Jest ona często używana do wykonywania zapytań HTTP do zdalnych serwerów i przetwarzania odpowiedzi. Jej prosty interfejs i możliwość dostosowania do różnych potrzeb sprawiają, że jest bardzo użyteczna przy tworzeniu aplikacji mobilnych, które wymagają integracji z serwerami internetowymi.

- Współpracuje z biblioteką `OkHttp`: `Retrofit` bazuje na bibliotece `OkHttp` do zarządzania połączeniami HTTP. Dzięki temu możesz korzystać z funkcji takich jak zarządzanie ciasteczkami, buforowanie, logowanie żądań i odpowiedzi oraz wiele innych.
- Definiowanie interfejsu API: Głównym celem `Retrofit` jest ułatwienie tworzenia interfejsów do zdalnych API. Tworzysz interfejs, a następnie określasz metody, które odpowiadają różnym zapytaniom HTTP. `Retrofit` **automatycznie** generuje implementację tego interfejsu, co pozwala na prostą i przejrzystą komunikację z serwerem.
- Serializacja i deserializacja: `Retrofit` domyślnie obsługuje przekształcanie danych między formatem JSON a obiektami Javy/Kotlin. Możesz dostosować sposób serializacji i deserializacji, korzystając z różnych konwerterów (np. Gson, Moshi), lub też możesz użyć niestandardowych rozwiązań.
- Obsługa różnych typów zapytań HTTP: `Retrofit` obsługuje różne typy zapytań HTTP, takie jak `GET`, `POST`, `PUT`, `DELETE`, `PATCH`, itp. Możesz określić ścieżkę, parametry, nagłówki i ciało żądania w prosty i czytelny sposób.
- Obsługa wielu endpointów: W jednej aplikacji możesz używać wielu różnych interfejsów API, co ułatwia integrację z różnymi serwerami i usługami.
- Obsługa błędów: Retrofit umożliwia definiowanie obsługi błędów, co pozwala na reagowanie na różne sytuacje podczas komunikacji z serwerem. Możesz określić, jakie działania podjąć w przypadku błędnej odpowiedzi HTTP lub problemów z połączeniem.
- Kotlin Coroutines: `Retrofit` jest często używany w połączeniu z Kotlin Coroutines, co umożliwia asynchroniczną obsługę zapytań HTTP i reaktywne programowanie.
- `Retrofit` obsługuje różne mechanizmy autentykacji, takie jak tokeny `OAuth`, autoryzacja `Basic`, czy też niestandardowe rozwiązania.

Retrofit

- `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'
```

Uniwersytet Wrocławski

Upoważnienie

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

Retrofit

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

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

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

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

Retrofit

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

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

```
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
)
```

Retrofit

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

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

```
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)
    }
}
```

Retrofit

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

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

```
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)
    }
}
```

Retrofit + MVVM

```
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()
        }
    }
}
```

Retrofit + MVVM + RecyclerView

```
class PostViewHolder(private val binding: RvItemBinding) : RecyclerView.ViewHolder(binding.root) {
    fun bind(item: Post) {
        binding.apply {
            title.text = item.title
            content.text = item.content
            userId.text = item.userId.toString()
        }
    }
}
```

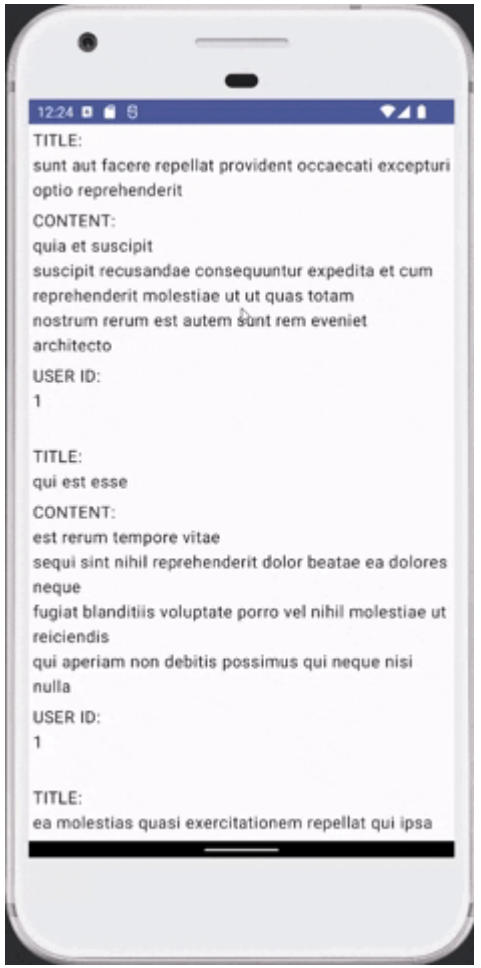
```
class PostComparator : DiffUtil.ItemCallback<Post>() {
    override fun areItemsTheSame(oldItem: Post, newItem: Post): Boolean {
        return oldItem === newItem
    }

    override fun areContentsTheSame(oldItem: Post, newItem: Post): Boolean {
        return oldItem == newItem
    }
}
```

