Investment Performance Mobile App: Winter Term Progress Report

Tyler Jones, Aviral Sinha, Sam Cooney

February 16, 2018

Abstract

The purpose of the following document is to review and discuss the current state of our investment performance mobile application as our development has begun over the last 6 weeks of Winter term. Our group will break down the struggles that we faced with the project as individuals, as well as struggles with the project overall.

CONTENTS

1 INTRODUCTION TO PROJECT

Our project is an investment performance mobile application. Our main goal with this application is to provide users with a way to enter and track the progress of their investments, so they can gain unique financial insight about the quality of said investments. Typically, through a brokerage, for example, users may only see what investments they have, and they are not always given a composite performance, as is the goal with our project. We aim to give the user a metric for both their entire investment portfolio, as well as the ability to drill down into individual investments for a more localized look. At the beginning of development, we sat down as a group and went over the pieces of the project and what would be a good way to breakdown the work. Essentially, we decided that the project could be divided into 3 major components, being the API, the development of the UI, and the backend/database. As will be discussed in this progress report, Avi has taken the lead on the API, Sam on UI development, and Tyler on the backend/database.

2 SAM COONEY - UI DEVELOPMENT

2.1 Description of Current Status

As of our alpha release, our UI is fully functional with all pages, buttons, and placeholders for charts/data implemented. Because of issues regarding iOS testing, the initial functional UI was built using Android libraries. Since I am new to UI design, I am using the Android test function UI to learn Xamarin and layout design. Since the functional Android UI is used to test connections to other parts of our project, I decided to create a Mock Up of our planned Beta release for the UI. We are using Xamarin.Forms for the Beta release which is a set of libraries that allows for a single codebase UI that at runtime, elements are compiled into the platform specific views. The mock-up for this beta release was created using Marvel, a mobile app mock-up site, offering free mock-up design.

This is the Portfolio View where from here users can add or remove assets from their portfolio, view vital financial data from a summarized point of all the assets currently in the portfolio. Our chart is a placeholder and will be replaced with a Google Charts designed pie chart that will be received from the web APIs.

Fig. 1. Portfolio View Example
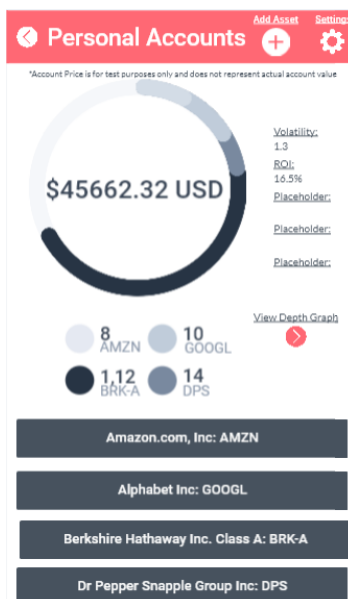
The Portfolio view will be where users spend majority of their time. Since we are looking to have design be built for quick, on-the-go access, the information in this view needs to be referenced quickly. This has created a challenge with balancing both simplicity but also retaining enough information that the user understands what they are looking at. Not having taken any graphical design or artistic classes, this is new and difficult for me but is proving to be an exciting task.

Our current UI functions exactly as viewed in the mock-up, what is missing is the design and the web calls. The asset view is similarly programmed. It functions very closely to the portfolio, except the view expresses the individual asset data instead of the portfolio as a whole.

Fig. 2. Asset View Overview
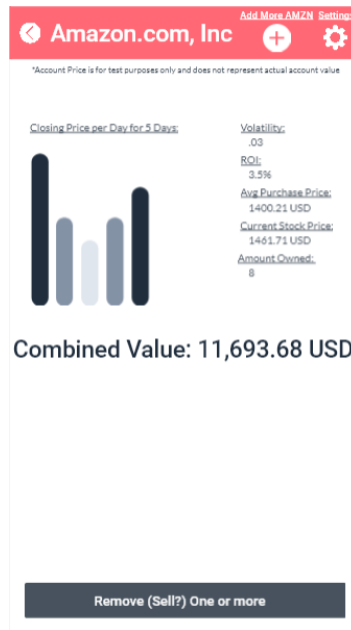
The asset view is where users will spend the second most time. From here they can view things like volatility, price changes in the last few days, return on investment since addition, and other financial data retrieved from the API and database. Much like in the last view, it is proving to be an arduous task in balancing the design.

