



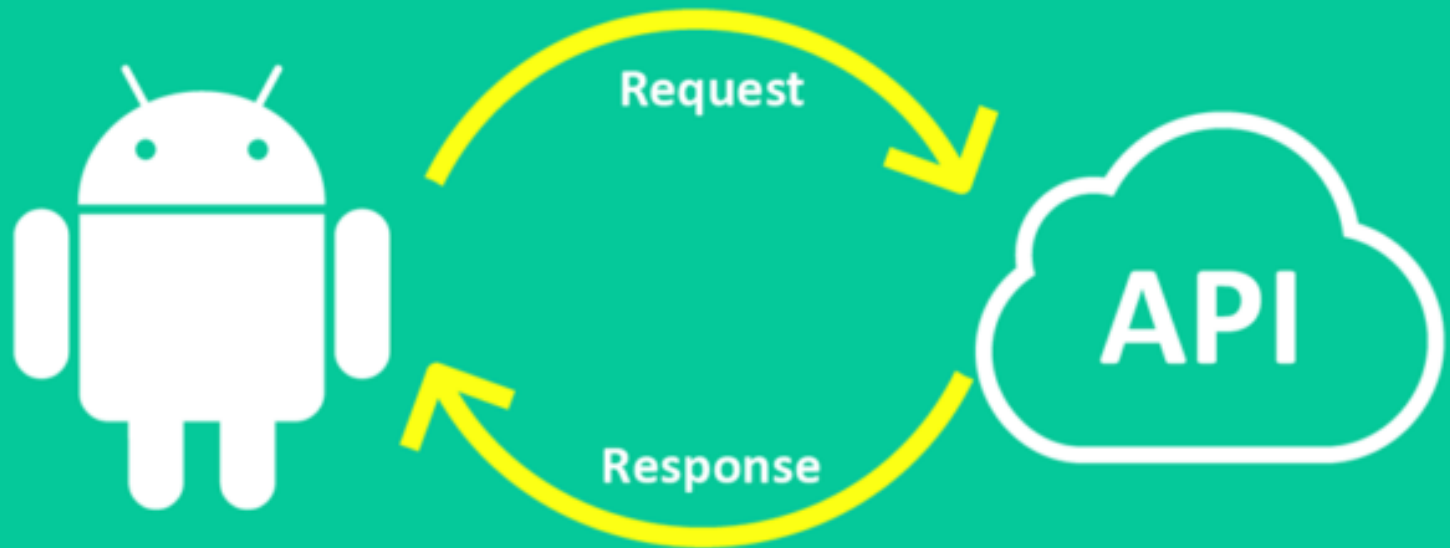
PROGRAMOWANIE URZĄDZEŃ MOBILNYCH 2

WYKŁAD 10

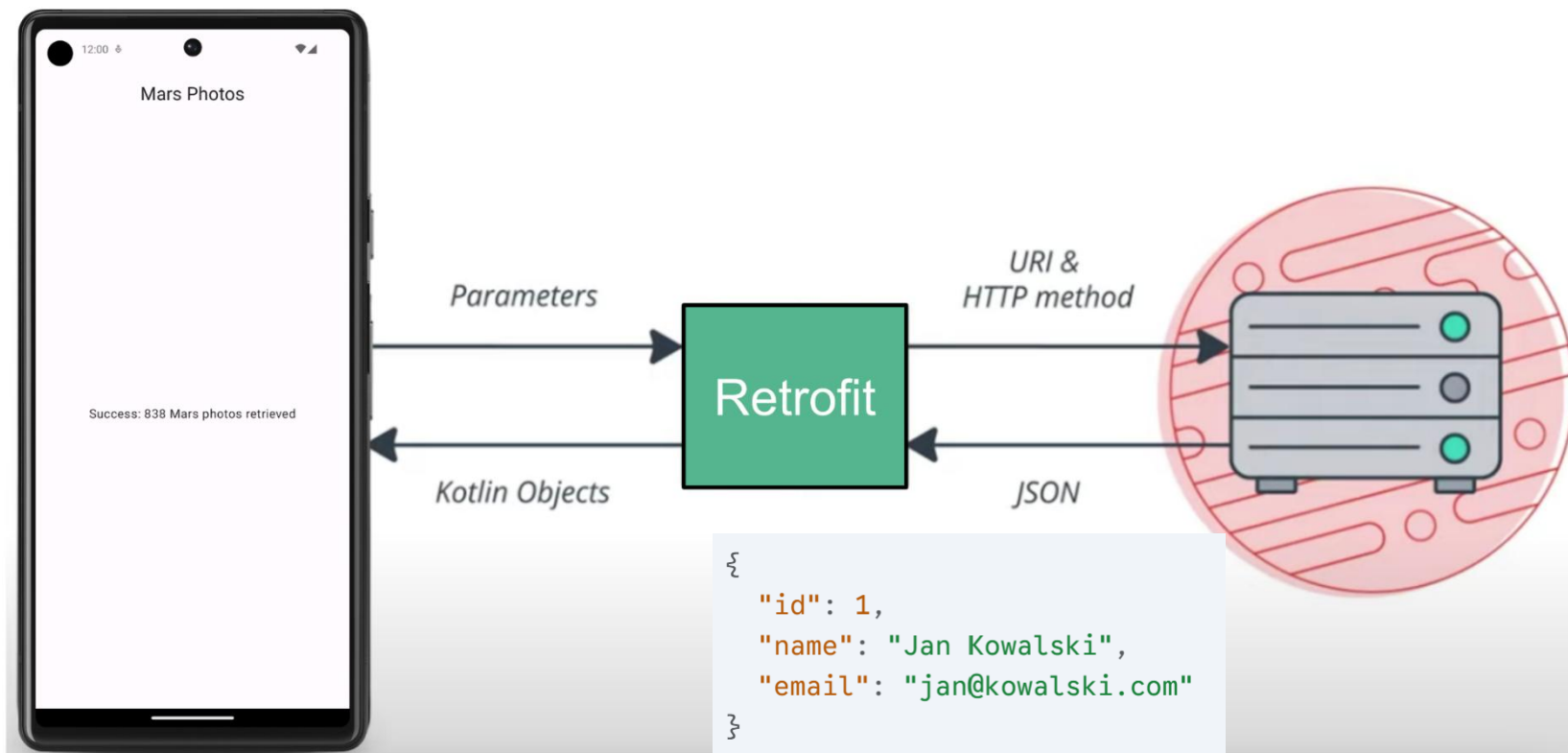
Praca z zewnętrznymi źródłami danych:

- Retrfit2
- Operacje Asynchroniczne
- Integracja z MVVM

Retrofit Library in Android



Retrofit



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 ...

- **GET - Pobieranie** danych.

