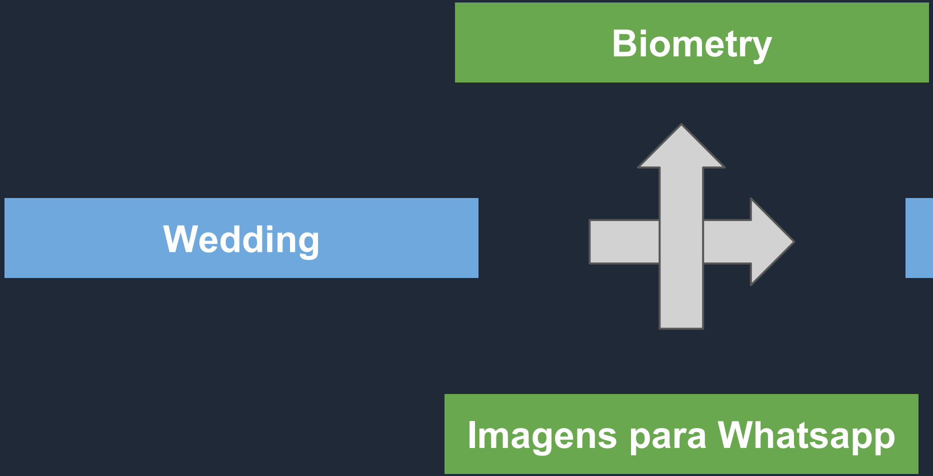
Android App Architecture

Gerson Silva Filho

June 2018

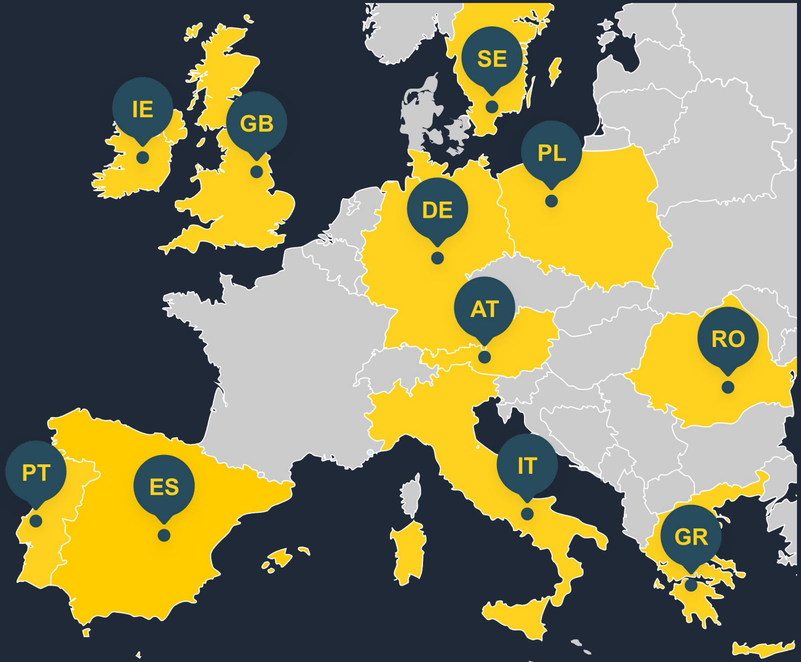
Who am I

- Android Dev mytaxi
- 10 + apps in the stores
- CEFETE (UTFPR)
- Apple Developer Academy
- FotoFun (Eternal)



mytaxi - die taxi app

- Aplicativo de ride-hailing líder no mercado europeu
- 150 cidades
- 500+ funcionários
- 100.000 motoristas registrados
- 2.400 requests por minuto



motivation

- Rx
- DI
- Web / API
- Design patterns
- Good encapsulation
- Android framework (lifecycle, intents, fragments, UI, resources)
- Uses caching/ no unnecessary network calls
- Tests
- Kotlin

motivation

- ~~Rx~~
- ~~DI~~
- ~~Web / API~~
- **Design patterns** ⟵
- **Good encapsulation** ⟵
- **Android framework (lifecycle, intents, fragments, UI, resources)** ⟵
- ~~Uses caching/ no unnecessary network calls~~
- **Tests** ⟵
- **Kotlin** ⟵

