ANDROID PROJECT

Pierre CHAMPION
Rémi BOUVET
Tony MARTEAU

SmartWallRemote

MASTER 1 - INFORMATIQUE

18/01/2018 - 20/03/2018

https://github.com/Drakirus/SmartWallRemote

Introduction

This project is proposed as part of the embedded interface programming course. The objectives are to acquire fundamentals knowledge of programming graphical mobile interface and fundamentals knowledge of Androïd application development.

Our work group is composed of three students in apprenticeship. Pierre CHAMPION who has already made some Android application. Rémi BOUVET and Tony MARTEAU who are discovering the Androïd world.

We have chosen to divide the work into three categories:

1. Pierre did the general architecture of the application, the parsing of the JSON and the main view.
2. Rémi did the video choose view.
3. Tony did the layout, and the scenario choose.

The following parts of the report are the requirement of the customer; the solution what we imagined; the presentation of how we implement our solution; the possible evolutions and the conclusion of this project.

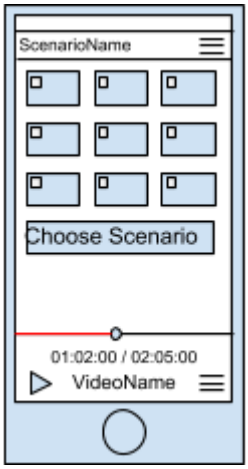
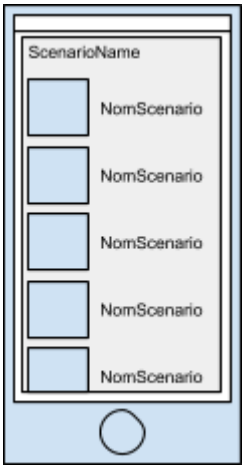
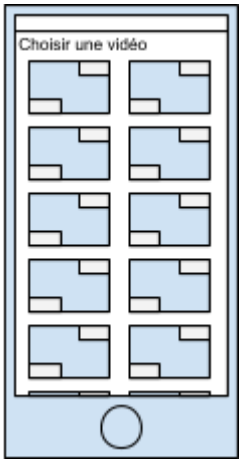
Requirement

The goal of the WallControl project that we called "SmartWallRemote" was to create a remote to manage a screen wall. The screen wall can display different type of scenarios. A scenario is a combination of several screen group with their video and the general layout.

The purpose of the remote is to command the wall by choosing one or multiple screens (group) and making an action like pause or play the video, change the video, change the layout, create a layout, create a scenario.
The remote is supposed to transmit a command to the API of the screen wall where the persistence is managed. That implies that a scenario list and a video list is retrieved from the wall API at the application initialization. The remote can also be used to commit new scenario created from it.

Solution

We decided to make a single view to facilitate the user interactions.

Main interface	Choose a layout/scenario	Choose a video
		

On the top of the main view, we can find the wall name. On the right corner, a menu button is available where we can choose the wall configuration, the server's settings and the information about the application.

Under the top headband, we can find the screen organization with screen selectable. When user select screen(s), they can do actions over, like play, pause the video, and he can move forward and backward the video timing. The users can also create a layout when they select screen(s) and save it.

Between the screen organization and the player, we can find a button used to select the layout and the scenario. When the user clicks over, a view appears in fade in, and the user can choose which scenario/layout he wants.

Swiping from bottom to top make a video chooser appear. Each video is described by, a name, a duration and an illustrated picture.

Implementation

The Languages

This Implementation makes use of the new Kotlin programing language.

One of the collaborators was already familiar with the android world, and he wanted to get himself a preview of the new Kotlin language.

Kotlin is now an official language on Android. It's expressive, concise, and powerful. Best of all, it's interoperable with our existing Android languages and runtime. Its integration in Android studio is very polished.

Kotlin allows us to write safer code and avoid one of the main cause of java errors the "NullPointerExceptions".

```
var output: String // Can only store a String
output = null      // Compilation error
// ==================================
val name: String? = null     // Nullable type
println(name.length())       // Compilation error
println(name?.length())      // Valid - print "null"
```

The app uses only one activity where multiple components interact with. The main activity was build using the new Anko DSL introduced by Kotlin. With Anko, you can create UI from Kotlin code. This DSL is a valid substitute for XML layouts in most cases. It is mature and stable enough for production.

