Uniwersytet Wrocławski

Rafał Lewandków

PROGRAMOWANIE URZĄDZEŃ MOBILNYCH 2

WYKŁAD 11

Wstrzykiwanie Zależności

- Dagger
- Hilt

Uniwersytet Wrocławski

Wstrzykiwanie zależności

W ostatnich przykładach musieliśmy ręcznie tworzyć instancje Repository i przekazywać je do ViewModelFactory, aby zbudować ViewModel. Dziś poznamy wzorzec, który **automatyzuje ten proces** – **Wstrzykiwanie Zależności** – i jego implementację w Androidzie, czyli bibliotekę **Hilt.**

```
// W Composable
val repository = PostRepository(RetrofitInstance.api)
val factory = PostViewModelFactory(repository)
PostsScreen(viewModel = viewModel(factory = factory))
```

Uniwersytet Wrocławski

Wstrzykiwanie zależności

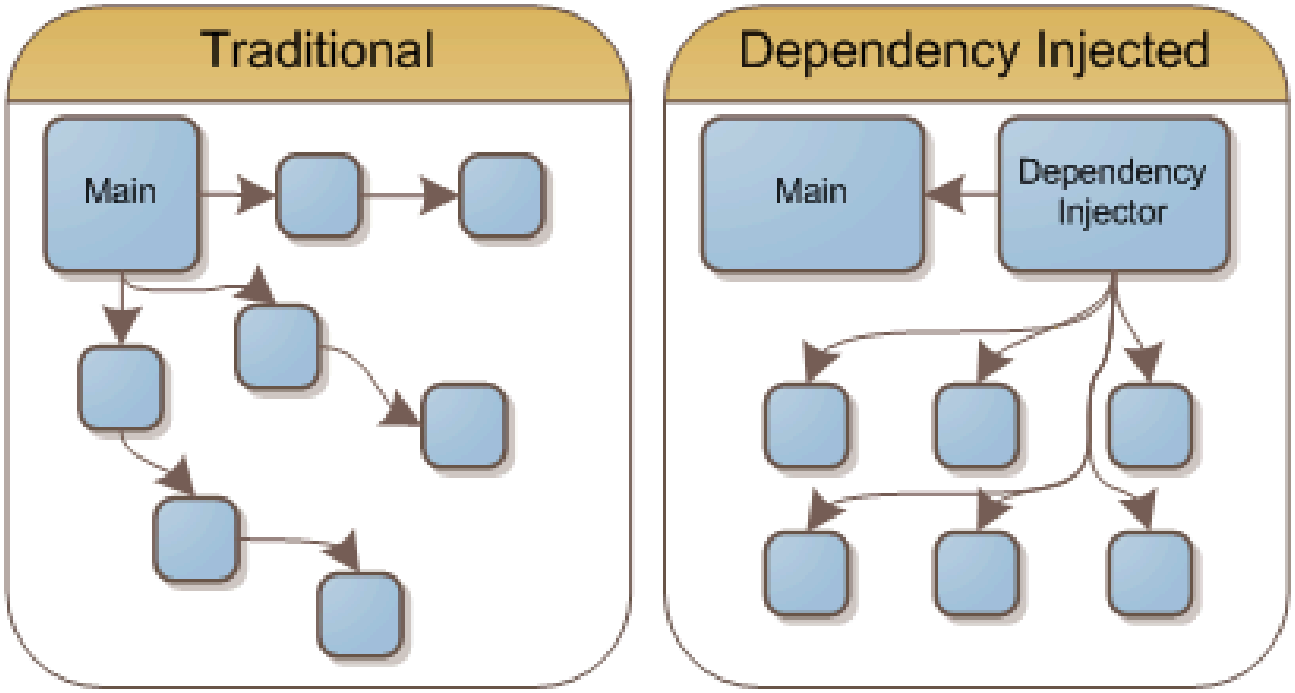
W ostatnich przykładach musieliśmy ręcznie tworzyć instancje Repository i przekazywać je do ViewModelFactory, aby zbudować ViewModel. Dziś poznamy wzorzec, który **automatyzuje ten proces** – **Wstrzykiwanie Zależności** – i jego implementację w Androidzie, czyli bibliotekę **Hilt.**

```
// W Composable
val repository = PostRepository(RetrofitInstance.api)
val factory = PostViewModelFactory(repository)
PostsScreen(viewModel = viewModel(factory = factory))
```

View *wie* za dużo: Komponent UI (PostsScreen) musi wiedzieć, jak zbudować Repository i Factory. To łamie zasadę jednej odpowiedzialności.