The rest of the pages not shown here have either been fully mocked up and the functionality has been implemented. The Android functional UI was displayed in the video, as well as the rest of the Beta design.

2.2 What's left to do

Our next steps are continuing development on the Beta UI release. Refactoring the Android functionality into Xamarin.Forms is proving to be a new challenge but fortunately there are lots of guides online that are extremely helpful. The portfolio view, asset view, adding new assets and portfolios all need to be implemented using Xamarin.Forms in the format of the mock-up UI. From here, we need to connect both our APIs for the charts and the financial data as well as the database storing user data - login info and portfolio/asset data. Avi and Tyler are using test UIs to plan how these connections will be made. Once these are finalized, we can begin to implement them into the Beta release.

2.3 Problems impeding progress/Solutions

Initially we ran into research problems in early development that lead to the functional UI design using the Android UI. First, Xamarin.Forms does not have the full functionality that each individual native

UI provides, and second I don't own a Mac which prevents testing deployment on the iOS platform. Since we did not want to halt development, we began work using Android libraries only. Since then I have aquired an older iMac and have since have also begun Xamarin.Forms testing on campus where Macs are available. The decision to switch to Xamarin.Forms was a group decision based off the goal to stick to our single code-base design. This on top of the fact that we are not graphic design gurus in any sense, Xamarin.Forms will not limit us in any ways in terms of development.

2.4 Other relevant/interesting information about project

Button creation has turned out to be extremely difficult. Finding the perfect mix of both small and simple but still providing enough information that the user needs to know where to navigate is very challenging. YouTube and the Xamarin guides are a nice guide but creating elements that are unique to our project is still a puzzle. This was something that I had not expected when first starting this project and so it has come as a big surprise to be one of the harder parts of the work.

3 AVI SINHA - APIS AND BUSINESS LOGIC

3.1 Description of Current Status

The API part of the project is being utilized through the use of 2 major tools: Quandl and Google Charts. We've started with the Quandl API as it is necessary for all the latest financial data to be fed into our application.Right now our project is just past the alpha level of development where the APIs we are implementing are currently being adapted into being compatible with Xamarin. The API part of the project is being utilized through the use of 2 major tools: Quandl and Google Charts. We've started with the Quandl API as it is necessary for all the latest financial data to be fed into our application. Since Xamarin is primarily written in C# we've had to take the existing documentation for Quandl and redevelop it to be functional within the application. The API consists of both time series and tables of various stocks and options. Time series data ranges from specific ones that can be outputted as follows in JSON:

```
{
  "dataset_data":{
    "limit":null,
    "transform":null,
    "column_index":null,
    "column_names":[
      "Date",
      "Open",
      "High",
      "Low"
    ],
    "start_date":"2015-05-24",
    "end_date":"2015-05-28",
    "frequency":"daily",
    "data":[
      [
        "2015-05-28",
        9.58,
        10.17,
        12.96
      ],
      [
        "2015-05-27",
        9.53,
```

```
            10.13 ,
            12.97
        ] ,
        [
            "2015−05−26",
            9.53 ,
            10.11 ,
            12.98
        ]
    ] ,
    "collapse":null ,
    "order":"desc"
  }
}
```

The code above shows how the data retrieved is takes a specific dataset, then outputs into the following columns: Date, Open, High, and Low. It also shows a daily frequency upload from May 26 to May 28th in a decreasing order. This JSON file will then be entered into the database and can be used for other information that the user will need to make a sound financial decision in maximizing their portfolio.

We can also send requests that will slice and customize the specific metadata of a time series data set prior to download. Using the following example we would get this

```
{
  "database":{
    "id":4922 ,
    "database_code":"WIKI",
    "name":"Facebook EOD Stock Prices",
    "description":"End of day stock prices , dividends and splits for 30000 US companies
    curated by the Quandl community
    and released into the public domain.",
    "datasets_count":3179 ,
    "downloads":186224033 ,
    "premium":false ,
  }
}
```

With this we get the overall listing of all end of day stock prices and any stock splits that have occurred. We also are able to see how many times this particular data set has been downloaded and if a premium membership is required for this. This type of alternative data is what makes Quandl superior over other data tools because we can get various types of financial data that can influence decisions and the market as a whole.

