# Documentation for the “Easy Finance” web application

## Table of Contents

**Foreword** ....................................................1

**The origins of this program** ....................................................1

**Summary of the features** ....................................................2

**File layout**....................................................3

**Known issues**....................................................4

## Foreword

The following is a summary of my final project for CS50’s Web Programming with Python and JavaScript. I have audited the course on edx.org. Disclaimer: I did not build this from scratch. This is a new iteration of a program with the same name, which I wrote earlier.

## The origins of this program

The students taking the CS50 course need to build a web application called “C$50 Finance” to complete Problem Set 7. This stock-trading application has the following elements:

- A registration form on the “/register” route that enters users into the database upon successful completion and sets their cash to $10,000.00.
- A login form on the “/login” route that logs users in upon successful completion and redirects them to the main page. This and the logout function were completed by the CS50 staff prior to the assignment.
- A table on the main page that contains information about the user’s stock portfolio. It needs to show the number of shares owned by the user, the current price per share, and the current value of their holdings, for each stock. It also needs to show users how much cash do they have and how the total value of their portfolio.
- A form that allows users to obtain the price of a selected stock on the “/quote” route. This element uses the Yahoo Finance API to get the information the user wants. Users have to enter the stock’s symbol for this function to work.
- A form on the “/buy” route that allows users to buy shares of the stock of their choice, as long as they have sufficient funds.
- A form on the “/sell” route that allows users to sell shares of stocks they own, as long as they have enough shares for the transaction.
- A table on the “/history” route that contains all stock transactions made by the user. It has to list the type (buy or sell) of the transaction, the stock bought or sold, the number of shares involved, the price of the stock at the time of the transaction, and a timestamp.
- A form on the “/change-pw” route that allows users to change their passwords.

(Pictures of each element are included in the “CS50_Finance_pics” folder.) Python (Flask), SQL, Jinja2, HTML and CSS were used in the making of this program. As you can see, no JavaScript of any sort is present. The program communicates with the user via redirects.

I have built the original version of the Easy Finance app based on this program. I have decided to add a “financial tracking” component to it and combine it with the stock-trading “simulator”. My aim was to create a simple and easy-to-use, yet wholesome application. I am somewhat invested in the topic of personal finance, so the idea was formed quickly.

In this iteration, I have rebuilt and expanded the “original” version of the program. The main changes were as follows: substituting Django for Flask-SQLAlchemy on the backend, adding unit tests and functional tests to the app and improving the “add recurring entries” feature (so that it actually works, however shabby it is).

## Summary of the features

With all this said, it is time to summarize the functions and elements of the “Easy Finance” application:

- A registration form on the “/register” route. In addition to entering their usernames and passwords, users are encouraged to enter the starting balance for their accounts. Filling this element of the form out is optional, though. If leaves it empty, the starting balance will be set to $10,000.00 by default.
- A login form on the “/login” route and a logout function. These two are virtually unchanged.
- A table on the main page that shows a summary of the user’s income, expenses, cash-flow (income minus expenses) and their cash. In hindsight, showing the total value of the user’s portfolio may be desirable.
- Another table on the main page that lists recurring income/expense elements the user has. It shows the following details: the ID, the type (income or expense), the title, the amount and the frequency (daily, weekly or monthly) of the entries. It also shows when these entries were added.
- A form on the main page that allows the user to add recurring entries. The users can select from three frequency options, two type options. They need to enter a title and an amount which is higher than zero.
- Edit recurring entries buttons on the main page: each recurring entry in the table is equipped with an edit button. Clicking on this button displays the edit recurring entry form and allows users to change the selected entry.
- Delete recurring entries buttons on the main page: each recurring entry in the table is also equipped with a delete button. Pressing one of these buttons removes the corresponding entry from the database and refreshes the page.
- Delete all button on the main page: the recurring entries table can be emptied by simply clicking on the delete all button, found in the bottom right corner of the table.
- A table on the “/stocks” route that contains information about the user’s portfolio. This table was also left unchanged.
- The “quote”, “buy” and “sell” forms were moved to the “/stocks” route. They are arranged in this order, below the table I mentioned earlier. The quote function now shows the price of the selected stock via a JavaScript alert. Other than this, the forms are the same as they were.
- A table on the “/history” route that lists all entries in the user’s financial history. It shows the ID, the type (balance, income, expense), the title and the amount for each entry, as well as the time when they were entered. The first entry is the starting balance of the user.