why

- Scalability
- Testability
- Separation of concerns
- Maintainability

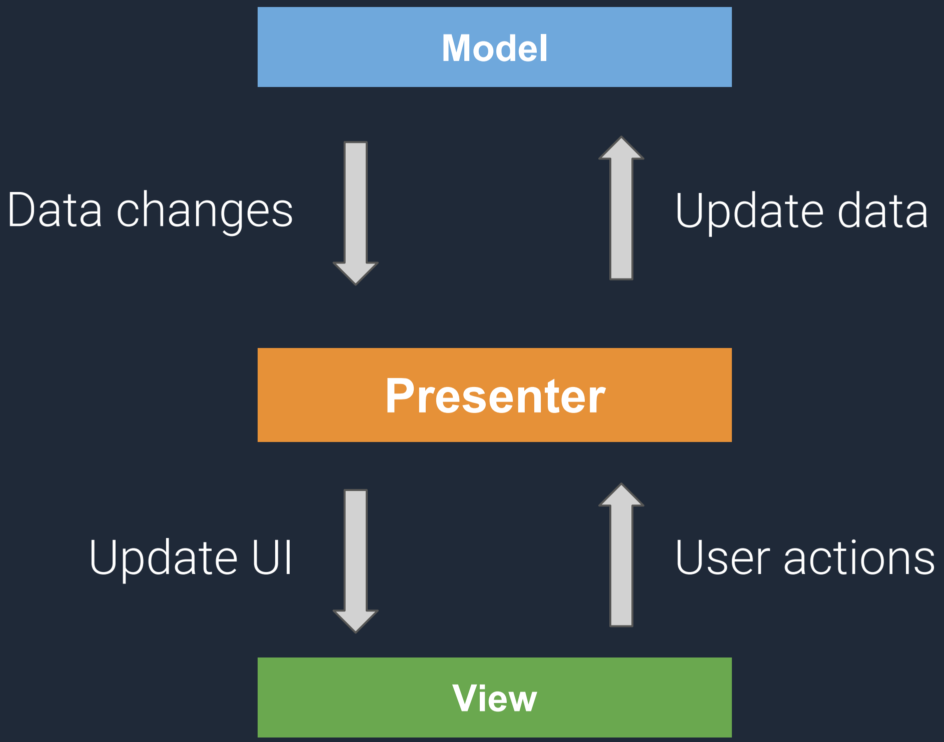
patterns

- MVC
- MVP
- MVVM
- Single flows (Flux / Redux)

