# Project 3

RentWell (A Housing App) - Due 3/10 @ 11:59 pm

For Easy Reading, Go to View -> Disable Print Layout

## RELEVANT RESOURCES

- Demo Video
- Starter Code
- Server Code
- Design Specs <- contains assets that are not already in the application

## BACKGROUND

With the ongoing pandemic finding housing as well as the requirements for housing get tricker to communicate. In order to combat this problem, you will be implementing an App that provides a seamless process of gathering and posting data via a remote database much like a modern day app to help users find appropriate housing.

## TECHNICAL DESCRIPTION

This project covers core technical aspects from previous projects as well as new concepts.

Some technical components you should already be familiar with implementing are: Recycler View, Jetpack Navigation,

Some new technical components include connecting to a web server with RESTful API calls and displaying images from the internet.

This project models the more modern applications that may be found on the app store where a database is stored on a remote server.

# NOTE

As android is an evolving language, at times you may get alerts from AndroidStudio saying certain portions are deprecated, however please don't try to update such areas since the project may not work as intended if you do.

# KEY CONCEPTS

- Kotlin and Xml
- Retrofit
- Using Content Receivers (getting local Images)
- Jetpack Navigation

# REQUIREMENTS

## Screen 1: List Of Posts

### Description

This is the "Home Page" and is the place that shows the list of posts made by the users to navigate through when trying to find housing.

### Each item in the list should display

| Data | Type *(Notes)* |
|---|---|
| Image | String |
| Type | String *(ie: Mansion, Apartment, Cabin, etc)* |
| Location | String *(ie: Davis, CA)* |
| Number of Beds | Int |
| Number of Bathrooms | Int |
| Covid Tested | String |
| Price | Int |

**Note:** The list of items only display a portion of all the attributes that are associated with a Post

### Interactions

Clicking a post in the list should send the user to **Screen 6: View Posts**, where they can view the full contents of the housing listing. At the top of the screen there should be an option to create a new post (via clickable 'POST' textview) which will send users to **Screen 2**, where they can begin filling out each field for their home listing. If the user is not currently logged in, then the user will NOT be navigated to **Screen 2**, and will instead be brought to **Screen 7** where they will be prompted to sign in again.

## Screens 2-5: Create Post

There will be 4 screens used as a form to post a housing listing. Each screen will collect the following information listed below into a Post object. On Screen 5 (the 4th screen in the form) an API call will be made to send the post and image to the server. It might be useful to use the below information to construct your Post object to send data between screens and make the API call.

### Display Post inputs fields

| Screen | Name | Type | Description |
|---|---|---|---|
| - | id | Int | Unique Identifier for the Listing |
| any | email | String | Email Address (ie: android@google.com) |
| 2 | location | String | Address (ie: Davis, CA) |
| 3 | type | String | Type of Rental Unit (ie: House, Apartments, Condo, Etc...) |
| 3 | bed | Int | Number of bedrooms in housing unit |
| 3 | bath | Int | Number of bathrooms in housing unit |
| 3 | price | Int | Monthly rent cost |
| 3 | moveIn | String | Move in date |
| 4 | desc | String | Additional information about the housing (ex. if water or wifi is included, etc.) |

| | | | |
|---|---|---|---|
| 4 | covidTested | String | Was everyone in the unit tested for covid |
| 5 | date | String | Date posted |
| 5 | image | String | Image url for Housing Picture |

Note: **Email** and **Date** are not mentioned in this xml file. These values should not be changeable by the user, and will instead be initialized. Emails can be fetched using Firebase and the Date can be grabbed using a Kotlin Date Object.

### Interactions

In screens 2, 3, and 4 there should be a 'CONTINUE' button that will incrementally send the user to the next screen in the post creation process from screen 2-5 (ie: 2->3->4->5). On Screen 5 there should be a 'DONE' button which would make the API call to submit data onto the server when clicked and also should navigate back to the home feed for the user to see the new listing created.

