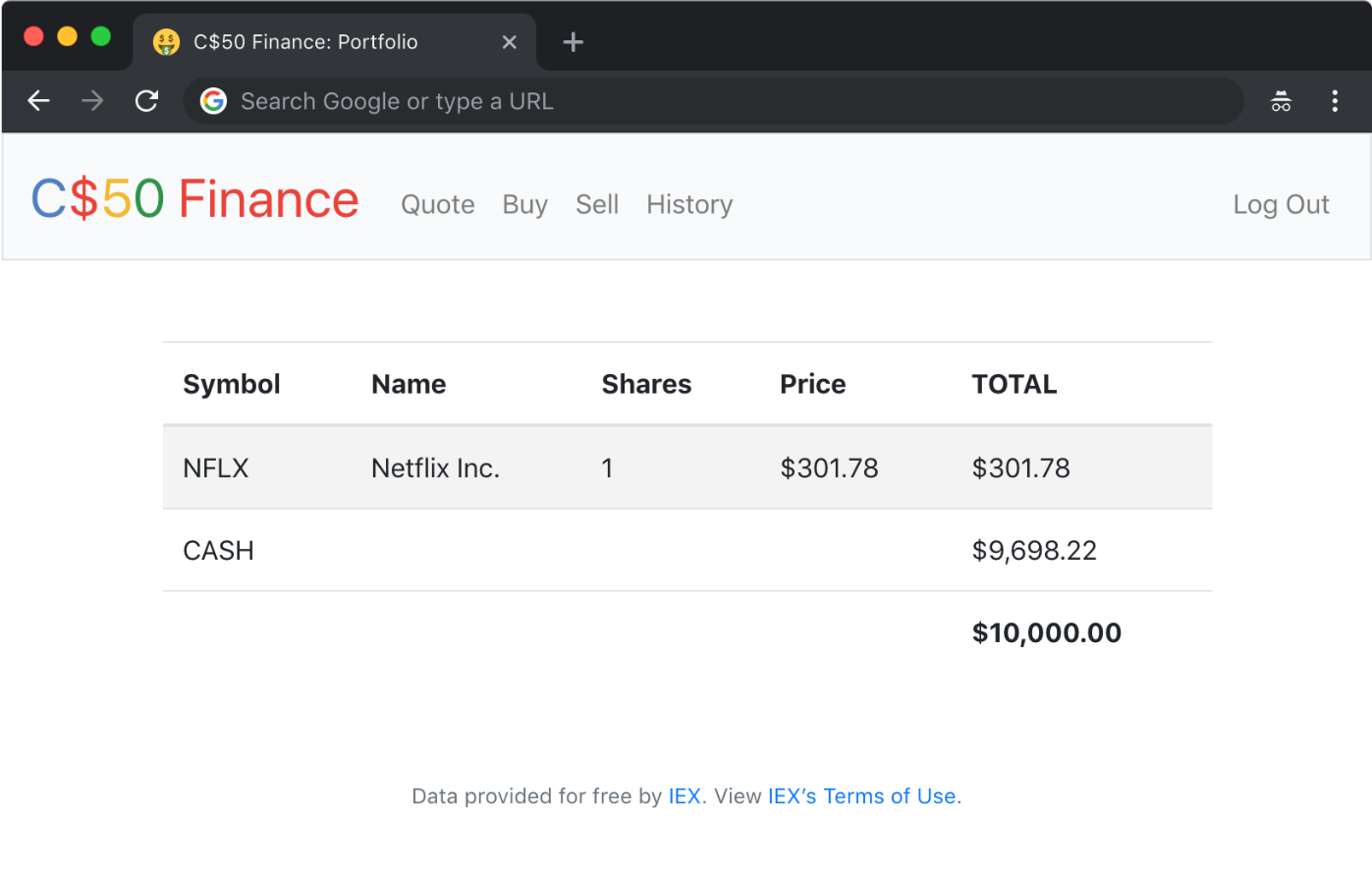
C$50 Finance

[tl;dr](https://docs.cs50.net/2019/x/psets/8/finance/finance.html#tldr)

Implement a website via which users can "buy" and "sell" stocks, a la the below.



