Titan Trading

Technical Documentation

Group 7

Steven Adler, Kristene Aguinaldo, Timothy Liu, Nicholas Lurski, Avanish Mishra, Safa Shaikh, Brooks Tawil, Kristian Wu and David Zhang

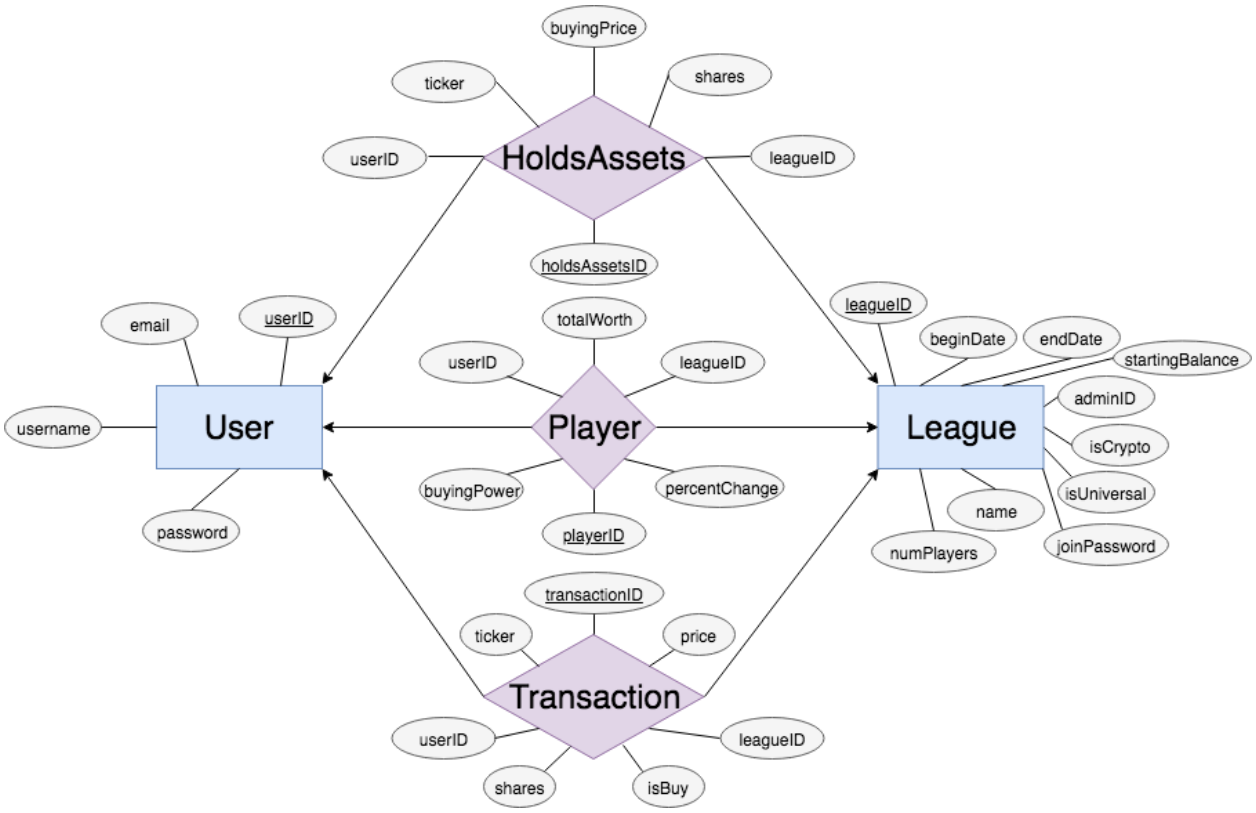
Front End Overview

We implemented the front end of our website using Bootstrap, Django, HTML, and CSS. HTML, or Hyptertext Markup Language, is a system for tagging text files on web pages. While HTML is used to actually create the content of the page, CSS, or Cascade Styling Sheets, is what is actually responsible for the design/styling of the webpage. HTML gives our website functionality while CSS makes our website aesthetically pleasing. For our HTML and CSS files, we used Bootstrap, an open-source front-end library for designing websites. Bootstrap contains many HTML and CSS templates which allowed us to really customize each page's functionality and design. In addition toe HTML, CSS and Bootstrap, Django is the other framework that was a key to our front end implementation. Django is an open-source web framework written in Python. It follows model-view-template (MVT) architectural pattern. MVT allows for a URL request to go to view, which then calls the appropriate model. The model will then perform functions to prepare output data that will then be passed to the template where the output is displayed.

