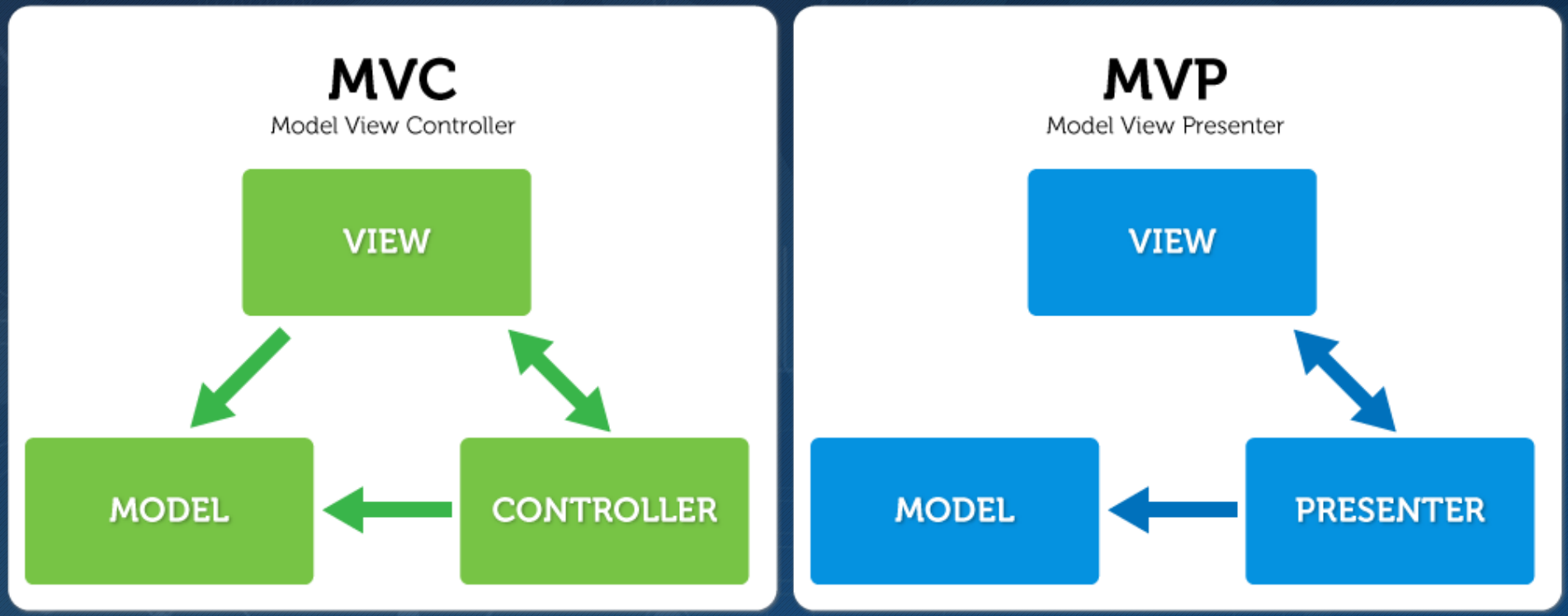
MVVM in Android

СЛЕДИ КАКВО ПРЕДСТОИ В ГРУПАТА НА DEV.BG/MOBILEDEVELOPMENT

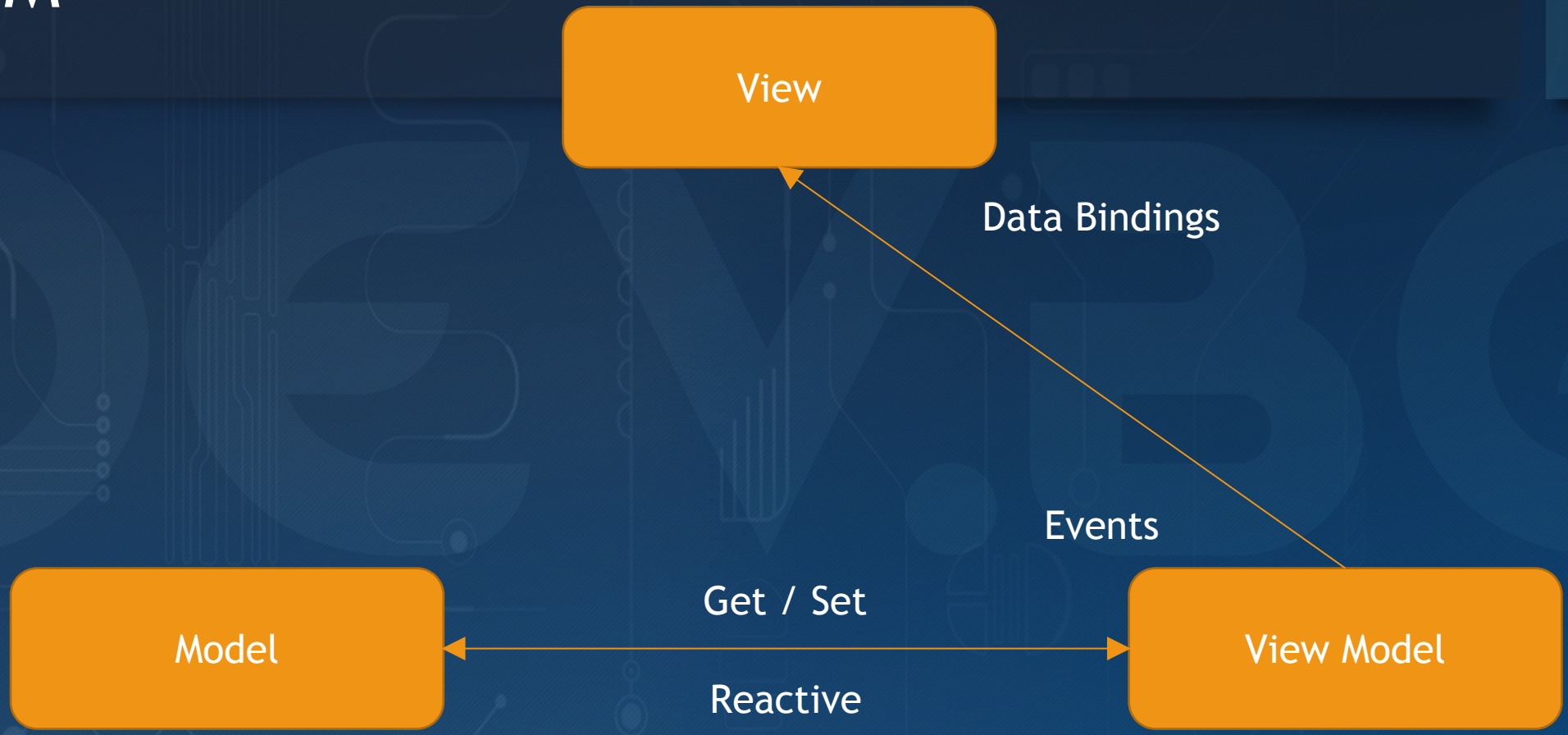
www.officesuitenow.com
www.mobisystems.com

СЛЕДИ КАКВО ПРЕДСТОИ В ГРУПАТА НА DEV.BG/MOBILEDEVELOPMENT

Архитектури



MVVM


View
Data Bindings
Events
Model
Get / Set
Reactive
View Model

Model

- Data
- Business logic

- Get / set methods
- Observables
- Reactive Java
- LiveData

View Model

- UI като интерфейс на клас
- Всяко поле има get метод
- Методи, които променят полетата
- Observable pattern

View

- Предимно ресурси
- Контролите се свързват с полета от ViewModel-а
- Събитията си имат методи от ViewModel-а

Databinding library for Android

- https://developer.android.com/topic/libraries/data-binding
- Позволява да имате bindings в xml ресурсите с описанието на вашите layout-и

build.gradle

```
apply plugin: 'kotlin-kapt'
...
android {
    ...
    dataBinding {
        enabled = true
    }
}
```

View Model class

- Не го бъркайте с android.arch.lifecycle.ViewModel
- Обикновен клас с пропъртита на Kotlin или getProp setProp методи на Java или просто данни

```
Kotlin: data class User(val firstName: String, val lastName: String)
 Java: public class User {
          public final String firstName;
          public final String lastName;
          public User(String firstName, String lastName) {
               this.firstName = firstName;