If you want to learn more about Anko check out those link:

- 400% faster layouts with Anko
- Why every Android developer should use Anko

Here is a simple UI wrote with Anko DSL: (this is the complete layout code)

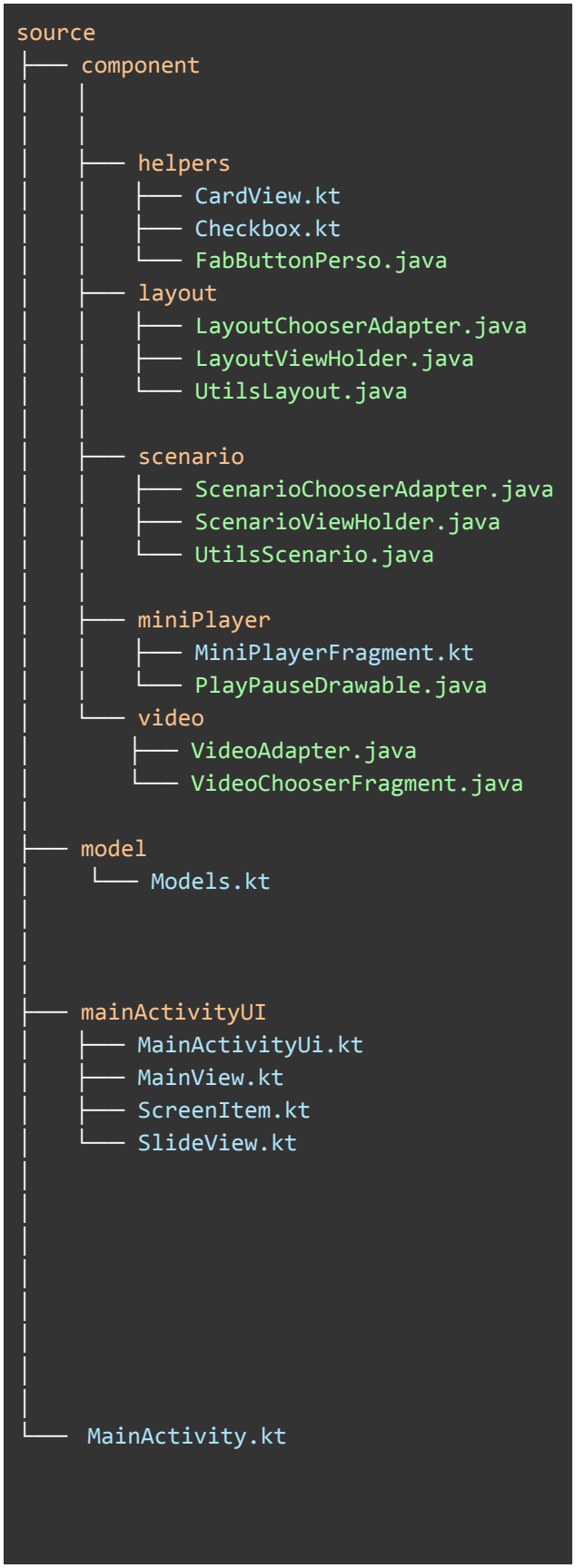
```
verticalLayout {
  val name = editText()
  button("Say Hello") {
    onClick {toast("Hello, ${name.text}!")}
  }
}
```

The code creates a button inside a LinearLayout and attaches an OnClickListener to that button. Moreover, onClick accepts a suspend lambda, so you can write your asynchronous code right inside the listener! **No XML is required!**

Since everyone wasn't ready to make the switch to Kotlin and Anko. Some of the components were builds using the old Java + XML combination.

The Structure

The app's file structure is this presented below.



The component package contains some graphical part of the app. The only requirement is model.

The helpers are some kotlin extension function to add methods to an existing android class

---

The layout and scenario packages contain the Adapters for the Layout and the scenario chooser. Those Adapters are designed to work with the Dialogplus dependency.

The miniPlayer expose a Fragment. It provide an interactive way of controlling the Video. Inspire by Phonograph

---

The video expose a Fragment. It provide a RecyclerView that display the thumbnails of the videos. And a SearchView for the user to find a video by it's name

---

The Models.kt is a 150 lines file containing all the Kotlin Data class. Using a single file to describe all of your **models** make it very readable and shareable.

The mainActivityUI packages is the **view** of our app. The entry point is MainActivityUI.kt, and since we are using a Sliding Up Panel, this view is straightforward, we just need to have 2 views the mainView.kt and the slideView.kt. One advantage of Anko is that we can mix the for loop that generates the grid and the component itself this ability allows us to create the UI without doing it in Java. (which would be really painful..)

