APLIKACJE MOBILNE

Wykład 03

dr Artur Bartoszewski

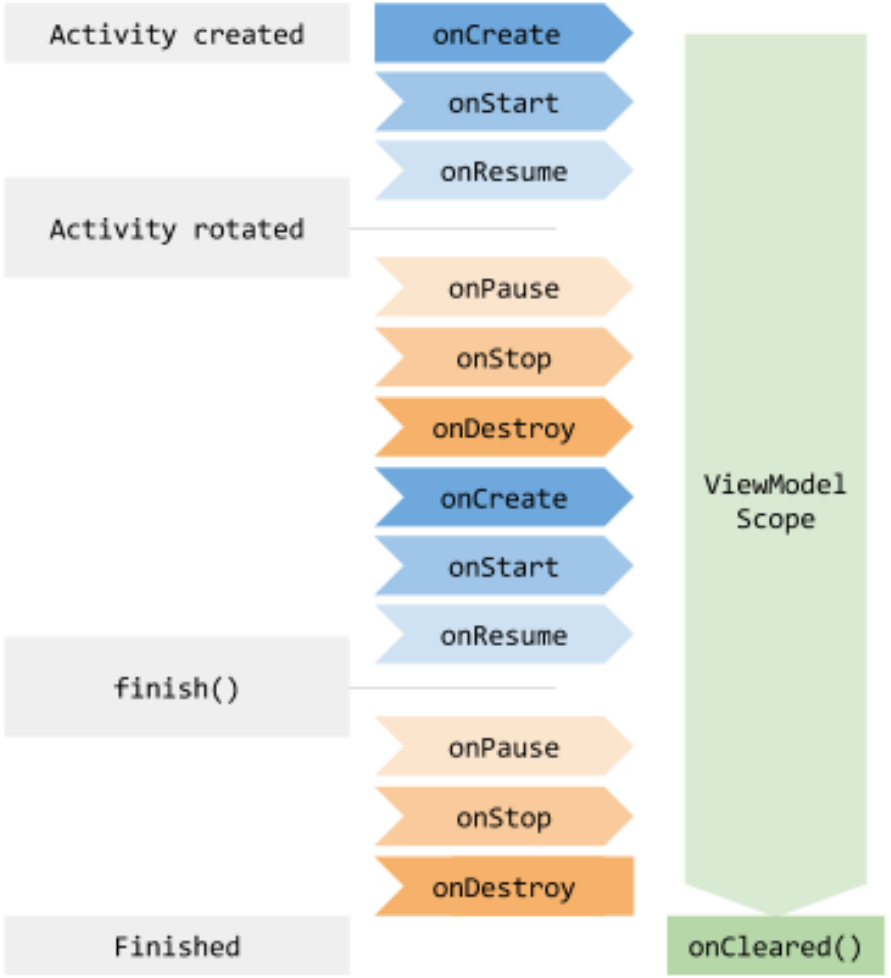
ViewModel

ViewModels

Klasa ViewModel (w Android Jetpack) to komponent architektury aplikacji, który służy do przechowywania i zarządzania danymi związanymi z interfejsem użytkownika (UI).

Głównym celem ViewModel jest oddzielenie logiki UI od logiki biznesowej i danych, co ułatwia testowanie, utrzymanie i skalowanie aplikacji.

ViewModels istnieje (w pewnym sensie) poza cyklem życia aktywności.



ViewModels

Główne cechy i zastosowania ViewModel:

- ViewModel przetrwa zmiany konfiguracji aktywności, takie jak obrót ekranu, czy zmiana języka. Dzięki temu dane przechowywane w ViewModel nie zostaną utracone, gdy aktywność zostanie zniszczona i ponownie utworzona.
- Może być używany do udostępniania danych między różnymi komponentami UI, takimi jak aktywności, fragmenty czy widoki niestandardowe.
- Pozwala na oddzielenie logiki UI od logiki biznesowej i danych. Dzięki temu kod jest bardziej czytelny, łatwiejszy w testowaniu i utrzymaniu.

ViewModels

Korzyści z używania ViewModel:

- Kod jest bardziej czytelny i łatwiejszy w utrzymaniu, ponieważ logika UI jest oddzielona od logiki biznesowej i danych.
- ViewModel można łatwo testować jednostkowo, ponieważ nie zależy od komponentów Android.
- ViewModel przetrwa zmiany konfiguracji, co zapobiega ponownemu pobieraniu danych i poprawia wydajność aplikacji. Podsumowując: ViewModel to ważny komponent architektury aplikacji Android, który ułatwia tworzenie aplikacji, które są łatwiejsze w testowaniu, utrzymaniu i skalowaniu.