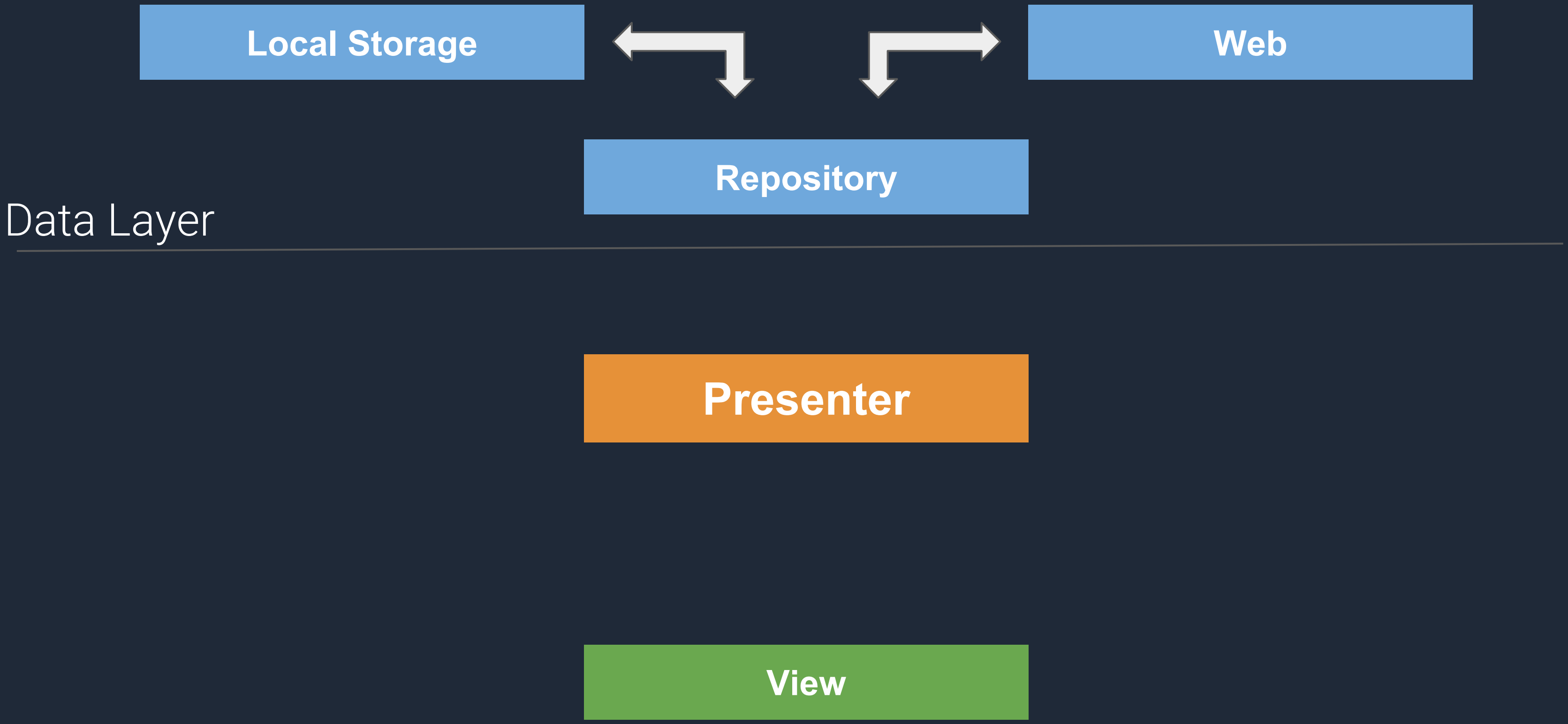
patterns

- ~~MVC~~
- MVP
- MVVM
- ~~Single flows (Flux / Redux)~~

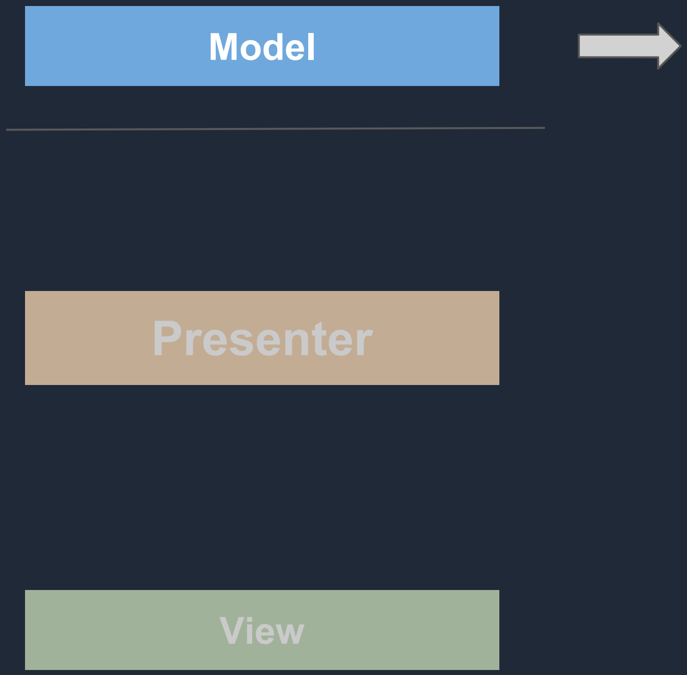
MVP


Model
Data changes
Update data
Presenter
Update UI
User actions
View

MVP

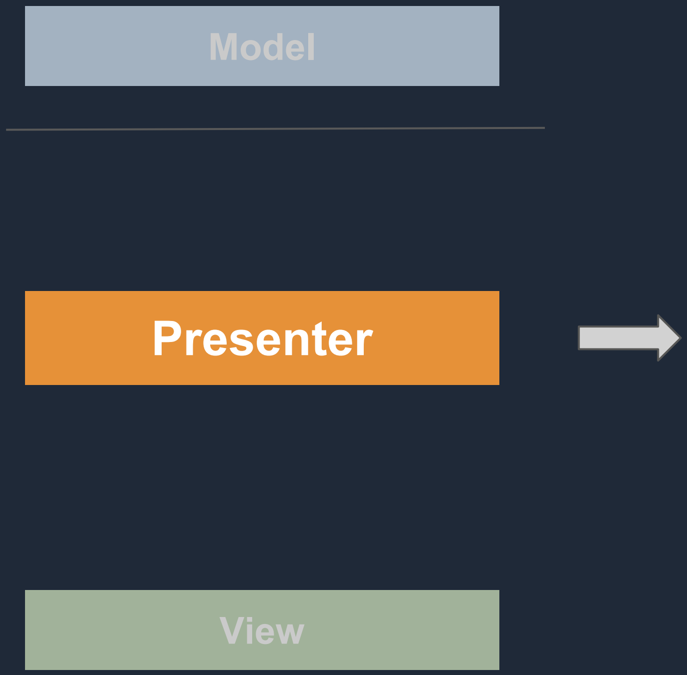


MVP



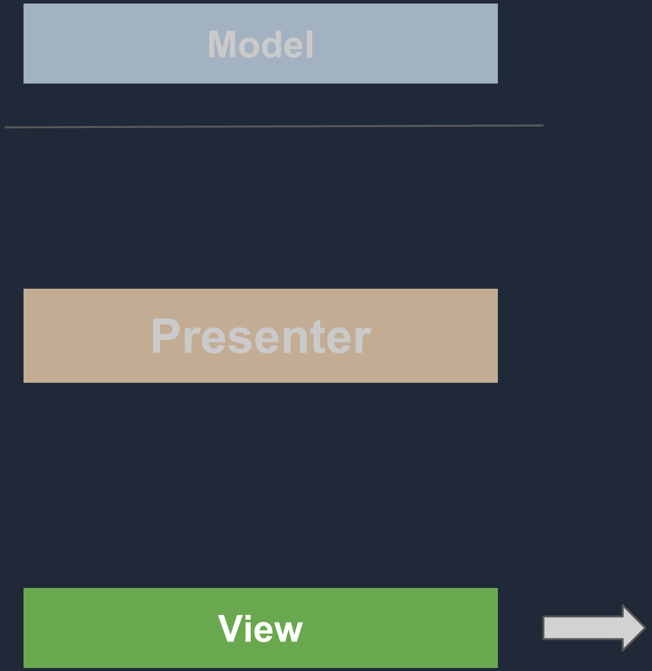
- Business Rules
- Pull / Push Data
- Models / Abstractions