Metoda GET służy wyłącznie do odczytywania zasobów z serwera. Jest to najczęstsza operacja w internecie – każdorazowe wejście na stronę internetową to zapytanie GET.

- **GET - Pobieranie** danych.

Metoda GET służy wyłącznie do odczytywania zasobów z serwera. Jest to najczęstsza operacja w internecie – każdorazowe wejście na stronę internetową to zapytanie GET.

- **POST - Tworzenie** nowego zasobu.

Metoda POST służy do wysyłania danych na serwer w celu utworzenia nowego wpisu (np. nowego użytkownika, nowego posta na blogu). Dane do utworzenia zasobu są przesyłane w ciele (body) zapytania.

- **GET - Pobieranie** danych.

Metoda GET służy wyłącznie do odczytywania zasobów z serwera. Jest to najczęstsza operacja w internecie – każdorazowe wejście na stronę internetową to zapytanie GET.

- **POST - Tworzenie** nowego zasobu.

Metoda POST służy do wysyłania danych na serwer w celu utworzenia nowego wpisu (np. nowego użytkownika, nowego posta na blogu). Dane do utworzenia zasobu są przesyłane w ciele (body) zapytania.

- **PUT - Całkowita aktualizacja** lub **zastąpienie** istniejącego zasobu.

Metoda PUT służy do aktualizacji zasobu. Wymaga przesłania w ciele zapytania **kompletnej, nowej reprezentacji** tego zasobu. Jeśli zasób o danym ID nie istnieje, PUT może go utworzyć.

- **GET - Pobieranie** danych.

Metoda GET służy wyłącznie do odczytywania zasobów z serwera. Jest to najczęstsza operacja w internecie – każdorazowe wejście na stronę internetową to zapytanie GET.

- **POST - Tworzenie** nowego zasobu.

Metoda POST służy do wysyłania danych na serwer w celu utworzenia nowego wpisu (np. nowego użytkownika, nowego posta na blogu). Dane do utworzenia zasobu są przesyłane w ciele (body) zapytania.

- **PUT - Całkowita aktualizacja** lub **zastąpienie** istniejącego zasobu.

Metoda PUT służy do aktualizacji zasobu. Wymaga przesłania w ciele zapytania **kompletnej, nowej reprezentacji** tego zasobu. Jeśli zasób o danym ID nie istnieje, PUT może go utworzyć.

- **DELETE - Usuwanie** zasobu.

Metoda DELETE służy do usuwania konkretnego zasobu na serwerze, identyfikowanego przez jego URL.

- **GET - Pobieranie** danych.

Metoda GET służy wyłącznie do odczytywania zasobów z serwera. Jest to najczęstsza operacja w internecie – każdorazowe wejście na stronę internetową to zapytanie GET.

- **POST - Tworzenie** nowego zasobu.

Metoda POST służy do wysyłania danych na serwer w celu utworzenia nowego wpisu (np. nowego użytkownika, nowego posta na blogu). Dane do utworzenia zasobu są przesyłane w ciele (body) zapytania.

- **PUT - Całkowita aktualizacja** lub **zastąpienie** istniejącego zasobu.

Metoda PUT służy do aktualizacji zasobu. Wymaga przesłania w ciele zapytania **kompletnej, nowej reprezentacji** tego zasobu. Jeśli zasób o danym ID nie istnieje, PUT może go utworzyć.

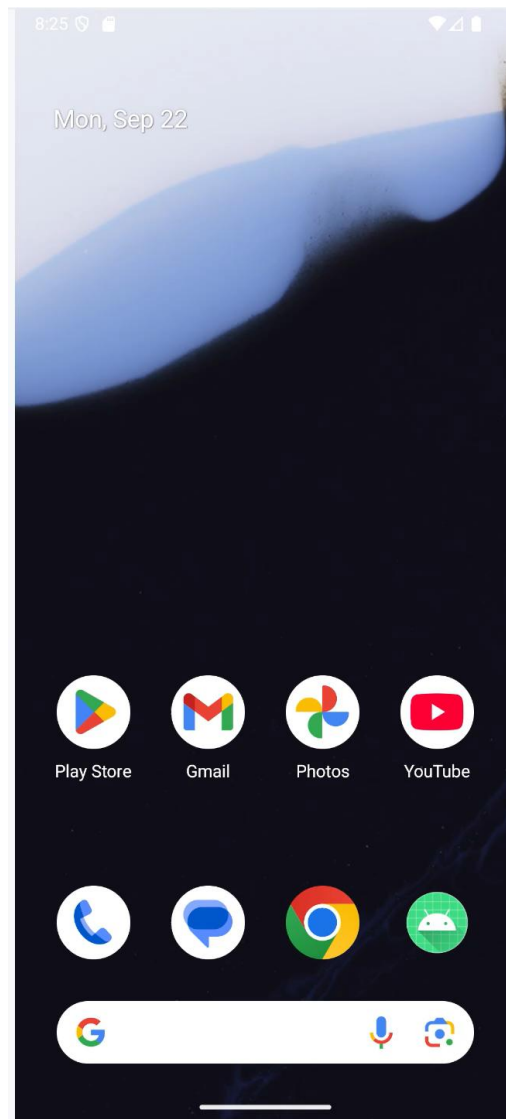
- **DELETE - Usuwanie** zasobu.

Metoda DELETE służy do usuwania konkretnego zasobu na serwerze, identyfikowanego przez jego URL.

- **PATCH - Częściowa aktualizacja** istniejącego zasobu.

Metoda PATCH jest podobna do PUT, ale znacznie bardziej elastyczna. Zamiast wysłać całą nową reprezentację zasobu, wysyłamy tylko te pola, które chcemy **zmienić**.

- **Gson Converter (converter-gson):** Konwertuje dane w formacie JSON na obiekty Kotlin/Javy (i z powrotem) przy użyciu biblioteki Gson od Google.
- **Moshi Converter (converter-moshi):** Konwertuje dane JSON za pomocą zoptymalizowanej dla Kotlin biblioteki Moshi od Square, znanej z wydajności i bezpieczeństwa.
- **Jackson Converter (converter-jackson):** Konwertuje dane JSON przy użyciu biblioteki Jackson.
- **Scalars Converter (converter-scalars):** Umożliwia traktowanie odpowiedzi serwera jako prostych typów prymitywnych, takich jak String, Int czy Boolean, bez konieczności tworzenia dedykowanej klasy modelu.
- **Simple XML Converter (converter-simplexml):** Konwertuje dane w formacie XML na obiekty Kotlin/Javy za pomocą biblioteki Simple XML.
- **Protobuf Converter (converter-protobuf):** Obsługuje Protocol Buffers, czyli wydajny, binarny format serializacji danych od Google, często używany w komunikacji gRPC.





```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

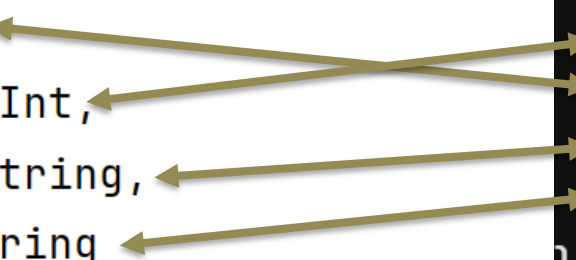
    <uses-permission android:name="android.permission.INTERNET" />

    <application...>

</manifest>
```