```
class PostAdapter(userComparator: PostComparator) : ListAdapter<Post, PostViewHolder>(userComparator) {
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): PostViewHolder {
        return PostViewHolder(
            RvItemBinding.inflate(
                LayoutInflater.from(parent.context), parent, false
            )
        )
    }

    override fun onBindViewHolder(holder: PostViewHolder, position: Int) {
        val item = getItem(position)
        holder.bind(item)
    }
}
```

Uniwersytet Wrocławski

Retrofit + MVVM + RecyclerView

```
class PostFragment : Fragment() {

    private lateinit var binding: FragmentPostBinding

    private val viewModel: PostViewModel by viewModels()

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        binding = FragmentPostBinding.inflate(inflater)

        val userAdapter = PostAdapter(PostComparator())
        viewLifecycleOwner.lifecycleScope.launch {
            viewModel.posts.collectLatest { posts ->
                userAdapter.submitList(posts)
            }
        }

        binding.recycler.apply{
            adapter = userAdapter
            layoutManager = LinearLayoutManager(requireContext())
        }

        return binding.root
    }
}
```



Uniwersytet Wrocławski

Retrofit + MVVM + Compose

```
@Composable
fun PostScreen(){
    val viewModel: PostViewModel = viewModel()

    val posts by viewModel.posts.collectAsStateWithLifecycle()

    LazyColumn {
        items(posts) { post ->
            Column {
                Text(
                    text = "TITLE:\n" + post.title,
                    Modifier.padding(4.dp)
                )
                Text(
                    text = "CONTENT:\n" + post.content,
                    Modifier.padding(4.dp)
                )
                Text(
                    text = "USER ID:\n" + post.userId.toString(),
                    Modifier.padding(4.dp)
                )
                Spacer(modifier = Modifier.padding(12.dp))
            }
        }
    }
}
```

Uniwersytet Wrocławski

Obiekt Response


```
[
  {
    "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\nnihil 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"
  },
```


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

Uniwersytet Wrocławski

Obiekt Response

```
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>>
}
```

Uniwersytet Wrocławski

Obiekt Response

Zwróćmy uwagę na zastosowanie obiektu `Response`, jest to obiekt, który reprezentuje odpowiedź HTTP otrzymaną od serwera po wysłaniu żądania HTTP. Jest to ważny element przy pracy z `Retrofit`, ponieważ umożliwia analizę wyników operacji sieciowych.

- Reprezentacja odpowiedzi HTTP: `Response` zawiera wszystkie informacje otrzymane od serwera w odpowiedzi na żądanie HTTP. Obejmuje to kod stanu HTTP, nagłówki, dane i inne metadane.
- Kod stanu HTTP: Obiekt `Response` zawiera kod stanu HTTP, który informuje o wyniku żądania. Na przykład, kod stanu 200 oznacza sukces, a kody stanu 4xx i 5xx oznaczają błędy. Możesz użyć kodu stanu, aby określić, czy operacja zakończyła się sukcesem czy błędem.
- Dane odpowiedzi: `Response` może zawierać dane przesłane przez serwer. Odpowiedź może być w formacie JSON, XML, tekstu lub innym, w zależności od formatu danych przekazywanych między klientem a serwerem.
- Nagłówki HTTP: Obiekt `Response` może również zawierać nagłówki HTTP przekazane przez serwer. Nagłówki mogą zawierać różne metadane dotyczące odpowiedzi, takie jak typ treści, długość treści, dane uwierzytelniające itp.
- Przetwarzanie odpowiedzi: Za pomocą obiektów `Response` można przetwarzać odpowiedzi w celu wyodrębnienia potrzebnych danych lub informacji zwrotnych z serwera.
- Obsługa błędów: `Response` umożliwia obsługę błędów HTTP. Jeśli serwer zwraca błąd, można to wykryć na podstawie kodu stanu i podjąć odpowiednie kroki, takie jak wyświetlenie komunikatu o błędzie użytkownikowi.
- Sprawdzanie poprawności odpowiedzi: Przy użyciu `Response` można sprawdzać, czy odpowiedź jest poprawna i zawiera oczekiwane dane.
- Przykładowe operacje na obiekcie `Response` w `Retrofit` obejmują sprawdzanie kodu stanu za pomocą `response.isSuccessful()`, odczytywanie danych za pomocą `response.body()`, pobieranie nagłówków za pomocą `response.headers()`, a także obsługę błędów, jeśli odpowiedź jest niepoprawna.
- Obiekty typu `Response` są używane w celu skonkretyzowania i analizy wyników operacji sieciowych.

Uniwersytet Wrocławski

Wzorzec ResourceBound

```
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()
}
```

Uniwersytet Wrocławski

Wzorzec ResourceBound

```
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.

Uniwersytet Wrocławski

Wzorzec ResourceBound

```
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())
    }
}
```

Uniwersytet Wrocławski

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
    }
}
```

Uniwersytet Wrocławski

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