MVP



- Presentation Layer
- NO ANDROID DEPENDENCIES (pure kotlin)
- No Android Lifecycle
- Interfaces Everywhere
- 100% test coverage
- **knows View**

MVP



- Android Framework (Activities / Fragments)
- UI Manipulation - As Dummy as Possible
- Create / Inject the interface of presenter

MVP

Model

Presenter

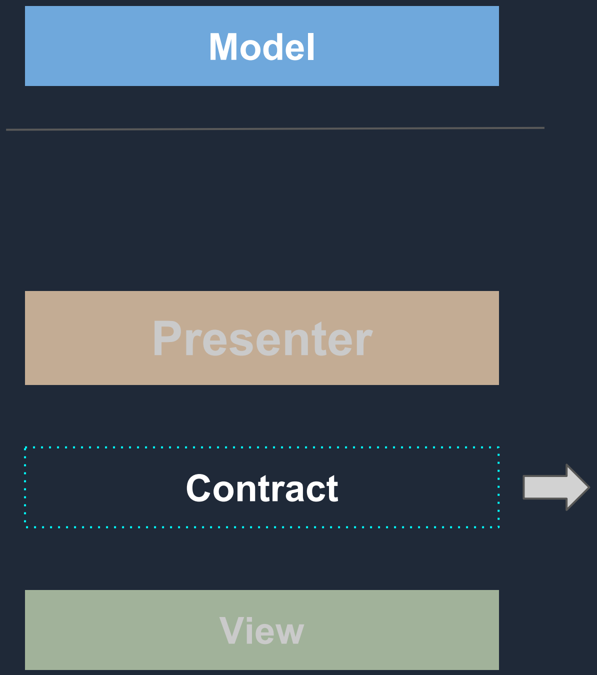
View

```
data class Game(
    var score1:Int, var team1:String,
    var score2:Int, var team2:String)
```

```
interface IGameRep {
    fun getGame(year: Int): Game?
}
```

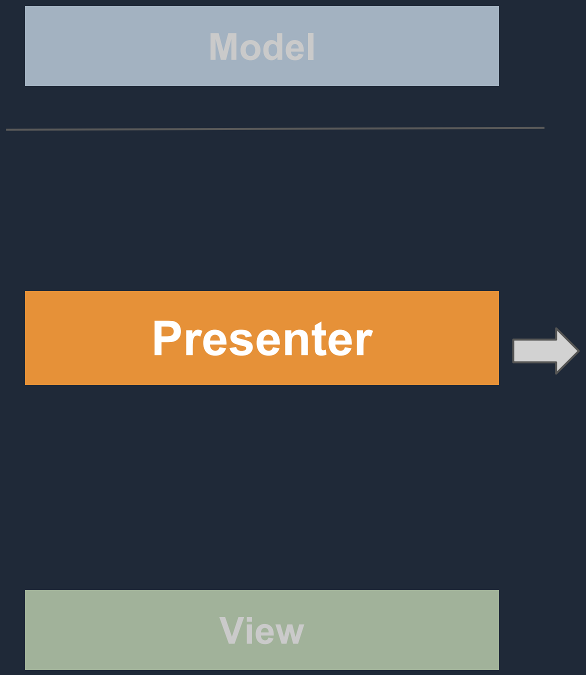
```
class GameRepository : IGameRepository {
    override fun getGame(year: Int): Game? {
        return when (year) {
            2002 -> Game(2, "Brazil", 0, "Germany")
            2018 -> Game(0, "Germany", 1, "Brazil")
            else -> null
        }
    }
}
```

MVP



```
interface ScoreContract {
    interface View{
        fun showGame(placar: String)
        fun setTeam1(team1: String)
        fun setTeam2(team2: String)
        fun showError(error: String)
    }
    interface Presenter{
       fun getGame(ano:String)
    }
}
```

MVP



```
class ScorePresenter(
     val view:ScoreContract.View,
     val repository: IGameRepository
): ScoreContract.Presenter{ ... }
```

```
override fun getGame(yearText: String) {
     val year = yearText.toIntOrNull()
     if (year == null) {
          showError("Invalid year")
     } else {
          val game = repository.getGame(year)
          if (game == null) {
               showError("No game found")
          } else {
               setGame(game)
          }
     }
}
```

MVP

Model

Presenter

View

```
private fun showError(error: String) {
     view.showError(error)
     view.setTeam1("")
     view.setTeam2("")
}

private fun setGame(game: Game) {
     view.showGame("${game.score1} : ${game.score2}")
     view.setTeam1(game.team1)
     view.setTeam1(game.team2)
}
```

MVP

Model

Presenter

View

```
class ScoreActivityMvp : AppCompatActivity(),
ScoreContract.View {

    lateinit var presenter: ScoreContract.Presenter

    override fun onCreate(savedInstanceState: Bundle?)
    {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_score)

        //Should be injected
        presenter = ScorePresenter(this)
        setButtonClick()
    }
}
```

