Work Based Project
Ada College

Language Settings Migration
Replace an existing custom implementation of an Android component with a single, simple and easy to use generic one.

—

By Veronica Sonzini

INTRODUCTION

Aims and goals of the project

Simplicity is better than complexity.

Simpler things are easier to understand, easier to build, easier to debug, and easier to maintain. Easier to understand is the most important, because it leads to the others.

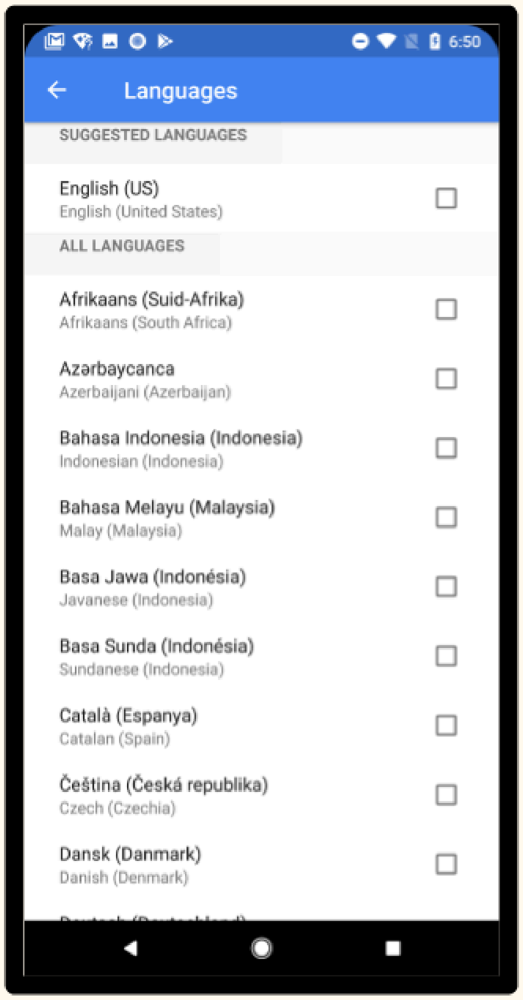
Complexity is multiplicative. In a system like Google, that is assembled from components, every time you make one part more complex, some added complexity is reflected in the other components.

Even though simplicity takes work, and it is hard to design, its benefits are exponential.

Plus, simpler systems run faster.

Continuing on these lines, I can say that the main objective of my project is to simplify the code base of another team, by replacing an Android component that they are currently using on their feature, with a simpler version:

- Create a list of languages in Android's Settings app, replacing their custom - single use - RecyclerView adapter by our generic and reusable RecyclerView adapter.
- These text items will display the language name and a checkbox for language selection, divided into two different sections:
  - Suggested Languages
  - All Languages



SUGGESTED LANGUAGES

English (US)
English (United States)

ALL LANGUAGES

Afrikaans (Suid-Afrika)
Afrikaans (South Africa)

Azərbaycanca
Azerbaijani (Azerbaijan)

Bahasa Indonesia (Indonesia)
Indonesian (Indonesia)

Bahasa Melayu (Malaysia)
Malay (Malaysia)

Basa Jawa (Indonésia)
Javanese (Indonesia)

Basa Sunda (Indonésia)
Sundanese (Indonesia)

Català (Espanya)
Catalan (Spain)

Čeština (Česká republika)
Czech (Czechia)

Dansk (Danmark)
Danish (Denmark)

Beneficiaries of the work done

This project will directly benefit two different teams:

- The team I'm a part of, by using this as a proof of concept, where the infrastructure built by my team was applied to another team's feature. This can be thought as internal "advertising" for feature developers to see how our UI framework works.
- The team where the feature is going to be implemented, considering that their code base is being simplified and reduced: less code and less to maintain.

Scope of the project

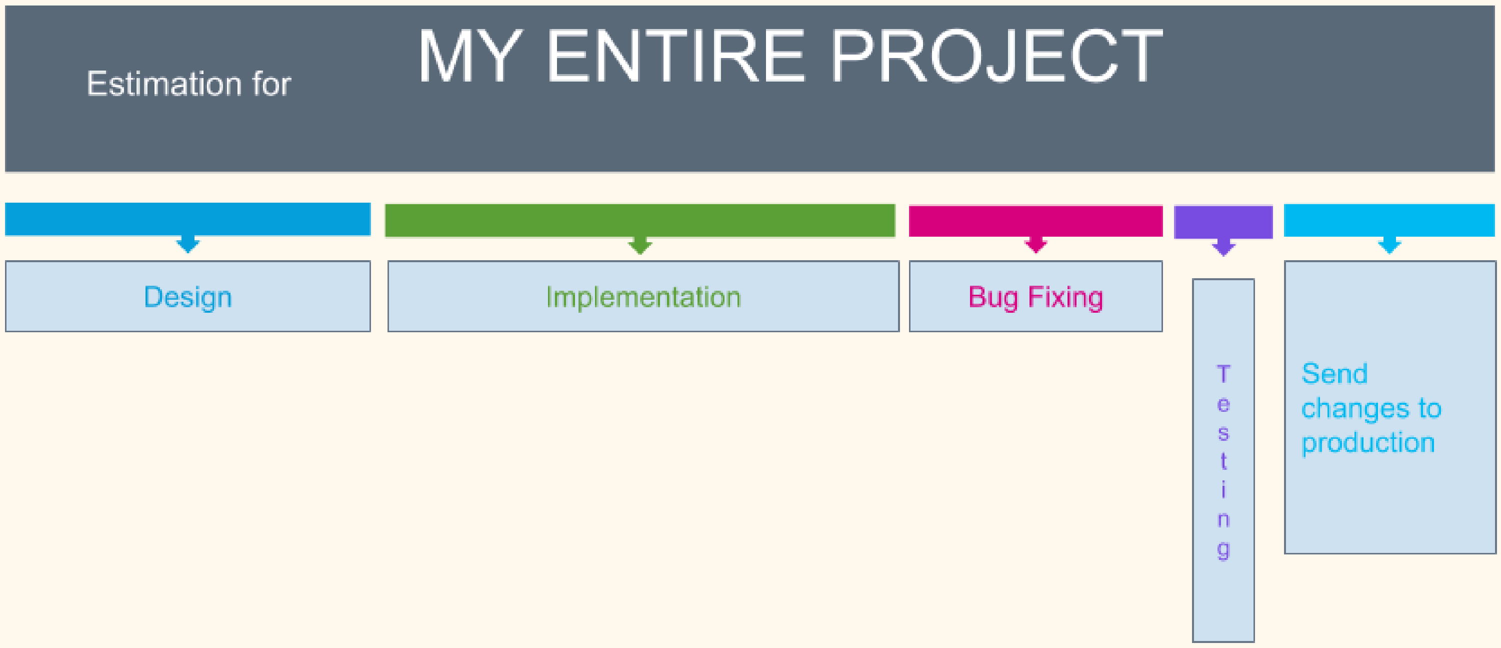
This project involves adapting the existing code from the Languages setting in Settings app to work with the new design my team developed.

The intention of the change in the architecture design of the Language settings is to simplify the written code, making it simpler to modify and to extend.

This will involve new design documentation for the architecture of the Language settings feature that will be changed, as well as various meetings with the team to discuss the best approach to achieve the objective.

There is no hard deadline for this task other than the submission date of the Work Based Project report, however I set a time limit myself for organizing purposes. I committed myself to take no longer than 10 weeks to complete this code migration, breaking down the assignment in the following sub-tasks:

- Design: one week including team planning, and various meetings to get to the final design.
- Implementation: the code replacement itself. This will combine the adaptation of the existing code with new files and changes necessary to make a successful migration. I have assigned to this section two weeks to complete.
- Problem solving: many complications are expected to raise with this code remodeling, therefore I have allocated three weeks to work in this.
- Testing: considering the extent of the changes involved, I assume testing will be a major part in the project, thus I have assigned to this task two weeks.
- Team / peer reviewing: I will need to allow at least 2 weeks for this part, considering that this is a mid- size project and I am expecting some comments from my reviewers on my changes.
- Sending new changes to production: this part will be done after the whole project is concluded.



Approach used in carrying out the project

I'm going to take an adaptive staging approach for this project, rather than a total waterfall approach or agile, dividing the process (in an informal way) into phases, stages and decision points. The process begins with a framework (design) that can be modified, if necessary, by new information. Decision points mark the transition between stages of the project implementation. At these points, an evaluation of the results obtained and information acquired is made and then the decision of which is the optimal path to proceed. Flexibility, which allows adaptation of the approach toward agreed overarching goals, will be maintained throughout.

This approach emphasizes continuous learning throughout project development: integration of new knowledge is anticipated. Similarly, decision points will allow me to adapt and incorporate new information throughout the process. Stages are defined to pursue continuous improvements until success is reached.

Assumptions on which the work is based

There are several assumptions about this project, but the main ones are:

- Feature developers will be interested in using this feature to simplify their code structure for a specific part of their own projects.

BACKGROUND

Software framework

[1]A software framework is a platform for developing software that provides a foundation on which developers can build features for a specific platform (in the case of this project, Android). The purpose of a framework is to improve the efficiency of creating new software. They can improve developer productivity and improve the quality, reliability and robustness of new software.

My team has created a Framework for UI features in the Android Google Search App. By using our UI framework, feature developers can benefit from many performance improvements when using it, like:

- Developers can focus on the unique requirements of their application instead of spending time on application infrastructure (plumbing).

This UI framework can be used to create UI features, following a specific Design Pattern that makes these elements lightweight (adaptive and flexible).

Android Views

[2]The UI consists of a hierarchy of objects called *views* - every element on the screen is a *view* - The View class represents the basic building block for all UI components, and the base class for interactive UI components including buttons, check boxes, and text entry fields.

A view has location, expressed as a pair of left and top coordinates, and two dimensions, expressed as width and height.

The Android system provides hundreds of predefined views, including those that display:

- Text (TextView)
- Fields for entering and editing text (EditText)

[1] “Software framework” - Techopedia : https://www.techopedia.com/definition/14384/software-framework

[2]“ Android Views” - Android Developers : https://developer.android.com/guide/topics/ui/custom-components

- Buttons users can tap (Buttons)
- Scrollable texts (ScrollView) and scrollable items (RecyclerView)
- Images (ImageView)

I'm going to be focusing i[3]n a type of view in particular, the RecyclerView.