Back End Overview

We implemented the back end of our website using PostgreSQL and hosted it on ElephantSQL. PostgreSQL is an open source object-relational management system. In comparison to other database management systems, such as MySQL, PostgreSQL is advantageous because it supports various data types, such as multidimensional arrays, JSON data, and network addresses. Our website uses the Alpha Vantage Finance API, which returns JSON data. Since PostgreSQL supports JSON data, using it as opposed to MySQL will make our website more streamlined. In addition, it gives us the flexibility to add additional functionality to our site. ElephantSQL, hosted by Amazon Web Services, hosts databases in the cloud, has an easy-to-use interface, and is compatible with PostgreSQL. For these reasons, we chose to use ElephantSQL to host our database for this project.

On the next page is our entity-relation diagram for the backend of our website. We have two entities: User and League. Between these two entities, we have three relationships: HoldAssets, Player, and Transaction. Through these relationships, we can see how User and League share information and how information is stored in the database.



Python files:

Path: "/SE18/stockgame2/home/"

admin.py

This python file imports models from models.py. It instantiates an AdminSite and reads metadata from each of the models.

apps.py

Not yet configured.

forms.py

This python file creates forms. It tells Django the type of form it is and the fields that should end up in the form.

methods.py

This python file was used for testing methods.

models.py

This python file contains the model maps. Each model contains essential fields and behaviors of the type of data that is being stored.

tests.py

This python file is used to test Django's test case module.

urls.py

This python file is used to store URL. When a url is called, Django runs through the URLs and stops at the first one that matches. It then imports and calls a given function.

views.py

This python file is used to store python functions that take a web request and return a web response. It can contain anything that is necessary to return that respones.

HTML Pages:

Path: “/SE18/stockgame2/home/templates/registration”

login.html

The focus of this page is a form created in HTML. The fill-outs are created using the input tag and setting the type attribute to “text”. The form calls a built in Django method, “{% url 'login' %}. When the user clicks the “Sign in!” button, the method checks the database for the user. When the user is authenticated, the user is able to see the navbar at the top of the page that redirects with the use of the href tag to pages including home, dashboard, about titan dropdown, and logout dropdown. If the user is not authenticated, the page is refreshed with error messages.

Path: “/SE18/stockgame2/home/templates/”

aboutus.html

The focus of this page is a container created in HTML. All pictures were set to 200px by 200px so that the images would align properly and look neat. A paragraph tag was used to allocate enough space for each founder to write their own description.

anonuser.html

The focus of this page is a container created in HTML. The errors message was emphasized using a “strong” tag to make it stand out. In addition, a button was created to redirect the user to signup.html when clicked through the href attribute. Other pages are redirected to this page when a user is not logged in (a guest), and tried to access unauthenticated pages. Authentication is done through Django’s user authentication.

buypage.html

The focus of this page is a form created in HTML. The fill-outs are created using the input tag and setting the type attribute to “text”. The checkbox is made by setting the type attribute to “checkbox”. The form calls a method, “/submitBuy”. This method creates a tuple in the

“home_transaction” table of the database that is auto populated by the user inputs after the user clicks the “confirm purchase” button. Functionality to create a tuple in the “home_asset” table of the database will be added. The cart shows two values, a stock ticker value and a total value. The stock ticker value is automatically filled in with the value of the stock ticker grabbed from either a stock or crypto API. The total is then filled in by calling a calculator method that multiplies the stock ticker value by the number of shares entered. If an incorrect input is entered into the form, the page will refresh will error messages. The authentication and errors are done with built in Django methods.

createleague.html

The focus of this page is a form created in HTML. The form calls a method, “/newLeague”. This method creates a tuple in the “home_league” table of the database that is auto populated by the user inputs after the user clicks the “create league” button. If an incorrect input is entered into the form, the page will refresh will error messages.