MVP

Model

Presenter

View

```
private fun setButtonClick() {
    buttonGetGame.setOnClickListener {
        presenter.getGame(editTextYear.text.toString())
    }
}

override fun showGame(placar: String) {
    textViewResult.text = placar
}

override fun setTeam1(team1: String) {
    textViewTeam1.text = team1
}

override fun showError(error: String) {
    textViewResult.text = error
}
```

MVP

Model

Presenter

View

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout>
   <EditText
       android:id="@+id/editTextYear"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content" />

   <Button
       android:id="@+id/buttonGetGame"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content" />

   <TextView
       android:id="@+id/textViewResult"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"/>
...
</android.support.constraint.ConstraintLayout>
```

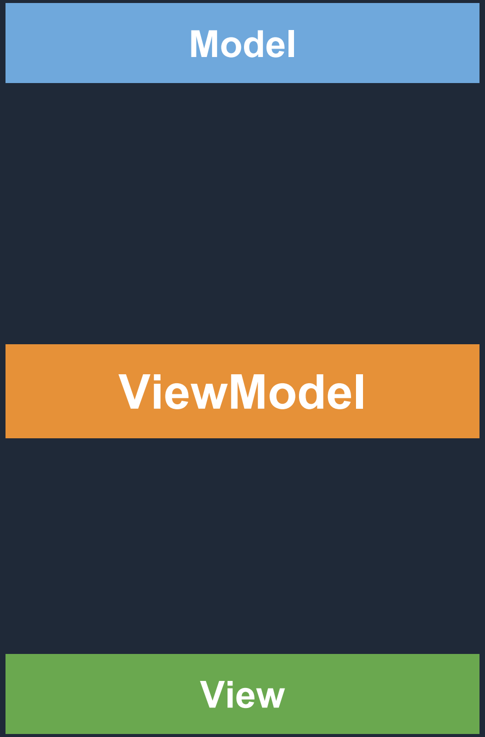
MVP – Unit Test

```
class PresenterTest {
    val repositoryMock: IGameRepository = mock()
    val viewMock: ScoreContract.View = mock()
    lateinit var presenter: ScorePresenter