[Background](https://docs.cs50.net/2019/x/psets/8/finance/finance.html#background)

If you’re not quite sure what it means to buy and sell stocks (i.e., shares of a company), head to <http://www.investopedia.com/university/stocks/> for a tutorial.

You’re about to implement C$50 Finance, a web app via which you can manage portfolios of stocks. Not only will this tool allow you to check real stocks' actual prices and portfolios' values, it will also let you buy (okay, "buy") and sell (okay, "sell") stocks by querying [IEX](https://iextrading.com/developer/) for stocks' prices.

Indeed, IEX lets you download stock quotes via their API (application programming interface) using URLs like <https://cloud-sse.iexapis.com/stable/stock/nflx/quote?token=API_KEY>. Notice how Netflix’s symbol (NFLX) is embedded in this URL; that’s how IEX knows whose data to return. That link won’t actually return any data because IEX requires you to use an API key (more about that in a bit), but if it did, you’d see a response in JSON (JavaScript Object Notation) format like this:

{

"symbol": "NFLX",

"companyName": "Netflix, Inc.",

"primaryExchange": "NASDAQ",

"calculationPrice": "close",

"open": 317.49,

"openTime": 1564752600327,

"close": 318.83,

"closeTime": 1564776000616,

"high": 319.41,

"low": 311.8,

"latestPrice": 318.83,

"latestSource": "Close",

"latestTime": "August 2, 2019",

"latestUpdate": 1564776000616,

"latestVolume": 6232279,

"iexRealtimePrice": null,

"iexRealtimeSize": null,

"iexLastUpdated": null,

"delayedPrice": 318.83,

"delayedPriceTime": 1564776000616,

"extendedPrice": 319.37,

"extendedChange": 0.54,

"extendedChangePercent": 0.00169,

"extendedPriceTime": 1564876784244,

"previousClose": 319.5,

"previousVolume": 6563156,

"change": -0.67,

"changePercent": -0.0021,

"volume": 6232279,

"iexMarketPercent": null,

"iexVolume": null,

"avgTotalVolume": 7998833,

"iexBidPrice": null,

"iexBidSize": null,

"iexAskPrice": null,

"iexAskSize": null,

"marketCap": 139594933050,

"peRatio": 120.77,

"week52High": 386.79,

"week52Low": 231.23,

"ytdChange": 0.18907500000000002,

"lastTradeTime": 1564776000616

}

Notice how, between the curly braces, there’s a comma-separated list of key-value pairs, with a colon separating each key from its value.

Let’s turn our attention now to this problem’s distribution code!

[Distribution](https://docs.cs50.net/2019/x/psets/8/finance/finance.html#distribution)

[Downloading](https://docs.cs50.net/2019/x/psets/8/finance/finance.html#downloading)

$ wget <https://cdn.cs50.net/2019/x/psets/8/finance.zip>

$ unzip finance.zip

$ rm finance.zip

$ cd finance

$ ls

application.py helpers.py static/

finance.db requirements.txt templates/

[Configuring](https://docs.cs50.net/2019/x/psets/8/finance/finance.html#configuring)

Before getting started on this assignment, we’ll need to register for an API key in order to be able to query IEX’s data. To do so, follow these steps:

1. Visit [iexcloud.io/cloud-login#/register/](https://iexcloud.io/cloud-login#/register/).
2. Enter your email address and a password, and click "Create account".
3. Once you’ve confirmed your account via a confirmation email, sign in to [iexcloud.io](https://iexcloud.io/).
4. Click **API Tokens**.
5. Copy the key that appears under the **Token** column (it should begin with pk\_).
6. In a terminal window within CS50 IDE, execute:

export API\_KEY=value

where value is that (pasted) value, without any space immediately before or after the =. You also may wish to paste that value in a text document somewhere, in case you need it again later.

[Running](https://docs.cs50.net/2019/x/psets/8/finance/finance.html#running)

1. Start Flask’s built-in web server (within finance/):

flask run

Visit the URL outputted by flask to see the distribution code in action. You won’t be able to log in or register, though, just yet!