ViewGroups and Layouts

The ViewGroup is a subclass of ***View*** and provides an invisible container that holds other Views or other ViewGroups and defines their layout properties.

There are different ***layouts*** which are subclasses of ViewGroup class and typical layouts define the visual structure for an Android user interface. You can declare your layout using a simple XML file, which would be located in the ***re/layout*** folder of the project.

RecyclerView

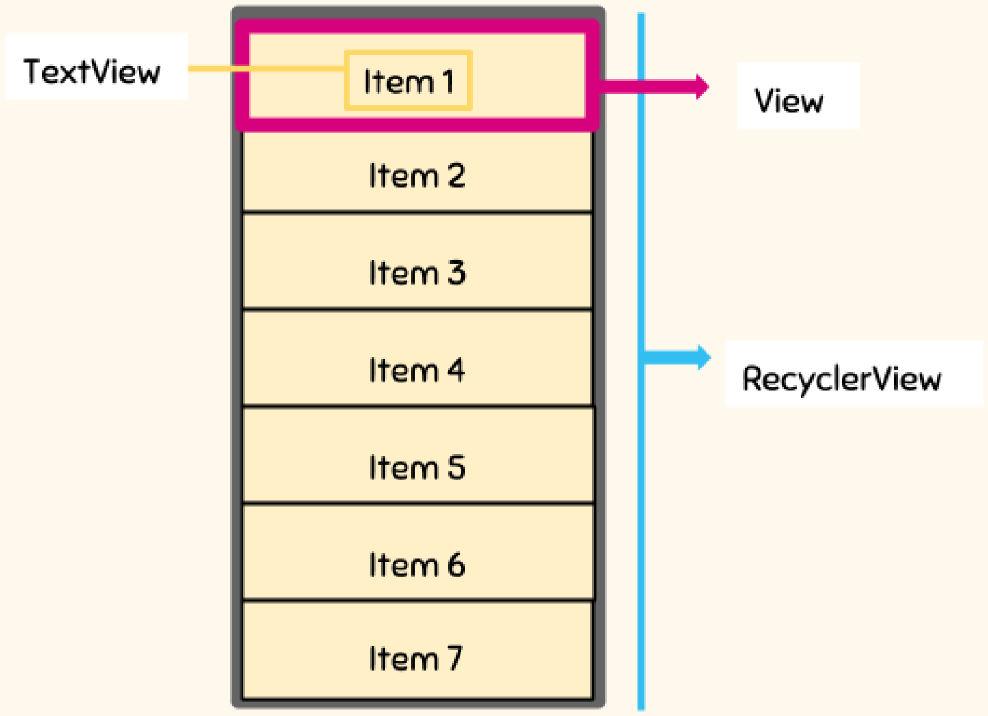
[4]The RecyclerView is a new ViewGroup prepared to render any adapter-based view in a similar way.

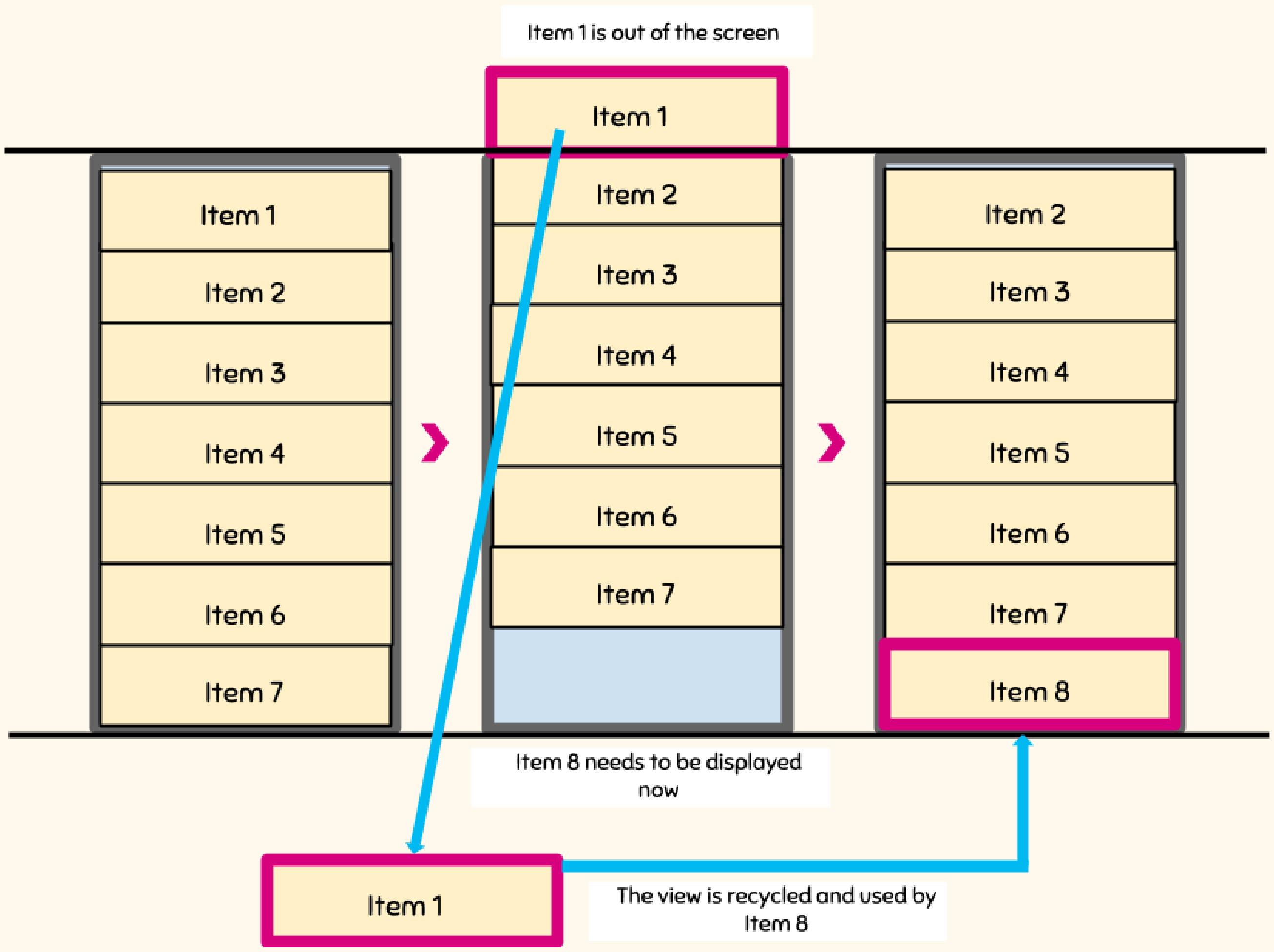
In the mobile development world, regardless of the platform, lists that display data to the user are commonly used. The Android platform gives us two different types of views that can be leveraged to display lists of data - the ***ListView*** and the ***RecyclerView***.

[3]"Software framework" - Techopedia : https://www.techopedia.com/definition/14384/software-framework

[4]" RecyclerViews" - Ray Wenderlich:
https://www.raywenderlich.com/170075/android-recyclerview-tutorial-kotlin

RecyclerView is a powerful and flexible tool available to Android developers.





Components of a RecyclerView

- [5]**LayoutManagers**: it positions item views inside a ***RecyclerView*** and determines when to reuse item views that are no longer visible to the user.
- **RecyclerView. Adapter**: it is the piece that will connect the data to the RecyclerView and determine the ***ViewHolder*** (s) that would need to be used to display that data.
- **Item animator**: it will animate the ViewGroup modifications like add/delete/select that are notified to the adapter.

[5] Componenets of RecyclerView - Ray Wenderlich : https://www.raywenderlich.com/170075/android-recyclerview-tutorial-kotlin

Using the RecyclerView

1. Add RecyclerView support library to the build file[6].

```
dependencies {
    def VERSION_ANDROID_SUPPORT = '24.2.1'
    ...
    compile "com.android.support:appcompat-v7:$VERSION_ANDROID_SUPPORT"
    compile "com.android.support:recyclerview-v7:$VERSION_ANDROID_SUPPORT"
    ...
}
```

   To begin setting the project we have to make sure that the RecyclerView dependency is included in our project libraries.

2. Define a model class to use as the data source.

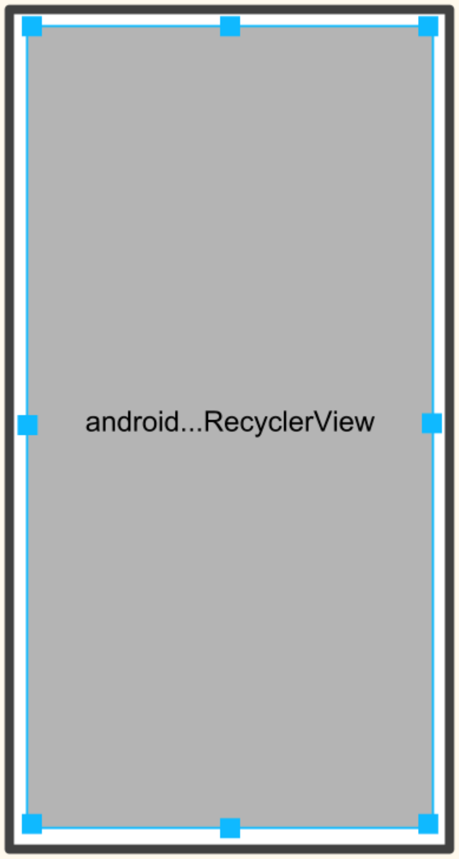
   Wi will need to create a common object that will hold our data.

```
public class SimpleViewModel {
    private String simpleText;

    public SimpleViewModel(@NonNull final String simpleText) {
        setSimpleText(simpleText);
    }

    @NonNull
    public String getSimpleText() {
        return simpleText;
    }

    public void setSimpleText(@NonNull final String simpleText) {
        this.simpleText = simpleText;
    }
}
```

[6] Using the RecyclerView - Ray Wenderlich : https://www.raywenderlich.com/170075/android-recyclerview-tutorial-kotlin

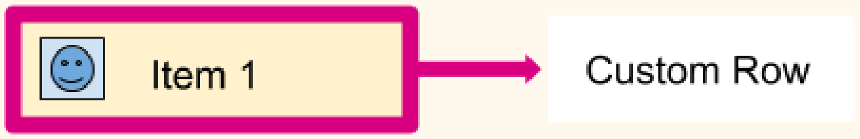
3. Add RecyclerView to your activity to display items.