Każda aplikacja na Androida, która chce **połączyć się z internetem**, musi mieć **jawnie zadeklarowane** uprawnienie INTERNET w swoim **manifeście**. Bez tego system operacyjny ze względów bezpieczeństwa zablokuje wszystkie próby wykonania zapytań sieciowych

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



The diagram illustrates the mapping between a JSON object and a Kotlin data class. Four arrows point from the JSON fields to the data class properties:

- From `"userId": 1,` to `val id: Int,`
- From `"id": 1,` to `val userId: Int,`
- From `"title": "sunt aut` to `val title: String,`
- From `"body": "quia et su` to `val body: String`

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

W interfejsie definiujemy **zestaw operacji**, które można wykonać na API. Nie zawiera żadnej logiki, a jedynie opisuje, co jest możliwe.

```
interface ApiService {  
    1 Usage  
    @GET( value = "posts")  
    suspend fun getPosts(): List<Post>  
}  
1 Usage  
object RetrofitInstance {  
    1 Usage  
    private const val BASE_URL = "https://jsonplaceholder.typicode.com/"  
  
    val api: ApiService by lazy {  
        Retrofit.Builder()  
            .baseUrl( baseUrl = BASE_URL)  
            .addConverterFactory( factory = GsonConverterFactory.create())  
            .build()  
            .create(ApiService::class.java)  
    }  
}
```

W interfejsie definiujemy **zestaw operacji**, które można wykonać na API. Nie zawiera żadnej logiki, a jedynie opisuje, co jest możliwe.

To adnotacja Retrofit, informująca o wykonaniu HTTP GET. W nawiasie podajemy **endpoint**, czyli ścieżkę do zasobu, który chcemy pobrać. Pełny adres URL będzie połączeniem baseUrl i tej ścieżki (<https://jsonplaceholder.typicode.com/posts>)

```
interface ApiService {  
    1 Usage  
    @GET( value = "posts")  
    suspend fun getPosts(): List<Post>  
}  
1 Usage  
object RetrofitInstance {  
    1 Usage  
    private const val BASE_URL = "https://jsonplaceholder.typicode.com/"  
  
    val api: ApiService by lazy {  
        Retrofit.Builder()  
            .baseUrl( baseUrl = BASE_URL )  
            .addConverterFactory( factory = GsonConverterFactory.create() )  
            .build()  
            .create(ApiService::class.java)  
    }  
}
```

To adnotacja Retrofit, informująca o wykonaniu HTTP GET. W nawiasie podajemy **endpoint**, czyli ścieżkę do zasobu, który chcemy pobrać. Pełny adres URL będzie połączeniem baseUrl i tej ścieżki (<https://jsonplaceholder.typicode.com/posts>)

```
interface ApiService {  
    1 Usage  
    @GET( value = "posts")  
    suspend fun getPosts(): List<Post>  
}  
1 Usage  
object RetrofitInstance {  
    1 Usage  
    private const val BASE_URL = "https://jsonplaceholder.typicode.com/"  
  
    val api: ApiService by lazy {  
        Retrofit.Builder()  
            .baseUrl( baseUrl = BASE_URL)  
            .addConverterFactory( factory = GsonConverterFactory.create())  
            .build()  
            .create(ApiService::class.java)  
    }  
}
```

W interfejsie definiujemy **zestaw operacji**, które można wykonać na API. Nie zawiera żadnej logiki, a jedynie opisuje, co jest możliwe.

Funkcja pobierająca dane z API. Konkretna implementacja zostanie automatycznie dostarczona przez Retrofit

To adnotacja Retrofit, informująca o wykonaniu HTTP GET. W nawiasie podajemy **endpoint**, czyli ścieżkę do zasobu, który chcemy pobrać. Pełny adres URL będzie połączeniem baseUrl i tej ścieżki (<https://jsonplaceholder.typicode.com/posts>)

W interfejsie definiujemy **zestaw operacji**, które można wykonać na API. Nie zawiera żadnej logiki, a jedynie opisuje, co jest możliwe.

Funkcja pobierająca dane z API. Konkretna implementacja zostanie automatycznie dostarczona przez Retrofit

Definiuje bazowy, stały adres URL serwera API

```
interface ApiService {  
    1 Usage  
    @GET( value = "posts")  
    suspend fun getPosts(): List<Post>  
}  
1 Usage  
object RetrofitInstance {  
    1 Usage  
    private const val BASE_URL = "https://jsonplaceholder.typicode.com/"  
  
    val api: ApiService by lazy {  
        Retrofit.Builder()  
            .baseUrl( baseUrl = BASE_URL)  
            .addConverterFactory( factory = GsonConverterFactory.create())  
            .build()  
            .create(ApiService::class.java)  
    }  
}
```

To adnotacja Retrofit, informująca o wykonaniu HTTP GET. W nawiasie podajemy **endpoint**, czyli ścieżkę do zasobu, który chcemy pobrać. Pełny adres URL będzie połączeniem baseUrl i tej ścieżki (<https://jsonplaceholder.typicode.com/posts>)

```
interface ApiService {  
    1 Usage  
    @GET( value = "posts")  
    suspend fun getPosts(): List<Post>  
}
```

W interfejsie definiujemy **zestaw operacji**, które można wykonać na API. Nie zawiera żadnej logiki, a jedynie opisuje, co jest możliwe.

Funkcja pobierająca dane z API. Konkretna implementacja zostanie automatycznie dostarczona przez Retrofit

```
object RetrofitInstance {  
    1 Usage  
    private const val BASE_URL = "https://jsonplaceholder.typicode.com/"  
  
    val api: ApiService by lazy {  
        Retrofit.Builder()  
            .baseUrl( baseUrl = BASE_URL)  
            .addConverterFactory( factory = GsonConverterFactory.create())  
            .build()  
            .create(ApiService::class.java)  
    }  
}
```

Ustawia główny adres serwera

Wzorzec Budowniczy

Definiuje bazowy, stały adres URL serwera API

W interfejsie definiujemy **zestaw operacji**, które można wykonać na API. Nie zawiera żadnej logiki, a jedynie opisuje, co jest możliwe.