Wstrzykiwanie zależności

Wstrzykiwanie Zależności to **wzorzec projektowy**, w którym **zależności obiektu** (czyli inne obiekty, których potrzebuje do działania) są mu **dostarczane z zewnątrz**, zamiast być tworzone przez niego samego.



https://www.google.com/url?sa=i&url=https%3A%2F%2Fblog.stackademic.com%2Friverpod-inversion-of-control-dependency-injection-dependency-inversion-and-service-locator-24a1c6972ed6

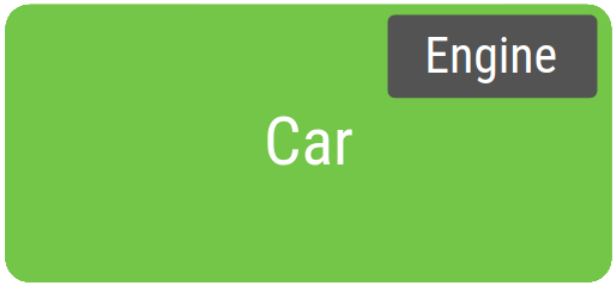
Wstrzykiwanie zależności

```
class Car {

    private Engine engine = new Engine();

    public void start() {
        engine.start();
    }
}


class MyApp {
    public static void main(String[] args) {
        Car car = new Car();
        car.start();
    }
}
```



Wstrzykiwanie zależności

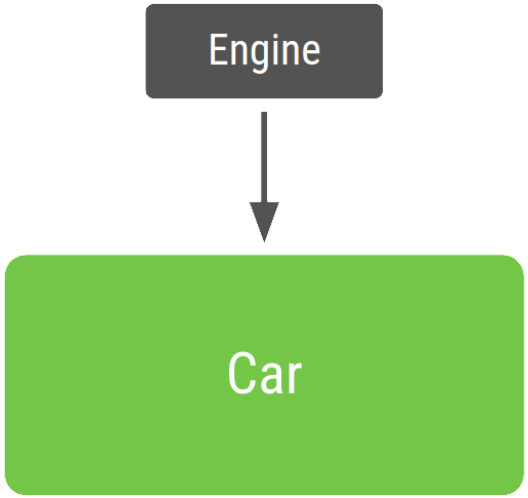
```
class Car {

    private final Engine engine;

    public Car(Engine engine) {
        this.engine = engine;
    }

    public void start() {
        engine.start();
    }
}


class MyApp {
    public static void main(String[] args) {
        Engine engine = new Engine();
        Car car = new Car(engine);
        car.start();
    }
}
```