We can get the monthly percent changes for FBs closing price at the end of the day. All of the requests will output to a JSON file that will be accessed by the database and be visible in the user interface. The main purpose of the API is for it to provide type safety guarantees and basic validation on data before requesting. Everything is based on the concept of a request with all the download requests being implemented into a single interface.

3.2   What's left to do

In terms of what we have left for this is setting up the various requests that a user will be making for retrieving their financial data. These will consist of download, Multiset Download, Metadata Download, Favorites Download, and Search.

In addition to this well also be implementing a Google Charts API which hasnt begun development yet. Using the data from the SQL server the data will be binding the financial data together that is pulled from two major sources. The financial information provided by Quandl and then the user data

of what specific investments the portfolio consists of. Using the ASP.NET environment we will use SQL to show which tables are being utilized and have the queries binded for the specific information that the chart is asking for. Then using .NET we pass the data into google charts in a JSON format. Since the ASP.NET environment will connect the SQL data.

3.3 Problems impeding progress/Solutions

The main problem impeding our progress with the API is that it was started much later than both the front and back end development. Originally we were going to divide both iOS and Android development with Xamarin but then we figured out that Xamarin has a feature that allows for the same codebase to easily be shared so we began porting the existing code to forms and then starting the development.

3.4 Other relevant/interesting information about project

Below we have provided a few code snippets of what the Quandl API interface will look like within the application, these functions will call additional files that will house most of the data necessary for accurate information to be provided.

Download:

```
using System;
using QuandlCS.Requests;
using QuandlCS.Types;

namespace QuandlCSTest
{
  class Program
  {
    static void Main(string[] args)
    {
      QuandlDownloadRequest request = new QuandlDownloadRequest();
      request.APIKey = "1234-FAKE-KEY-4321";
      request.Datacode = new Datacode("PRAGUESE", "PX"); // PRAGUESE is the source, PX is th
      request.Format = FileFormats.JSON;
      request.Frequency = Frequencies.Monthly;
      request.Truncation = 150;
      request.Sort = SortOrders.Ascending;
      request.Transformation = Transformations.Difference;

      Console.WriteLine("The request string is : {0}", request.ToRequestString());
    }
  }
}
```

This specific code, shows a download request, asks for the API key, the specific datacode. The file format by default will also be in JSON and consistently updates at a frequency of once per month and ascends up.

Metadata Download

```
using System;
using QuandlCS.Requests;
using QuandlCS.Types;

namespace QuandlCSTest
{
```

```
class Program
{
  static void Main(string[] args)
  {
    QuandlMetadataRequest request = new QuandlMetadataRequest();
    request.APIKey = "1234-FAKE-KEY-4321";
    request.Datacode = new Datacode("NSE", "OIL");
    request.Format = FileFormats.XML;

    Console.WriteLine("The request string is : {0}", request.ToRequestString());
  }
}
}
```

The multiset download is similar to the previous one but instead looks at the market as a whole and compares NASDAQ to a specific investment you want to look at. These are the first couple requests that a user will want to use and then make their portfolio more detailed while also looking at it in a macro level.

Overall the progress with the API's we need has been going smoothly in terms of getting everything set up. More than likely we will be running into hurdles with having be properly compatible with Xamarin but that can be troubleshooted. Once the API's are completed we will begin setting up the business logic of the financial data that will be displayed. We will need to set up functions that properly calculate, volatility, return on investment, risk, and any other points that will be key to having displaying proper metrics.

4   TYLER JONES - USER DATA AND BACKEND

4.1   Description of Current Status

Coming into this project I have never worked with mobile applications, and I haven't taken the databases course, yet I am handling the backend for a mobile application so I have had quite the ramp up time in terms of learning how to handle my portion of the work. During fall term, it was hard to gauge if my planning and research was complete, because I didn't know what I didn't know, and if there were components that I was perhaps missing or not understanding, which definitely ended up being the case.