Funkcja pobierająca dane z API. Konkretna implementacja zostanie automatycznie dostarczona przez Retrofit

Ustawia główny adres serwera

Wskazuje, który konwerter zostanie wykorzystany. GsonConverterFactory będzie *tłumaczyć* JSON na obiekty Kotliny

```
interface ApiService {  
    1 Usage  
    @GET( value = "posts")  
    suspend fun getPosts(): List<Post>  
}  
1 Usage  
object RetrofitInstance {  
    1 Usage  
    private const val BASE_URL = "https://jsonplaceholder.typicode.com/"  
  
    val api: ApiService by lazy {  
        Retrofit.Builder()  
            .baseUrl( baseUrl = BASE_URL )  
            .addConverterFactory( factory = GsonConverterFactory.create() )  
            .build()  
            .create(ApiService::class.java)  
    }  
}
```

To adnotacja Retrofit, informująca o wykonaniu HTTP GET. W nawiasie podajemy **endpoint**, czyli ścieżkę do zasobu, który chcemy pobrać. Pełny adres URL będzie połączeniem baseUrl i tej ścieżki (https://jsonplaceholder.typicode.com/posts)

Definiuje bazowy, stały adres URL serwera API

Wzorzec Budowniczy

W interfejsie definiujemy **zestaw operacji**, które można wykonać na API. Nie zawiera żadnej logiki, a jedynie opisuje, co jest możliwe.

Funkcja pobierająca dane z API. Konkretna implementacja zostanie automatycznie dostarczona przez Retrofit

Ustawia główny adres serwera

Wskazuje, który konwerter zostanie wykorzystany. GsonConverterFactory będzie *tłumaczyć* JSON na obiekty Kotliny

```
interface ApiService {  
    1 Usage  
    @GET( value = "posts")  
    suspend fun getPosts(): List<Post>  
}  
1 Usage  
object RetrofitInstance {  
    1 Usage  
    private const val BASE_URL = "https://jsonplaceholder.typicode.com/"  
  
    val api: ApiService by lazy {  
        Retrofit.Builder()  
            .baseUrl( baseUrl = BASE_URL )  
            .addConverterFactory( factory = GsonConverterFactory.create() )  
            .build()  
            .create(ApiService::class.java)  
    }  
}
```

To adnotacja Retrofit, informująca o wykonaniu HTTP GET. W nawiasie podajemy **endpoint**, czyli ścieżkę do zasobu, który chcemy pobrać. Pełny adres URL będzie połączeniem baseUrl i tej ścieżki (https://jsonplaceholder.typicode.com/posts)

Definiuje bazowy, stały adres URL serwera API

Wzorzec Budowniczy

Przekazujemy do Retrofit interfejs ApiService ze zdefiniowanymi metodami. Po wywołaniu create **automatycznie generowana** jest konkretna **implementacja interfejsu**, dzięki której możemy wykonać zdefiniowane w nim **żądania sieciowe**

Klasa stanu UI, która w jednym obiekcie **grupuje** wszystkie **informacje** potrzebne do narysowania ekranu.

```
data class PostUiState(  
    val posts: List<Post> = emptyList(),  
    val isLoading: Boolean = true,  
    val error: String? = null  
)  
  
3 Usages  
class PostViewModel(private val repository: PostRepository) : ViewModel()  
    4 Usages  
    private val _uiState = MutableStateFlow( value = PostUiState())  
    1 Usage  
    val uiState: StateFlow<PostUiState> = _uiState.asStateFlow()  
    init {  
        fetchPosts()  
    }  
    1 Usage  
    private fun fetchPosts() {  
        viewModelScope.launch {  
            _uiState.update { it.copy(isLoading = true) }  
            try {  
                val posts = repository.getPosts()  
                _uiState.update { PostUiState(posts = posts,  
                    isLoading = false) }  
            } catch (e: Exception) {  
                _uiState.update { PostUiState(error =  
                    "Nie udało się załadować danych: ${e.message}",  
                    isLoading = false) }  
            }  
        }  
    }  
}
```

Retrofit

Przechowuje listę postów do wyświetlenia; domyślnie jest pusta.

Przechowuje komunikat o błędzie, jeśli wystąpi; domyślnie null, oznaczając brak błędu.

Klasa stanu UI, która w jednym obiekcie **grupuje** wszystkie **informacje** potrzebne do narysowania ekranu.

Informuje, czy dane są w **trakcie ładowania**; domyślnie true, aby pokazać wskaźnik ładowania przy starcie

```
data class PostUiState(  
    val posts: List<Post> = emptyList(),  
    val isLoading: Boolean = true,  
    val error: String? = null  
)  
3 Usages  
class PostViewModel(private val repository: PostRepository) : ViewModel()  
4 Usages  
    private val _uiState = MutableStateFlow<PostUiState> { value = PostUiState() }  
1 Usage  
    val uiState: StateFlow<PostUiState> = _uiState.asStateFlow()  
    init {  
        fetchPosts()  
    }  
1 Usage  
    private fun fetchPosts() {  
        viewModelScope.launch {  
            _uiState.update { it.copy(isLoading = true) }  
            try {  
                val posts = repository.getPosts()  
                _uiState.update { PostUiState(posts = posts, isLoading = false) }  
            } catch (e: Exception) {  
                _uiState.update { PostUiState(error = "Nie udało się załadować danych: ${e.message}", isLoading = false) }  
            }  
        }  
    }  
}
```


Retrofit

Przechowuje listę postów do wyświetlenia; domyślnie jest pusta.

Przechowuje komunikat o błędzie, jeśli wystąpi; domyślnie null, oznaczając brak błędu.

Klasa stanu UI, która w jednym obiekcie **grupuje** wszystkie **informacje** potrzebne do narysowania ekranu.

Informuje, czy dane są w **trakcie ładowania**; domyślnie true, aby pokazać wskaźnik ładowania przy starcie

przechowuje aktualny stan UI

```
data class PostUiState(  
    val posts: List<Post> = emptyList(),  
    val isLoading: Boolean = true,  
    val error: String? = null  
)  
3 Usages  
class PostViewModel(private val repository: PostRepository) : ViewModel()  
4 Usages  
    private val _uiState = MutableStateFlow<PostUiState> { value = PostUiState() }  
1 Usage  
    val uiState: StateFlow<PostUiState> = _uiState.asStateFlow()  
    init {  
        fetchPosts()  
    }  
1 Usage  
    private fun fetchPosts() {  
        viewModelScope.launch {  
            _uiState.update { it.copy(isLoading = true) }  
            try {  
                val posts = repository.getPosts()  
                _uiState.update { PostUiState(posts = posts, isLoading = false) }  
            } catch (e: Exception) {  
                _uiState.update { PostUiState(error = "Nie udało się załadować danych: ${e.message}", isLoading = false) }  
            }  
        }  
    }  
}
```

Retrofit

Przechowuje listę postów do wyświetlenia; domyślnie jest pusta.