1. Via CS50’s file browser, double-click **finance.db** in order to open it with phpLiteAdmin. Notice how finance.db comes with a table called users. Take a look at its structure (i.e., schema). Notice how, by default, new users will receive $10,000 in cash. But there aren’t (yet!) any users (i.e., rows) therein to browse.

Here on out, if you’d prefer a command line, you’re welcome to use sqlite3 instead of phpLiteAdmin.

[Understanding](https://docs.cs50.net/2019/x/psets/8/finance/finance.html#understanding)

[application.py](https://docs.cs50.net/2019/x/psets/8/finance/finance.html#application-py)

Open up application.py. Atop the file are a bunch of imports, among them CS50’s SQL module and a few helper functions. More on those soon.

After configuring [Flask](http://flask.pocoo.org/), notice how this file disables caching of responses (provided you’re in debugging mode, which you are by default on CS50 IDE), lest you make a change to some file but your browser not notice. Notice next how it configures [Jinja](http://jinja.pocoo.org/) with a custom "filter," usd, a function (defined in helpers.py) that will make it easier to format values as US dollars (USD). It then further configures Flask to store [sessions](http://flask.pocoo.org/docs/1.0/quickstart/#sessions) on the local filesystem (i.e., disk) as opposed to storing them inside of (digitally signed) cookies, which is Flask’s default. The file then configures CS50’s SQL module to use finance.db, a SQLite database whose contents we’ll soon see!

Thereafter are a whole bunch of routes, only two of which are fully implemented: login and logout. Read through the implementation of login first. Notice how it uses db.execute (from CS50’s library) to query finance.db. And notice how it uses check\_password\_hash to compare hashes of users' passwords. Finally, notice how login "remembers" that a user is logged in by storing his or her user\_id, an INTEGER, in session. That way, any of this file’s routes can check which user, if any, is logged in. Meanwhile, notice how logout simply clears session, effectively logging a user out.

Notice how most routes are "decorated" with @login\_required (a function defined in helpers.py too). That decorator ensures that, if a user tries to visit any of those routes, he or she will first be redirected to login so as to log in.

Notice too how most routes support GET and POST. Even so, most of them (for now!) simply return an "apology," since they’re not yet implemented.

[helpers.py](https://docs.cs50.net/2019/x/psets/8/finance/finance.html#helpers-py)

Next take a look at helpers.py. Ah, there’s the implementation of apology. Notice how it ultimately renders a template, apology.html. It also happens to define within itself another function, escape, that it simply uses to replace special characters in apologies. By defining escape inside of apology, we’ve scoped the former to the latter alone; no other functions will be able (or need) to call it.

Next in the file is login\_required. No worries if this one’s a bit cryptic, but if you’ve ever wondered how a function can return another function, here’s an example!

Thereafter is lookup, a function that, given a symbol (e.g., NFLX), returns a stock quote for a company in the form of a dict with three keys: name, whose value is a str, the name of the company; price, whose value is a float; and symbol, whose value is a str, a canonicalized (uppercase) version of a stock’s symbol, irrespective of how that symbol was capitalized when passed into lookup.

Last in the file is usd, a short function that simply formats a float as USD (e.g., 1234.56 is formatted as $1,234.56).

[requirements.txt](https://docs.cs50.net/2019/x/psets/8/finance/finance.html#requirements-txt)

Next take a quick look at requirements.txt. That file simply prescribes the packages on which this app will depend.

[static/](https://docs.cs50.net/2019/x/psets/8/finance/finance.html#static)

Glance too at static/, inside of which is styles.css. That’s where some initial CSS lives. You’re welcome to alter it as you see fit.

[templates/](https://docs.cs50.net/2019/x/psets/8/finance/finance.html#templates)

Now look in templates/. In login.html is, essentially, just an HTML form, stylized with [Bootstrap](http://getbootstrap.com/). In apology.html, meanwhile, is a template for an apology. Recall that apology in helpers.py took two arguments: message, which was passed to render\_template as the value of bottom, and, optionally, code, which was passed to render\_template as the value of top. Notice in apology.html how those values are ultimately used! And [here’s why](https://github.com/jacebrowning/memegen). 0:-)

Last up is layout.html. It’s a bit bigger than usual, but that’s mostly because it comes with a fancy, mobile-friendly "navbar" (navigation bar), also based on Bootstrap. Notice how it defines a block, main, inside of which templates (including apology.html and login.html) shall go. It also includes support for Flask’s [message flashing](http://flask.pocoo.org/docs/1.0/patterns/flashing/) so that you can relay messages from one route to another for the user to see.