## Screen 6: View Post

This screen should *display* everything from **Screens 2-5: Create Post** with the exception of id and email. Refer to the Designs to see how to properly display the housing listing details. From this screen a user will additionally have the option to contact the listing owner via email. By clicking a designated button on the bottom of the screen called "✉ EMAIL" the user should be sent to an email app using intents and automatically fill in the "subject" and "recipient" fields of the email. Please feel free to customize the pre-set email subject, here is a possible subject: "RentWell Listing Inquiry regarding (LISTING ADDRESS HERE)."

### Interactions

Clicking on a button will send the user to an Email app, with the recipient field automatically filled in with the home listing poster's email address. The user can now contact the poster via email, and will automatically be brought back to the same View Post Screen in the Home Listing App after sending the email

## Screen 7: Sign In

This is the "Sign In Page" where the user will be brought to if the user is attempting to create a Post, but has not yet logged into their Google account. This screen will integrate with Firebase

to authenticate the user. After this step, the user should be immediately directed to Screen 2 to create a post.

### Interactions

A sign in button which will open the Google Sign In Client, where you can sign into your Google account

# ARCHITECTURE BREAKDOWN

## Given Core Pieces

The project starter code is on GitHub

## Project File Structure

- **Bold Black files -** you need to complete the codebase with logic
- **Bold Blue files** - you need to create and implement yourselves
- Any non-formatted files should be left alone

- ❏ app
  - ❏ java
    - ❏ Com.android.example.housingconnect
      - ❏ data
        - ❏ **HousingService**
        - ❏ **ImageUploadResponse**
        - ❏ **Message**
        - ❏ **Post**
      - ❏ screens
        - ❏ **FormDescriptionFragment**
        - ❏ **FormDetailsFragment**
        - ❏ **FormImageFragment**
        - ❏ **FormLocationFragment**
        - ❏ **HousingDisplayFragment**
        - ❏ **HousingFeedFragment**
        - ❏ **SignInFragment**
      - ❏ **HousingListAdapter.kt**

            - ❏ **MainActivity**
    - ❏ res
        - ❏ drawable
            - ❏ -> contains some of the vector assets you should need, feel free to add more to your liking. Any additional assets can be found in the drive with all the app designs and assets.
        - ❏ layout
            - ❏ **activity_main.xml**
            - ❏ **fragment_form_description.xml**
            - ❏ **fragment_form_details.xml**
            - ❏ **fragment_form_image.xml**
            - ❏ **fragment_form_location.xml**
            - ❏ **fragment_housing_display.xml**
            - ❏ **fragment_housing_feed.xml**
            - ❏ **fragment_sign_in.xml**
            - ❏ **housing_list_item.xml**
        - ❏ navigation
            - ❏ **nav_graph.xml**

Gradle Scripts

- Do not touch anything in here

# PHASES

## Phase 0: Understanding the Codebase

There is code already provided to you, make sure that you can understand and coordinate with the given code. This is done to make your jobs easier so that you can make apps with more functionality. The code base is sprinkled with all the phases. There are some key files to look through, but look through as many as you see fit.

In the previous project, essential project dependencies were already included in the starter code to help speed up the process of development. In order to prepare all students to be able to make applications in the future without any starter code, we will provide guidance for adding project dependencies but will leave it up to you to add them in.

## Phase 1: Setting up the Server/Using the Server

To set up the server we will use a combination of github and replit:

1. You can fork or clone the server repository into your personal github.
2. Login to repl.it with your github for importing the server easily.
3. Create a new Node.js project by clicking the plus icon under the header 'Create'
4. Switch tabs in the pop up to 'Import from Github' and select the github repository that holds the housing server. Finally hit the blue button 'import from github'. If you are prompted to pay for the service, make your repository public. You should **NOT** pay for anything, we can use the service for free for the purposes of this project.