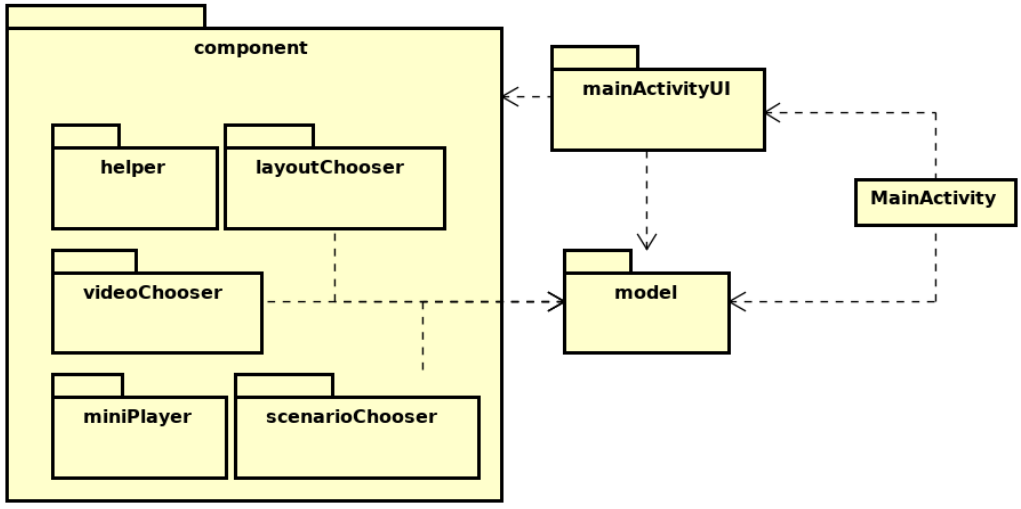
To glue everything together, the MainAcrivity.kt class act as your **controller**.

The Dependency used

As you saw, to reach our needs, we used quite a bit of libraries. Here a list of them:

Anko	The DSL used to generate a lot of component in the app
Sliding Up Panel	A simple way to add a draggable sliding up panel
Dialogplus	Advanced dialog solution for android
Fuel	The easiest HTTP networking library for Kotlin/Android
Gson	Serialization/deserialization to convert Java Objects into JSON and back
Toasty	The usual Toast, but with style
Colorpicker	A simple color picker library for Android
FabButton	A Floating ActionButton with a progress indicator ring
Faker	Used to generates fake data
KeyboardVisibilityEvent	A Library to handle soft keyboard visibility change event.