[Specification](https://docs.cs50.net/2019/x/psets/8/finance/finance.html#specification)

[register](https://docs.cs50.net/2019/x/psets/8/finance/finance.html#register)

Complete the implementation of register in such a way that it allows a user to register for an account via a form.

* Require that a user input a username, implemented as a text field whose name is username. Render an apology if the user’s input is blank or the username already exists.
* Require that a user input a password, implemented as a text field whose name is password, and then that same password again, implemented as a text field whose name is confirmation. Render an apology if either input is blank or the passwords do not match.
* Submit the user’s input via POST to /register.
* INSERT the new user into users, storing a hash of the user’s password, not the password itself. Hash the user’s password with [generate\_password\_hash](http://werkzeug.pocoo.org/docs/0.14/utils/#werkzeug.security.generate_password_hash).
* Odds are you’ll want to create a new template (e.g., register.html) that’s quite similar to login.html.

Once you’ve implemented register correctly, you should be able to register for an account and log in (since login and logoutalready work)! And you should be able to see your rows via phpLiteAdmin or sqlite3.

[quote](https://docs.cs50.net/2019/x/psets/8/finance/finance.html#quote)

Complete the implementation of quote in such a way that it allows a user to look up a stock’s current price.

* Require that a user input a stock’s symbol, implemented as a text field whose name is symbol.
* Submit the user’s input via POST to /quote.
* Odds are you’ll want to create two new templates (e.g., quote.html and quoted.html). When a user visits /quote via GET, render one of those templates, inside of which should be an HTML form that submits to /quote via POST. In response to a POST, quotecan render that second template, embedding within it one or more values from lookup.

[buy](https://docs.cs50.net/2019/x/psets/8/finance/finance.html#buy)

Complete the implementation of buy in such a way that it enables a user to buy stocks.

* Require that a user input a stock’s symbol, implemented as a text field whose name is symbol. Render an apology if the input is blank or the symbol does not exist (as per the return value of lookup).
* Require that a user input a number of shares, implemented as a text field whose name is shares. Render an apology if the input is not a positive integer.
* Submit the user’s input via POST to /buy.
* Odds are you’ll want to call lookup to look up a stock’s current price.
* Odds are you’ll want to SELECT how much cash the user currently has in users.
* Add one or more new tables to finance.db via which to keep track of the purchase. Store enough information so that you know who bought what at what price and when.
  + Use appropriate SQLite types.
  + Define UNIQUE indexes on any fields that should be unique.
  + Define (non-UNIQUE) indexes on any fields via which you will search (as via SELECT with WHERE).
* Render an apology, without completing a purchase, if the user cannot afford the number of shares at the current price.
* You don’t need to worry about race conditions (or use transactions).

Once you’ve implemented buy correctly, you should be able to see users' purchases in your new table(s) via phpLiteAdmin or sqlite3.

[index](https://docs.cs50.net/2019/x/psets/8/finance/finance.html#index)

Complete the implementation of index in such a way that it displays an HTML table summarizing, for the user currently logged in, which stocks the user owns, the numbers of shares owned, the current price of each stock, and the total value of each holding (i.e., shares times price). Also display the user’s current cash balance along with a grand total (i.e., stocks' total value plus cash).

* Odds are you’ll want to execute multiple SELECTs. Depending on how you implement your table(s), you might find [GROUP BY](https://www.google.com/search?q=SQLite+GROUP+BY), [HAVING](https://www.google.com/search?q=SQLite+HAVING), [SUM](https://www.google.com/search?q=SQLite+SUM), and/or [WHERE](https://www.google.com/search?q=SQLite+WHERE) of interest.
* Odds are you’ll want to call lookup for each stock.

[sell](https://docs.cs50.net/2019/x/psets/8/finance/finance.html#sell)

Complete the implementation of sell in such a way that it enables a user to sell shares of a stock (that he or she owns).