You should now see a screen with a code editor. You will not need to edit any code here but you will need to do a bit of set up to run the server:

1. Select the shell tab on the right and type 'npm install' and hit Enter
2. Wait until the all the server dependencies are installed
3. Now the server is ready to run. Hit the 'Run' button at the top to turn the server on

- You will need to keep the webpage loaded in order to interact with the server, this means you may need to hit the run button whenever you need to test server interactions from your application
- If you get an error message after coming back to the server, run 'npm install' again

WARNING: **Do Not Upload Any Personal or Sensitive Information** Into The Server From Your Application. The Data is Not Private.

## Phase 2: Setup REST API Client

In this phase, you'll create client-side code for interacting with the API server you hosted.
Reference: https://square.github.io/retrofit/

### Phase 2.1: Add Dependencies

Add the following dependencies to the Gradle file for App Module

```
// Retrofit
implementation 'com.squareup.retrofit2:retrofit:2.5.0'

// GSON Converter for Retrofit
// (this library handles conversion between JSON string and Kotlin objects)
```

```
implementation 'com.squareup.retrofit2:converter-gson:2.5.0'
```

Add Glide dependency in the Gradle file for App Module

```
implementation 'com.github.bumptech.glide:glide:4.11.0'
annotationProcessor 'com.github.bumptech.glide:compiler:4.12.0'
```

## Phase 2.2: Add Data Models

Create the following data classes and add fields to each according to server-side code

- Post.kt *(Hint: check the SQL for creating the housing table, this is NOT an Entity)*
- Message.kt
    - Stores a variable of type String called message
- ImageUploadResponse.kt
    - Stores two variables of type String called message and type

## Phase 2.3: Define API Request Interface

- Implement the Kotlin interface called HousingService (HousingService.kt)
- By looking at the server-side code, create abstract functions each modeling the one of the following APIs (Each function in this interface defines the contract of each REST API call to the backend server)

| Request Type | Endpoint | Return Type | Description |
|---|---|---|---|
| GET | `/housing/all` | List<Post> | Get all the posts |
| POST | `/housing` | Message | Create a new post |
| POST | `/images/upload` | ImageUploadResponse | Upload an image |
| GET | `/images/{id}` | Gives an Image Binary | Get image to display |

NOTE: You do not need to implement the last endpoint. It will only be utilized with Glide to display images that are stored on the server

## Phase 2.4: Create Singleton for Accessing API Service

- In MainActivity.kt, create a public, late-initialized variable of type HousingService
- Initialize the service variable in onCreate using Retrofit Builder

- This single instance of the API service can be shared across all fragments, since all fragments belong to the MainActivity where the NavHostFragment is.