    @Before
    fun setup() { presenter = ScorePresenter(viewMock, repositoryMock)}

    @Test
    fun `test game found`() {
        val game = Game(1, "Brasil", 0, "Germany")
        whenever(repositoryMock.getGame(any())).thenReturn(game)
        presenter.getGame("1991")
        verify(viewMock).setTeam1(game.team1)
        verify(viewMock).setTeam2(game.team2)
        verify(viewMock).showGame("1 : 0")
    }
}
```

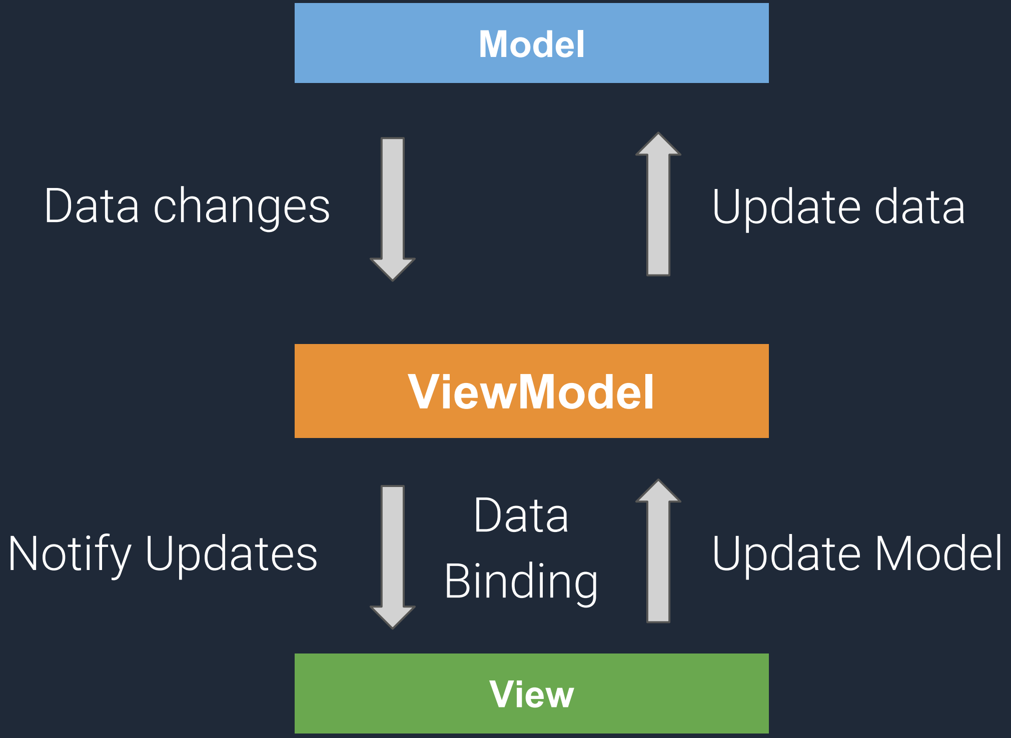
MVP

- Easy to understand
- Clear separation
- Testable
- Easy to mock stuff
- UI Independent
- Some boilerplate – a lot of classes

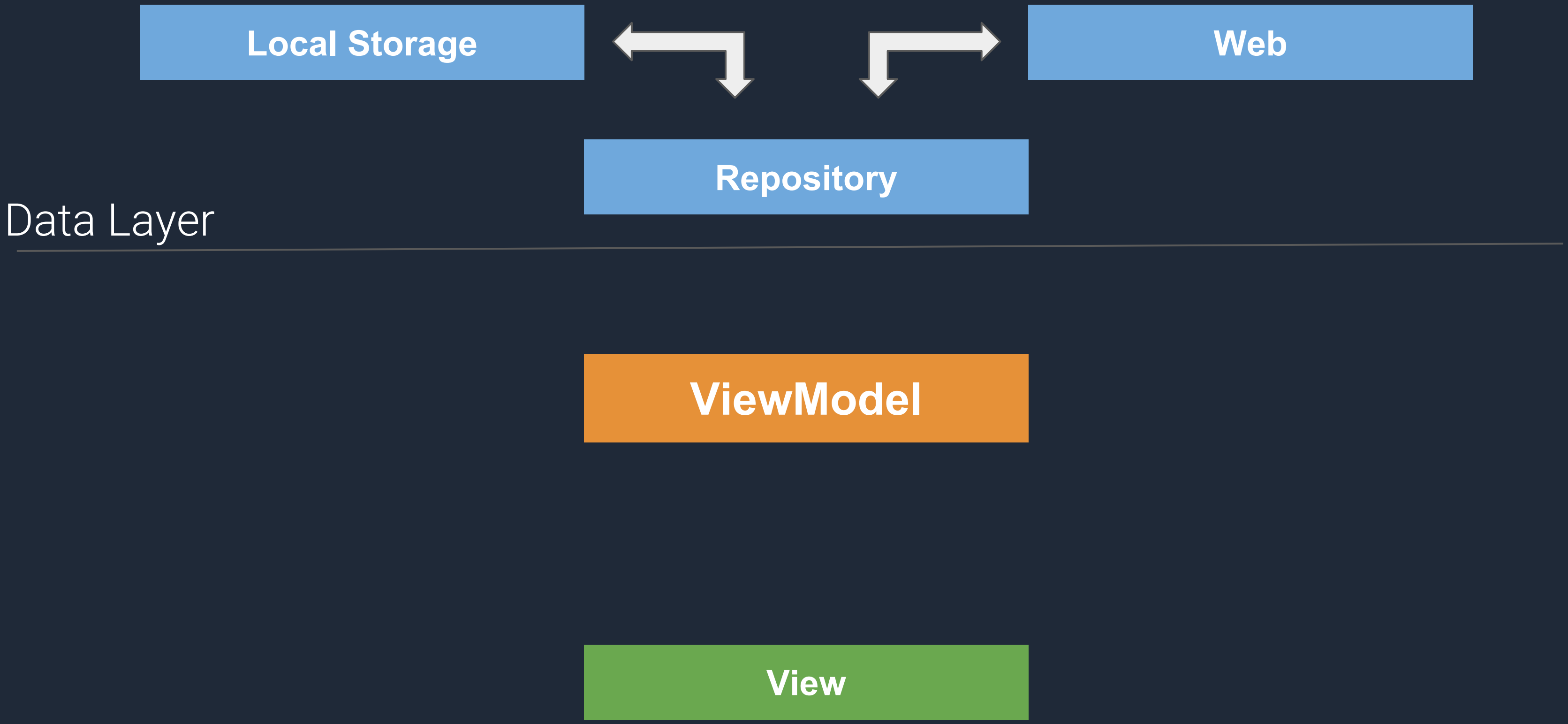
MVVM



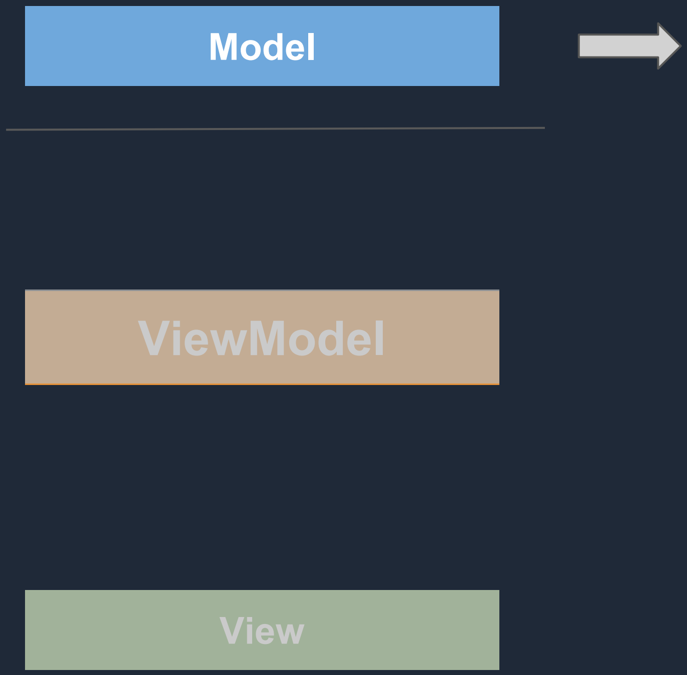