               this.lastName = lastName;
          }
       }
```

<layout>

```
<layout xmlns:android="http://schemas.android.com/apk/res/android">

    <data>
        <variable name="user" type="com.example.User"/>
    </data>

    <TextView android:text="@{user.firstName}" />

</layout>
```

Generated bindings class

- За всеки layout се генерира служебен клас, който се грижи за свързването на View с ViewModel

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    ActivityMainBinding binding = DataBindingUtil.setContentView(this, R.layout.activity_main);
    User user = new User("Test", "User");
    binding.setUser(user);
}

ActivityMainBinding binding = ActivityMainBinding.inflate(getLayoutInflater());
```

Binding expressions

```
android:text="@{String.valueOf(index + 1)}"
android:visibility="@{age < 13 ? View.GONE : View.VISIBLE}"
android:text="@{user.displayName ?? user.lastName}"
android:text="@{list[index]}"
android:padding="@{large? @dimen/largePadding : @dimen/smallPadding}"
android:text="@{@string/nameFormat(firstName, lastName)}"
android:text="@{@plurals/banana(bananaCount)}"
```

Event handlers

View.OnClickListenre -> onClick() -> android:onClick

```
android:onClick="@{() -> currencies.sync()}"
android:onClick="@{currencies::sync}"
```

```
fun onClickFriend(view: View, friend:  User) { ... }
```

```
android:onClick="@{(view) -> user.onClickFriend(view, friend)}"
```

Variables