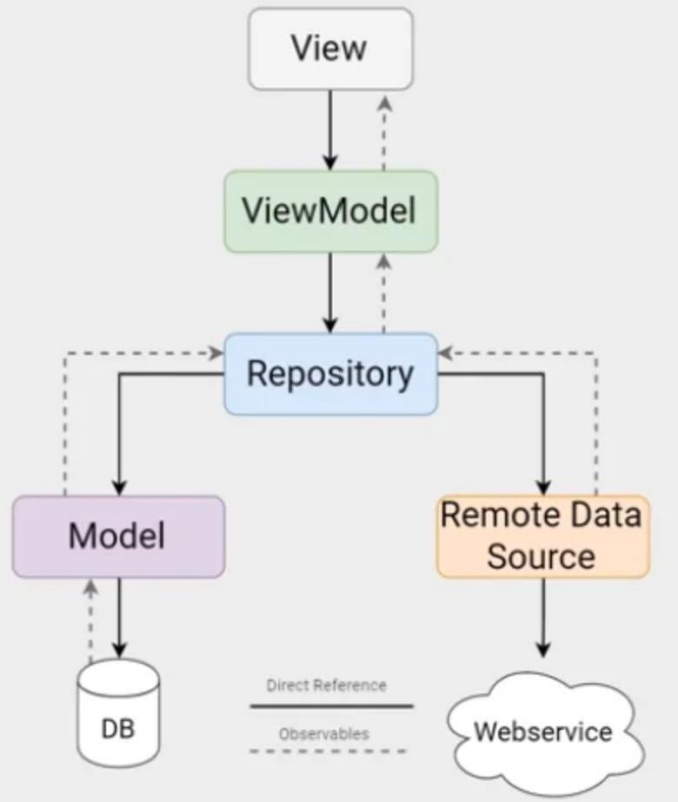


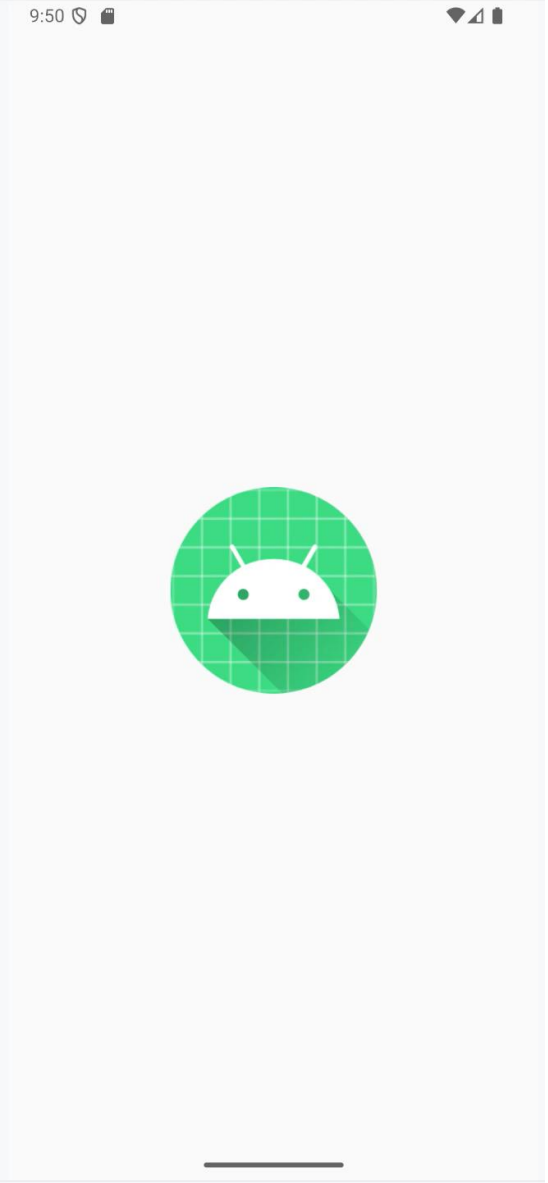
Dagger-Hilt

Dagger2: To standard DI w świecie Androida. Działa w czasie kompilacji, jest niezwykle wydajny, ale jego konfiguracja jest bardzo skomplikowana.

Hilt: To biblioteka zbudowana na Daggerze, stworzona przez Google specjalnie dla Androida. Upraszcza ona Daggera, wprowadzając standardowe adnotacje i automatyzując większość konfiguracji.

Dagger-Hilt


View
ViewModel
Repository
Model
Remote Data Source
DB
Direct Reference
Observables
Webservice


9:50

Dagger-Hilt

Adnotacja z biblioteki Hilt, która oznacza tę klasę jako **główny kontener zależności** dla aplikacji. Jej obecność uruchamia **mechanizm generowania kodu** przez Hilt, który tworzy i zarządza **grafem zależności** dla całego cyklu życia aplikacji.

```
@HiltAndroidApp
class NewsApplication : Application()
```

```
<?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
        android:name=".NewsApplication"
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
```

Uniwersytet Wrocławski

Dagger-Hilt

Adnotacja z biblioteki Hilt, która oznacza tę klasę jako **główny kontener zależności** dla aplikacji. Jej obecność uruchamia **mechanizm generowania kodu** przez Hilt, który tworzy i zarządza **grafem zależności** dla całego cyklu życia aplikacji.

```
@HiltAndroidApp
class NewsApplication : Application()
```

Ten atrybut jest instrukcją, aby **nie używać** domyślnej klasy Application. Jako **punkt startowy** powinna zostać użyta klasa NewsApplication

```
<?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
        android:name=".NewsApplication"
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
```

Dagger-Hilt

Hilt generuje w tle odpowiedni komponent zależności powiązany z **cyklem życia** Activity

Informuje procesor adnotacji, że MainActivity będzie potrzebować dostępu do **grafu zależności**.

Gdy system Android tworzy instancję MainActivity, Hilt integruje się z jej cyklem życia i **wstrzykuje** do klasy **wszystkie pola**, które są oznaczone adnotacją **@Inject**.

```
@AndroidEntryPoint
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContent {
            MaterialTheme {
                AppNavigation()
            }
        }
    }
}
```

Dagger-Hilt – budowanie grafu zależności

1. **@Inject constructor - Instrukcja Tworzenia -** To podstawowy sposób, w jaki *uczymy* Hilt tworzyć nasze własne klasy.

```
// Hilt wie, że aby stworzyć Repository, potrzebuje ApiService
class PostRepository @Inject constructor(private val apiService: ApiService) { ...
```

```
@HiltViewModel
class PostViewModel @Inject constructor(private val repository: PostRepository) :
```

@HiltViewModel - Specjalna adnotacja dla ViewModeli, która pozwala na wstrzykiwanie do nich zależności.

2. **Moduły Hilt (@Module, @Provides)**

Wykorzystywany gdy nie mamy dostępu do implementacji klasy którą chcemy wstrzyknąć (np. Retrofit, Room).

Moduł to ***klasa-instrukcja***, która zawiera *przepisy* na tworzenie takich obiektów.

Uniwersytet Wrocławski

Dagger-Hilt – moduły

@Module: Oznacza klasę jako moduł Hilt**.**

@InstallIn(...): Deklaruje do których komponentów (cyklu życia) opisana klasa/obiekt powinna zostać dodana podczas generacji obiektów przez Hilt (np. SingletonComponent dla obiektów na poziomie całej aplikacji).