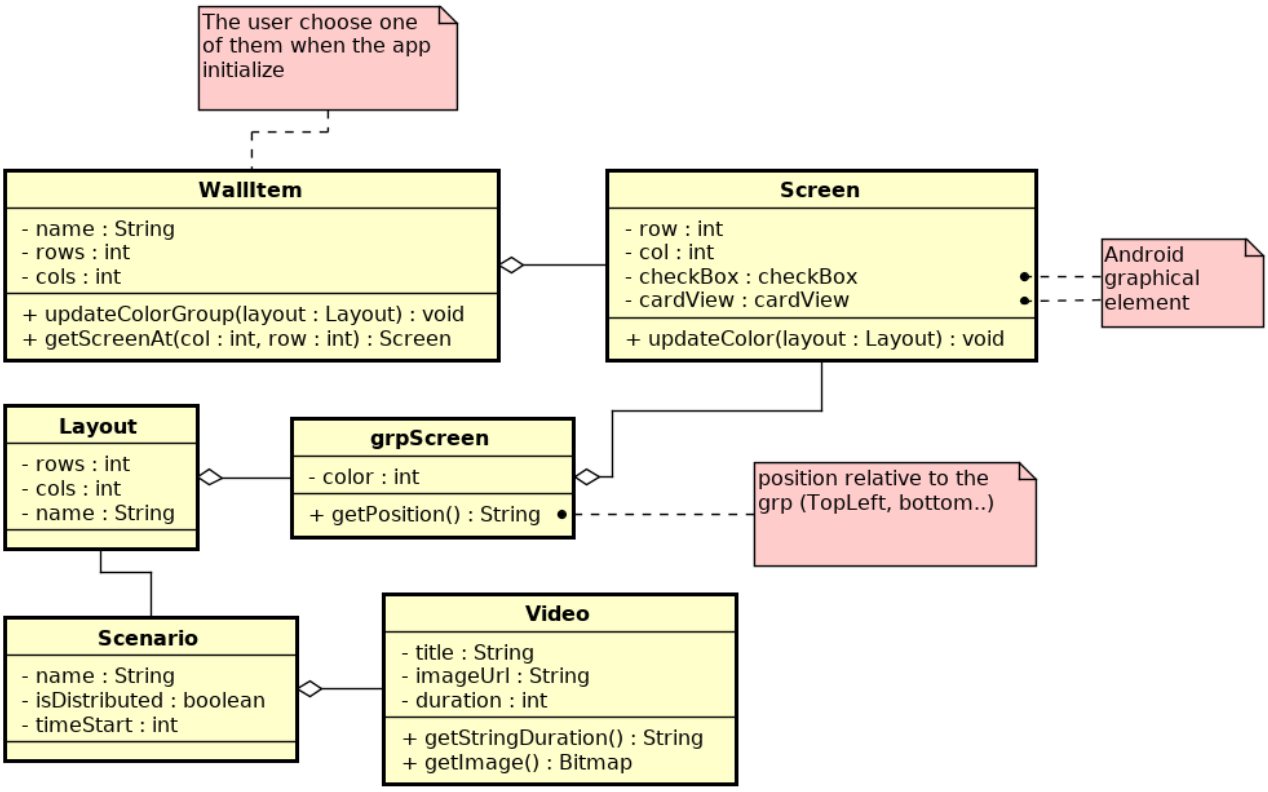
Functional dependency diagram



Since the class diagram of an MVC project is always the same, we decided to make a functional dependency diagram, combined with the Structure description above it should give you a pretty good understanding of our architecture.

One of the alternatives would be to have a part of the model in each of the components. This way we could have a better outsource of the component packages, since every component would have its own model.

Class Diagram of the Model



This model is **initialized at runtime** using the library Fuel and the JSON files in the json directory of our Github. Some evolution need to be made to track the current time of the videos played on the screens.

Current state & Evolution

At the time that we deliver the project, the interface is complete, and we are loading every part dynamically from JSON files. Since the API to the screen wall was not provided, we did not implement the command functionality, so the remote isn't functional.

We can provide some projection of the possible evolution of our application. We could add a progress bar that shows the current state of the video/layout that the user can see the progress of each video displaying with a quick look so he can react to them efficiently.
The wall state should be saved on the wall, and the API should manage the synchronization.

Conclusion

The goal of this project was for us, to learn a new framework and architecture by making by making an Android application while dealing with the creation of user interface.
The skills we learn are :

- Discover the Android Studio IDE
- Learn to use the Android Framework
- Discover a new programing language
- Make use of pre-existing android's libraries

Tony Marteau

The discovery of Androïd was complicated at first because I am not familiar with Java. But the team motivated me and allowed me to discover a new language and the Androïd Framework of an application. Today, even if I do not think to make Android app later, this open-mindedness was refreshing for my general computer science background.

Rémi Bouvet

As a first time dealing with an Android project, it was an exciting way to improve my understanding of how we design user interfaces and embedded programming. In an increasingly mobile world, I think it is vital to have an overview of the creation of mobile application as a developer. The project allowed me to improve my Java skills and to discover Kotlin and Anko.

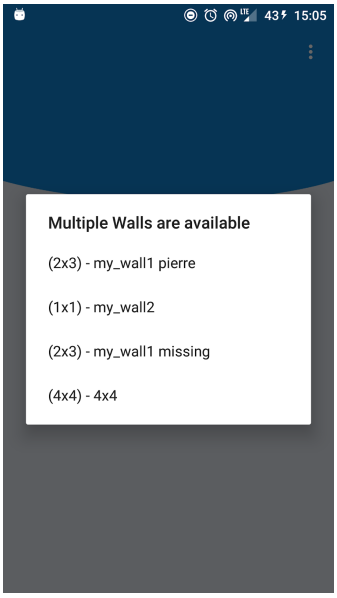
Pierre Champion

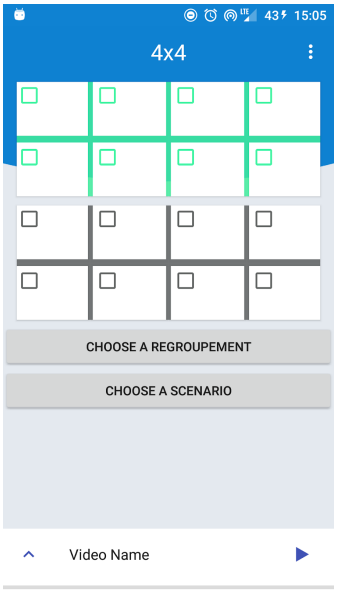
Using Java and Kotlin together weren't an issue. Instead, it was quite refreshing to have a look at another programing language. I think that making the dynamic grid generation in Java would have been too wordy, that way by making it using Anko and Kotlin we keep a clean and maintainable app. I think that Java app (not necessary Android one) are going to use more and more Kotlin since the two are 100% interoperable. Kotlin allows the user to do more thing without having the Java overthinking and while keeping all of the excellent libraries made in Java.