Przechowuje komunikat o błędzie, jeśli wystąpi; domyślnie null, oznaczając brak błędu.

Klasa stanu UI, która w jednym obiekcie **grupuje** wszystkie **informacje** potrzebne do narysowania ekranu.

Informuje, czy dane są w **trakcie ładowania**; domyślnie true, aby pokazać wskaźnik ładowania przy starcie

przechowuje aktualny stan UI

Aktualizuje stan, aby pokazać **wskaźnik ładowania**. Użycie `update` jest bezpieczne wątkowo.

```
data class PostUiState(  
    val posts: List<Post> = emptyList(),  
    val isLoading: Boolean = true,  
    val error: String? = null  
)  
  
3 Usages  
class PostViewModel(private val repository: PostRepository) : ViewModel()  
    4 Usages  
    private val _uiState = MutableStateFlow<PostUiState> { value = PostUiState() }  
    1 Usage  
    val uiState: StateFlow<PostUiState> = _uiState.asStateFlow()  
    init {  
        fetchPosts()  
    }  
    1 Usage  
    private fun fetchPosts() {  
        viewModelScope.launch {  
            _uiState.update { it.copy(isLoading = true) }  
            try {  
                val posts = repository.getPosts()  
                _uiState.update { PostUiState(posts = posts,  
                    isLoading = false) }  
            } catch (e: Exception) {  
                _uiState.update { PostUiState(error =  
                    "Nie udało się załadować danych: ${e.message}",  
                    isLoading = false) }  
            }  
        }  
    }  
}
```


Retrofit

Przechowuje listę postów do wyświetlenia; domyślnie jest pusta.

Przechowuje komunikat o błędzie, jeśli wystąpi; domyślnie null, oznaczając brak błędu.

Klasa stanu UI, która w jednym obiekcie **grupuje** wszystkie **informacje** potrzebne do narysowania ekranu.

Informuje, czy dane są w **trakcie ładowania**; domyślnie true, aby pokazać wskaźnik ładowania przy starcie

przechowuje aktualny stan UI

Aktualizuje stan, aby pokazać **wskaźnik ładowania**. Użycie update jest bezpieczne wątkowo.

W przypadku sukcesu, aktualizuje stan **nową listą postów** i ukrywa wskaźnik ładowania.

```
data class PostUiState(  
    val posts: List<Post> = emptyList(),  
    val isLoading: Boolean = true,  
    val error: String? = null  
)  
  
3 Usages  
class PostViewModel(private val repository: PostRepository) : ViewModel()  
    4 Usages  
    private val _uiState = MutableStateFlow<PostUiState> { value = PostUiState() }  
    1 Usage  
    val uiState: StateFlow<PostUiState> = _uiState.asStateFlow()  
    init {  
        fetchPosts()  
    }  
    1 Usage  
    private fun fetchPosts() {  
        viewModelScope.launch {  
            _uiState.update { it.copy(isLoading = true) }  
            try {  
                val posts = repository.getPosts()  
                _uiState.update { PostUiState(posts = posts,  
                    isLoading = false) }  
            } catch (e: Exception) {  
                _uiState.update { PostUiState(error =  
                    "Nie udało się załadować danych: ${e.message}",  
                    isLoading = false) }  
            }  
        }  
    }  
}
```

Przechowuje listę postów do wyświetlenia; domyślnie jest pusta.

Przechowuje komunikat o błędzie, jeśli wystąpi; domyślnie null, oznaczając brak błędu.

Klasa stanu UI, która w jednym obiekcie **grupuje** wszystkie **informacje** potrzebne do narysowania ekranu.

Informuje, czy dane są w **trakcie ładowania**; domyślnie true, aby pokazać wskaźnik ładowania przy starcie

przechowuje aktualny stan UI

Aktualizuje stan, aby pokazać **wskaźnik ładowania**. Użycie update jest bezpieczne wątkowo.

W przypadku sukcesu, aktualizuje stan **nową listą postów** i ukrywa wskaźnik ładowania.

W przypadku błędu (wyjątku), aktualizuje stan **komunikatem o błędzie** i również ukrywa wskaźnik ładowania.

```
data class PostUiState(  
    val posts: List<Post> = emptyList(),  
    val isLoading: Boolean = true,  
    val error: String? = null  
)  
  
3 Usages  
class PostViewModel(private val repository: PostRepository) : ViewModel()  
    4 Usages  
    private val _uiState = MutableStateFlow<PostUiState> { value = PostUiState() }  
    1 Usage  
    val uiState: StateFlow<PostUiState> = _uiState.asStateFlow()  
    init {  
        fetchPosts()  
    }  
    1 Usage  
    private fun fetchPosts() {  
        viewModelScope.launch {  
            _uiState.update { it.copy(isLoading = true) }  
            try {  
                val posts = repository.getPosts()  
                _uiState.update { PostUiState(posts = posts,  
                    isLoading = false) }  
            } catch (e: Exception) {  
                _uiState.update { PostUiState(error =  
                    "Nie udało się załadować danych: ${e.message}",  
                    isLoading = false) }  
            }  
        }  
    }  
}
```

```
Scaffold(topBar = { TopAppBar(title = { Text(text = "Posty z JSONPlaceholder") }) }) {  
    Box(...) {  
        when {  
            uiState.isLoading -> {  
                CircularProgressIndicator()  
            }  
            uiState.error != null -> {  
                Text(text = uiState.error!!, color = MaterialTheme.colorScheme.error)  
            }  
            else -> {  
                LazyColumn(contentPadding = PaddingValues(all = 16.dp)) {  
                    items(items = uiState.posts) { post ->  
                        PostItem(post = post)  
                        Spacer(modifier = Modifier.height(height = 8.dp))  
                    }  
                }  
            }  
        }  
    }  
}
```

Jeśli isLoading ma wartość true, wyświetlany jest tylko CircularProgressIndicator.

Jeśli pole error nie jest null, wyświetlany jest Text z komunikatem o błędzie.

Jeśli nie ma ładowania ani błędu, oznacza to, że dane zostały pobrane pomyślnie. Wyświetlana jest LazyColumn z listą postów

1. Adnotacja @Query

- **Cel:** Dodawanie dynamicznych parametrów do adresu URL zapytania. Jest to podstawowy mechanizm do **filtrowania, sortowania lub stronicowania** wyników.