MVVM


Model
Data changes
Update data
ViewModel
Notify Updates
Data Binding
Update Model
View

MVVM

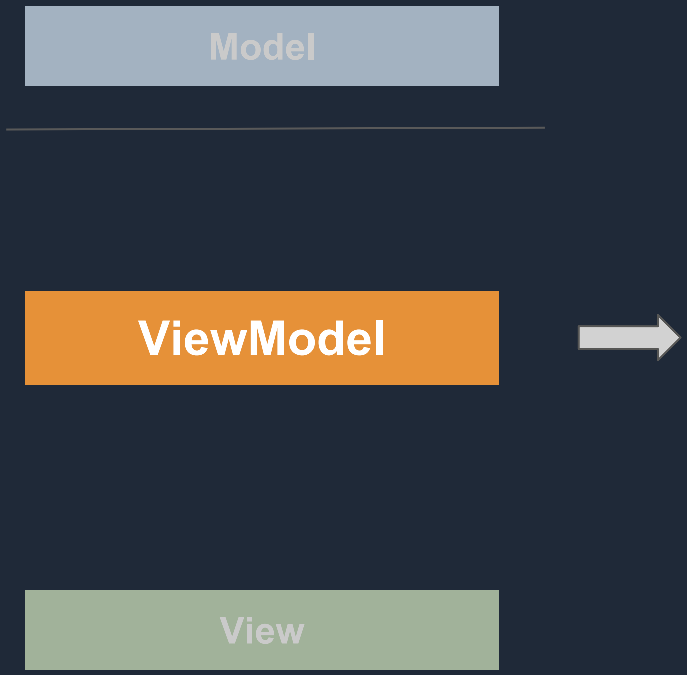


MVVM



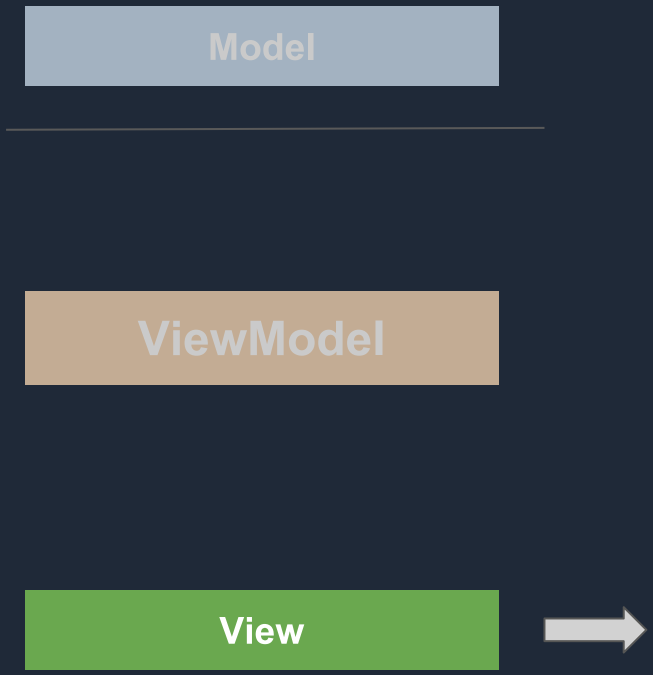
- Business Rules
- Pull / Push Data
- Models / Abstractions

MVVM



- Presentation Layer
- NO ANDROID DEPENDENCIES (pure kotlin)
- No Android Lifecycle
- 100% test coverage by Unit Tests
- **Don't know View**
- android.arch (Architecture Components)
- **Data Binding**

MVVM



- Android Framework (Activities / Fragments)
- UI Manipulation - As Dummy as Possible
- Create / Inject the ViewModel
- UiTests with espresso
- DataBinding in the xml

MVVM

Model

ViewModel

View

