**How to Use this Template**

1. Create a new document, and copy and paste the text from this template into your new document [ Select All → Copy → Paste into new document ]
2. Name your document file: "**Capstone_Stage1**"
3. Replace the text in green

---



**GitHub Username**: Your GitHub username here

# Income Portfolio Tracker

## Description

The Income Portfolio Tracker application will be an investment application that will help users track the income generated from their stock or ETF investments. Most investment applications track gains in price, but this application will track the income return of a person's investment portfolio in addition to the portfolio's gains. The intended user is a value investor that is interested in gauging what monthly cash flow their investment portfolio provides as opposed to just unrealized capital gains.

## Intended User

The intended user of the Income Portfolio Tracker application is the value investor that is interested in gauging what monthly cash flow their investment portfolio provides in dollar and percentage terms as opposed to just unrealized capital gains.

## Features

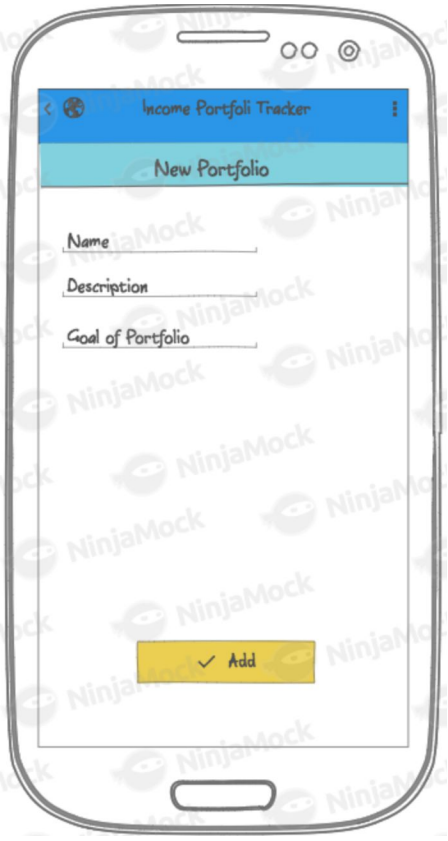
- Lookup any stock listed in the US exchanges
- Build one or more portfolios of stock holdings
- Track the yearly yield (percentage and total) of the portfolios based on current value and original investment
- Track the unrealized capital gains of the portfolios

# User Interface Mocks

## Portfolio List Screen (landing screen)


The portfolio list screen is the application landing page; it will list the portfolios saved for the user and will provide an option to create a new one. Clicking on a portfolio in the list will take you to the portfolio details page.

## New Portfolio Screen




The new portfolio screen will allow the user to create a new portfolio entity meant to contain multiple stock holdings. The stock holdings will be added in a later screen. Any portfolio added will be persisted in the device SQLlite DB

## Portfolio Details Screen


The portfolio details screen is the central screen of the application, as it will provide the user a view of the cash flow generated by the portfolio as well as the sector diversification of the portfolio. A list of the holdings will also offer a quick performance analysis of each stock holding.

## New Stock Holding Screen


The new stock holding screen will allow a user to add a stock position to a portfolio. The symbol entry will be validated with the web service to verify its validity, and any non market price information (name, symbol, industry sector) about the stock will be pulled down from the web service and stored in the device's SQLlite DB.

# Key Considerations

### How will your app handle data persistence?

The Income Portfolio Tracker application will handle persistence on the device using Android Architecture Components, specifically the Room library. It will also sync the data into the cloud using Firebase when a network connection is ready.

### Describe any edge or corner cases in the UX.

Potential edge cases to handle:

- Inputs not formatted properly; the application will be based on floating point number entries, so checks before data persistence will have to be done. I'll also explore if keyboard restrictions for those inputs is possible.
- Stock symbols change, so checks when current price data is pulled from the server will have to be done to avoid NullPointerExceptions.
- The main data structure will be the portfolio, so should the user delete all their portfolios, the application will have to handle future stock position additions accordingly, by creating a portfolio first.

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

- Room for data persistence; this makes SQLlite persistence much simpler and error free.
- Espresso for testing the portfolio creation and stock position additions/deletions, as well as the calculation displays.
- Butterknife for dependency injection of views; this makes for more readable code.

### Describe how you will implement Google Play Services or other external services.

- The Google Firebase Realtime DB service will be used for data persistence in the cloud. Only the stock portfolios and stock positions will be persisted to the cloud, since all other additional information is market data that can be downloaded from REST API calls.
- The Google AdMob service will also be used, to show advertisements on the home screen and the portfolio detail screen.
- The Iex Trading (https://iextrading.com/developer/docs/#stocks) REST service will be used to retrieve stock market data; it provides feeds on current market data and dividend payouts needed for the required cash flow analysis of the application.

# Next Steps: Required Tasks

The tasks described below will be broken down not by screen functionality but by ground up implementation. The tasks are:

1. Project setup and JSON parsing
    - Create the project and add in all the necessary libraries for the REST calls; implement the JSON parsing using static inputs and unit test
2. REST call implementation
    - Implement the REST calls and integrate with the JSON parsing and all associated integration/unit tests
3. Data Persistence
    - Implement all the data persistence with Room, as well as any integration tests
4. Create and hook up the UI
    - Create the views and integrate with the REST and persistence
5. Google Services
    - Add in the Google Firebase and AdMob services and test

## Task 1: Project Setup and JSON parsing

- Create the application project and set up the github for it
- Implement the JSON parsing and the data objects to use
- Unit test the JSON parsing with static JSON inputs

## Task 2: REST call implementation

- Implement the REST calls for the stock information
- Integrate the REST calls with the JSON parsing
- Write integration tests for the REST to Java POJO interface

## Task 3: Data Persistence

- Add in the Room library
- Design the persistence classes; ideally reuse the JSON parsing POJOs
- Write integration tests for the adding and deleting of objects

## Task 4: Create and Hook up the UI

- Create the activities and layouts in the order detailed above
- Hook up the activities with the REST calls and Room persistence
- Write Espresso tests for integration tests

## Task 5: Integrate Google Services

- Add in persisting to the Firebase DB
- Write integration tests for the Firebased DB persistence functionality
- Add in Google Ad services
- Espresso test the Google Ad services functionality

---

**Submission Instructions**

- After you've completed all the sections, download this document as a PDF [ File → Download as PDF ]
    - Make sure the PDF is named "**Capstone_Stage1.pdf**"
- Submit the PDF as a zip or in a GitHub project repo using the project submission portal

If using GitHub:

- Create a new GitHub repo for the capstone. Name it "**Capstone Project**"
- Add this document to your repo. Make sure it's named "**Capstone_Stage1.pdf**"