dashboard.html

The focus of this page is a container created in HTML. The container can contain different values based off the user. The top-left section of the container will always contain the Universal League. The information inside describing the league is then auto populated with information queried from the database. Similar information is populated for each league that the user is a part of. The container also contains two buttons, “Create a League” and “Join a League”, that will redirect the user to a createleague.html and joinleague.html through the href attribute. The dashboard page is specific to the leagues that the user has joined.

faq.html

The focus of this page is a container created in HTML. This page contains multiple sub containers with collapsible headers. This is done mainly through the use of the data-toggle attribute. By setting the data-toggle attribute to collapse, the collapse of the tabs are able to show the answers to the question shown.

greet.html

The focus of this page is a container created in HTML. The name of the company was emphasized using a “strong” tag to make it stand out. In

addition, a button was created to redirect the user to the signup.html when clicked through the href attribute.

home.html

The focus of this page is a container created in HTML. The page centers on three containers and a button. The first container is a widget taken from https://www.tradingview.com/symbols/NASDAQ-AAPL/ and formatted to fit the space. Next, a leaderboard created using an ordered list in HTML. The names and assets are manually entered now, but will eventually be queried from the database. Finally, at the bottom, there is a client-side script that takes and displays news sources relevant to the stock market. The “play now” button in the middle of the page redirects the user to dashboard.html through the href attribute.

individualleague.html

The focus of this page is a container created in HTML. The page centers on four pieces of information. First, a leaderboard created using an ordered list in HTML. Similar to home.html, the leaderboard will be able to get the information by querying from the database. Next, all the information describing and pertaining to the individual league will be filled by querying into the database as well. Finally, there are two buttons linking to buypage.html and sellform.html through the href attribute.

joinleague.html

The focus of this page is a form created in HTML. After the league name and password are entered and the user clicks on the button, the form will call a method that queries the database and authenticates the information. If it is authenticated, the user dashboard will be updated with the new league information. If not, the page will refresh with display error messages. The authentication and errors are done using built in methods from the Django framework.

mission.html

The focus of this page is a container created in HTML. The blocks of text are spaced out using the paragraph tag. In addition, an unordered list is used to highlight individual mission goals.

profile.html

The focus of this page are two containers created in HTML. The first container consists of the top section. The picture is put in using an img tag. At the moment, it is manually entered, but functionality for changing the picture based on the user will be implemented. The user name underneath the picture is put in through a header tag by requesting the username using a built in Django method. The second container consists of the information pertaining to the user's profile organized using header and paragraph tags. Finally, the table at the bottom was created using a "list-group" class from the Bootstrap library.

sellform.html

The focus of this page is a form created in HTML. The fill-outs are created using the input tag and setting the type attribute to "text". The checkbox is made by setting the type attribute to "checkbox". The form calls a method, "/submitSell. This method creates a tuple in the "home_transaction" table of the database that is auto populated by the user inputs after the user clicks the "confirm purchase" button. Functionality to remove a tuple in the "home_asset" table of the database will be added. The cart shows two values, a stock ticker value and a total value. The stock ticker value is automatically filled in with the value of the stock ticker grabbed from either a stock or crypto API. The total is then filled in by calling a calculator method that multiplies the stock ticker value by the number of shares entered. If an incorrect input is entered into the form, the page will refresh with error messages. The authentication is done by a built in Django method.

settings.html

The focus of this page will be a container created in HTML. The page will eventually contain information organized by header, paragraph tags, and ordered lists.

signup.html

The focus of this page is a form created in HTML. It was formatted using bootstrap grid classes. This divides the page into 12 columns and lets format it based off an offset. The form calls the method "/submitSignup". This method creates a tuple in "home_user" table in the database that is auto populated by the form inputs. The fill-outs are created using the input tag and setting the type attribute to "text". The "Sign-Up!" button redirects

to home on success and back to the signup page on failure. The authentication is done by a built in Django method.

universalleague.html

The focus of this page is a container created in HTML. This page is similar to individualleague.html. However, the information filled out from querying in the database will be grabbed from all users instead of those pertaining to a particular league.

user.html

This page has not yet been put together yet.