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">


    <android.support.v7.widget.RecyclerView
        android:id="@+id/simple_recyclerview"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</FrameLayout>
```



4. Create a custom Row layout XML file to visualize the item.

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_margin="8dp">

    <TextView
        android:id="@+id/simple_text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        tools:text="This is some temp text" />
</FrameLayout>
```



5. Create a RecyclerView adapter.

   Next, we will need an adapter. Create a class that extends ***RecyclerView. Adapter***.

```
public class SimpleAdapter extends RecyclerView.Adapter {
    …
}
```

6. Create a ViewHolder

   The ViewHolder is more than just a dumb object that holds the item’s views. It is the object that represents each item in our collection and will be used to display it.

```
public class SimpleViewHolder extends RecyclerView.ViewHolder {
    private TextView simpleTextView;

    public SimpleViewHolder(final View itemView) {
        super(itemView);
        simpleTextView = (TextView) itemView.findViewById(R.id.simple_text);
    }
}
```

7. Bind the adapter to the data source to populate the RecyclerView

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        SimpleAdapter adapter = new SimpleAdapter(generateSimpleList());
        RecyclerView recyclerView = (RecyclerView)findViewById(R.id.simple_recyclerview);
        recyclerView.setHasFixedSize(true);
        recyclerView.setLayoutManager(new LinearLayoutManager(this));
        recyclerView.setAdapter(adapter);
    }

    private List<SimpleViewModel> generateSimpleList() {
        List<SimpleViewModel> simpleViewModelList = new ArrayList<>();

        for (int i = 0; i < 100; i++) {
            simpleViewModelList.add(new SimpleViewModel(String.format(Locale.US, "This is item %d", i)));
        }