@Provides: Oznacza funkcję jako *przepis* na stworzenie obiektu.

```
@Module
@InstallIn(SingletonComponent::class)
object NetworkModule {

    @Provides
    @Singleton // Gwarantuje, że będzie tylko jedna instancja Retrofit
    fun provideRetrofit(): Retrofit {
        return Retrofit.Builder()
            .baseUrl("https://jsonplaceholder.typicode.com/")
            .addConverterFactory(GsonConverterFactory.create())
            .build()
    }

    @Provides
    @Singleton
    fun provideApiService(retrofit: Retrofit): ApiService {
        return retrofit.create(ApiService::class.java)
    }
}
```

Uniwersytet Wrocławski

Dagger-Hilt – moduły

Hilt component	Injector for
`SingletonComponent`	`Application`
`ActivityRetainedComponent`	N/A
`ViewModelComponent`	`ViewModel`
`ActivityComponent`	`Activity`
`FragmentComponent`	`Fragment`
`ViewComponent`	`View`
`ViewWithFragmentComponent`	`View` annotated with `@WithFragmentBindings`
`ServiceComponent`	`Service`

Dagger-Hilt

Wzorzec **DTO – Data Transfer Object**.
Jest to klasa, której **jedynym zadaniem** jest **transportowanie danych** pomiędzy **różnymi warstwami** lub systemami.
DTO stanowi warstwę pośrednią. Repository odbiera dane w formacie ArticleDto, a następnie mapuje je na wewnętrzny model aplikacji (ArticleEntity), który jest używany w bazie danych i ViewModelu.

API (JSON) ➔ Retrofit ➔ ***ArticleDto*** *➔ (mapowanie w Repozytorium) ➔* ***ArticleEntity*** *➔ ViewModel i UI*

```
// --- Warstwa Danych: Sieć (Retrofit) ---
1 Usage
data class NewsApiResponse(val articles: List<ArticleDto>)
1 Usage
data class ArticleDto(val title: String?, val description: String?)
15 Usages
interface ApiService {
    1 Usage
    @GET( value = "v2/top-headlines?country=us&category=technology")
    suspend fun getTopHeadlines
                (@Header( value = "X-Api-Key") apiKey: String): NewsApiResponse
}
```

Dagger-Hilt

Adnotacja oznacza obiekt AppModule jako moduł

```
// --- Moduł Hilt (Wstrzykiwanie Zależności) ---
Usages
@Module
@InstallIn( ...value = SingletonComponent::class)
object AppModule {
    // {...}
    1 Usage
    @Provides
    @Singleton
    fun provideRetrofit(): Retrofit {
        return Retrofit.Builder()
            .baseUrl( baseUrl = "https://newsapi.org/")
            .addConverterFactory( factory = GsonConverterFactory.create())
            .build()
    }
    1 Usage
    @Provides
    @Singleton
    fun provideApiService(retrofit: Retrofit): ApiService =
        retrofit.create(ApiService::class.java)
}
```

Dagger-Hilt

Adnotacja oznacza obiekt AppModule jako moduł

Wszystkie obiekty z tego modułu mają być dostępne w **globalnym zakresie aplikacji**

```
// --- Moduł Hilt (Wstrzykiwanie Zależności) ---
Usages
@Module
@InstallIn( ...value = SingletonComponent::class)
object AppModule {
    // {...}
    1 Usage
    @Provides
    @Singleton
    fun provideRetrofit(): Retrofit {
        return Retrofit.Builder()
            .baseUrl( baseUrl = "https://newsapi.org/")
            .addConverterFactory( factory = GsonConverterFactory.create())
            .build()
    }
    1 Usage
    @Provides
    @Singleton
    fun provideApiService(retrofit: Retrofit): ApiService =
        retrofit.create(ApiService::class.java)
}
```

Dagger-Hilt

Adnotacja oznacza obiekt AppModule jako moduł

Wszystkie obiekty z tego modułu mają być dostępne w **globalnym zakresie aplikacji**

Informuje Hilt, że ta funkcja jest instrukcją tworzenia obiektu

```
// --- Moduł Hilt (Wstrzykiwanie Zależności) ---
Usages
@Module
@InstallIn( ...value = SingletonComponent::class)
object AppModule {
    // {...}
    1 Usage
    @Provides
    @Singleton
    fun provideRetrofit(): Retrofit {
        return Retrofit.Builder()
            .baseUrl( baseUrl = "https://newsapi.org/")
            .addConverterFactory( factory = GsonConverterFactory.create())
            .build()
    }
    1 Usage
    @Provides
    @Singleton
    fun provideApiService(retrofit: Retrofit): ApiService =
        retrofit.create(ApiService::class.java)
}
```

Uniwersytet Wrocławski

Dagger-Hilt

Adnotacja oznacza obiekt AppModule jako moduł

Wszystkie obiekty z tego modułu mają być dostępne w **globalnym zakresie aplikacji**

Informuje Hilt, że ta funkcja jest instrukcją tworzenia obiektu

Gwarantuje, że ta funkcja zostanie wywołana tylko raz, a stworzona instancja bazy danych będzie współdzielona w całej aplikacji

```
// --- Moduł Hilt (Wstrzykiwanie Zależności) ---
Usages
@Module
@InstallIn( ...value = SingletonComponent::class)
object AppModule {
    // {...}
    1 Usage
    @Provides
    @Singleton
    fun provideRetrofit(): Retrofit {
        return Retrofit.Builder()
            .baseUrl( baseUrl = "https://newsapi.org/")
            .addConverterFactory( factory = GsonConverterFactory.create())
            .build()
    }
    1 Usage
    @Provides
    @Singleton
    fun provideApiService(retrofit: Retrofit): ApiService =
        retrofit.create(ApiService::class.java)
}
```