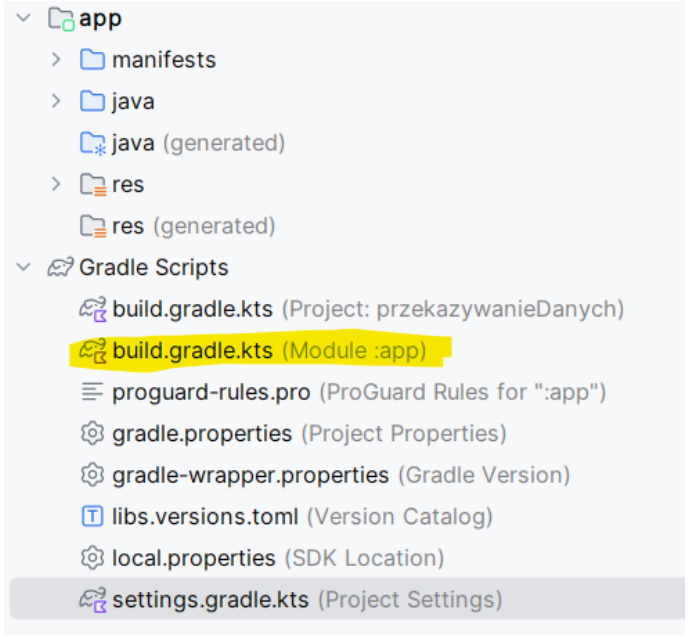
ViewModels

Korzyści z używania ViewModel:

- Kod jest bardziej czytelny i łatwiejszy w utrzymaniu, ponieważ logika UI jest oddzielona od logiki biznesowej i danych.
- ViewModel można łatwo testować jednostkowo, ponieważ nie zależy od komponentów Android.
- ViewModel przetrwa zmiany konfiguracji, co zapobiega ponownemu pobieraniu danych i poprawia wydajność aplikacji. Podsumowując: ViewModel to ważny komponent architektury aplikacji Android, który ułatwia tworzenie aplikacji, które są łatwiejsze w testowaniu, utrzymaniu i skalowaniu.

Wykorzystanie ViewModel – Krok 1

Upewnij się, że w pliku build.gradle (Module: app) są dodane zależności dla komponentów ViewModel



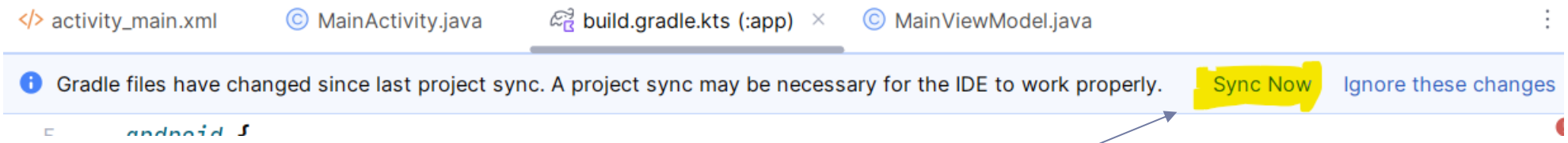

```
dependencies {
    val activity_version = "1.9.1"
    // Java language implementation
    implementation("androidx.activity:activity:$activity_version")
    // Kotlin
    implementation("androidx.activity:activity-ktx:$activity_version"))
    implementation(libs.appcompat)
    implementation(libs.material)
    implementation(libs.activity)
    implementation(libs.constraintlayout)
    testImplementation(libs.junit)
    androidTestImplementation(libs.ext.junit)
    androidTestImplementation(libs.espresso.core)
}
```

Aktualny opis pobrać można ze strony:

https://developer.android.com/jetpack/androidx/releases/activity#kts

Wykorzystanie ViewModel – Krok 1

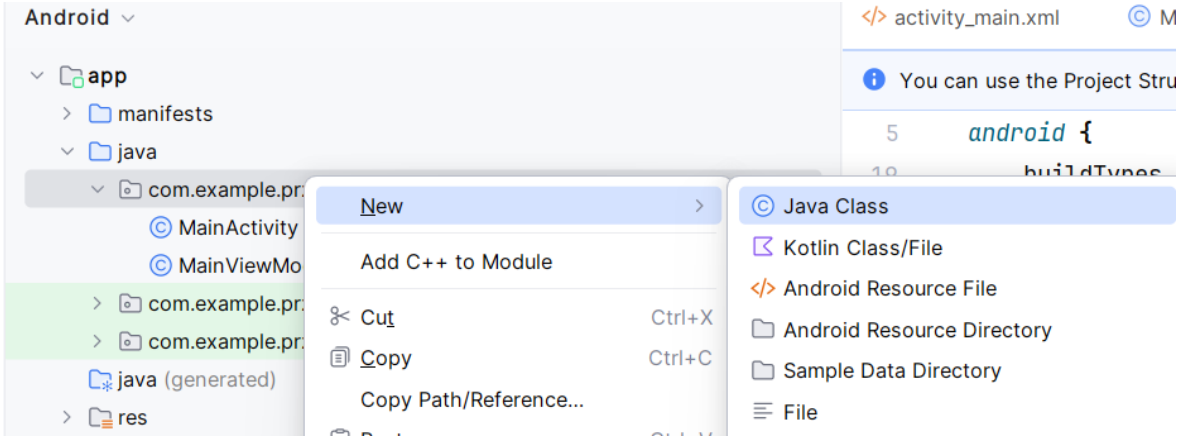
Upewnij się, że w pliku build.gradle (Module: app) są dodane zależności dla komponentów ViewModel