        return simpleViewModelList;
    }
}
```

The problem identified

The Adapter

[7]The adapter's role is to convert an object at a position into a list Row item to be inserted. Right now, feature developers need to create a customized instance of this adapter. To achieve this there are a basic set of methods that feature developers need to add to the adapter for it to work:

- **onCreateViewHolder(ViewGroup, int)**

  This method is called right when the adapter is created and is used to initialize the ViewHolder(s).

- **onBindViewHolder(RecyclerView. ViewHolder, int)**

  This method is called for each ViewHolder to bind it to the adapter. This is where the data is passed to the ViewHolder.

- **getItemCount()**

  This method returns the size of the collection that contains the item we want to display.

- **getItemViewType(int)**

  This method returns an integer which represents the view type. Since the Android system stores a static reference to each layout as an integer in the "***R***" (resources) class, it can simply return the layout resource id to be used in the ***onCreateViewHolder()*** method.

Creating a custom adapter to use on RecyclerView is a valid way to do the work, but if every feature developer would create a custom adapter every time they need to use a RecyclerView, then Google's code base would be filled with duplicate and boilerplate code.

[7] "RecyclerView Adapters" - the App Development: https://willowtreeapps.com/ideas/android-fundamentals-working-with-the-recyclerview-adapter-and-viewholder-pattern/

Demonstrate the deficiencies your project intends to address

The code at Google is huge and complex. New hires in the company struggle to grasp it and enormous resources are spent in training and mentoring them so they can cope. Complexity just happens and its costs are exponential.

However, simplicity takes work - but it's all upfront. Simplicity is hard to design, but it's easier to build and much easier to maintain.

Simpler code is more readable, easier to test, easier to explain and most important easier to fix when they fail.

Duplication comes in several forms, ranging from copy/paste used for a handful of lines, to branches of large systems, to boiler-plate repeated every time another piece of code is added.

At any time a feature developer must implement a RecyclerView, they would have to create an adapter, customized to their needs, and because these are "custom" adapters, then they can't be reused by other feature developers. This creates duplication, and Google code base is no stranger to this. If instead of having this number of repeated code, we could replace all this instances with a single, general one, can you imagine how many lines of code could be reduced?

The solution my team came with

To enable recycling in my team's UI Framework, we proposed a new simpler and generic recycling Adapter.

Using this new adapter would help **:

- Reduce initial load time of the app.
- Reduce memory usage.
- Recycle common view elements.

(The way this is achieved is ***Google Confidential,* for that reason I will reserve this from further explanation).

SPECIFICATION & DESIGN

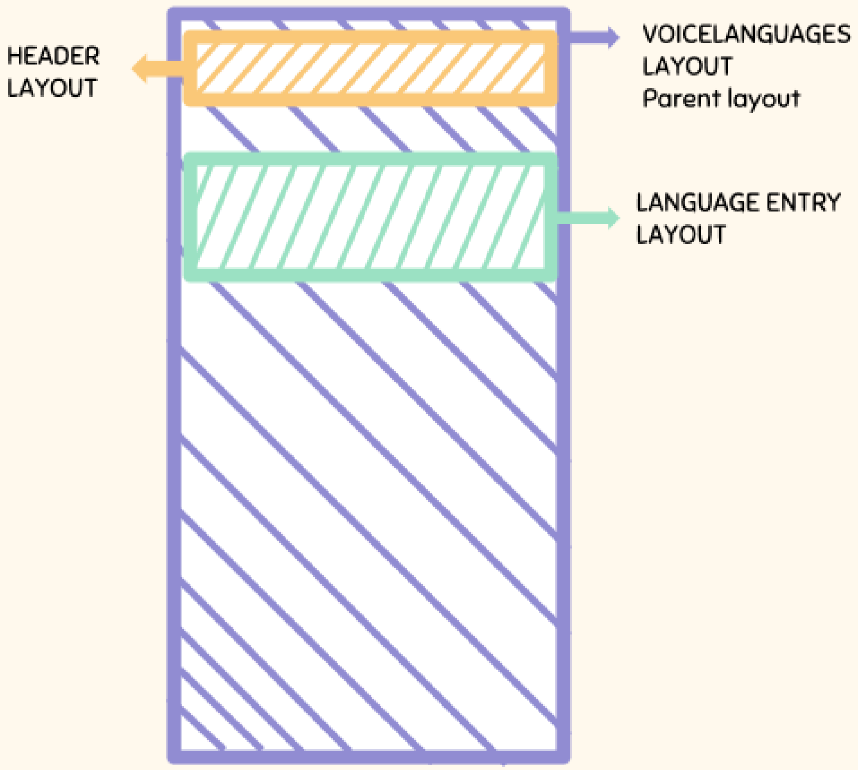
To accomplish the objectives previously stated, the design was build with extensibility and scalability in mind. Great care has been taken in designing something that can be updated easily.

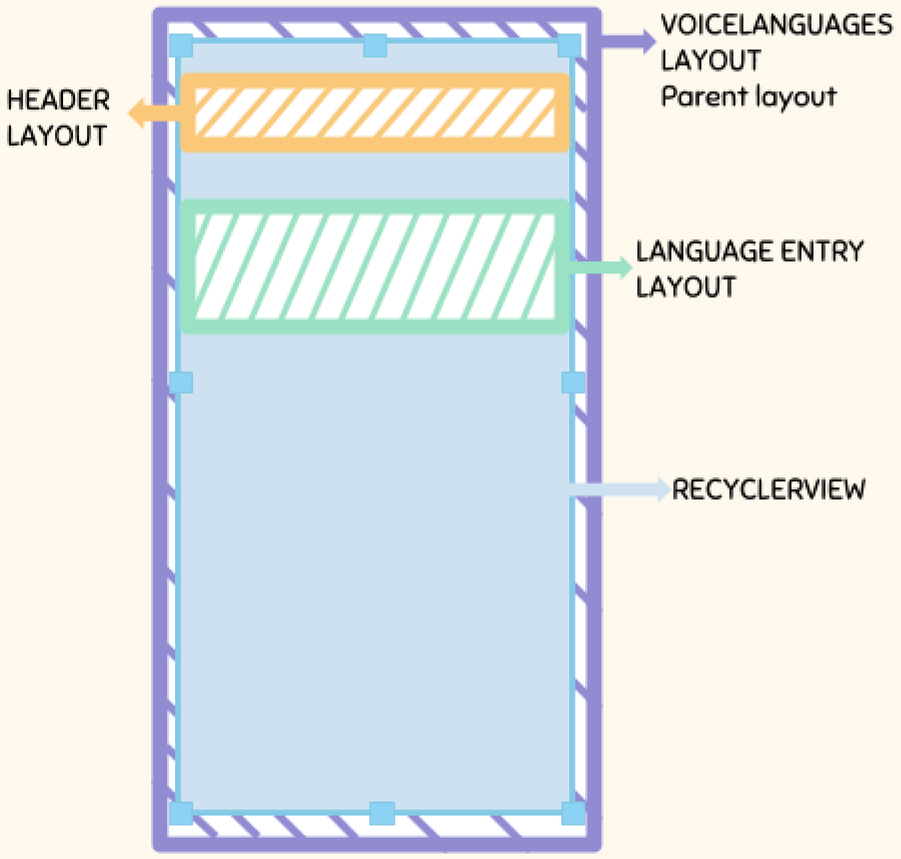
The user interface

Due to this project being a code migration, the UI was kept from the original version, making only minor changes to adapt the new structure to the existing UI.

The original UI consisted of three different layouts:

1. Section header layout
2. Language entry layout
3. Voice Languages layout -Whole screen layout-: contains the RecyclerView widget, and inside the RecyclerView there will be the Header layout and the LanguageEntry layout.





What data types are implemented in the system

The central data structure used in the project is the ***List***[8]. This is part of the original design and wasn't touched or improved in any way, since one of the main objectives -besides the replacement of the RecyclerView Adapter- was to minimize changing -breaking- the original functionality of the Voice Languages app.

The data coming from the service is stored in a list, later traversed with the purpose of identifying each component to attach it to its corresponding view.

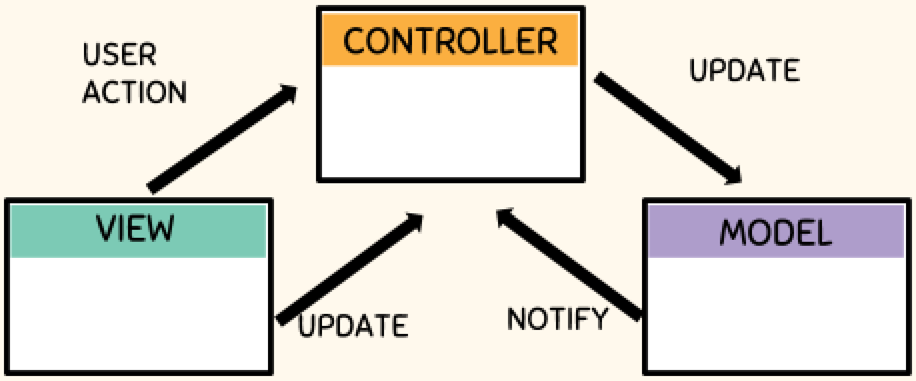
[8] Java List - java Point: https://www.javatpoint.com/java-list

How code is partitioned into modules

MVC Design Pattern

[9]From a high level, MVC is as straightforward as it's name. It's made up of three layers:

- The model: where data resides. Persistence, model objects, parsers and networking code lives here.
- The view: the face of the app. Its classes are typically reusable, since there aren't any domain specific logic in them. For example TextView is a view that presents text on the screen, and it's easily reusable.
- The controller: it mediates between the view and the model. In the ideal scenario, the controller entity won't know with what the concrete view it's dealing with.



To adapt the existing code from Language Settings, the current model will be split into different components, following a design pattern similar to the MVP. This will give a clearer view of what each component does.

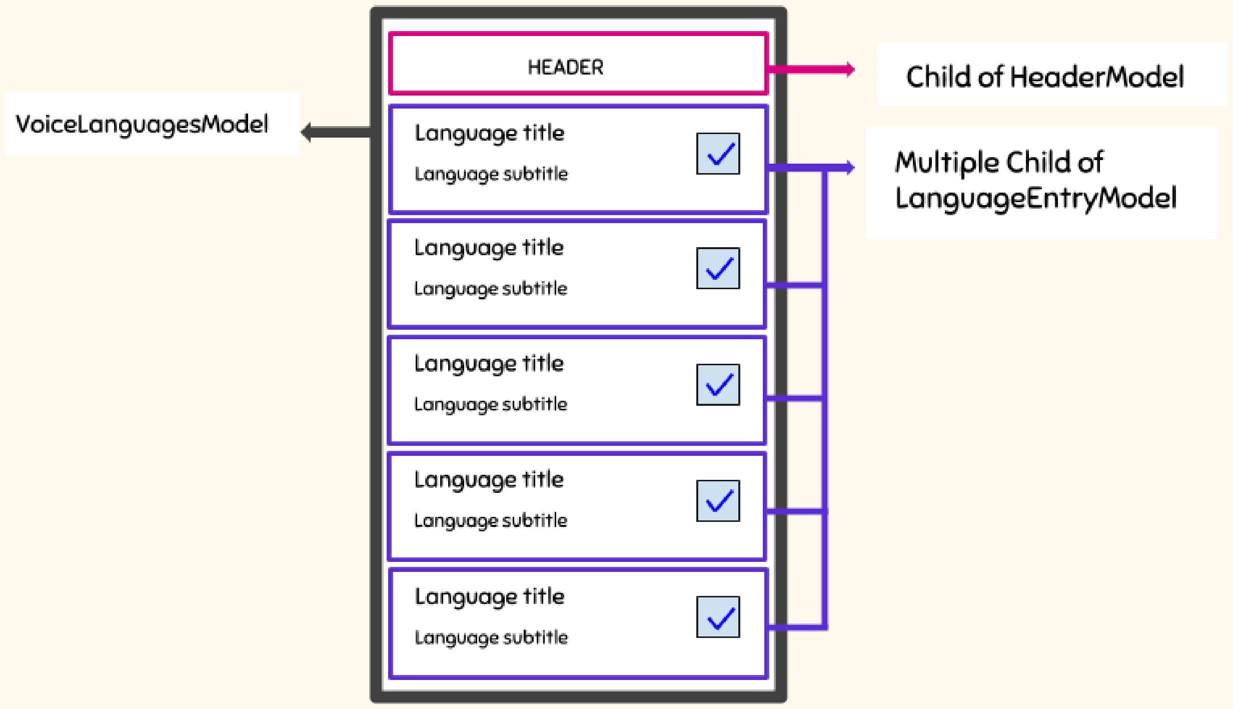
Splitting the model

There will be three different models:

1. **HeaderModel**: can specify the text and subtest of a language entry

[9] "MVP model" - Ray Wenderlich : https://www.raywenderlich.com/132662/mvc-in-ios-a-modern-approach

2. **LanguageEntryModel**: can have information related to the header
3. **VoiceLanguagesModel**: will have a child HeaderModel and multiple children from languageEntryModel.



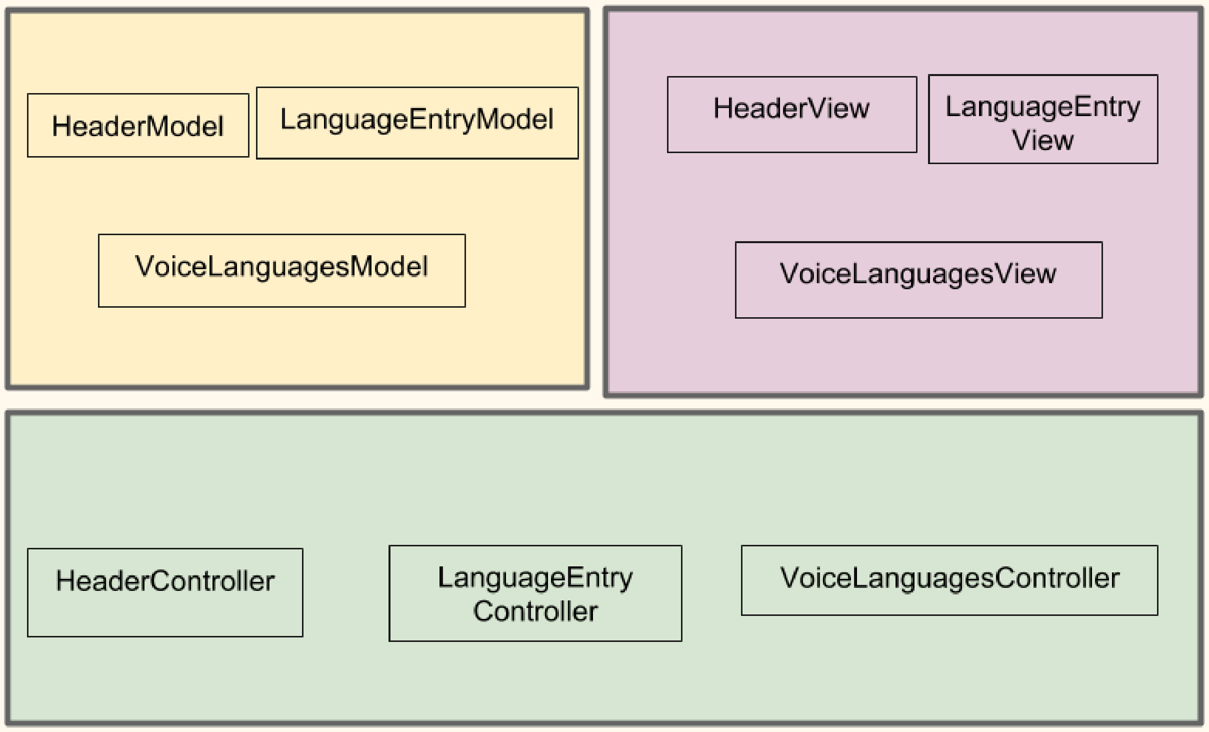
Adding Views and Controllers for all the models

The RecyclerView will contain:

- Children of Header (for as many headers as needed)
- Children of LanguageEntryModel (for as many language entries as needed)

Events:

- What happens when the ***back*** button is clicked.
- What happens when the ***checkbox*** is clicked.



How data flows through the system

The Languages List

The language list is formed of language entries. Each language entry is composed of languages and headers. Each of them is identified with a type. The type is a Boolean.

LanguageEntry

Enum Type {

SUGGESTED_LANGUAGES_HEADER = 0;

ALL_LANGUAGES_HEADER = 1

LANGUAGE = 2;

SUGGESTED_LANGUAGE = 3;

}

LanguageList {

Repeated LanguageEntry;

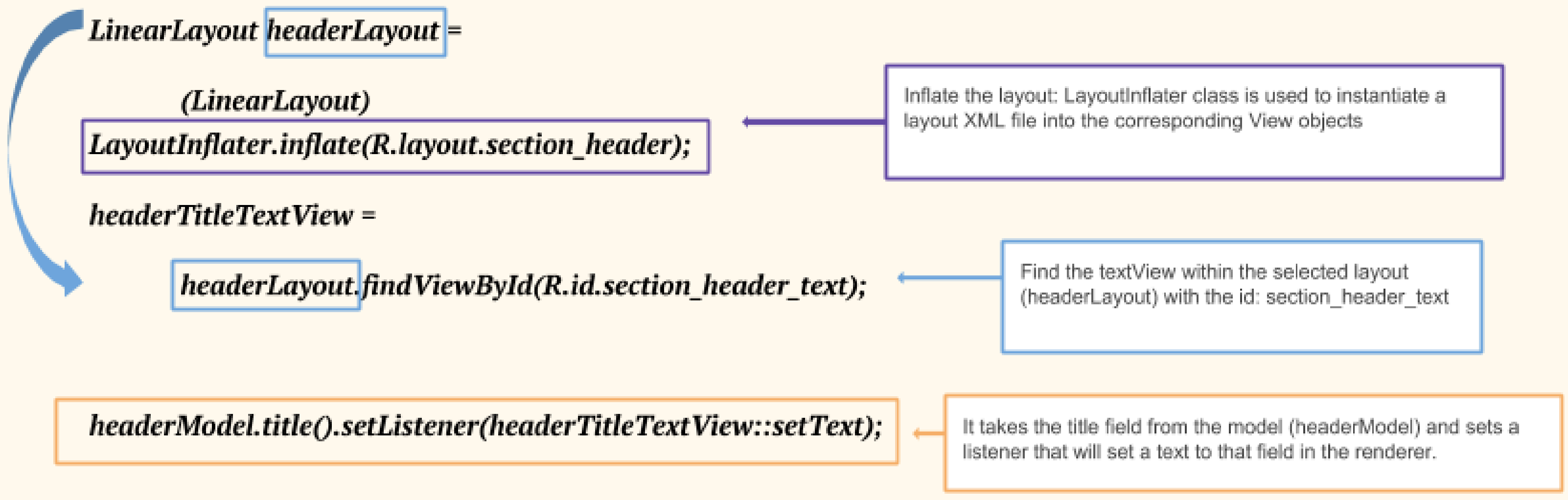
}

Views

The View will create everything that is visible on the screen:

- Header view container
- Header textView
- Language entry container view
- Language entry titleTextView
- LanguageEntry subtitleTextView
- Checkbox view

For this purpose, there must be a layout inflated, each UI element (TextView, Checkbox) needs to be identified within the corresponding layout -there is one layout per model created-, and a listener must be set for each UI element. This is demonstrated in the following example:



```
LinearLayout headerLayout =
      (LinearLayout)
LayoutInflater.inflate(R.layout.section_header);

headerTitleTextView =
      headerLayout.findViewById(R.id.section_header_text);

headerModel.title().setListener(headerTitleTextView::setText);
```

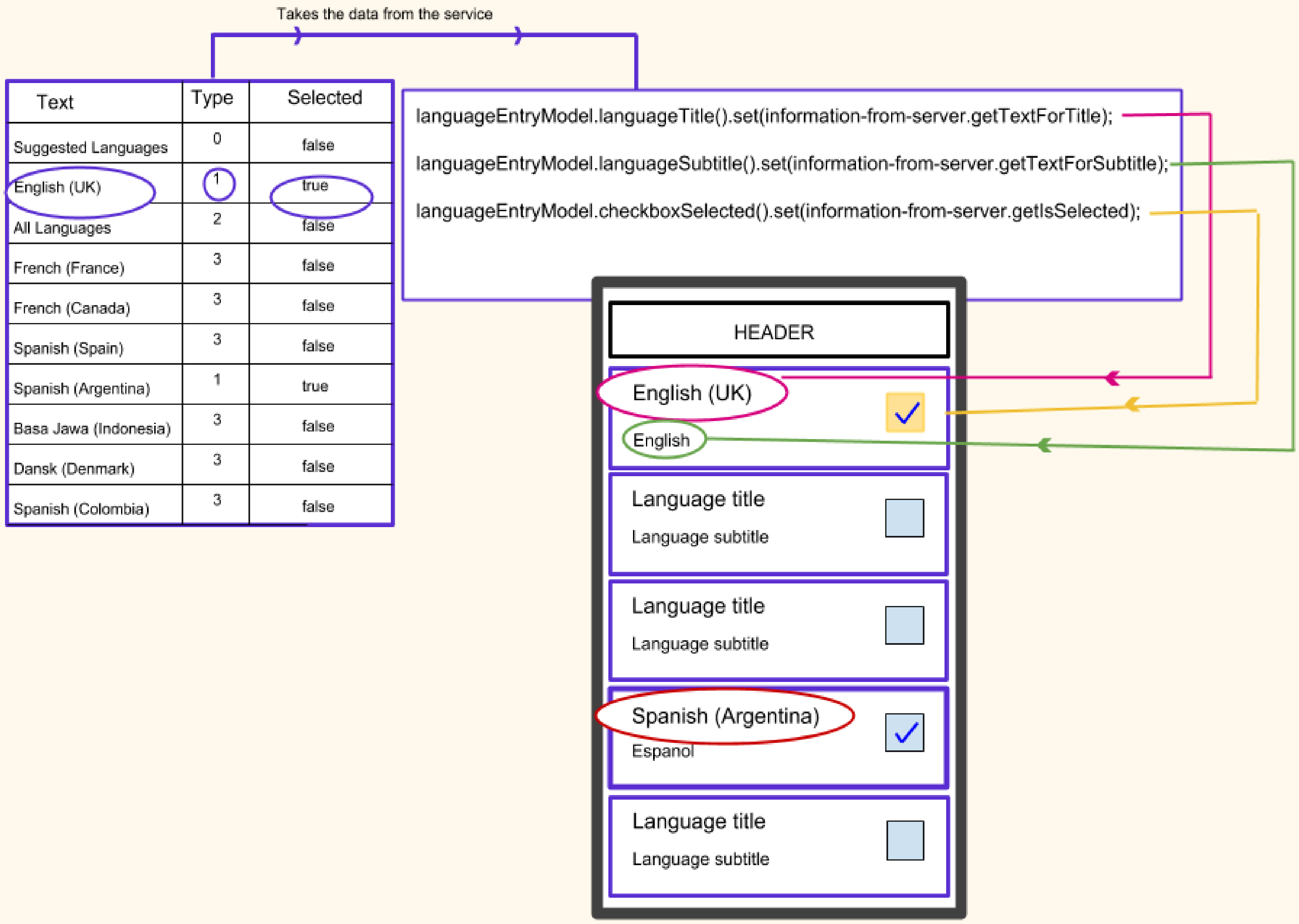
Inflate the layout: LayoutInflater class is used to instantiate a layout XML file into the corresponding View objects

Find the textView within the selected layout (headerLayout) with the id: section_header_text

It takes the title field from the model (headerModel) and sets a listener that will set a text to that field in the renderer.

Controllers and the dynamic behavior of the system

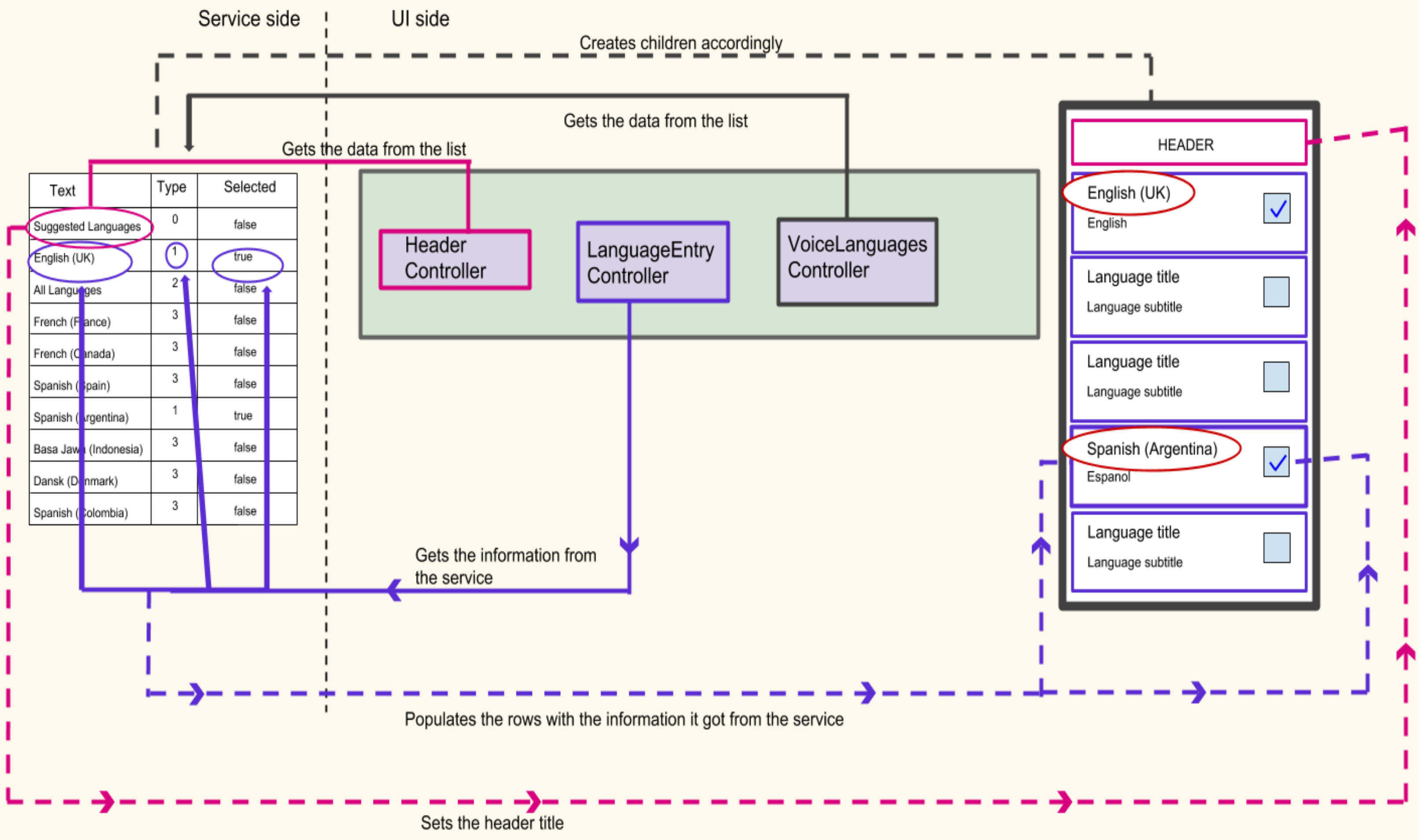
Each Controller will get the data from the Languages List and will update the model with that data.



Text	Type	Selected
Suggested Languages	0	false
English (UK)	1	true
All Languages	2	false
French (France)	3	false
French (Canada)	3	false
Spanish (Spain)	3	false
Spanish (Argentina)	1	true
Basa Jawa (Indonesia)	3	false
Dansk (Denmark)	3	false
Spanish (Colombia)	3	false