Dagger-Hilt

```
// --- Warstwa Danych: Baza Danych (Room) ---
24 Usages
@Entity(tableName = "articles")
data class ArticleEntity(
    @PrimaryKey val title: String,
    val description: String?
)
19 Usages  1 Implementation
@Dao
interface ArticleDao {
    1 Usage  1 Implementation
    @Query( value = "SELECT * FROM articles")
    fun getArticlesStream(): Flow<List<ArticleEntity>>
    1 Usage  1 Implementation
    @Query( value = "SELECT * FROM articles WHERE title = :title")
    fun getArticleByTitle(title: String): Flow<ArticleEntity?>
    1 Usage  1 Implementation
    @Upsert
    suspend fun insertArticles(articles: List<ArticleEntity>)
    1 Usage  1 Implementation
    @Query( value = "DELETE FROM articles")
    suspend fun clearAll()
}
11 Usages  1 Implementation
@Database(entities = [ArticleEntity::class], version = 1, exportSchema = false)
abstract class AppDatabase : RoomDatabase() {
    1 Usage  1 Implementation
    abstract fun articleDao(): ArticleDao
}
```

Połączenie **Update i Insert** - wstaw, jeśli nie istnieje, lub zaktualizuj, jeśli istnieje

Uniwersytet Wrocławski

Dagger-Hilt

```
@Module
@InstallIn( ...value = SingletonComponent::class)
object AppModule {
    1 Usage
    @Provides
    @Singleton
    fun provideAppDatabase(@ApplicationContext context: Context): AppDatabase {
        return Room.databaseBuilder(
            context,
            klass = AppDatabase::class.java,
            name = "news_database_db")
            .build()
    }
    1 Usage
    @Provides
    @Singleton
    fun provideArticleDao(database: AppDatabase): ArticleDao = database.articleDao()
    1 Usage
    @Provides
    @Singleton
    fun provideRetrofit(): Retrofit {
        return Retrofit.Builder()
            .baseUrl( baseUrl = "https://newsapi.org/")
            .addConverterFactory( factory = GsonConverterFactory.create())
            .build()
    }
    1 Usage
    @Provides
    @Singleton
    fun provideApiService(retrofit: Retrofit): ApiService = retrofit.create(ApiService::class.java)
```

Dagger-Hilt

Adnotacja z biblioteki Javy, adoptowana przez Hilt. Wskazuje konstruktor, który ma być użyty do **utworzenia obiektu.**

Wskazuje, że adnotacja dotyczy **głównego konstruktora klasy.** (Wstrzykiwanie przez konstruktor).

Elementy, które mają zostać dostarczone przez Hilt. Instrukcje znajdują się w AppModule

```
// --- Warstwa Danych: Repozytorium ---
14 Usages
class ArticleRepository @Inject constructor(
    private val apiService: ApiService,
    private val articleDao: ArticleDao
) {
    1 Usage
    val articlesStream: Flow<List<ArticleEntity>> =
        articleDao.getArticlesStream()
    1 Usage
    fun getArticleByTitle(title: String): Flow<ArticleEntity?> =
        articleDao.getArticleByTitle(title)
    1 Usage
    suspend fun refreshNews() {...}
}
```

Uniwersytet Wrocławski

Dagger-Hilt

W aplikacji są dwa ViewModele, ponieważ każdy z nich jest odpowiedzialny za logikę i stan **jednego, konkretnego ekranu**, co jest zgodne z zasadą **separacji odpowiedzialności.**

```
@HiltViewModel
class ArticleDetailViewModel @Inject constructor(
    repository: ArticleRepository,
    savedStateHandle: SavedStateHandle
) : ViewModel() {
    1 Usage
    private val articleTitle: String =
        savedStateHandle.get<String>("articleTitle") ?: ""
    1 Usage
    val uiState: StateFlow<ArticleDetailState> = repository
        .getArticleByTitle( title = Uri.decode( s = articleTitle))
        .map {...}
        .stateIn(...)
}
```

```
@HiltViewModel
class ArticlesViewModel @Inject constructor(
    private val repository: ArticleRepository
) : ViewModel() {
    3 Usages
    private val _isLoading = MutableStateFlow( value = false)
    1 Usage
    val uiState: StateFlow<ArticlesUiState> = combine(...) {...}
        .stateIn(
        scope = viewModelScope,
        started = SharingStarted.WhileSubscribed( stopTimeoutMillis = 5000),
        initialValue = ArticlesUiState(isLoading = true)
    )

    init {...}
    1 Usage
    fun refresh() {...}
}
```

Uniwersytet Wrocławski

Dagger-Hilt

W aplikacji są dwa ViewModele, ponieważ każdy z nich jest odpowiedzialny za logikę i stan **jednego, konkretnego ekranu**, co jest zgodne z zasadą **separacji odpowiedzialności.**