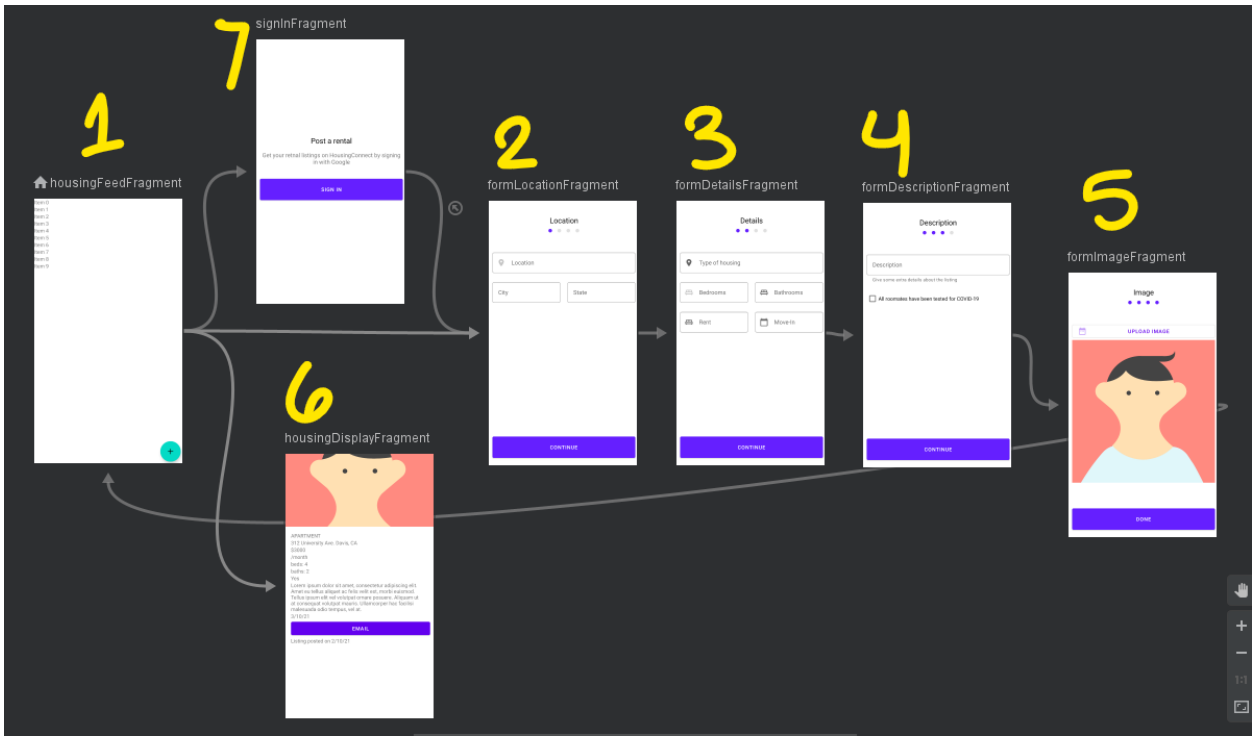
## Phase 3: Set up Jetpack Navigation for pages

In this phase we want to have all the connections and screens (fragments) laid out prior to diving into making POST and GET operations from our server. We can add our Buttons to switch screens respectively.

### Phase 3.0: Navigation Graph Setup

Here is an example of how the navigational graph should look. Note that the Layout design is not the end result, it is just to give you an idea of identifying respective files and how connections may look like.

**Each of the numbers 1-7 represent the respective screens that must be implemented**




- Create the navigation graph
- Add fragments (use provided fragments to add into the navigation graph)
- Add NavHostFragment to activity_main with navigation graph attached
- Add all actions (arrows)
- Add an argument to the following destinations:
    - **FormDetailsFragment** (Screen 3)

    - **FormDescriptionFragment** (Screen 4)
    - **FormImageFragment** (Screen 5)
    - **HousingDisplayFragment** (Screen 6)
- Can choose to pass a **Post**, **String**, or **Int** object to these respective screens/fragments based on the data that needs to be passed in

## Phase 3.1: Adding a RecyclerView to HousingFeedFragment

In order to successfully create a list for the Posts, a RecyclerView must be implemented:

- **HousingFeedFragment** & **fragment_housing_feed.xml**
    - Establish a RecyclerView similar to Project 2, inside of a fragment
    - The user will navigate to **FormLocationFragment** when clicking the 'POST' textview only if they are logged in, else they will navigate to **SignInFragment**

- **HousingListAdapter** & **housing_list_item.xml**
    - Define views in ViewHolder
    - Update views in the layout with the Post data from the server
    - If you prefer to make your own adapter from scratch, feel free to do so

- **Add Glide dependency if you have not already in the Gradle file for App Module**

```
implementation 'com.github.bumptech.glide:glide:4.11.0'
annotationProcessor 'com.github.bumptech.glide:compiler:4.12.0'
```

- **Use Glide in your recycler view adapter to display images for each post**

```
Glide.with(holder.itemView).load("https://.../" +
item.image).into(your_image_view)
```

## Phase 3.2: Creating a Post (Screens 2-5)

This is where the user will fill in the details when creating their home posts:

- **FormLocationFragment**
    - Location
        - Address (editText)
        - State (editText, make dropdown if time permits)
            - No need to list every state
        - City (editText)