```
languageEntryModel.languageTitle().set(information-from-server.getTextForTitle);

languageEntryModel.languageSubtitle().set(information-from-server.getTextForSubtitle);

languageEntryModel.checkboxSelected().set(information-from-server.getIsSelected);
```

The header controller is in charge of looking through the languages list and find all the Strings that have

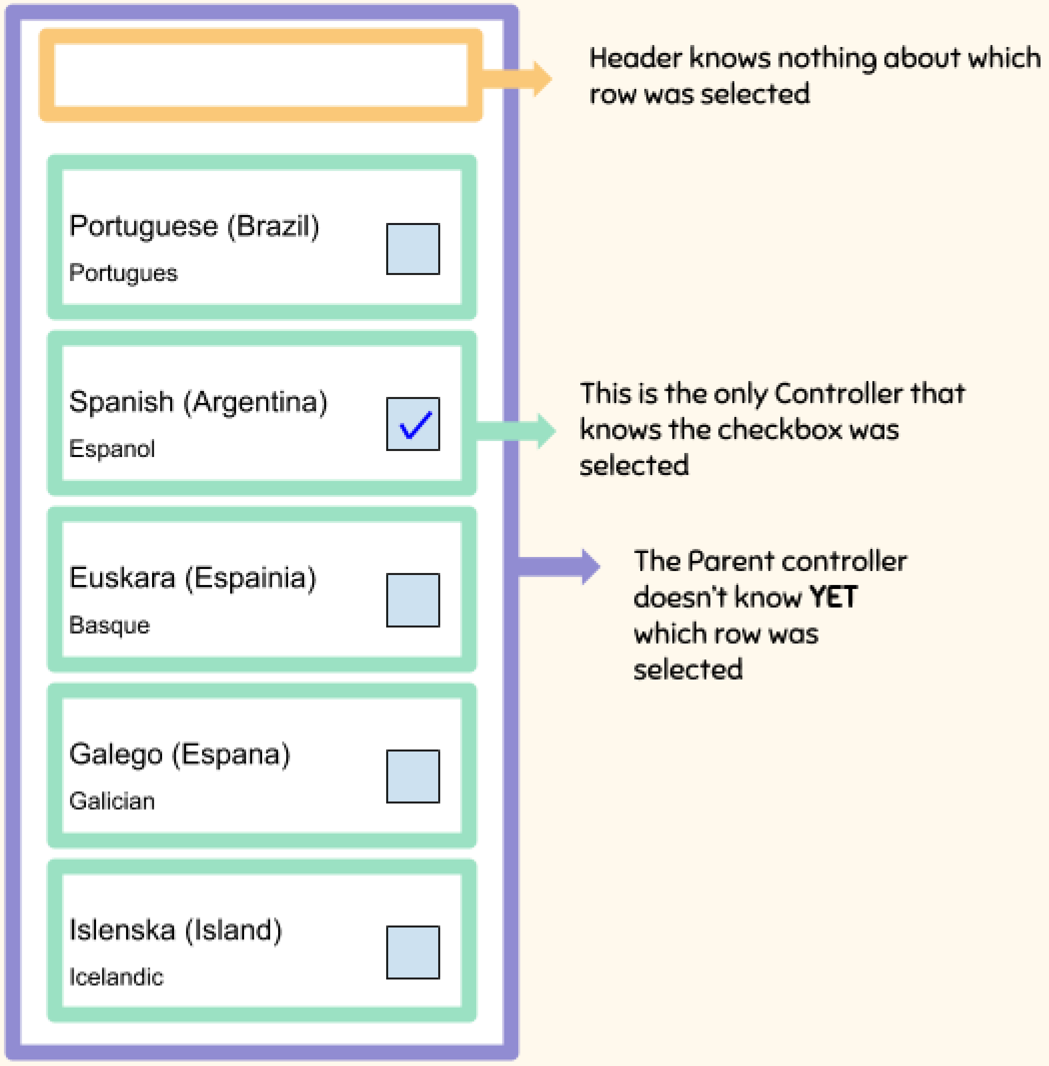


Text	Type	Selected
Suggested Languages	0	false
English (UK)	1	true
All Languages	2	false
French (France)	3	false
French (Canada)	3	false
Spanish (Spain)	3	false
Spanish (Argentina)	1	true
Basa Jawa (Indonesia)	3	false
Dansk (Denmark)	3	false
Spanish (Colombia)	3	false

Events

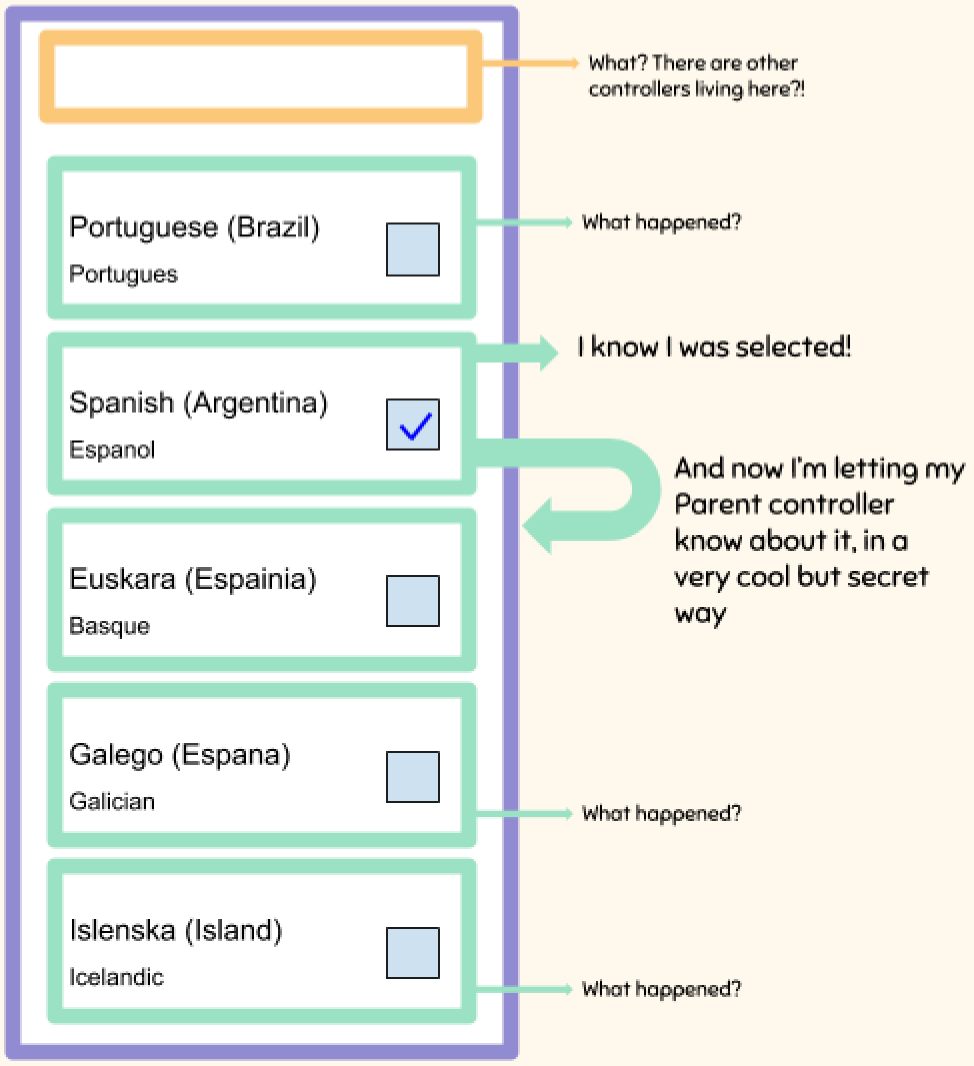
Events in Android can take various forms, but are usually generated in response to an external action. The most common form of events, particularly for devices like smartphones and tablets, involve some form of interaction with the touch screen. Such events fall into the category of input events.

Each object created is only aware of what happens within itself, but they can communicate to other objects this information. For example if the ***Spanish(Argentina)*** is selected from the languages list -checkbox is selected - then ***LanguageEntryView*** will receive this information first, and will share this with its controller: ***LanguageEntryController*** through an ***event.*** This will be the only controller that will be aware of the change of state of ***Spanish(Argentina).***


Header knows nothing about which row was selected
Portuguese (Brazil)
Portugues
Spanish (Argentina)
Espanol
This is the only Controller that knows the checkbox was selected
Euskara (Espainia)
Basque
The Parent controller doesn't know YET which row was selected
Galego (Espana)
Galician
Islenska (Island)
Icelandic

My team has created a way of communication between child controllers and the parent controller. Of course this is ***Google confidential*** stuff, so I won't be able to explain how this works, but trust me there is a way compatible with the whole UI framework design.

The child controller notifies the parent controller about this event taking place.



And now the Parent Controller can deal with this: it will update the ***SelectedList*** of languages accordingly and will have a new list ready for the model to update with it.

The events that would be implemented in the project are:

- onCkeckboxChecked() -> to implement from LanguageEntryController
- OnReturnButtonPressed() -> to implement from VoiceLanguagesController.

IMPLEMENTATION

The project was implemented on Android platform, following OOP[10].

Following the reasoning of the previous sections, I have separated the implementation into multiple tasks:

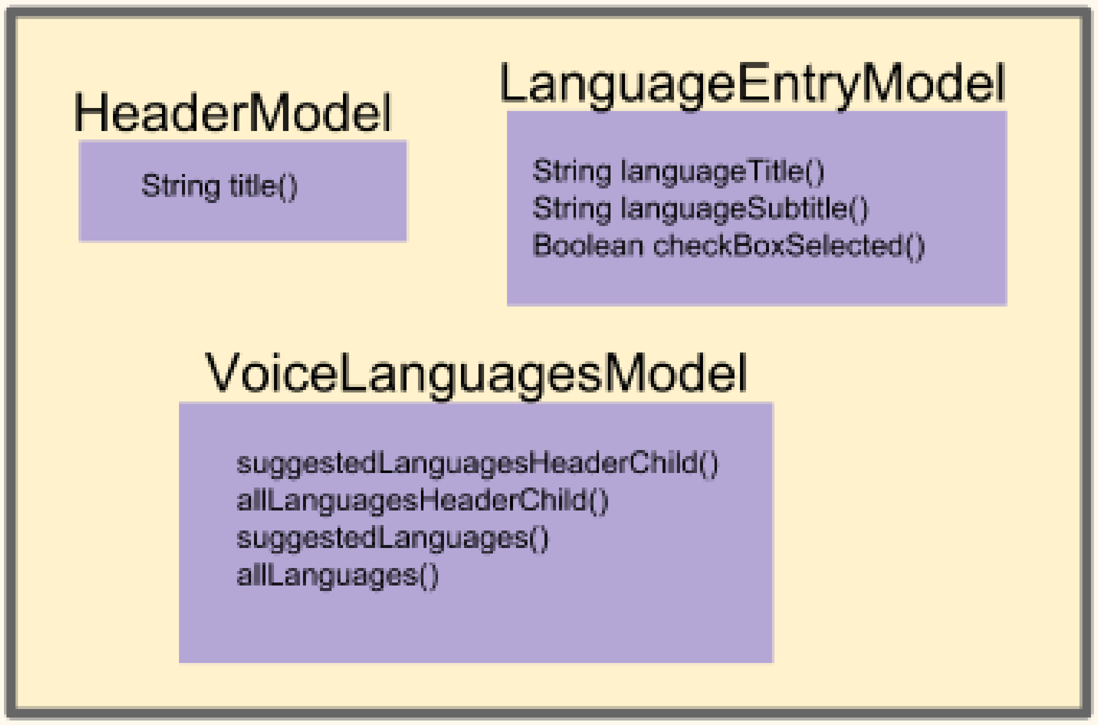
1. Creating all the Models.
2. Creating the Views.
3. Adapting the existing Controller to adopt the new design.
4. Splitting the current Events Interface into 2 different ones:
    a. Events handled by LanguageEntry -> checkbox clicked. This event should be handled from the child controller that knows about what is happening at its own level.