* Require that a user input a stock’s symbol, implemented as a select menu whose name is symbol. Render an apology if the user fails to select a stock or if (somehow, once submitted) the user does not own any shares of that stock.
* Require that a user input a number of shares, implemented as a text field whose name is shares. Render an apology if the input is not a positive integer or if the user does not own that many shares of the stock.
* Submit the user’s input via POST to /sell.
* You don’t need to worry about race conditions (or use transactions).

[history](https://docs.cs50.net/2019/x/psets/8/finance/finance.html#history)

Complete the implementation of history in such a way that it displays an HTML table summarizing all of a user’s transactions ever, listing row by row each and every buy and every sell.

* For each row, make clear whether a stock was bought or sold and include the stock’s symbol, the (purchase or sale) price, the number of shares bought or sold, and the date and time at which the transaction occurred.
* You might need to alter the table you created for buy or supplement it with an additional table. Try to minimize redundancies.

[check](https://docs.cs50.net/2019/x/psets/8/finance/finance.html#check)

Complete the implementation of check in such a way that it checks whether a username is available.

* The route should accept, via GET, an HTTP parameter called username.
* If the value of username is of length at least 1 and does not already belong to a user in the database, the route should return, in JSON format, true, signifying that the username is (as of that moment) available. Else it should return, in JSON format, false. Recall that jsonify in Flask can return a value in JSON format.

Finally, enhance your template for register with some JavaScript in such a way that it prevents submission of your registration form if the inputted username is already taken, letting the user know, as via alert or via a Bootstrap [alert](http://getbootstrap.com/docs/4.1/components/alerts/) or [validation](http://getbootstrap.com/docs/4.1/components/forms/#validation).

* Use Ajax (e.g., jQuery’s [$.get](https://api.jquery.com/jquery.get/) method) to query /check to check whether the inputted username is available.
* Recall that you can prevent submission of a form with [preventDefault](https://www.w3schools.com/jsref/event_preventdefault.asp), and you can programmatically induce submission of a form (once validated by you) with [submit](https://www.w3schools.com/jsref/met_form_submit.asp).

[personal touch](https://docs.cs50.net/2019/x/psets/8/finance/finance.html#personal-touch)

Implement at least one personal touch of your choice:

* Allow users to change their passwords.
* Allow users to add additional cash to their account.
* Allow users to buy more shares or sell shares of stocks they already own via index itself, without having to type stocks' symbols manually.
* Require users' passwords to have some number of letters, numbers, and/or symbols.
* Implement some other feature of comparable scope.

[Walkthroughs](https://docs.cs50.net/2019/x/psets/8/finance/finance.html#walkthroughs)

**Instead of**pwd\_context.encrypt**, which Zamyla mentions, be sure to use**generate\_password\_hash**instead.**

**Note that Zamyla does not discuss**/check**, but you should still implement it!**

<https://youtu.be/sy1o1yLob5U?list=PLhQjrBD2T380UP0XpPRhaUNxWbhftko_b>

[Testing](https://docs.cs50.net/2019/x/psets/8/finance/finance.html#testing)

Be sure to test your web app manually too, as by

* inputting alpabetical strings into forms when only numbers are expected,
* inputting zero or negative numbers into forms when only positive numbers are expected,
* inputting floating-point values into forms when only integers are expected,
* trying to spend more cash than a user has,
* trying to sell more shares than a user has,
* inputting an invalid stock symbol, and
* including potentially dangerous characters like ' and ; in SQL queries.

[Correctness](https://docs.cs50.net/2019/x/psets/8/finance/finance.html#correctness)

check50 cs50/problems/2019/x/finance

[Style](https://docs.cs50.net/2019/x/psets/8/finance/finance.html#style)

style50 application.py

[Staff’s Solution](https://docs.cs50.net/2019/x/psets/8/finance/finance.html#staffs-solution)

You’re welcome to stylize your own app differently, but here’s what the staff’s solution looks like!

<http://finance.cs50.net/>

Feel free to register for an account and play around. Do **not** use a password that you use on other sites.

It is **reasonable** to look at the staff’s HTML and CSS.