```
<data>
        <import type="android.graphics.drawable.Drawable"/>
        <variable name="user" type="com.example.User"/>
        <variable name="userImage" type="Drawable"/>
</data>
```

- Ако данните имплементират Observable ще бъдат наблюдавани за промени
- Генерират се get и set методи за всяка променлива генерирания клас за свързване

Includes

- Можете да преизползвате layout-и в други и да им подавате данни

```
<layout>
  <data>
    <variable name="user" type="com.example.User"/>
  </data>
  <LinearLayout>
    <include layout="@layout/name" bind:user="@{user}"/>
    <include layout="@layout/contact" bind:user="@{user}"/>
  </LinearLayout>
</layout>
```

Observable interface

- void addOnPropertyChangedCallback (Observable.OnPropertyChangedCallback callback)
- void removeOnPropertyChangedCallback (Observable.OnPropertyChangedCallback callback)

- void onPropertyChanged (Observable sender, int propertyId)

Observable fields

- ObservableBoolean, ObservableInt, ObservableDouble ...
- ObservableField<T>

```
class User {
    val firstName = ObservableField<String>()
    val lastName = ObservableField<String>()
    val age = ObservableInt()
}
```

```
user.firstName = "Ivan"
val age = user.age
```

BaseObservable

- Наследете BaseObservable
- Анотирайте get методите с @Bindable
- В set методите трябва да се извика notifyPropertyChanged(BR.propName)

```
@get:Bindable
var firstName: String = ""
    set(value) {
        field = value
        notifyPropertyChanged(BR.firstName)
    }
```

```
private String firstName;

@Bindable
public String getFirstName() {  return this.firstName; }

public void setFirstName(String firstName) {
    this.firstName = firstName;
    notifyPropertyChanged(BR.firstName);
}
```

Adapters

- Автоматично намиране на метод по името на атрибута

```
android:text -> setText()
app:scrimColor -> setScrimColor()
```

BindingAdapter

- Добавяне на логика при сетването на атрибут BindingAdapter

```
@BindingAdapter("android:paddingLeft")
fun setPaddingLeft(view: View, padding: Int) {
    view.setPadding(padding,
            view.getPaddingTop(),
            view.getPaddingRight(),
            view.getPaddingBottom())
}
```

```
@BindingAdapter("android:paddingLeft")
public static void setPaddingLeft(View view, int padding) {
    view.setPadding(padding,
            view.getPaddingTop(),
            view.getPaddingRight(),
            view.getPaddingBottom());
}
```

- Възможно е с един метод да се обработват няколко атрибута

```
@BindingAdapter("imageUrl", "error")
fun loadImage(view: ImageView, url: String, error: Drawable) {
    Picasso.get().load(url).error(error).into(view)
}
```

BindingAdapters with listeners

- Ако view има listener с един метод, може директно да се закачите към него с метод.

```
@BindingAdapter("android:onLayoutChange")
fun setOnLayoutChangeListener(view: View,
        oldValue: View.OnLayoutChangeListener?,
        newValue: View.OnLayoutChangeListener?) {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
        if (oldValue != null)
            view.removeOnLayoutChangeListener(oldValue)
        if (newValue != null)
            view.addOnLayoutChangeListener(newValue)
    }
}
```

```
<View android:onLayoutChange="@{() -> handler.layoutChanged()}"/>
```

BindingConverters

- Статични методи които конвертират от един тип в друг.

```
@BindingConversion
fun convertColorToDrawable(color: Int) = ColorDrawable(color)
```

Two way converters

```
object Converter {
    @InverseMethod("stringToDate")
    fun dateToString(value: Long): String {
        // Converts long to String.
    }

    fun stringToDate(value: String): Long {
        // Converts String to long.
    }
}
```

```
<EditText
    android:id="@+id/birth_date"
    android:text="@={Converter.dateToString(user.birthDate)}"/>
```

Two way adapters

```
object Adapters {
    @BindingAdapter( attribute="android:text" )
    fun applyDate(view: EditText, oldValue: Long, value: Long) {
        // Converts long to String and apply to view. Check for same value.
    }
    @InverseBindingAdapter( attribute="android:text" )
    fun getDate(view: EditText): Long {
        // Read text from view and convert it to date.
    }
}
```

```
<EditText
    android:id="@+id/birth_date"
    android:text="@={user.birthDate}"/>
```

Testing

Testing Model, ViewModel and Adapters

- Стандартни тестове за бизнес логика.
- Добре е връзката с модела да е през интерфейс, за да може да си имаме тестов модел или мок обект.
- Голяма част от UI логиката е във ViewModel-а.
- Converters са идеални за unit test.
- Adapters също могат да се unit тестват.

Testing View

- Обикновенно се ползват стандартни контроли, които са тествани.
- UI Automation тестване
- Ръчно тестване

Demo

Kotlin / Java

Thank you!

Контакти: kaloyan.donev@mobisystems.com

Използвана литература:

https://developer.android.com/topic/libraries/data-binding/
https://android.jlelse.eu/better-testing-with-mvvm-ae74d4d872bd
https://android.jlelse.eu/