1. Adnotacja @Query

- **Cel:** Dodawanie dynamicznych parametrów do adresu URL zapytania. Jest to podstawowy mechanizm do **filtrowania, sortowania lub stronicowania** wyników.
- **Zakres stosowalności:**
 - **Filtrowanie wyników:** ?userId=1, ?status=published (pobierz zasoby o określonym statusie).

1. Adnotacja @Query

- **Cel:** Dodawanie dynamicznych parametrów do adresu URL zapytania. Jest to podstawowy mechanizm do **filtrowania, sortowania lub stronicowania** wyników.
- **Zakres stosowalności:**
 - **Filtrowanie wyników:** ?userId=1, ?status=published (pobierz zasoby o określonym statusie).
 - **Sortowanie:** ?sortBy=date&order=desc (posortuj wyniki po dacie, malejąco).

1. Adnotacja @Query

- **Cel:** Dodawanie dynamicznych parametrów do adresu URL zapytania. Jest to podstawowy mechanizm do **filtrowania, sortowania lub stronicowania** wyników.
- **Zakres stosowalności:**
 - **Filtrowanie wyników:** ?userId=1, ?status=published (pobierz zasoby o określonym statusie).
 - **Sortowanie:** ?sortBy=date&order=desc (posortuj wyniki po dacie, malejąco).
 - **Paginacja:** ?page=2&limit=20 (pobierz drugą stronę wyników, po 20 na stronie).

1. Adnotacja @Query

- **Cel:** Dodawanie dynamicznych parametrów do adresu URL zapytania. Jest to podstawowy mechanizm do **filtrowania, sortowania lub stronicowania** wyników.
- **Zakres stosowalności:**
 - **Filtrowanie wyników:** ?userId=1, ?status=published (pobierz zasoby o określonym statusie).
 - **Sortowanie:** ?sortBy=date&order=desc (posortuj wyniki po dacie, malejąco).
 - **Paginacja:** ?page=2&limit=20 (pobierz drugą stronę wyników, po 20 na stronie).
 - **Wyszukiwanie:** ?q=kotlin (prześlij zapytanie do wyszukiwarki).

1. Adnotacja @Query

- **Cel:** Dodawanie dynamicznych parametrów do adresu URL zapytania. Jest to podstawowy mechanizm do **filtrowania, sortowania lub stronicowania** wyników.
- **Zakres stosowalności:**
 - **Filtrowanie wyników:** ?userId=1, ?status=published (pobierz zasoby o określonym statusie).
 - **Sortowanie:** ?sortBy=date&order=desc (posortuj wyniki po dacie, malejąco).
 - **Paginacja:** ?page=2&limit=20 (pobierz drugą stronę wyników, po 20 na stronie).
 - **Wyszukiwanie:** ?q=kotlin (prześlij zapytanie do wyszukiwarki).

2. Adnotacja @Header

- **Cel:** Dodawanie **nagłówków** do zapytania HTTP. Nagłówki przenoszą **metadane** o samym zapytaniu, a nie o zasobie, który chcemy pobrać.

1. Adnotacja @Query

- **Cel:** Dodawanie dynamicznych parametrów do adresu URL zapytania. Jest to podstawowy mechanizm do **filtrowania, sortowania lub stronicowania** wyników.
- **Zakres stosowalności:**
 - **Filtrowanie wyników:** ?userId=1, ?status=published (pobierz zasoby o określonym statusie).
 - **Sortowanie:** ?sortBy=date&order=desc (posortuj wyniki po dacie, malejąco).
 - **Paginacja:** ?page=2&limit=20 (pobierz drugą stronę wyników, po 20 na stronie).
 - **Wyszukiwanie:** ?q=kotlin (prześlij zapytanie do wyszukiwarki).

2. Adnotacja @Header

- **Cel:** Dodawanie **nagłówków** do zapytania HTTP. Nagłówki przenoszą **metadane** o samym zapytaniu, a nie o zasobie, który chcemy pobrać.
- **Zakres stosowalności:**
 - **Autoryzacja / Uwierzytelnianie:** Przesyłanie tokenów dostępu (np. Authorization: Bearer <token>). To najczęstsze zastosowanie.

1. Adnotacja @Query

- **Cel:** Dodawanie dynamicznych parametrów do adresu URL zapytania. Jest to podstawowy mechanizm do **filtrowania, sortowania lub stronicowania** wyników.
- **Zakres stosowalności:**
 - **Filtrowanie wyników:** ?userId=1, ?status=published (pobierz zasoby o określonym statusie).
 - **Sortowanie:** ?sortBy=date&order=desc (posortuj wyniki po dacie, malejąco).
 - **Paginacja:** ?page=2&limit=20 (pobierz drugą stronę wyników, po 20 na stronie).
 - **Wyszukiwanie:** ?q=kotlin (prześlij zapytanie do wyszukiwarki).

2. Adnotacja @Header

- **Cel:** Dodawanie **nagłówków** do zapytania HTTP. Nagłówki przenoszą **metadane** o samym zapytaniu, a nie o zasobie, który chcemy pobrać.
- **Zakres stosowalności:**
 - **Autoryzacja / Uwierzytelnianie:** Przesyłanie tokenów dostępu (np. Authorization: Bearer <token>). To najczęstsze zastosowanie.
 - **Określanie Typu Treści:** Informowanie serwera, w jakim formacie akceptujemy odpowiedź (Accept: application/json) lub w jakim formacie wysyłamy dane (Content-Type: application/json).

1. Adnotacja @Query

- **Cel:** Dodawanie dynamicznych parametrów do adresu URL zapytania. Jest to podstawowy mechanizm do **filtrowania, sortowania lub stronicowania** wyników.
- **Zakres stosowalności:**
 - **Filtrowanie wyników:** ?userId=1, ?status=published (pobierz zasoby o określonym statusie).
 - **Sortowanie:** ?sortBy=date&order=desc (posortuj wyniki po dacie, malejąco).
 - **Paginacja:** ?page=2&limit=20 (pobierz drugą stronę wyników, po 20 na stronie).
 - **Wyszukiwanie:** ?q=kotlin (prześlij zapytanie do wyszukiwarki).

2. Adnotacja @Header

- **Cel:** Dodawanie **nagłówków** do zapytania HTTP. Nagłówki przenoszą **metadane** o samym zapytaniu, a nie o zasobie, który chcemy pobrać.
- **Zakres stosowalności:**
 - **Autoryzacja / Uwierzytelnianie:** Przesyłanie tokenów dostępu (np. Authorization: Bearer <token>). To najczęstsze zastosowanie.
 - **Określanie Typu Treści:** Informowanie serwera, w jakim formacie akceptujemy odpowiedź (Accept: application/json) lub w jakim formacie wysyłamy dane (Content-Type: application/json).
 - **Zarządzanie Cache:** Przesyłanie instrukcji dotyczących buforowania (Cache-Control: no-cache).

