



Reskilling 4Employment Software Developer

Acesso móvel a sistemas de informação

Bruno Santos

bruno.santos.mcv@msft.cesae.pt

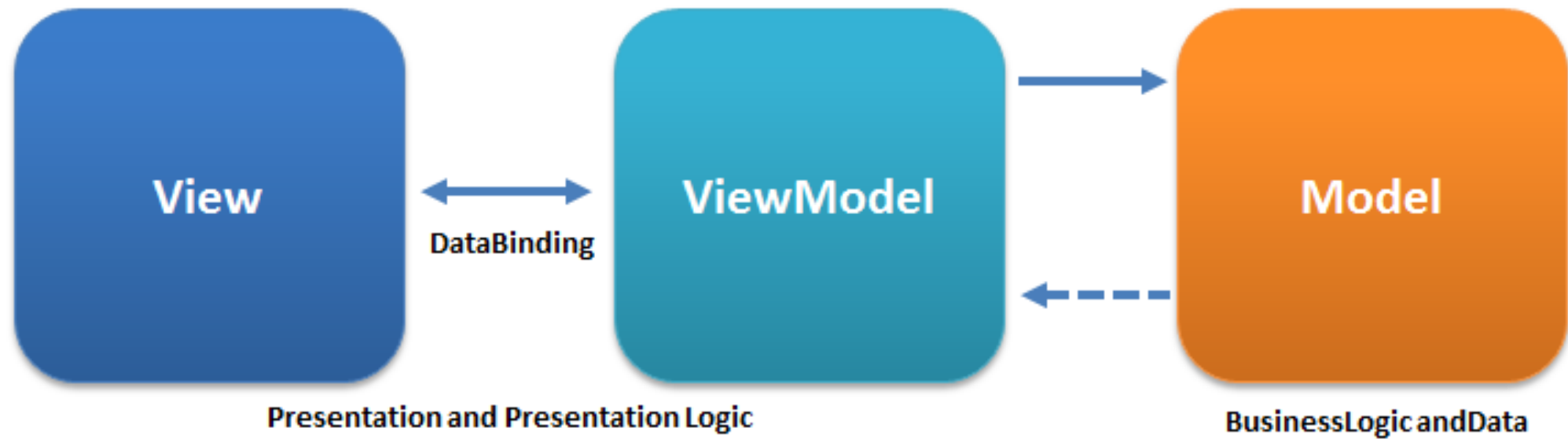
Tópicos

- MVVM (Model-View-ViewModel)

Android MVVM

- MVVM é um padrão de desenvolvimento de software, isto é, uma solução genérica e reutilizável para resolver um problema que acontece numa arquitetura de um software dado um contexto.
- O objetivo do MVVM é promover uma separação de responsabilidades, entre a view e sua lógica, aumentando, entre outros, a testabilidade da aplicação. Uma das características do MVVM face ao MVP (Model View Presenter) é a ausência de referência da View no ViewModel, sendo esta ligação feita indiretamente através de bindings.