    b. Events handled by VoiceLanguages -> return button clicked. This event should be managed by the parent controller.
5. Replace the RecyclerView Adapter.

Creating Models

The feature models created are simple:

- HeaderModel: features one String for the title of the header.
- LanguageEntryModel: has two Strings (one for title and one for subtitle) and a Boolean representing the selection of the checkbox.
- VoiceLanguagesModel: contains the children of HeaderModel and LanguageEntryModel.

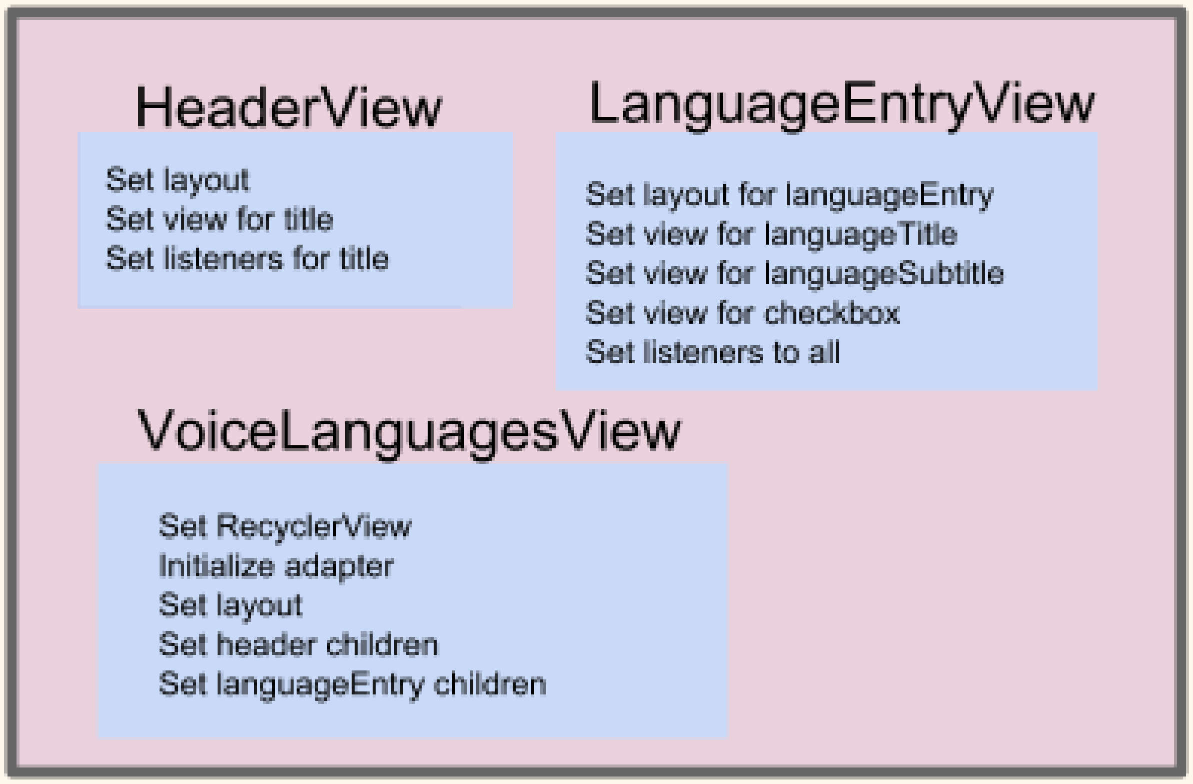
[10] "OOP": Object Oriented Programing
https://medium.com/@richardeng/a-simple-explanation-of-oop-46a156581214



Creating the Views

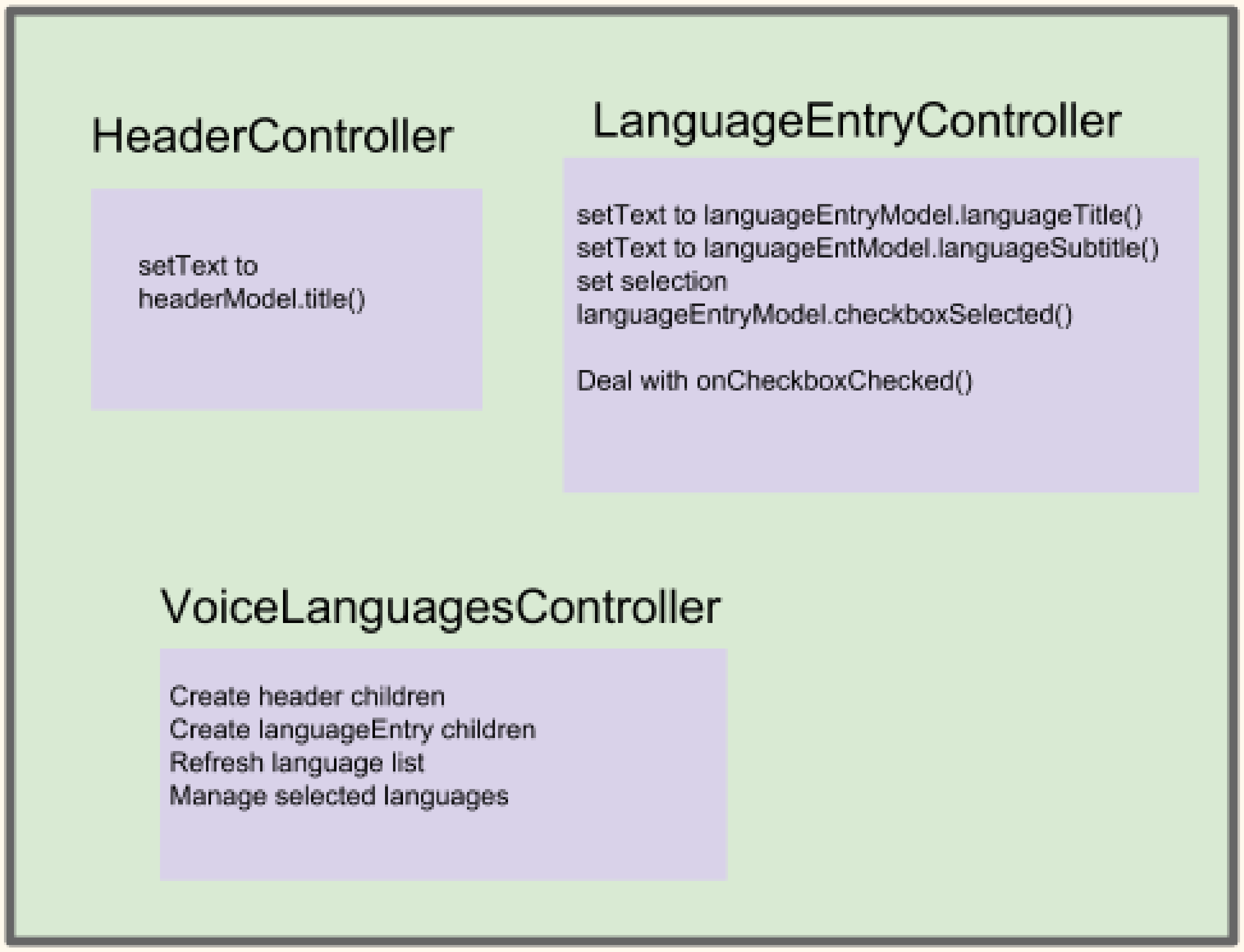
The Views will specify how the data from the model should be presented. If the model data changes, the view must update its presentation as needed.

- HeaderView: Will set the layout for the screen, and the TextView for the title of the header.
- LanguageEntryView: Will set the layout for the screen, the TextViews for the title and subtitle and the CheckboxView.
- VoiceLanguagesView: will deal with creating the layout, setting the RecyclerView, and instantiating the children from the models inside of the RecyclerView Adapter.



Creating the Controllers

The controllers will need to ‘translate’ the user’s interaction with the view into actions that the model will perform.


HeaderController
setText to headerModel.title()
LanguageEntryController
setText to languageEntryModel.languageTitle()
setText to languageEntModel.languageSubtitle()
set selection languageEntryModel.checkboxSelected()
Deal with onCheckboxChecked()
VoiceLanguagesController
Create header children
Create languageEntry children
Refresh language list
Manage selected languages

How it all works together

The View registers as a listener on the Model. Any changes to the underlying data of the Model immediately results in a broadcast change notification, which the View receives. The Model it's not aware of the view or the Controller - it simply broadcasts change notifications to all interested listeners.

The Controller is between the Model and the View. The View objects will use the Controller to translate user's actions into updates on the Controller objects.

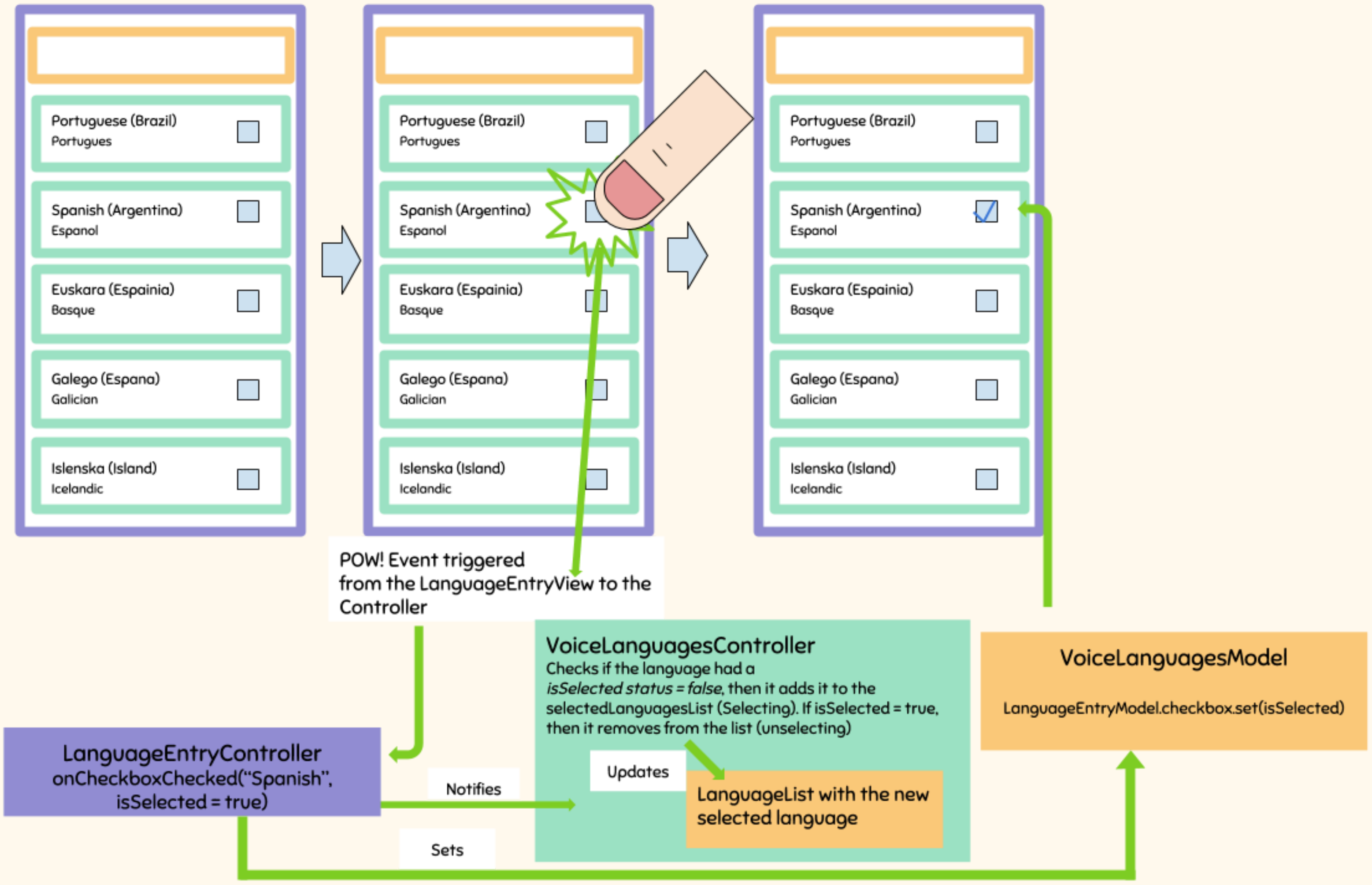
The Controller is given a reference to the underlying Model.

Once the user interacts with the View, the following action occurs:

The View recognizes that a UI action -for example, a language was selected or the screen was scrolled - has occurred, using a listener method registered to be called when such action occurs. The View calls the appropriate method on the Controller.

The Controller accesses the Model, updating it in a way appropriate to the user's action. If the model has been altered, notifies interested listeners, like the View, of the change.

As mentioned earlier the Model does not carry a reference to the View but instead uses an **event** notification to notify interested parties of the change - onCheckboxSelected() or onBackButtonPressed().


Portuguese (Brazil)
Portugues
Spanish (Argentina)
Espanol
Euskara (Espainia)
Basque
Galego (Espana)
Galician
Islenska (Island)
Icelandic