Note: For location you should have three separate editTextViews: Address, City, and State which **should add up to a single string for location**

    - "CONTINUE" button to advance onto the next fragment
- **FormDetailsFragment**
    - Type (editText, make dropdown if time permits)
        - ie: house, apartment, condo, etc...
    - Number of Bedrooms (editText)
    - Number of Bathrooms (editText)
    - Price (editText)
    - Move In Date (editText)
    - "CONTINUE" button to advance onto the next fragment
- **FormDescriptionFragment**
    - Description (editText)
    - Covid Status (Check Box)
    - "CONTINUE" button to advance onto the next fragment
- **FormImageFragment**
    - "SELECT IMAGE" button for opening gallery and selecting an image
    - Housing Image to show which image was selected from gallery
    - Automatically able to click "upload"to upload a different image instead
    - "DONE" button to submit your home listing form and to return back to the **housingFeedFragment**
- **Associated Kotlin Files**
    - These four fragments will contain a set of onClickListers on the "CONTINUE" and "DONE" buttons so that the user can navigate to the next respective screen
    - You will need to come up with a way to pass all the attributes from each fragment into a single Post object to be used in Screen 5 when making an API POST request. The code for making this API request will be in **Phase 6**

## Phase 3.3: Viewing a Post

This is where the basics must be set up for viewing a note:

- **fragment_housing_display.xml**
    - Make ten Text Views and one Image View for users to provide the following information for each Home Listing Post:
        - Send an Email to the owner (Button)
        - Image (ImageView)
        - Number of Beds (TextView)
        - Number of Bathrooms (TextView)

        - Price (TextView)
        - Move In Date (TextView)
        - Location (TextView)
        - Type (TextView)
        - Date Posted (TextView)
        - Covid Status (TextView)
        - Description (TextView)
- **HousingDisplayFragment.kt**
    - Make sure to have navigation controller ready for navigating to the NoteFeedFragment
    - Need to set data through grabbing the post from safe args
    - Each action should respond accordingly to the next call
    - Create an onClickListener for the email contact button that starts an implicit intent to open up the email app and passes the email string data as well, so that it automatically fills the recipients field

### Phase 3.4: Sign In Screen

This is where the user will be brought to if they attempt to create a post, but have not logged into their Google account yet

- **fragment_sign_in.xml**
    - Have a textView for the title of the login screen
    - Have a Button for starting the Google Sign In Client
- **SignInFragment.kt**
    - Do not begin until **Phase 5**

## Phase 4: Get Housing List

- This is where we will make a retrofit call
- On Screen 1, retrieve all posts from server
- Use the Singleton service defined in the main activity to fetch the posts on the server
- Use your interface and call the endpoint accordingly for launching the retrofit call

## Phase 5: Login with Firebase

- The user should only be brought to this screen if they are currently not logged and are attempting to create a post

- You will not need to add any dependencies manually for this process, as Android Studio will do that for you through Firebase Integration (PROVIDE LINK)
- https://firebase.google.com
- That being said, make sure that the following dependencies are implemented:

```
implementation platform('com.google.firebase:firebase-bom:26.2.0')
implementation 'com.google.firebase:firebase-analytics-ktx'
implementation 'com.google.firebase:firebase-auth-ktx'
implementation 'com.google.android.gms:play-services-auth:19.0.0'
```

## Phase 5.1: Configure Google Sign In

- Create a global variable of type GoogleSignInClient
- In onCreate(), create a variable for our Google SignIn Options (gso), and assign it to a Google Sign in Builder which requests the following information:
    - Id Token
    - Email
- Assign our global GoogleSignInClient variable to a GoogleSignIn object and get a client member using our activity context and the gso we made earlier as parameters
- In our onViewCreated(), set an on click listener to our sign in button, and start the activity result for our sign in. The result we expect to get back is a request code to know whether or not starting this activity was successful (This request code can be any integer)
- In our onActivityResult(), if the request code that we get back from our result matches the one we sent in, then we call our handleSignInResult function