1. Adnotacja @Query

- **Cel:** Dodawanie dynamicznych parametrów do adresu URL zapytania. Jest to podstawowy mechanizm do **filtrowania, sortowania lub stronicowania** wyników.
- **Zakres stosowalności:**
 - **Filtrowanie wyników:** ?userId=1, ?status=published (pobierz zasoby o określonym statusie).
 - **Sortowanie:** ?sortBy=date&order=desc (posortuj wyniki po dacie, malejąco).
 - **Paginacja:** ?page=2&limit=20 (pobierz drugą stronę wyników, po 20 na stronie).
 - **Wyszukiwanie:** ?q=kotlin (prześlij zapytanie do wyszukiwarki).

2. Adnotacja @Header

- **Cel:** Dodawanie **nagłówków** do zapytania HTTP. Nagłówki przenoszą **metadane** o samym zapytaniu, a nie o zasobie, który chcemy pobrać.
- **Zakres stosowalności:**
 - **Autoryzacja / Uwierzytelnianie:** Przesyłanie tokenów dostępu (np. Authorization: Bearer <token>). To najczęstsze zastosowanie.
 - **Określanie Typu Treści:** Informowanie serwera, w jakim formacie akceptujemy odpowiedź (Accept: application/json) lub w jakim formacie wysyłamy dane (Content-Type: application/json).
 - **Zarządzanie Cache:** Przesyłanie instrukcji dotyczących buforowania (Cache-Control: no-cache).
 - **Przesyłanie Kluczy API:** Niektóre serwisy wymagają przesyłania klucza API w nagłówku, a nie w URL.

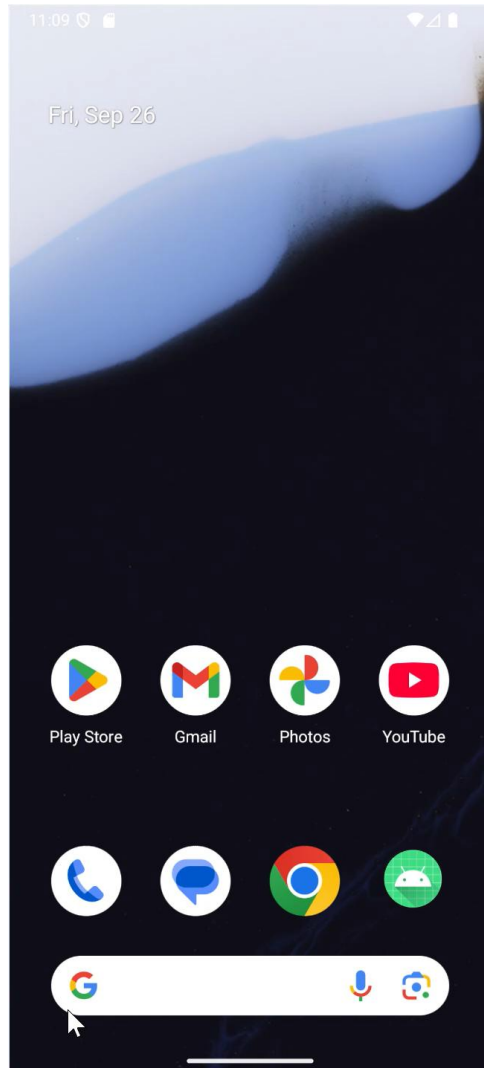
Pobierz posty tylko dla konkretnego użytkownika

```
interface ApiService {  
    // Funkcja pobierze posty tylko dla użytkownika o podanym ID  
    @GET("posts")  
    suspend fun getPostsByUser(@Query("userId") id: Int): List<Post>  
}
```

```
[
  {
    "userId": 1,
    "id": 1,
    "title": "sunt aut facere repellat provident occaecati excepturi optio reprehenderit",
    "body": "quia et suscipit\nsuscipit recusandae consequuntur expedita et cum\nreprehenderit",
  },
  {
    "userId": 1,
    "id": 2,
    "title": "qui est esse",
    "body": "est rerum tempore vitae\nsequi sint nihil reprehenderit dolor beatae ea dolores  
nisi nulla"
  },
]
```

<https://icanhazdadjoke.com/>

Nagłówek Accept zmienia format odpowiedzi serwera



Adnotacja Retrofit służąca do dodawania **statycznych nagłówków** do zapytania HTTP

```
data class Joke(val id: String, val joke: String, val status: Int)
3 Usages
interface JokeApiService {
    1 Usage
    @Headers( ...value = "Accept: application/json")
    @GET( value = "/")
    suspend fun getJokeAsJson(): Joke

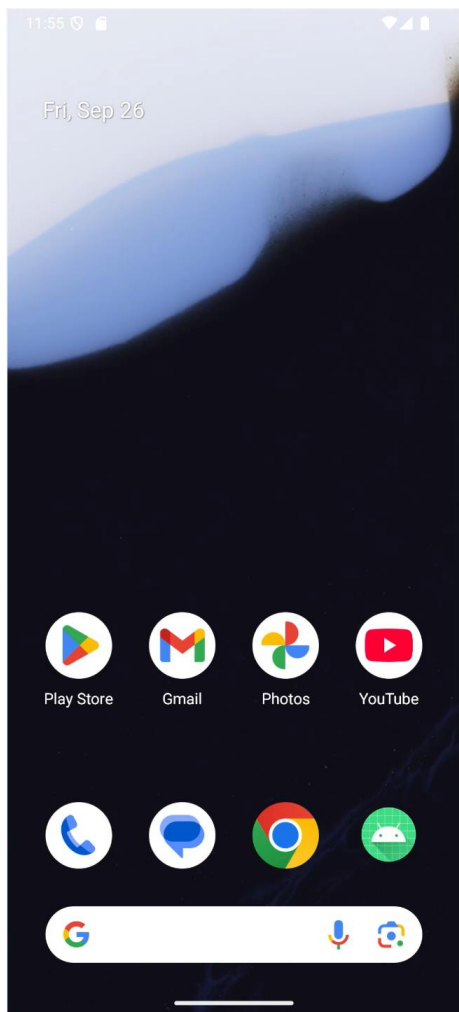
    1 Usage
    @Headers( ...value = "Accept: text/plain")
    @GET( value = "/")
    suspend fun getJokeAsText(): String
}
```

