Capstone Project Guidelines

GitHub Username: Johnny Jem

Workouts for Imgur

Create your own workout album or download pre-existing ones via Imgur!

A workout app that downloads public albums found at Imgur.com and transforms them into organized workouts. Features "Most Popular" and "Recommended" pages of community-made workouts. Logging to keep track of workouts completed plus reminders. In-app web access to imgur.com for easy browsing of potential albums for download.

Intended User

WMy intended user are users interested in health and fitness, particularly those found on communities such as imgur.com and reddit.com/r/bodyweightfitness and r/fitness.

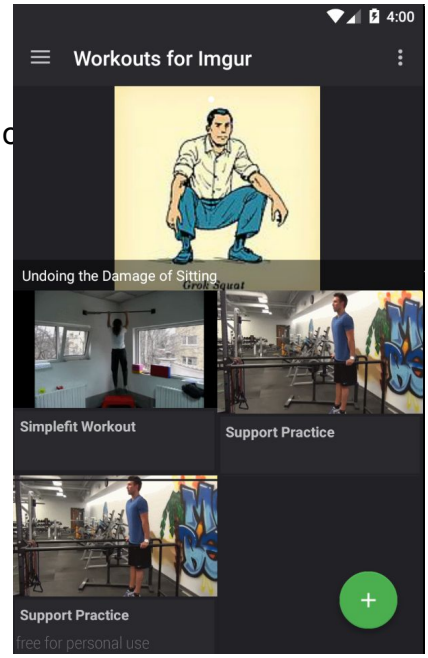
Features

List the main features of your app. For example:

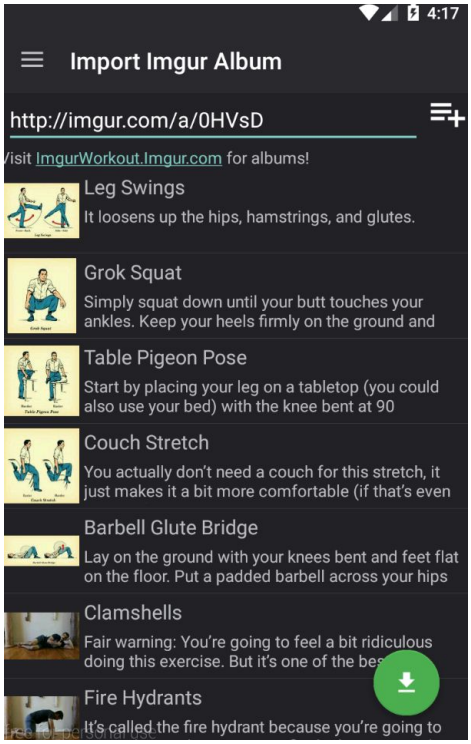
- Supports on screen title and instructions
- Offline support after downloading your album
- Timer or Sets & Reps per Image
- Rest Countdown
- Text-to-Speech
- Auto-complete logs after workout
- Easy in-app browsing of imgur.com to retrieve albums

User Interface Mocks - the Main Points

Screen 1 - MainActivity



Title Screen: Displays “Most popular” slide on top and users library o albums below. FAB button allows user to quickly search & add albums.
Revisions to make: Add BottomSheet that keeps track of currently playing album.