Nie zapomnij zsynchronizować projektu

Wykorzystanie ViewModel – Krok 2

Utwórz klasę, która dziedziczy po ViewModel




```
package com.example.przekazywaniedanych;

import androidx.lifecycle.ViewModel;

public class MainViewModel extends ViewModel {
  //Przechowywane zmienne
   int count = 0;
}
```

Do klasy która powstanie auromatycznie dodajemy informację, że dziedziczyć ma po ViewModel

Wykorzystanie ViewModel – Krok 3

- Uzyskaj instancję ViewModel w aktywności.
- Użyj ViewModelProvider do uzyskania instancji ViewModel w aktywności.

```
public class MainActivity extends AppCompatActivity {

    private MainViewModel viewModel;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        viewModel = new ViewModelProvider(this).get(MainViewModel.class);
    }
}
```

Wykorzystanie ViewModel – Krok 4

- Danymi zawartymi w ViewModel możemy posługiwać się jak polami obiektu.

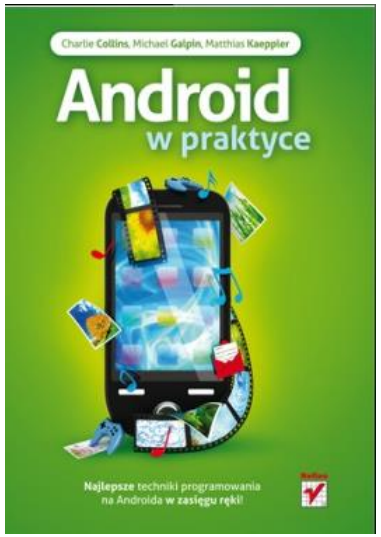
```
viewModel.count++;

tekst.setText(viewModel.count + "");
```

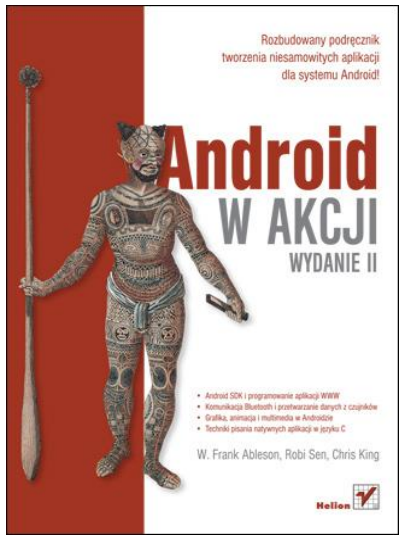
Możliwości klasy ViewModel są oczywiście dużo większe Wykorzystano tu najprostsze jego zastosowanie - czyli przechowam danych, które dostępne są z dowolnej aktywności lub fragmentu i nie zależą od cyklu życia aplikacji.

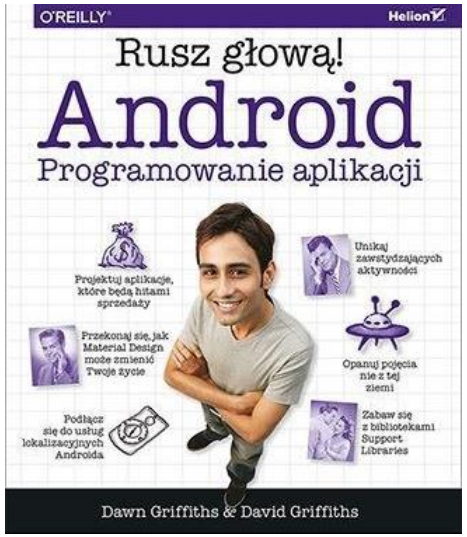
Dane z ViewModel znikają dopiero po zamknięciu aplikacji (patrz cykl życia aktywności).

Literatura



https://developer.android.com





https://javastart.pl/baza-wiedzy/android/

https://forum.android.com.pl