SavedStateHandle: Umożliwia automatyczne wstrzyknięcie obiektu SavedStateHandle, który pozwala na **odczytanie argumentów** przekazanych przez **Compose Navigation**.

```
@HiltViewModel
class ArticleDetailViewModel @Inject constructor(
    repository: ArticleRepository,
    savedStateHandle: SavedStateHandle
) ViewModel() {
    1 Usage
    private val articleTitle: String =
        savedStateHandle.get<String>("articleTitle") ?: ""
    1 Usage
    val uiState: StateFlow<ArticleDetailState> = repository
        .getArticleByTitle( title = Uri.decode( s = articleTitle))
        .map {...}
        .stateIn(...)
}
```

```
@HiltViewModel
class ArticlesViewModel @Inject constructor(
    private val repository: ArticleRepository
) : ViewModel() {
    3 Usages
    private val _isLoading = MutableStateFlow( value = false)
    1 Usage
    val uiState: StateFlow<ArticlesUiState> = combine(...) {...}
        .stateIn(
        scope = viewModelScope,
        started = SharingStarted.WhileSubscribed( stopTimeoutMillis = 5000),
        initialValue = ArticlesUiState(isLoading = true)
    )

    init {...}
    1 Usage
    fun refresh() {...}
}
```

Uniwersytet Wrocławski

Dagger-Hilt

W aplikacji są dwa ViewModele, ponieważ każdy z nich jest odpowiedzialny za logikę i stan **jednego, konkretnego ekranu**, co jest zgodne z zasadą **separacji odpowiedzialności.**

SavedStateHandle: Umożliwia automatyczne wstrzyknięcie obiektu SavedStateHandle, który pozwala na **odczytanie argumentów** przekazanych przez **Compose Navigation**.

@HiltViewModel to specjalna adnotacja Hilt, która musi być użyta zamiast @Inject constructor w przypadku ViewModeli. Informuje Hilt, że ta klasa jest ViewModelem i ma być **przygotowana do wstrzykiwania zależności.**

```
@HiltViewModel
class ArticleDetailViewModel @Inject constructor(
    repository: ArticleRepository,
    savedStateHandle: SavedStateHandle
) ViewModel() {
    1 Usage
    private val articleTitle: String =
        savedStateHandle.get<String>("articleTitle") ?: ""
    1 Usage
    val uiState: StateFlow<ArticleDetailState> = repository
        .getArticleByTitle( title = Uri.decode( s = articleTitle))
        .map {...}
        .stateIn(...)
}
```

```
@HiltViewModel
class ArticlesViewModel @Inject constructor(
    private val repository: ArticleRepository
) : ViewModel() {
    3 Usages
    private val _isLoading = MutableStateFlow( value = false)
    1 Usage
    val uiState: StateFlow<ArticlesUiState> = combine(...) {...}
        .stateIn(
        scope = viewModelScope,
        started = SharingStarted.WhileSubscribed( stopTimeoutMillis = 5000),
        initialValue = ArticlesUiState(isLoading = true)
    )

    init {...}
    1 Usage
    fun refresh() {...}
}
```

Uniwersytet Wrocławski

Dagger-Hilt

W aplikacji są dwa ViewModele, ponieważ każdy z nich jest odpowiedzialny za logikę i stan **jednego, konkretnego ekranu**, co jest zgodne z zasadą **separacji odpowiedzialności.**

SavedStateHandle: Umożliwia automatyczne wstrzyknięcie obiektu SavedStateHandle, który pozwala na **odczytanie argumentów** przekazanych przez **Compose Navigation**.

@HiltViewModel to specjalna adnotacja Hilt, która musi być użyta zamiast @Inject constructor w przypadku ViewModeli. Informuje Hilt, że ta klasa jest ViewModelem i ma być **przygotowana do wstrzykiwania zależności.**

ViewModele mają specjalny **cykl życia** Adnotacja @HiltViewModel zapewnia, że Hilt będzie tworzył i **dostarczał instancje** ViewModelu w sposób **zgodny z tym cyklem.**

```
@HiltViewModel
class ArticleDetailViewModel @Inject constructor(
    repository: ArticleRepository,
    savedStateHandle: SavedStateHandle
) ViewModel() {
    1 Usage
    private val articleTitle: String =
        savedStateHandle.get<String>("articleTitle") ?: ""
    1 Usage
    val uiState: StateFlow<ArticleDetailState> = repository
        .getArticleByTitle( title = Uri.decode( s = articleTitle))
        .map {...}
        .stateIn(...)
}
```

```
@HiltViewModel
class ArticlesViewModel @Inject constructor(
    private val repository: ArticleRepository
) : ViewModel() {
    3 Usages
    private val _isLoading = MutableStateFlow( value = false)
    1 Usage
    val uiState: StateFlow<ArticlesUiState> = combine(...) {...}
        .stateIn(
        scope = viewModelScope,
        started = SharingStarted.WhileSubscribed( stopTimeoutMillis = 5000),
        initialValue = ArticlesUiState(isLoading = true)
    )

    init {...}
    1 Usage
    fun refresh() {...}
}
```