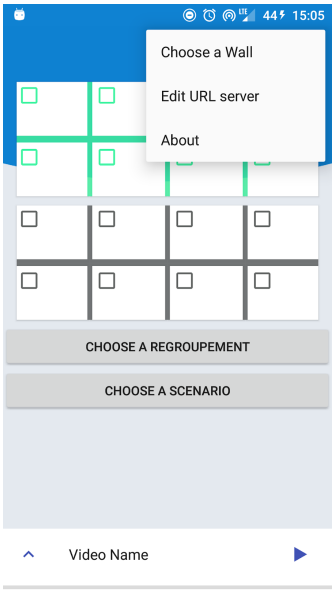
After making the app, I made myself a good opinion of the 'new way' of creating an Android app. I don't think I could come back to the classic Java + XML way since Kotlin is a very appealing language that solves a lot of the Java issue with Style.

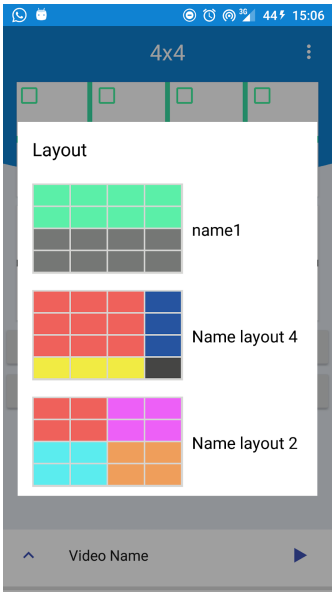
Side Note: Since the android studio is based on IntelliJ IDEA, made by the same company as the one that made Kotlin, I don't think that Google has had a lot of choices while choosing the new programming language for Android. This allows kotlin to have a secure long-term future in the Android industry. Kotlin is the Second Most Loved Language in StackOverflow Survey 2018.

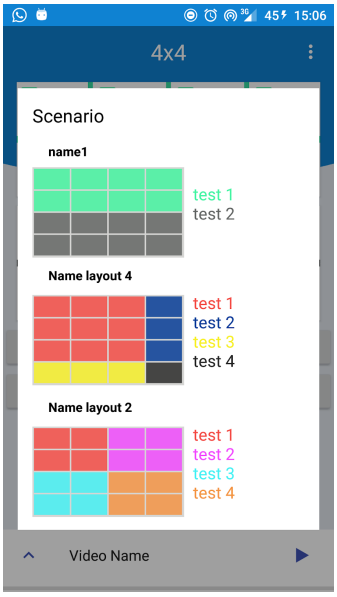
ANNEXES

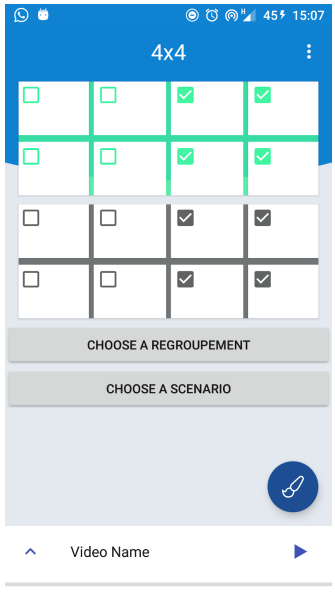

Multiple Walls are available
(2x3) - my_wall1 pierre
(1x1) - my_wall2
(2x3) - my_wall1 missing
(4x4) - 4x4