Android MVVM



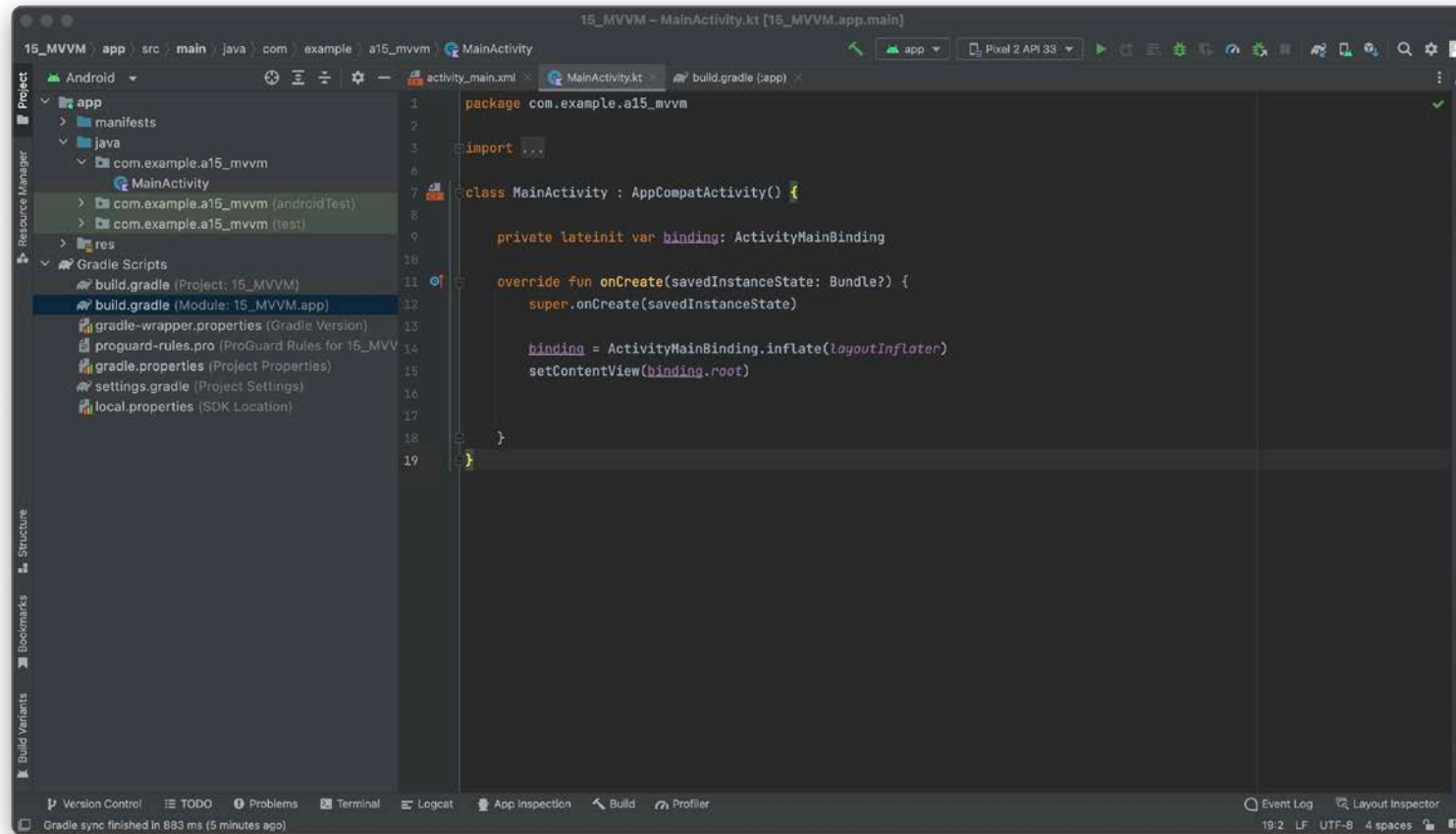
Android MVVM

- **View:** Entidade responsável por definir a estrutura, layout e aparência do que será exibido. Dentro do nosso contexto, as Views são as Activities, Fragments e elementos visuais criados.
- **Model:** Implementação do modelo de domínio da aplicação que inclui o modelo de dados, regras de negócio e validações de lógica.
- **ViewModel:** Camada intermédia entre a View e o Model, é o responsável por manusear o Model para ser utilizado pela View. Utiliza o databinding para notificar mudanças aos observadores (View).

Android MVVM

- Começamos por criar um projeto com uma Empty Activity

Android MVVM



Android MVVM

- De seguida criamos uma classe com o mesmo nome da Activity principal, alterando “Activity” por “ViewModel”
- E referenciamos a ViewModel na Activity

Android MVVM

MainActivity.kt

```
class MainActivity : AppCompatActivity() {  
  
    private lateinit var binding: ActivityMainBinding  
    private lateinit var viewModel: MainViewModel  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
  
        binding = ActivityMainBinding.inflate(layoutInflater)  
        setContentView(binding.root)  
  
        viewModel = ViewModelProvider(owner: this).get(MainViewModel::class.java)  
    }  
}
```

MainViewModel.kt

```
MainViewModel.kt  
1 package com.example.a15_mvvm  
2  
3 import androidx.lifecycle.ViewModel  
4  
5 class MainViewModel : ViewModel() {  
6     }  
7
```

Android MVVM

- Na classe ViewModel vamos referenciar os elementos de layout utilizando o LiveData para observar as alterações a serem executadas

Android MVVM

MainActivity.kt

```
viewModel = ViewModelProvider(owner: this).get(MainViewModel::class.java)

viewModel.welcome().observe(owner: this, Observer { it: String!
    binding.textWelcome.text = it
})
```

MainViewModel.kt

```
class MainViewModel : ViewModel() {

    private var textWelcome = MutableLiveData<String>()

    fun welcome(): LiveData<String> {
        return textWelcome
    }
}
```

Android MVVM

- Através da função de init vamos simular a atualização do valor da TextView

Android MVVM

```
class MainViewModel : ViewModel() {  
    private var textWelcome = MutableLiveData<String>()  
  
    init {  
        textWelcome.value = "Olá"  
    }  
  
    fun welcome(): LiveData<String> {  
        return textWelcome  
    }  
}
```