## Phase 5.2: Handling Sign In Results

- Create a function which takes in a task API of type Task<GoogleSignInAccount> as a parameter
- Create an exception handling statement which will

```
try {
    // set a variable to get the result of the GoogleSignInAccount
    // use firebase to authenticate the idToken of this variable
}
```

- If this try is not successful

```
catch {
    // print an exception
```

```
}
```

### Phase 5.3: Firebase Authentication With Google

- Create a function to authenticate Firebase with a Google Credential, and take in the idToken as a String argument
- Set a variable equal to your Google Credential
- Place an addOnComplete listener to a task in which you try to sign into Firebase with the Google Credential variable you made earlier

```
if (task is successful) {
    // then navigate to to the fragment_create_post.xml
} else {
    // print an error message
}
```

Note: The Android emulator will store your login information, so you will not be prompted to login again after the first time. If you would like to test the login feature again, you will need to goto the AVD manager and wipe the data from the device

## Phase 6: Upload Image and Retrieve Image

### Phase 6.1: Pick and Display Image from Gallery

- Create an event listener for the image view that starts an implicit intent to let the user pick an image from Gallery
- Observe how to override the onActivityResult function to display the image in the image view as given
- Make sure to have an onClickListener enabled button for uploading images

You should see that the onActivityResult is already implemented in FormImageFragment. You need to invoke an implicit intent that will open the phone's image gallery and provide the request code that matches what the function onActivityResult() is expecting.

```
startActivityForResult(Intent(Intent.ACTION_PICK,
    MediaStore.Images.Media.EXTERNAL_CONTENT_URI), REQUEST_CODE)
```

### Phase 6.2: Upload Image

Before sending the request to create a post, we need to send another request which uploads the image.

- Use ContentResolver to read image data and extension form the resource URI

```
val contentResolver = requireContext().contentResolver
val bytes = contentResolver.openInputStream(imageUri!!)!!.readBytes()
val extension = MimeTypeMap.getSingleton()
       .getExtensionFromMimeType(contentResolver.getType(imageUri!!))
```

- Create a Multipart Body that contains the image data and type

```
val requestBody = RequestBody.create(MediaType
.parse("multipart/form-data"), bytes)
val part = MultipartBody.Part.createFormData("image",
"image.$extension", requestBody)
```

- Call the image upload API

```
val service = (requireActivity() as MainActivity).service
service.uploadImage(part).enqueue(object :
Callback<ImageUploadResponse> {
    ...
}
```

    - Use the path returned by the image upload API in the post request to link the image with the post

## Phase 7: Create Post Page

- Connect the post screens' data with the server set up from earlier
- Send the data to the server using the "/housing" endpoint. Make sure the post object being used to send the data include all the accumulated information from all previous screens (including the path to the image)
- One spot where you can send this data is inside the onResponse section of the API call used to upload the image to the server

## Phase 8: UI Designs

Now that all the logic is set up. Do your best to match the provided designs. Designs are linked at the top of this prompt.