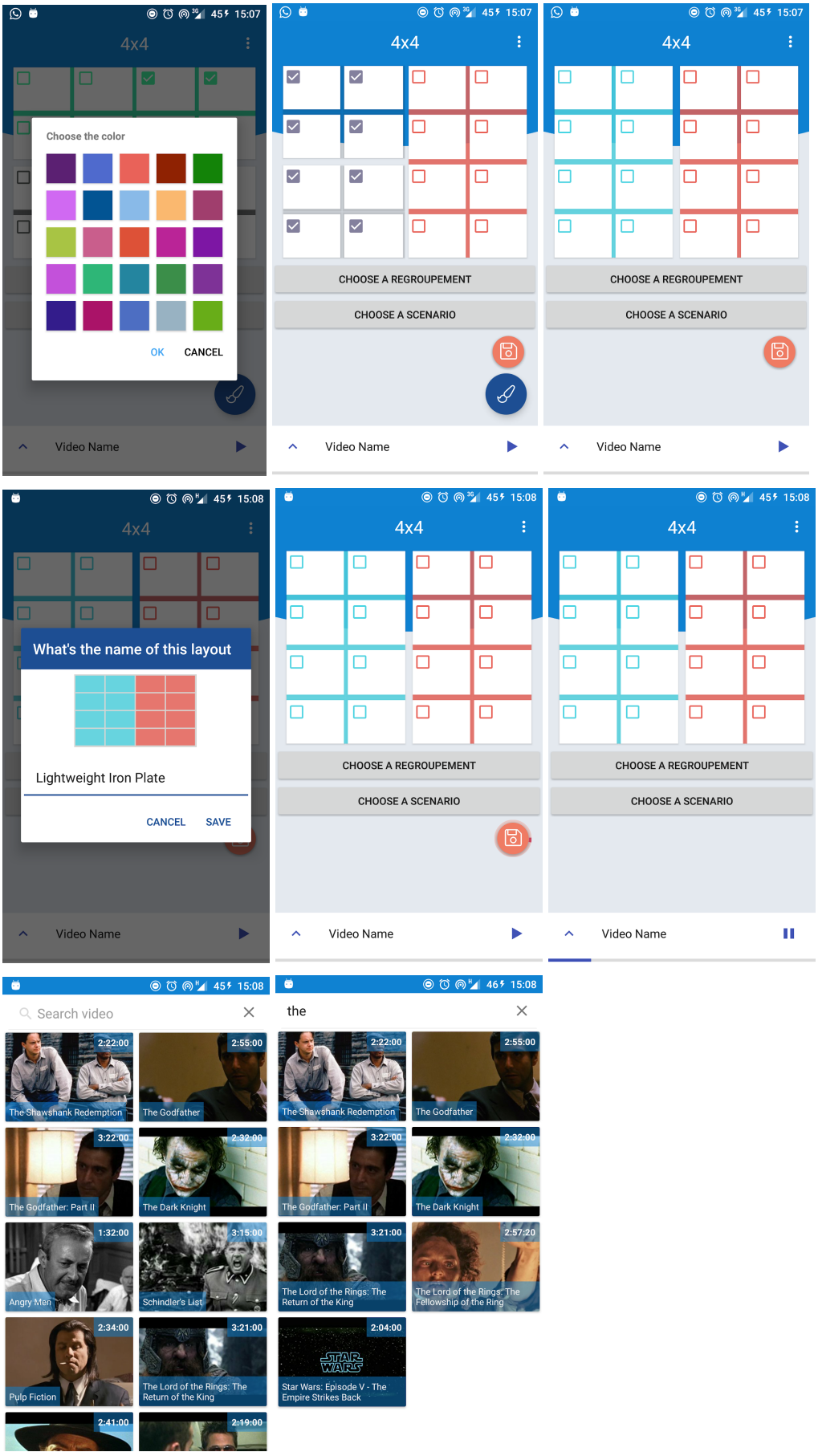

4x4
CHOOSE A REGROUPEMENT
CHOOSE A SCENARIO
Video Name


Choose a Wall
Edit URL server
About
CHOOSE A REGROUPEMENT
CHOOSE A SCENARIO
Video Name


4x4
Layout
name1
Name layout 4
Name layout 2
Video Name


4x4
Scenario
name1
test 1
test 2
Name layout 4
test 1
test 2
test 3
test 4
Name layout 2
test 1
test 2
test 3
test 4
Video Name


4x4
CHOOSE A REGROUPEMENT
CHOOSE A SCENARIO
Video Name


4x4
Choose the color
OK
CANCEL
CHOOSE A REGROUPEMENT
CHOOSE A SCENARIO
Video Name
What's the name of this layout
Lightweight Iron Plate
CANCEL
SAVE
Search video
the
The Shawshank Redemption
The Godfather
The Godfather: Part II
The Dark Knight
Angry Men
Schindler's List
The Lord of the Rings: The Return of the King
The Lord of the Rings: The Fellowship of the Ring
Pulp Fiction
Star Wars: Episode V - The Empire Strikes Back