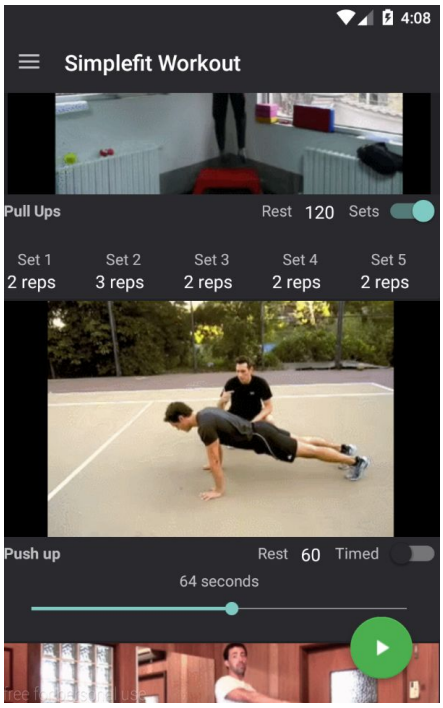
Screen 2 - Download albums screen

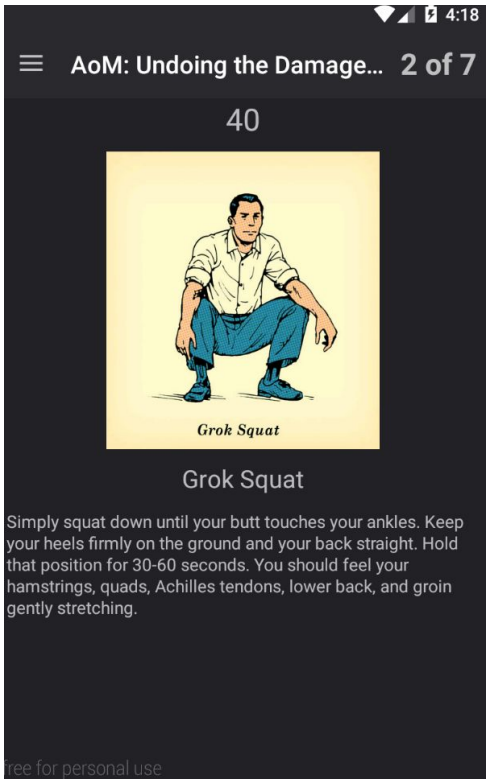
- User will enter screen either via mainActivity FAB, through a browsers shared intent when user has navigated to imgur album, or via in-app custom tab that points to imgur.com .
- User can download albums here

Revisions to consider: place “Search” in toolbar.

Screen 3 - Album Viewer

-Enter screen via MainActivity album selection.
-Recyclerview with cards that allow user to choose Sets, Reps, or timing.
Revisions to consider: Re-think card display to provide a smoother UI.



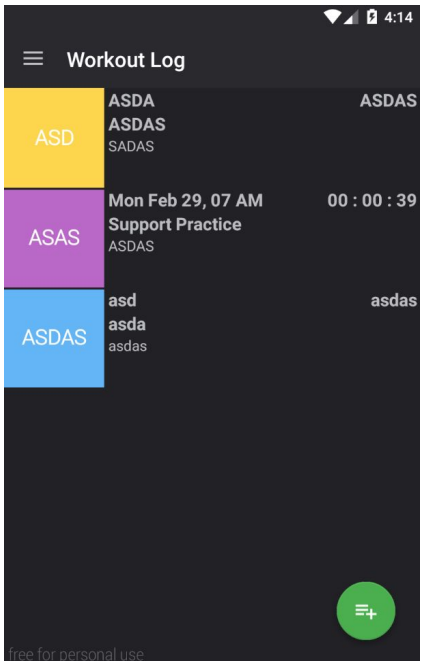


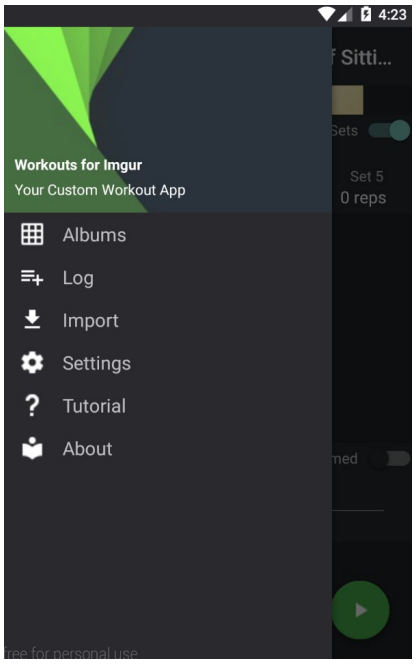
Screen 4 - Image Player

-Enter screen via screen 3 “Album Viewer”. Composed of a whole-screen viewpager that does not allow swiping until workout is complete. Green “next” FAB pops up once user is OK to move on to next exercise.
Revisions to consider: Clean up display and implement custom fonts.