Dagger-Hilt

Zadaniem funkcji hiltViewModel jest **automatyczne znalezienie** i **dostarczenie** prawidłowej, w pełni skonstruowanej **instancji ViewModelu** dla danego ekranu. Automatycznie wiąże **cykl życia** ViewModelu z **odpowiednim właścicielem**, którym najczęściej jest ekran nawigacji.

```
@OptIn( ...markerClass = ExperimentalMaterial3Api::class)
@Composable
fun ArticleListScreen(
    viewModel: ArticlesViewModel = hiltViewModel(),
    onArticleClick: (String) -> Unit
) {...}

1 Usage
@OptIn( ...markerClass = ExperimentalMaterial3Api::class)
@Composable
fun ArticleDetailScreen(viewModel: ArticleDetailViewModel = hiltViewModel()) {...}
```

Bez Hilt:

```
val repository = MyRepository(...)
val factory = MyViewModelFactory(repository)
val viewModel: MyViewModel = viewModel(factory = factory)
```

Z Hilt:

```
val viewModel: MyViewModel = hiltViewModel()
```

Uniwersytet Wrocławski

Dagger-Hilt - Podsumowanie

Hilt to biblioteka do **wstrzykiwania zależności** (Dependency Injection) zbudowana na Daggerze, która upraszcza i **automatyzuje dostarczanie obiektów** (*zależności*) do różnych części aplikacji Android.

- **@HiltAndroidApp** Umieszczana nad klasą Application, inicjuje Hilt dla całej aplikacji i tworzy **główny kontener zależności.**

Uniwersytet Wrocławski

Dagger-Hilt - Podsumowanie

Hilt to biblioteka do **wstrzykiwania zależności** (Dependency Injection) zbudowana na Daggerze, która upraszcza i **automatyzuje dostarczanie obiektów** (*zależności*) do różnych części aplikacji Android.

- **@HiltAndroidApp** Umieszczana nad klasą Application, inicjuje Hilt dla całej aplikacji i tworzy **główny kontener zależności.**
- **@AndroidEntryPoint** Oznacza komponent Androida (np. Activity, Fragment), informując Hilt, że ma do niego wstrzykiwać zależności.

Uniwersytet Wrocławski

Dagger-Hilt - Podsumowanie

Hilt to biblioteka do **wstrzykiwania zależności** (Dependency Injection) zbudowana na Daggerze, która upraszcza i **automatyzuje dostarczanie obiektów** (*zależności*) do różnych części aplikacji Android.

- **@HiltAndroidApp** Umieszczana nad klasą Application, inicjuje Hilt dla całej aplikacji i tworzy **główny kontener zależności.**
- **@AndroidEntryPoint** Oznacza komponent Androida (np. Activity, Fragment), informując Hilt, że ma do niego wstrzykiwać zależności.
- **@HiltViewModel** Specjalna adnotacja dla klas ViewModel, która umożliwia wstrzykiwanie do nich zależności i integruje je z cyklem życia.

Uniwersytet Wrocławski

Dagger-Hilt - Podsumowanie

Hilt to biblioteka do **wstrzykiwania zależności** (Dependency Injection) zbudowana na Daggerze, która upraszcza i **automatyzuje dostarczanie obiektów** (*zależności*) do różnych części aplikacji Android.

- **@HiltAndroidApp** Umieszczana nad klasą Application, inicjuje Hilt dla całej aplikacji i tworzy **główny kontener zależności.**
- **@AndroidEntryPoint** Oznacza komponent Androida (np. Activity, Fragment), informując Hilt, że ma do niego wstrzykiwać zależności.
- **@HiltViewModel** Specjalna adnotacja dla klas ViewModel, która umożliwia wstrzykiwanie do nich zależności i integruje je z cyklem życia.
- **@Inject** Używana na konstruktorze klasy, aby nauczyć Hilt, jak tworzyć jej instancje (**constructor injection**). Używana na polu w klasie z adnotacją **@AndroidEntryPoint**, aby poprosić Hilt o wstrzyknięcie tam zależności (**field injection**).

Uniwersytet Wrocławski

Dagger-Hilt - Podsumowanie

Hilt to biblioteka do **wstrzykiwania zależności** (Dependency Injection) zbudowana na Daggerze, która upraszcza i **automatyzuje dostarczanie obiektów** (*zależności*) do różnych części aplikacji Android.

- **@HiltAndroidApp** Umieszczana nad klasą Application, inicjuje Hilt dla całej aplikacji i tworzy **główny kontener zależności.**
- **@AndroidEntryPoint** Oznacza komponent Androida (np. Activity, Fragment), informując Hilt, że ma do niego wstrzykiwać zależności.
- **@HiltViewModel** Specjalna adnotacja dla klas ViewModel, która umożliwia wstrzykiwanie do nich zależności i integruje je z cyklem życia.
- **@Inject** Używana na konstruktorze klasy, aby nauczyć Hilt, jak tworzyć jej instancje (**constructor injection**). Używana na polu w klasie z adnotacją **@AndroidEntryPoint**, aby poprosić Hilt o wstrzyknięcie tam zależności (**field injection**).
- **@Module** Oznacza klasę lub obiekt jako moduł, czyli zbiór instrukcji tworzenia zależności, których **nie jesteśmy właścicielami**.