Currently, however, I am working on improving the database structure. As is, it has been a significant time gate for myself to research and understand how to implement our data relationship into a relational database, as well as how to effectively interact with such data from a mobile application. The basic relationship we need to have configured for our data is as follows: Multiple users can share a portfolio and a single user can have multiple portfolios. Moreover, a portfolio can and will be comprised of multiple investments, and a certain investment can belong to more than one portfolio (multiple people can buy Apple stock). Currently, with this multiple many to many relationship structure that needs to be setup, my database structure can be seen below.

A many to many relationship, as I have learned as I have been building my database, is best fixed by adding a 3rd joining table for the relationship. As is demonstrated by the screenshot of all the tables in my database, the naming conventions I have set up, for instance with UsersPortfolios, allows the relationship between multiple users and multiple portfolios to be achieved within the context of a relational database like MySQL. The same goes for PortfoliosInvestments when showing the multiple investments that make up a portfolio. Without going into too much detail, how the process will work, and is working currently, will be that when a user adds a new portfolio, our application will make a call to the API and push the information about that portfolio, including portfolio name and total amount invested in the portfolio into the Porfolios table, as well as insert a new entry into the UsersPortfolios table that specifies who owns that new portfolio, which could be one or multiple users. All of the data that will be queried from the database is from the pictured UsersPortfoliosView, which places all of

Fig. 3. Database Table Overview

the portfolios and their owners into a single table, or technically a view in this case, that is able to be queried by our API effectively. This same strategy is applied in the PortfoliosInvestments table, and will also have a corresponding view that allows for effective queries.

Another thing that I have been working on is the interaction of this data with my database. Coming into this term, I wasnt aware what an API was or what it was used for, but I ended up having to build and use one for our application. These first few weeks of the term have been dedicated to research, as well as constructing our REST web API on our web.engr hosted OSU site. This REST web API is based in php and returns JSON for our C# based application to parse and display for the user. The API is generalized so that calls are made to it in the form of /api.php/TableName, the output from such a URL is JSON of all the content of the desired table. I felt that making my API simple enough so that calls to it can be generalized and simply return all the content from the table allows me to do my complexities involving the data through the database and client side, rather than getting bogged down in making an overly complex API.

4.2 What's left to do

What I have left to do and where the project as a whole is going are two different things. When composing the final product and putting all of our pieces together, my backend portions will mostly be plugged in to Sam's work with the UI. The work I personally have left to do on the backend is registering a new user, adding another user to an existing portfolio, figuring out how to configure the database to handle specific numbers of shares being associated with a portfolio, and lastly, adding hashing functionality to our users' passwords, as right now they are being stored just as plain text. As far as the application as a whole is concerned, the biggest piece that we need working now are our calls to the financial API, Quandl, that will use fetched data in order to actually allow our app to do what it was intended to do. This will be a challenge not so much from a technical standpoint, but rather from a business standpoint, and will require verification from our client going forward that we are using the correct data, and generating and showing the user their investment performance properly. Overall, as we have gotten a much better sense over these first few weeks about what degree of work is going to be required for all the components, as well as a more complete and solid picture of how our app is going to work as a complete unit, we feel confident moving forward about the state of the project.

4.3 Problems impeding progress/Solutions

One bottleneck and problem, as I previously mentioned, that I have experienced so far is being thrown into many components of this project that are entirely new. The paradigm of mobile development, C# which our app is written in, MySQL for the database, php for the API, and JSON parsing within the app are all completely new tools that I am having to both learn and effectively use all without having any prior knowledge of.

4.4 Other relevant/interesting information about project

Regarding the work breakdown of the project, Sam, Avi, and I are all individually working on components of the application, so although we haven't put them all together yet, the act of doing so will be trivial. All of us are new to mobile development, as well as Xamarin, so we figured this work flow would be much more efficient while we more or less figure out how to get each of our necessary components working. Where my portion of the work currently stands is mostly with allowing the user to communicate with the database within the application. Currently, users have the ability to log-in, albeit insecurely at the moment, load their portfolios, and load the investments contained within said portfolios. Furthermore, Xamarin has proved to actually be an incredibly interesting tool for our development. Developing a cross-platform application in C# feels a bit like magic at times, as neither iOS or Android use C# natively. It has been interesting developing from the standpoint of making an app that is as cross-compatible as possible, and at times this means sacrificing some of the unique and powerful qualities that iOS and Android both uniquely offer.