Olá



Android MVVM

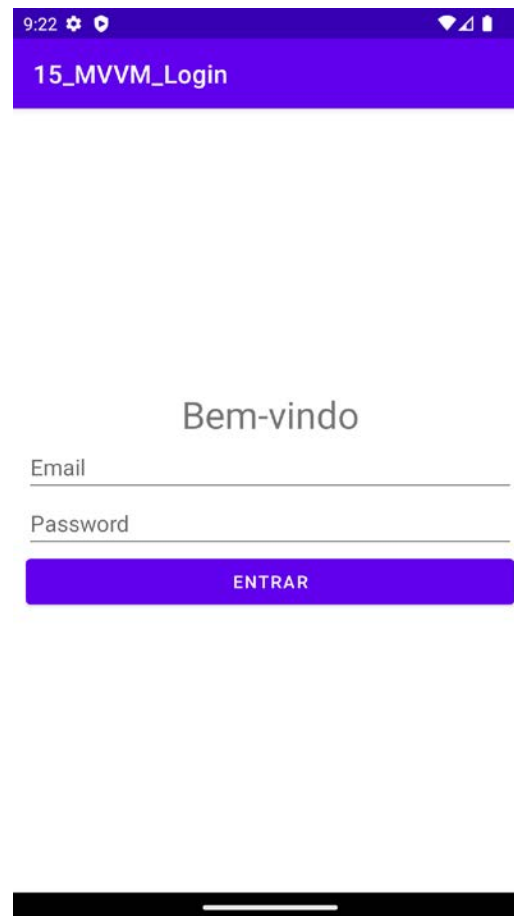
- Vamos agora criar um segundo exemplo simulando um ecrã de login numa aplicação.
- A aplicação terá um ecrã com dois campos editáveis para email e password e um botão para realizar o login.
- Existe também uma classe `PersonRepository` para simular a validação do login.
- Será criada a classe `ViewModel` para fazer a interligação entre o Modelo (`PersonRepository`) e a View (`MainActivity`)

Android MVVM

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res-auto"
3      xmlns:app="http://schemas.android.com/apk/res-auto"
4      xmlns:tools="http://schemas.android.com/tools"
5      android:layout_width="match_parent"
6      android:layout_height="match_parent"
7      android:gravity="center"
8      android:orientation="vertical"
9      android:padding="10dp"
10     tools:context=".MainActivity">
11
12     <TextView
13         android:layout_width="match_parent"
14         android:layout_height="wrap_content"
15         android:gravity="center"
16         android:text="Bem-vindo"
17         android:textSize="30sp" />
18
19     <EditText
20         android:id="@+id/edit_email"
21         android:layout_width="match_parent"
22         android:layout_height="wrap_content"
23         android:hint="Email" />
24
```

```
19     <EditText
20         android:id="@+id/edit_email"
21         android:layout_width="match_parent"
22         android:layout_height="wrap_content"
23         android:hint="Email" />
24
25     <EditText
26         android:id="@+id/edit_password"
27         android:layout_width="match_parent"
28         android:layout_height="wrap_content"
29         android:hint="Password" />
30
31     <Button
32         android:id="@+id/button_entrar"
33         android:layout_width="match_parent"
34         android:layout_height="wrap_content"
35         android:text="Entrar" />
36
37
38 </LinearLayout>
```

Android MVVM



Android MVVM

- Existe também uma classe `PersonRepository` para simular a validação do login.
- No método `login` existente estamos apenas a validar se o `username` é “admin” e a `password` “pass”, no entanto, num exemplo real, faríamos a validação utilizando, por exemplo uma base de dados.

Android MVVM

```
class PersonRepository {  
    fun login(email: String, password: String): Boolean {  
        return (email.equals("admin") && password.equals("pass"))  
    }  
}
```

Android MVVM

- Será criada a classe ViewModel para fazer a interligação entre o Modelo (PersonRepository) e a View (MainActivity)
- Aqui foram criadas duas funções:
 - login – atualiza o valor na View (LiveData)
 - doLogin – vai ao PersonRepository validar os dados de login e atualiza a variável de LiveData do ViewModel

Android MVVM

```
class MainViewModel : ViewModel() {  
  
    private var login = MutableLiveData<Boolean>()  
    private val personRepository = PersonRepository()  
  
    fun login(): LiveData<Boolean> {  
        return login  
    }  
  
    fun doLogin(email: String, password: String) {  
        login.value = personRepository.login(email, password)  
    }  
}
```

Android MVVM

- Na View (MainActivity):
 - Associamos a View com o ViewModel
 - Invocamos o método observe para apresentar o resultado da operação login
 - Implementamos o código para clique no botão e chamar a função doLogin do ViewModel

Android MVVM

```
class MainActivity : AppCompatActivity() {  
  
    private lateinit var binding: ActivityMainBinding  
    private lateinit var viewModel: MainViewModel  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        binding = ActivityMainBinding.inflate(layoutInflater)  
        setContentView(binding.root)  
  
        viewModel = ViewModelProvider(owner: this).get(MainViewModel::class.java)  
  
        viewModel.login().observe(owner: this, Observer { it: Boolean! } {  
            if (it) {  
                Toast.makeText(context: this, text: "Login OK", Toast.LENGTH_SHORT).show()  
            } else {  
                Toast.makeText(context: this, text: "Login Errado", Toast.LENGTH_SHORT).show()  
            }  
        })  
  
        binding.buttonEntrar.setOnClickListener { it: View! } {  
            val email = binding.editEmail.text.toString()  
            val password = binding.editPassword.text.toString()  
  
            viewModel.doLogin(email, password)  
        }  
    }  
}
```

Android MVVM