Screen 5 - Log Viewer

-Enter screen via Side drawer or completion of workout.
-Allows user to view past workouts and details in a recyclerview.
Revisions to consider: Consider re-designing input of details and card detail.





Screen 0 - Navigation Drawer

Includes links for top-level navigation.
Disabled during album playback to prevent interference with viewpager paging.

Key Considerations

How will your app handle data persistence?

My app will use a Realm database to handle data persistence of JSON info downloaded & paths of images downloaded onto device. Logic for duplicated elements will be implemented as well. Data persistence during app runtime will be handled via Model-View-Presenter implementation where the presenter is responsible for keeping track of caching information, until Realm DB is needed to persist information.

Describe any corner cases in the UX.

If a user is currently playing an album, the back button will return to Screen 3 - Album Viewer. In all other cases the User will be returned to the Main Activity, including using the back button when having launched the activity via intent from external application.

Describe any libraries you'll be using and share your reasoning for including them.

For my Android Nanodegree capstone project I plan to revamp this app with my updated standards including:

- Material Design - Design Support Library
    - Appeal to users familiarity with Android platform standards.
- Model-View-Presenter Architecture - Via mosby library.
    - Separation of concerns will allow for a more robust testable app.
- Lambda Expressions with RxJava (Reactive Extensions for the JVM)
    - Allows for composable, cleaner, asynchronous code.
- Dependency Injection - via Dagger 2
    - Easier testing and separation of concerns.
- Realm database
    - More fluent api with fast read speeds.
- Handle Networking with Retrofit
    - Using the standard for networking and allows for management of different endpoints. Supports RxJava Integration.
- Glide along with OkHttp to handle the loading and caching of images.
- Butterknife - Field and method binding for Android views. Facilitates cleaner code.

Next Steps: Required Tasks

Task 1: Project Setup

- Configure libraries
- Design layout
    - Design recyclerview elements.
- Apply MVP framework with the help of Mosby library.
- Setup Dagger 2 configurations.
- Setup Retrofit Endpoints

If it helps, imagine you are describing these tasks to a friend who wants to follow along and build this app with you.

Task 2: Implement UI for Each Activity and Fragment

List the subtasks. For example:

- Build UI for MainActivity
    - Design Recyclerview views
    - Navigation Drawer
    - Toolbar
- Build UI for Download Activity
    - Design recyclerview to replicate imgur mobile rows.
- Build UI for Album Viewer Activity
    - Design recyclerview views to per Youtube Music app's rows example.
    - Design popup fragment where user can better configure image options.

- Build UI for Album Player Activity
    - Design for both Tablets and phone specifications.
    - Look at youtube music player app for guidelines.
- Build UI for Workout Log Activity
    - Apply material design card views.
    - Implement fragment pop-up to insert details.

Task 3: Connect Services

Once all UI components have been designed move on to connecting services:

- Connect Realm library to recyclerview adapters and adapters to recyclerview
    - Make Realm models
- Connect Retrofit endpoints with imgur api.
    - Implement downloading behavior (with background service) for imgur albums
    - Setup “shared to our app” external app (browsers, etc) Intent behavior.
- Implement image library management logic to interface with Realm database.
- Add a setting page to control cache, offline downloads, logging data, and backup behavior.
- Add “About” fragment with license and credits.
- Add “Help” fragment with UI & app guidelines.

Task 4: Testing

- Implement Unit tests and functional UI tests via AssertJ-Android, Mockito, Expresso libraries.
- Run monkeyrunner tests
- Make sure there are no frame skipping.
- Test for leaks using LeakCanary.
- Handle error cases.

Task 5: Real-World Testing

- Implement Integrations tests for Api response and background services.
- Live-test app by using it myself.

Task 6: Assemble Google Play Resources

- Clean app, remove debug code & sensitive information.
- Assemble Google play screenshots and write app descriptions.
- Submit to Google play after reviewing app for compliance with all google play guidelines.