POW! Event triggered from the LanguageEntryView to the Controller
VoiceLanguagesController
Checks if the language had a *isSelected status = false*, then it adds it to the selectedLanguagesList (Selecting). If isSelected = true, then it removes from the list (unselecting)
Updates
LanguageList with the new selected language
VoiceLanguagesModel
LanguageEntryModel.checkbox.set(isSelected)
LanguageEntryController
onCheckboxChecked("Spanish", isSelected = true)
Notifies
Sets

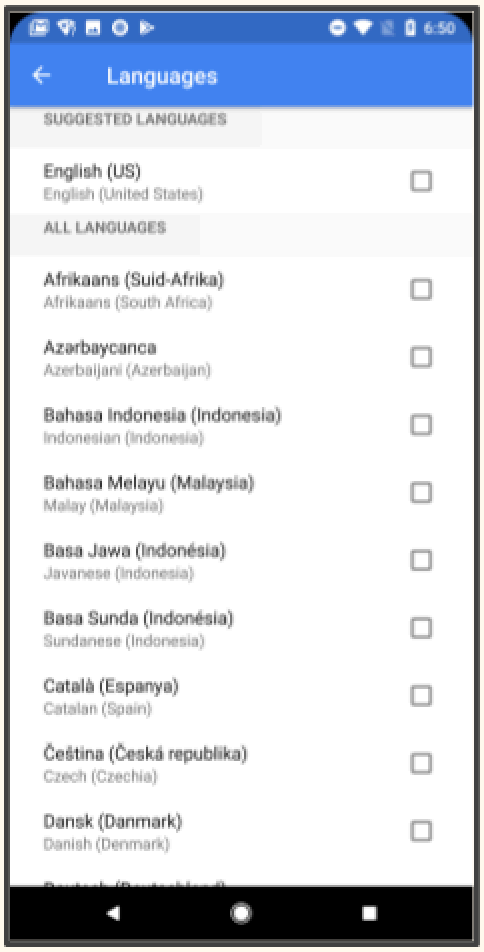
RESULT & EVALUATION

The good, the bad and the ugly

The result was positive: I could complete the migration without changing the original functionality of the application. The new Adapter was successfully added and the rows were recycled as expected.

The project took much more time that I had initially expected, leaving almost no time for testing. I initial estimation was wildly optimistic, full of unintentional bias. I should have balanced my optimism with realism. My approach to the estimation was taking into account the ***best case scenario*** rather than the ***worst case,*** and in the end it was a mixture of this two.

Before the migration



Languages

SUGGESTED LANGUAGES

English (US)
English (United States)

ALL LANGUAGES

Afrikaans (Suid-Afrika)
Afrikaans (South Africa)

Azərbaycanca
Azerbaijani (Azerbaijan)

Bahasa Indonesia (Indonesia)
Indonesian (Indonesia)

Bahasa Melayu (Malaysia)
Malay (Malaysia)

Basa Jawa (Indonésia)
Javanese (Indonesia)

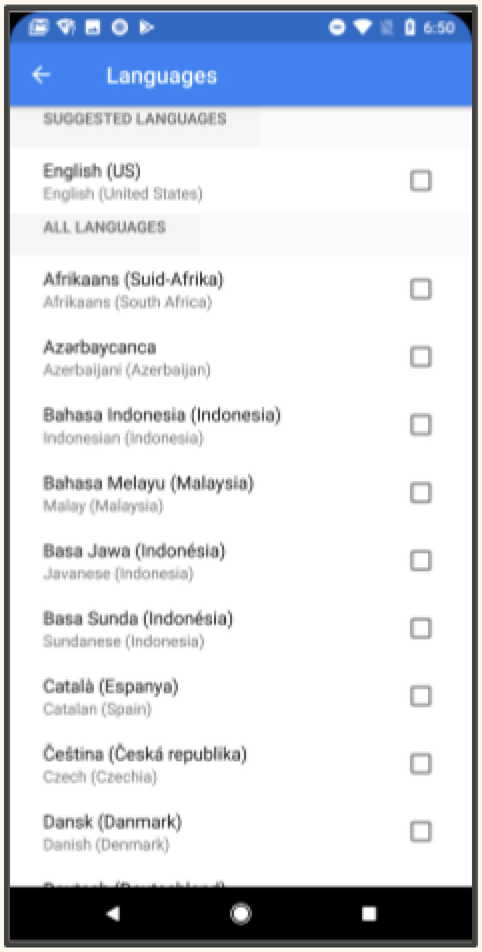
Basa Sunda (Indonésia)
Sundanese (Indonesia)

Català (Espanya)
Catalan (Spain)

Čeština (Česká republika)
Czech (Czechia)

Dansk (Danmark)
Danish (Denmark)

After the migration



Languages

SUGGESTED LANGUAGES

English (US)
English (United States)

ALL LANGUAGES

Afrikaans (Suid-Afrika)
Afrikaans (South Africa)

Azərbaycanca
Azerbaijani (Azerbaijan)

Bahasa Indonesia (Indonesia)
Indonesian (Indonesia)

Bahasa Melayu (Malaysia)
Malay (Malaysia)

Basa Jawa (Indonésia)
Javanese (Indonesia)

Basa Sunda (Indonésia)
Sundanese (Indonesia)

Català (Espanya)
Catalan (Spain)

Čeština (Česká republika)
Czech (Czechia)

Dansk (Danmark)
Danish (Denmark)

TOTAL SUCCESS!

FUTURE WORK

There are still some changes that need to be done in a future iteration of this migration.

Testing

More test cases should be added. Only VoiceLanguagesController - the Parent controller- was tested completely. The child controllers were not tested on their own.

CONCLUSIONS

This project presented a new option for feature developers when it comes to using a RecyclerView. It was proven how this new and generic Adapter for RecyclerViews would reduce the time and number of lines of code that the developer would need to invest when working on this task. It was shown how this new Adapter can be applied and used in multiple -various- projects. It was stated the use of this generic instance adapts to any type of project, minimizing the interference with the original code.

APPENDIX

The following information cannot be added to the report due to confidentiality.

- Design Documentation for VoiceLanguagesMigration
- Planning meetings
- Implementation of the RecyclerView migration: classes, interfaces, testing.