Uniwersytet Wrocławski

Dagger-Hilt - Podsumowanie

Hilt to biblioteka do **wstrzykiwania zależności** (Dependency Injection) zbudowana na Daggerze, która upraszcza i **automatyzuje dostarczanie obiektów** (*zależności*) do różnych części aplikacji Android.

- **@HiltAndroidApp** Umieszczana nad klasą Application, inicjuje Hilt dla całej aplikacji i tworzy **główny kontener zależności.**
- **@AndroidEntryPoint** Oznacza komponent Androida (np. Activity, Fragment), informując Hilt, że ma do niego wstrzykiwać zależności.
- **@HiltViewModel** Specjalna adnotacja dla klas ViewModel, która umożliwia wstrzykiwanie do nich zależności i integruje je z cyklem życia.
- **@Inject** Używana na konstruktorze klasy, aby nauczyć Hilt, jak tworzyć jej instancje (**constructor injection**). Używana na polu w klasie z adnotacją **@AndroidEntryPoint**, aby poprosić Hilt o wstrzyknięcie tam zależności (**field injection**).
- **@Module** Oznacza klasę lub obiekt jako moduł, czyli zbiór instrukcji tworzenia zależności, których **nie jesteśmy właścicielami**.
- **@InstallIn(...)** Określa, w jakim zakresie (scope) mają być dostępne instrukcje z modułu (np. @InstallIn(SingletonComponent::class) dla zasięgu całej aplikacji).

Uniwersytet Wrocławski

Dagger-Hilt - Podsumowanie

Hilt to biblioteka do **wstrzykiwania zależności** (Dependency Injection) zbudowana na Daggerze, która upraszcza i **automatyzuje dostarczanie obiektów** (*zależności*) do różnych części aplikacji Android.

- **@HiltAndroidApp** Umieszczana nad klasą Application, inicjuje Hilt dla całej aplikacji i tworzy **główny kontener zależności.**
- **@AndroidEntryPoint** Oznacza komponent Androida (np. Activity, Fragment), informując Hilt, że ma do niego wstrzykiwać zależności.
- **@HiltViewModel** Specjalna adnotacja dla klas ViewModel, która umożliwia wstrzykiwanie do nich zależności i integruje je z cyklem życia.
- **@Inject** Używana na konstruktorze klasy, aby nauczyć Hilt, jak tworzyć jej instancje (**constructor injection**). Używana na polu w klasie z adnotacją **@AndroidEntryPoint**, aby poprosić Hilt o wstrzyknięcie tam zależności (**field injection**).
- **@Module** Oznacza klasę lub obiekt jako moduł, czyli zbiór instrukcji tworzenia zależności, których **nie jesteśmy właścicielami**.
- **@InstallIn(...)** Określa, w jakim zakresie (scope) mają być dostępne instrukcje z modułu (np. @InstallIn(SingletonComponent::class) dla zasięgu całej aplikacji).
- **@Provides** Oznacza funkcję w module jako instrukcję, który mówi Hilt, jak stworzyć i dostarczyć konkretną zależność (np. instancję Retrofit lub Room).

Uniwersytet Wrocławski

Dagger-Hilt - Podsumowanie

Hilt to biblioteka do **wstrzykiwania zależności** (Dependency Injection) zbudowana na Daggerze, która upraszcza i **automatyzuje dostarczanie obiektów** (*zależności*) do różnych części aplikacji Android.

- **@HiltAndroidApp** Umieszczana nad klasą Application, inicjuje Hilt dla całej aplikacji i tworzy **główny kontener zależności.**
- **@AndroidEntryPoint** Oznacza komponent Androida (np. Activity, Fragment), informując Hilt, że ma do niego wstrzykiwać zależności.
- **@HiltViewModel** Specjalna adnotacja dla klas ViewModel, która umożliwia wstrzykiwanie do nich zależności i integruje je z cyklem życia.
- **@Inject** Używana na konstruktorze klasy, aby nauczyć Hilt, jak tworzyć jej instancje (**constructor injection**). Używana na polu w klasie z adnotacją **@AndroidEntryPoint**, aby poprosić Hilt o wstrzyknięcie tam zależności (**field injection**).
- **@Module** Oznacza klasę lub obiekt jako moduł, czyli zbiór instrukcji tworzenia zależności, których **nie jesteśmy właścicielami**.
- **@InstallIn(...)** Określa, w jakim zakresie (scope) mają być dostępne instrukcje z modułu (np. @InstallIn(SingletonComponent::class) dla zasięgu całej aplikacji).
- **@Provides** Oznacza funkcję w module jako instrukcję, który mówi Hilt, jak stworzyć i dostarczyć konkretną zależność (np. instancję Retrofit lub Room).
- **@Singleton** Adnotacja używana razem z @Provides (lub na klasie), która instruuje Hilt, aby stworzył tylko jedną instancję danej zależności dla całego zasięgu.