CPSC 304 2015W2
Project Part II

Group Members

Name	**Student No.**	**Unix ID**	**Email**
Albert Xing	40640104	z6k8	albert.xing@alumni.ubc.ca
Calvin Cheng	36090132	o7x8	calvin.cheng@alumni.ubc.ca
Kyle Stadnyk	52749025	b5p5	b5p5@ugrad.cs.ubc.ca

By typing our names and student numbers in the above table, we certify that the work in the attached assignment was performed solely by those whose names and student IDs are included above.

In addition, we indicate that we are fully aware of the hrules and consequences of plagiarism, as set forth by the Department of Computer Science and the University of British Columbia.

We realize that in using a platform unsupported by the course and course staff we assume full responsibilities for all technical issues that may arise due to our choice of platform.

Background

We are building a financial portfolio managment application. In general, this application will help investors and financial advisors track and manage their portfolios. Specifically, each portfolio is composed of any number of stock holdings.

Platform

We will use Ruby on Rails for our application. The application will be deployed and hosted through Heroku. Heroku uses PostgreSQL as the backing DB for Rails. The frontend will be built using HTML/CSS/JS. Frontend frameworks or libraries (React, Backbone, D3, jQuery) may be used.

Functionality

There will be three user groups that will have different roles in the application:

Investors Investors, or clients, will be able to monitor and make changes to their portfolios. They will also be able to view and update their personal information.

Advisors Financial advisors can manage portfolios on behalf of investors. They will be able to monitor and make changes to portfolios they are hired to manage. They will also be able to view their clients' contact information.

Administrators System administrators will have complete read and write access to all information in the system.

Specifically, updates to portolios will be executed as *transactions*. Each transaction is either a *buy* or *sell*. Transactions will affect the holdings in the portfolio - either by adding or removing a holding, or updating an existing holding. Other updates to portfolios include adding or withdrawing cash.

When an investor or advisor wishes to view holdings in a portfolio, multiple database queries are needed. The first query must retrieve all portfolios associated with an investor. For example, to list portfolios owned by investor '1234':

```
SELECT * FROM Portfolio WHERE owner_id = 1234;
```

Then, to list the holdings for a given porfolio ID (e.g. 4321):

```
SELECT * FROM Holding WHERE portfolio_id = 4321;
```

Each holding contains infomation about the stock associated with the holding, as well as the number of shares owned. The price of the stock is queried using an external API given the symbol. Then we can calculate the value of the holding by multiplying the stock price and the number of shared owned. The total value of a portfolio is the sum of the values of its holdings.

Queries can also be used to search for contact information of an investor given their name - for example, to find the email of users named "John Doe", we use the query:

```
SELECT email FROM User WHERE name = 'John Doe';
```

We can distinguish between investors and advisors by whether or not they are owners of some portfolio - advisors would never own a portfolio. Alternatively in implementation we can assign each user a type field that would eliminate the need for such a complex query.

General performance metrics for portfolios, investors, and advisors would require more complicated queries and analysis. For example, we might want to find the percentage return for a given portfolio. This can be calculated by dividing the current value of the portfolio by the initial principal invested. Similarly, the average return for an investor or advisor can be found by dividing the total value of all portfolios owned or managed, by the total of all princial investments for these portfolios.

Data

The main entities represented are Users, and Portfolios. Each User contains contact details as attributes, as well as an Address. Portfolios contain Holdings for a particular Stock. For more details, please reference our `project-schema`; an excerpt containing the database schemas is given below.

Database Schemas

Stock(symbol: CHAR(6), exchange: CHAR(6), name: VARCHAR(64))
Primary key: (symbol, exchange)
Functional dependencies: $symbol, exchange \to name$

The Stock entity represents a single stock, identified by a symbol and the exchange it is traded in. The given FD asserts that a unique symbol and exchange implies the name of the stock.

Holding(holding_id: INT, portfolio_id: INT, symbol: CHAR(6), exchange: CHAR(6), num_shares: INT, date: DATE, time: TIME, price: INT)
Primary key: (holding_id, portfolio_id)

Foreign key: portfolio_id references Portfolio
Foreign key: (symbol, exchange) references Stock
Functional dependencies: $holding\_id, portfolio\_id \rightarrow symbol, exchange, date, time, price$

A Holding is a number of shares of a stock inside a portfolio, with the date, time, and price (in USD) the shares were bought for. The FD states that a holding can be uniquely identified by a holding_id and the ID of the portfolio in which it belongs.

Portfolio(<u>id</u>: INT, purpose: VARCHAR(64), creation_date: DATE, principal: INT, cash: INT, owner_id: INT, manager_id: INT)
Primary key: id
Foreign key: owner_id references User
Foreign key: manager_id references User
Functional dependencies: $id \rightarrow purpose, creation\_date, principal, cash, owner\_id, manager\_id$

A Portfolio is a collection of stock holdings for a particular purpose. The FD states that a portfolio can be identified by its unique ID. Also note that portfolios for self-directed investors are both owned and managed by the same investor, while managed portfolios are owned by an investor and managed by an advisor.

User(<u>id</u>: INT, name: VARCHAR(64), email: VARCHAR(64), phone: CHAR(10), password: BINARY(60), address_id: INT)
Primary key: id
Foreign key: address_id references Address
Functional dependencies: $id \rightarrow name, email, phone, password, address\_id$

Users are either investors (who provide principal), advisors (who trade on behalf of investors), or self-directed investors (who assume both roles). Each user has a unique ID, as the FD states, through which they can be identified. We decided not to include email as a candidate key because it may be the case that a single individual would wish to open multiple accounts, for example a financial advisor who wants to open a separate individual account for their own use.

Address(<u>id</u>: INT, number: CHAR(5), street: VARCHAR(32), city: VARCHAR(32), country: VARCHAR(32), postal_code: CHAR(6))
Primary key: id
Functional dependencies: $id \rightarrow number, street, city, country, postal\_code$

An Address represents a particular mailing address with a street number, street, city, country, and postal code. For a more efficient implementation, a unique ID is assigned to each address which allows for faster queries through a single column instead of multiple columns. Note that although postal codes often uniquely identify cities, this is not generally the case especially internationally. We decided not to include the FDs $postal\_code \rightarrow city$ and $number, street, city \rightarrow postal\_code$ on this basis.

Division of Responsibilities

A rough division of duties is as follows:

Albert will be responsible for frontend development, including but not limited to interface design and development, user-facing forms, client-side API calls, and client-side analytics.

Calvin will be responsible for backend logic, including but not limited to server API handlers, Rails setup, maintenance, and deployment, and server-side markup and data generation.

Kyle will be responsible for data management, including but not limited to database interactions, database queries, and database maintenance.

The division of labour is not final, but may change as the project progesses.