```
data class Game(
    var score1:Int, var team1:String,
     var score2:Int, var team2:String)
```

```
interface IGameRep {
    fun getGame(year: Int): Game?
}
```

```
class GameRepository : IGameRepository {
    override fun getGame(year: Int): Game? {
        return when (year) {
            2002 -> Game(2, "Brazil", 0, "Germany")
            2018 -> Game(0, "Germany", 1, "Brazil")
            else -> null
        }
    }
}
```

MVVM

Model

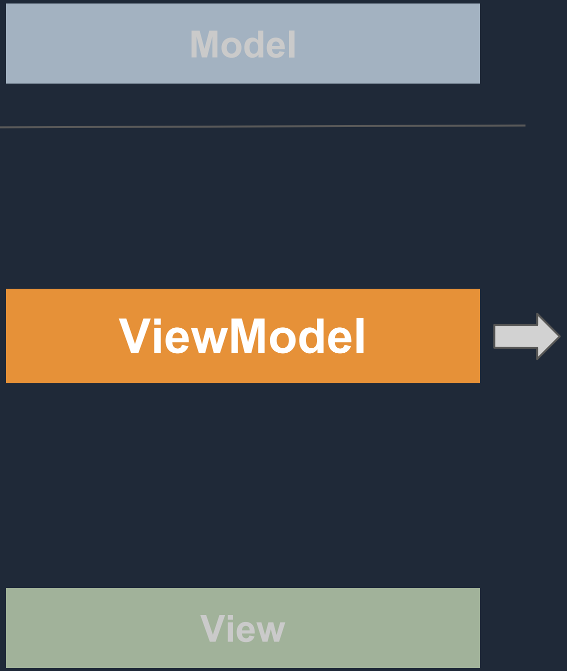
ViewModel

View

```
class ScoreViewModel(val repository:IGameRepository){
    var resultLabel = ObservableField("")
    var nameTeam1 = ObservableField("")
    var nameTeam2 = ObservableField("")
    var yearLabel: String = ""
}
```

```
fun getGame() {
    val year = yearLabel.toIntOrNull()
    if (year == null) {
        setError("Invalid year")
    } else {
        val game = repository.getGame(year)
        if (game == null) {
            setError("No game found.")
        } else {
       setGame(game) }
    }
}
```

MVVM



```
private fun setGame(game: Game) {
    resultLabel.set("${game.score1}:${game.score2}")
    nameTeam1.set(game.team1)
    nameTeam2.set(game.team2)
}

private fun setError(error: String) {
    resultLabel.set(error)
    nameTeam1.set("")
    nameTeam2.set("")
}
```

MVVM

Model

ViewModel

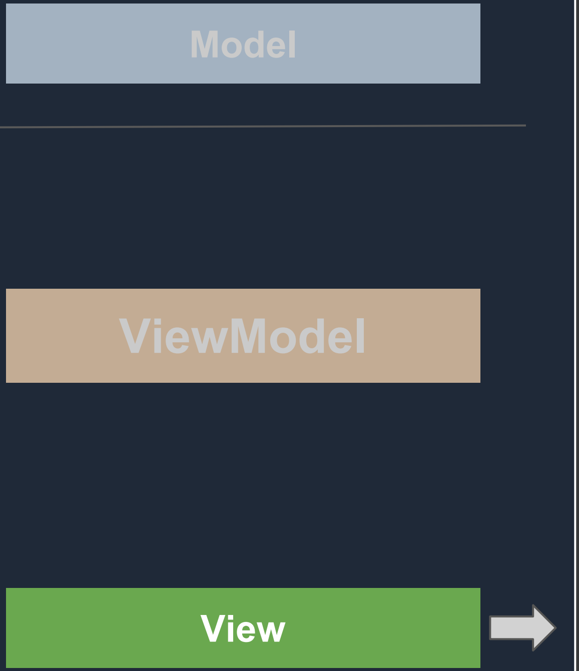
View

```
dataBinding {
    enabled = true
}
```

build.gradle

```
class ScoreActivityMvvm : AppCompatActivity() {
    lateinit var binding: ActivityScoreMvvmBinding
    override fun onCreate(savedInstanceState: Bundle?)
    {
        super.onCreate(savedInstanceState)
        binding = DataBindingUtil.setContentView(
            this,
            R.layout.activity_score_mvvm
        )

        //To be injected
        binding.vm = ScoreViewModel()
    }
}
```

MVVM



```
<layout>
    <data>
        <variable
            name="vm"
       type=".mvvm.ScoreViewModel" />
    </data>
    <android.support.constraint.ConstraintLayout>

     <EditText android:text=" @={vm.yearLabel}"/>
     <Button
     android:onClick=" @{() -> vm.getGame()}"/>

     <TextView android:text=" @{vm.resultLabel}"/>
     <TextView android:text=" @{vm.nameTeam1}"/>
     <TextView android:text=" @{vm.nameTeam2}"/>
</layout>
```

MVVM – Unit Tests

```
class ViewModelTest {
    val repositoryMock: IGameRepository = mock()
    lateinit var viewModel: ScoreViewModel

    @Before
    fun setup() { viewModel = ScoreViewModel(repositoryMock)}

    @Test
    fun `tests invalid year`() {
        whenever(repositoryMock.getGame(any())).thenReturn(null)
        viewModel.yearLabel = "test not year"
        viewModel.getGame()
        assertEquals(viewModel.nameTeam1.get(), "")
        assertEquals(viewModel.nameTeam2.get(), "")
        assertEquals(viewModel.resultLabel.get(), "Invalid year")
    }
}
```

MVVM

- Clear separation
- Testable
- UI Independent
- Less code than MVP
- Libraries (Android Architecture components)
- Data Binding

Obrigado

https://github.com/GersonSilvaFilho

https://www.linkedin.com/in/gersonsilvafilho/