# Resources

- Canvas Project 3 Page, Lectures, Demos & Links
- Piazza
- Discord
- Email

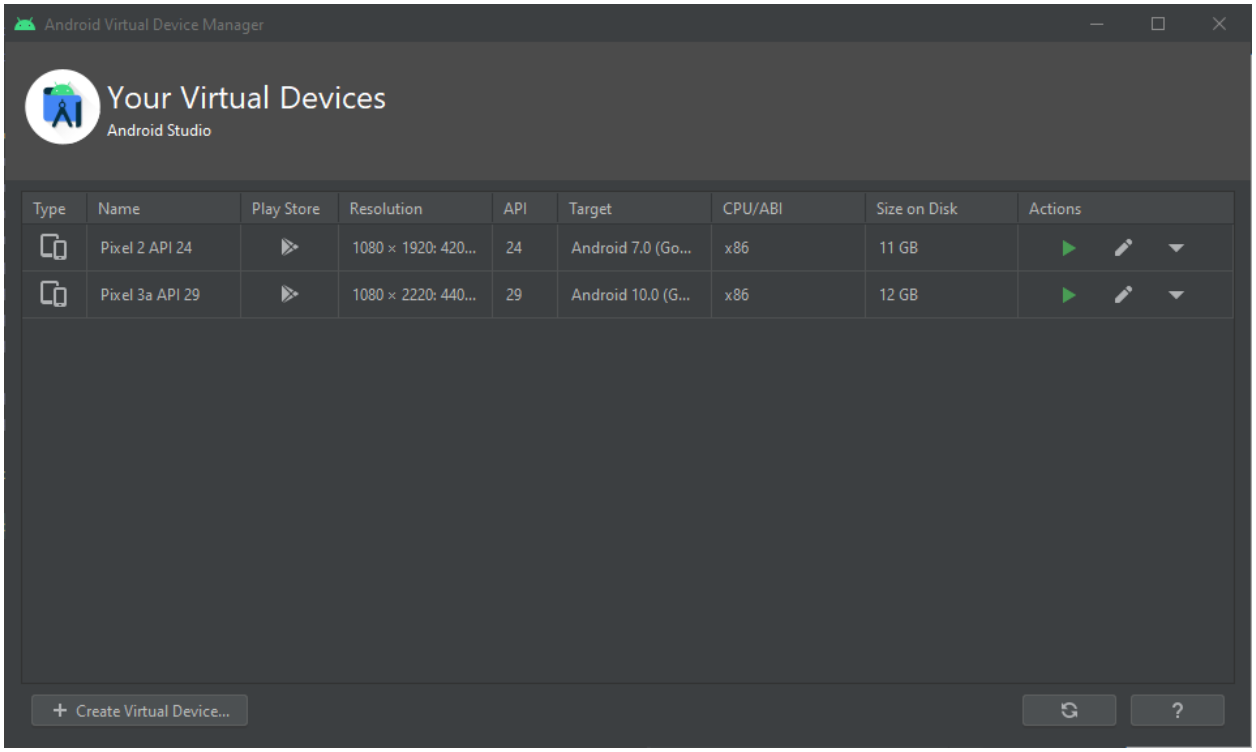
# Submission

Projects will be submitted through gradescope via github. You can access gradescope through the tabs in Canvas. Please only have one partner submit the project and add the other partner to the gradescope submission so they can see it. Here are some details for submitting the project

- Include a README.md that:
    - Describes the objective of the project in your own words.
    - Includes a link to the video code tour/demo
    - Make sure your server is set to public on replit and include a link to it (Once all the assignments are graded you can feel free to private the server or delete it)
    - A sample README.md has been added to the starter-code github
- Code Tour/Demo
    - No more than 7 minutes. If you can comprehensively cover the project in less than that, then there is no need to use the full 7 minutes. Concise and complete is better.
    - Provide a video recording of your project and codebase describing how you accomplished the end result. Make sure to show the app running and all the possible interactions. Also outline and show comprehension of the sections/phases that are coded
    - Suggested method for video submission: Make a Zoom cloud recording and add the link in the README.md. If you choose another method of recording make sure you can link a sharable video in the README.md

# Additional Notes

- Make sure your emulator supports the google play store. You can verify this by checking if there is an icon under the play store columbine the Android Virtual Device Manager..




- **Mac Users with M1 Chips ONLY**
    - Mac M1 laptops currently do not have full support for the Android Studio emulator, but Android has since put out a preview emulator
        - https://github.com/741g/android-emulator-m1-preview/releases/tag/0.1
        - Download the android-emulator-m1-preview.dmg file
    - The Android Emulator will be a piece of software which must be opened manually before clicking run in Android Studio
    - Launching the Web Page may not fully function as expected. You may only see a white screen pop up, but you should still see the correct URL being displayed in the web search bar