[Hints](https://docs.cs50.net/2019/x/psets/8/finance/finance.html#hints)

* Within cs50.SQL is an execute method whose first argument should be a str of SQL. If that str contains named parameters to which values should be bound, those values can be provided as additional named parameters to execute. See the implementation of login for one such example. The return value of execute is as follows:
  + If str is a SELECT, then execute returns a list of zero or more dict objects, inside of which are keys and values representing a table’s fields and cells, respectively.
  + If str is an INSERT, and the table into which data was inserted contains an autoincrementing PRIMARY KEY, then executereturns the value of the newly inserted row’s primary key.
  + If str is a DELETE or an UPDATE, then execute returns the number of rows deleted or updated by str.

If an INSERT or UPDATE would violate some constraint (e.g., a UNIQUE index), then execute returns None. In cases of error, execute raises a RuntimeError.

* Recall that cs50.SQL will log to your terminal window any queries that you execute via execute (so that you can confirm whether they’re as intended).
* Be sure to use named bind parameters (i.e., a [paramstyle](https://www.python.org/dev/peps/pep-0249/#paramstyle) of named) when calling CS50’s execute method, a la WHERE name=:name. Do **not** use f-strings, [format](https://docs.python.org/3.6/library/functions.html#format), or + (i.e., concatenation), lest you risk a SQL injection attack.
* If (and only if) already comfortable with SQL, you’re welcome to use [SQLAlchemy Core](http://docs.sqlalchemy.org/en/latest/index.html) or [Flask-SQLAlchemy](http://flask-sqlalchemy.pocoo.org/) (i.e., [SQLAlchemy ORM](http://docs.sqlalchemy.org/en/latest/index.html)) instead of cs50.SQL.
* You’re welcome to add additional static files to static/.
* Odds are you’ll want to consult [Jinja’s documentation](http://jinja.pocoo.org/docs/dev/) when implementing your templates.
* It is **reasonable** to ask others to try out (and try to trigger errors in) your site. Via **Share** in CS50 IDE’s top-right corner can you share your **Application** by making it **Public**. Take care not to share your **Editor**, which would provide access to your Python code and SQLite database.
* You’re welcome to alter the aesthetics of the sites, as via
  + <https://bootswatch.com/>,
  + <https://getbootstrap.com/docs/4.1/content/>,
  + <https://getbootstrap.com/docs/4.1/components/>, and/or
  + <https://memegen.link/>.

[FAQs](https://docs.cs50.net/2019/x/psets/8/finance/finance.html#faqs)

[ImportError: No module named 'application'](https://docs.cs50.net/2019/x/psets/8/finance/finance.html#importerror-no-module-named-application)

By default, flask looks for a file called application.py in your current working directory (because we’ve configured the value of FLASK\_APP, an environment variable, to be application.py). If seeing this error, odds are you’ve run flask in the wrong directory!

[OSError: [Errno 98] Address already in use](https://docs.cs50.net/2019/x/psets/8/finance/finance.html#oserror-errno-98-address-already-in-use)

If, upon running flask, you see this error, odds are you (still) have flask running in another tab. Be sure to kill that other process, as with ctrl-c, before starting flask again. If you haven’t any such other tab, execute fuser -k 8080/tcp to kill any processes that are (still) listening on TCP port 8080.

[check50 ran into an error while running checks!](https://docs.cs50.net/2019/x/psets/8/finance/finance.html#check50-ran-into-an-error-while-running-checks)

If, upon running check50, you see this error, odds are you have a bug in your code somewhere! Open up the CS50.me produced by check50 to see a detailed traceback to help you debug!

[check50 is taking longer than normal!](https://docs.cs50.net/2019/x/psets/8/finance/finance.html#check50-is-taking-longer-than-normal)

If you see this, make sure to remove or comment out any print statements from application.py and try check50 again.

[How to Submit](https://docs.cs50.net/2019/x/psets/8/finance/finance.html#how-to-submit)

Execute the below from within your ~/workspace/pset8/finance directory, logging in with your GitHub username and password when prompted. For security, you’ll see asterisks (\*) instead of the actual characters in your password.

submit50 cs50/problems/2019/x/finance