- Change starting balance button on the “/history” route: when the user clicks this button (situated above the financial history table), a form with a single element appears. This form allows the user to set a new starting balance for their account.
- A form on the “/history” route that allows users to add entries to their financial history. This form is almost identical to the “recurring entries” form. The only difference is the lack of the “frequency” element.
- Edit entry buttons in the financial statement table: statement entries are equipped with an edit button. Pressing one of these buttons displays the edit entry form and allows users to make changes to the selected form. The user’s balance is also adjusted when changes are made.
- Delete entry buttons in the financial statement table: entries in the table are also equipped with a delete button. Pressing a delete button deletes the entry that is associated with it. The page is then refreshed. The user’s balance is also adjusted whenever a statement entry is deleted.
- Delete all button on the history page: the financial statements table has a delete all button. Clicking this button will essentially reset the user’s account (aside from the recurring entries table): the stock portfolio is emptied, the transaction history is emptied and the financial statement table is emptied. The user’s balance is reset to 10,000.00 USD. The user has to confirm their decision by accepting a popup with a warning message before they can proceed.
- A table on the “/history” route summarizing the user’s transaction history. This table is the same as it was in the C$50 Finance app.
- Hide transaction history button (above the transaction history table): when the user clicks on this button the transaction history is hidden and the button is replaced with the show transaction history button.
- Show transaction history button: when the user clicks on this button the transaction history is shown to them and the button is replaced with the hide transaction history button. This button is not visible by default.
- A form on the “/change-pw” route that allows users to change their passwords. The users need to enter their old password and their new password twice. They are redirected to the main page upon success.

(Pictures of each element are included in the “Easy_Finance_pics” folder.) Python (Django), JavaScript, HTML and CSS were used in the making of this program. As I mentioned earlier, I have replaced the Flask-SQLAlchemy backend with Django. JQuery syntax was replaced by ES6 syntax in an earlier iteration. I have also added a total 137 tests to the program: 87 of them are unit tests and 50 of them are functional tests. I believe I have achieved a nice coverage of the code altogether.

## File layout

The program works with the usual Django architecture:

- models.py contains the database models (5 in total)
- views.py contains the server-side logic
- the static folder contains scripts.js and styles.css, the JavaScript and CSS files of the application, respectively
- the templates folder contains all html files
- unit tests can be found in the unit_tests folder, functional tests are in the functional_tests folder (all test files start with the prefix test, so they can be run without any issues)

One oddity can be found in the project's urls.py file. Aside from the url setup, it also contains the script that schedules background tasks (adding recurring entries, namely). Placing those scripts there allowed me to ensure that the schedule gets set up as soon as the server is started.

A file called helpers.py can be found in the application's folder. It contains a number of functions that are used in the program. Their intended purpose is to decrease complexity.

## Known issues

While some of the earlier issues have been fixed in this iteration, there is always room for improvement. There are two issues in particular that are worth mentioning (aside from a plethora of inconsistencies in logic and quirks in general).

- First, design-flaw. Somewhere down the line, I decided that the buying and selling of stock shares should also be recorded in the financial statement table. While this is not an issue in itself, it also affects the summary table on the main page. Since stock purchases are recorded as an expense, users who wish to amass shares of stocks in their portfolio will be greeted with a massive negative value of "cash flow" every time they log in. At the same time, selling of shares is classified as an income in the statements. This means they can be used to "cover up" poor finances, at least temporarily. A potential solution to this situation is to separate stock-related operations from the financial statements table. Instead, the value of the user's stock portfolio could be presented in the summary table.
- The second problem stems from the solution I found to running background tasks in Django. I'm using a module called django-background-tasks. It is lightweight (especially compared to solutions based on Celery) and it works well with Windows. These positive qualities come with some trade-offs, however. The main issue is that it isn't really an automatized solution. If you want to run tasks in the background, you need to open a second terminal window, go to the project folder and run "python manage.py process_tasks". Also, whenever a task is added to the schedule, it is executed once immediately. This may or may not be what the user intended to do, and this ambiguity is definitely an issue.