Nagłówek, który informuje serwer o formacie akceptowanej odpowiedzi przez aplikację.


```
object RetrofitInstance {  
    1 Usage  
    private const val BASE_URL = "https://icanhazdadjoke.com/"  
    1 Usage  
    private val retrofit by lazy {  
        Retrofit.Builder()  
            .baseUrl( baseUrl = BASE_URL )  
            // Ważne: dodajemy oba konwertery. Retrofit sam wybierze właściwy.  
            .addConverterFactory( factory = ScalarsConverterFactory.create() ) // Do obsługi String  
            .addConverterFactory( factory = GsonConverterFactory.create() ) // Do obsługi JSON  
            .build()  
    }  
    val api: JokeApiService by lazy {  
        retrofit.create(JokeApiService::class.java)  
    }  
}
```

<https://newsapi.org/>

Scenariusz: Pobierz najnowsze wiadomości z USA (country=us) na temat technologii (category=technology). Klucz API musi być wysłany w nagłówku X-API-Key.



```
data class Article(  
    val title: String,  
    val author: String?,  
    val description: String?)
```

2 Usages

```
data class NewsResponse(  
    val status: String,  
    val totalResults: Int,  
    val articles: List<Article>)
```

3 Usages

```
interface NewsApiService {
```

1 Usage

```
@GET( value = "v2/top-headlines")
```

```
suspend fun getTopHeadlines(  
    @Header( value = "X-API-Key") apiKey: String,  
    @Query( value = "country") countryCode: String,  
    @Query( value = "category") category: String  
): NewsResponse
```

```
}
```

Odpowiedź z serwera jest
bardziej skomplikowana

Szablon pojedynczego
artykułu

```
data class Article(  
    val title: String,  
    val author: String?,  
    val description: String?)
```

2 Usages

```
data class NewsResponse(  
    val status: String,  
    val totalResults: Int,  
    val articles: List<Article>)
```

3 Usages

```
interface NewsApiService {  
    1 Usage  
    @GET( value = "v2/top-headlines")  
    suspend fun getTopHeadlines(  
        @Header( value = "X-API-Key") apiKey: String,  
        @Query( value = "country") countryCode: String,  
        @Query( value = "category") category: String  
    ): NewsResponse  
}
```

Odpowiedź z serwera jest
bardziej skomplikowana



Przykłady

Szablon pojedynczego
artykułu

```
data class Article(  
    val title: String,  
    val author: String?,  
    val description: String?)
```

Odpowiedź z serwera jest
bardziej skomplikowana

Szablon całej odpowiedzi
z serwera

```
2 Usages  
data class NewsResponse(  
    val status: String,  
    val totalResults: Int,  
    val articles: List<Article>)
```

Metadane

Dane

```
3 Usages  
interface NewsApiService {  
    1 Usage  
    @GET( value = "v2/top-headlines")  
    suspend fun getTopHeadlines(  
        @Header( value = "X-API-Key") apiKey: String,  
        @Query( value = "country") countryCode: String,  
        @Query( value = "category") category: String  
    ): NewsResponse  
}
```

Szablon pojedynczego
artykułu

```
data class Article(  
    val title: String,  
    val author: String?,  
    val description: String?)
```

Odpowiedź z serwera jest
bardziej skomplikowana

Szablon całej odpowiedzi
z serwera

```
2 Usages  
data class NewsResponse(  
    val status: String,  
    val totalResults: Int,  
    val articles: List<Article>)
```

Metadane

Dane

3 Usages

```
interface NewsApiService {
```

1 Usage

```
@GET( value = "v2/top-headlines")  
suspend fun getTopHeadlines(  
    @Header( value = "X-API-Key") apiKey: String,  
    @Query( value = "country") countryCode: String,  
    @Query( value = "category") category: String  
): NewsResponse
```

```
}
```

Funkcja będzie wykonywać
żądanie HTTP GET do endpointa
/v2/top-headlines.

Szablon pojedynczego artykułu

```
data class Article(  
    val title: String,  
    val author: String?,  
    val description: String?)
```

Odpowiedź z serwera jest bardziej skomplikowana

Szablon całej odpowiedzi z serwera

```
2 Usages  
data class NewsResponse(  
    val status: String,  
    val totalResults: Int,  
    val articles: List<Article>)
```

Metadane

Dane

3 Usages

```
interface NewsApiService {
```

1 Usage

```
@GET( value = "v2/top-headlines")
```

```
suspend fun getTopHeadlines(  
    @Header( value = "X-API-Key") apiKey: String,  
    @Query( value = "country") countryCode: String,  
    @Query( value = "category") category: String  
): NewsResponse
```

```
}
```

Funkcja będzie wykonywać żądanie HTTP GET do endpointa /v2/top-headlines.

Standardowy sposób przesyłania kluczy autoryzacyjnych

Parametry zapytania

Szablon pojedynczego artykułu

```
data class Article(  
    val title: String,  
    val author: String?,  
    val description: String?)
```

Odpowiedź z serwera jest bardziej skomplikowana

Szablon całej odpowiedzi z serwera

```
2 Usages  
data class NewsResponse(  
    val status: String,  
    val totalResults: Int,  
    val articles: List<Article>)
```

Metadane

Dane

3 Usages

```
interface NewsApiService {
```

1 Usage

```
@GET( value = "v2/top-headlines")
```

```
suspend fun getTopHeadlines(  
    @Header( value = "X-API-Key") apiKey: String,  
    @Query( value = "country") countryCode: String,  
    @Query( value = "category") category: String  
): NewsResponse
```

```
}
```

Funkcja będzie wykonywać żądanie HTTP GET do endpointa /v2/top-headlines.

Standardowy sposób przesyłania kluczy autoryzacyjnych

Parametry zapytania

```
GET /v2/top-headlines?country=pl&category=technology HTTP/1.1  
Host: newsapi.org  
X-API-Key: TWÓJ_KLUCZ_API
```


- **Deklaratywna Definicja API:** Definiujemy zapytania sieciowe jako metody w interfejsie Kotlin, używając prostych adnotacji, takich jak **@GET**, **@POST**, **@Path** itd.
- **Bezpieczeństwo Typologiczne:** Retrofit dba o to, by odpowiedzi serwera w formacie JSON (lub innym) były poprawnie konwertowane na zdefiniowane klasy danych (data class), co eliminuje błędy rzutowania.
- **Integracja z Korutinami:** Funkcje w interfejsie można oznaczyć jako **suspend**, dzięki czemu operacje sieciowe są wykonywane **asynchronicznie**, w tle, bez blokowania wątku UI.
- **Wymienne Konwertery:** Obsługuje różne formaty danych dzięki wymiennym bibliotekom konwertującym. Najpopularniejsze to Gson i Moshi do obsługi formatu JSON.
- **Upraszcza Kod:** Zastępuje skomplikowany, ręczny kod do obsługi zapytań HTTP i parsowania odpowiedzi prostym, czytelnym i łatwym w utrzymaniu interfejsem.