

Deliverable 4 - Design

Project C.A.T. - Capital Activity Tracker

Group: “The Stockings”

Clayton Williams, Kaitlyn Lee, Brandon Horner, Savannah Fischer

Course: CS 386 Software Engineering, Spring 2018

Instructor: Marco Gerosa

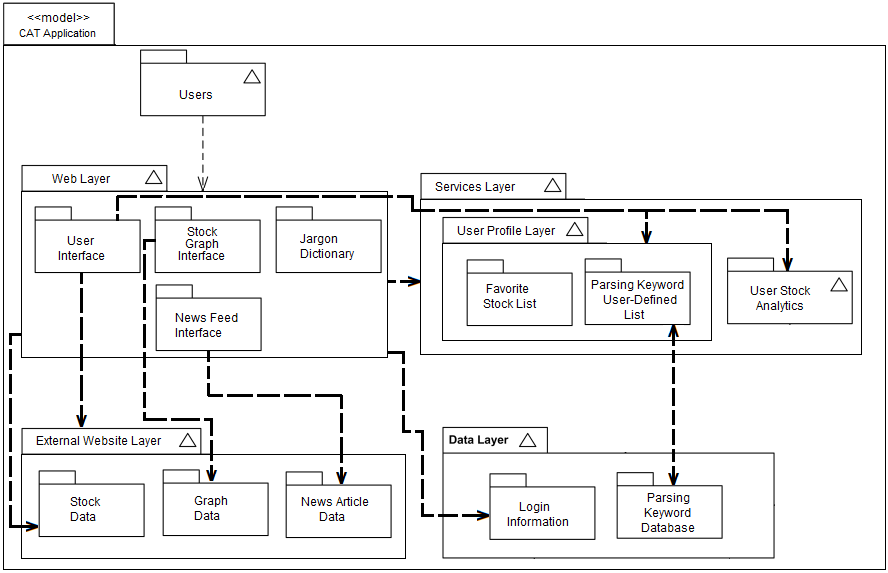
GitHub Link: <https://github.com/kdl222/CAT>

1. Description

CAT is a stock market analysis application that brings data visualizations and information-gathering together in an easily-accessible and affordable program, making it perfect for new and experienced stock traders alike. Equipped with live transaction suggestions and in-depth company evaluations, CAT allows traders to make more informed financial decisions than could be reliably accomplished alone.

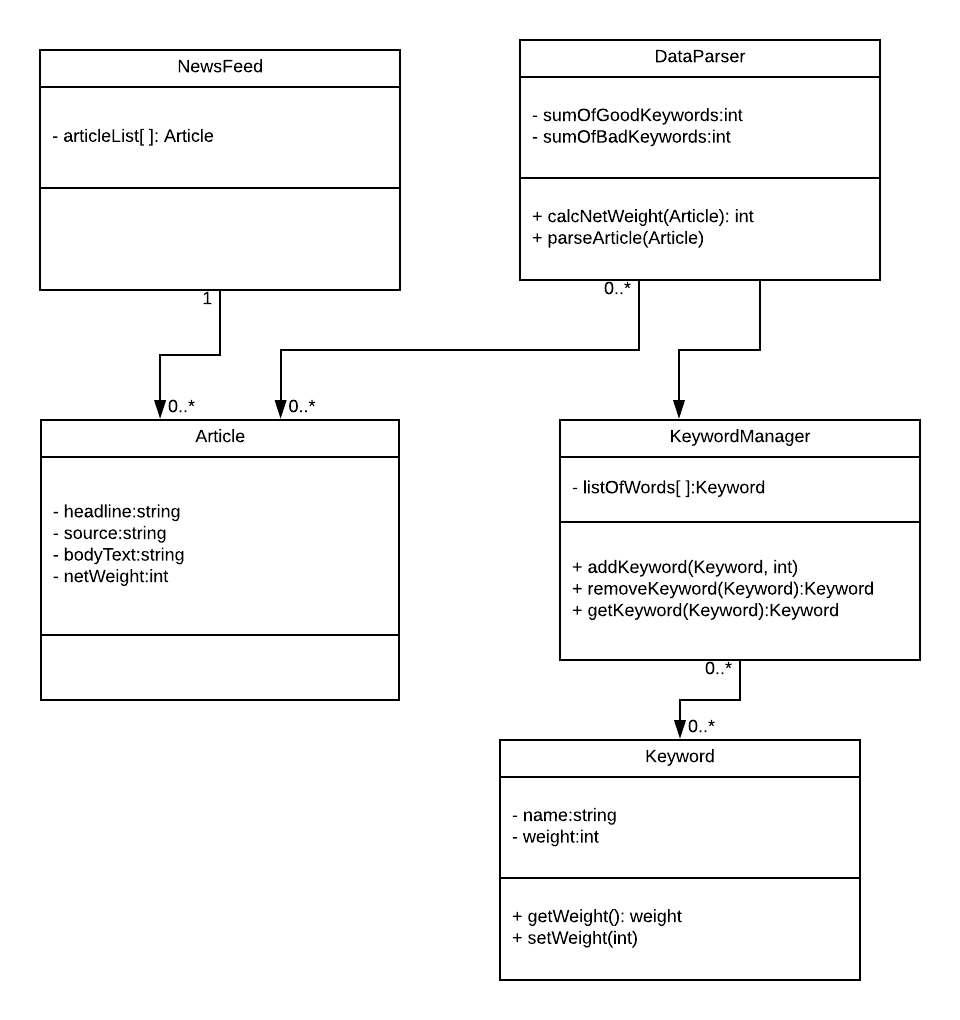
Users can create a secure and private account on the CAT web application, where they can easily view and remove any stocks, and their accompanied real-time activity graphs, that they have selected to watch. The ranges of time covered by stock activity graphs can be adjusted to the user’s preference. Users can view a news feed containing many articles. Users can add and remove keywords in a keyword manager, set the weight of each word, and watch as the manager parses through the news feed to show the net weight of each article.

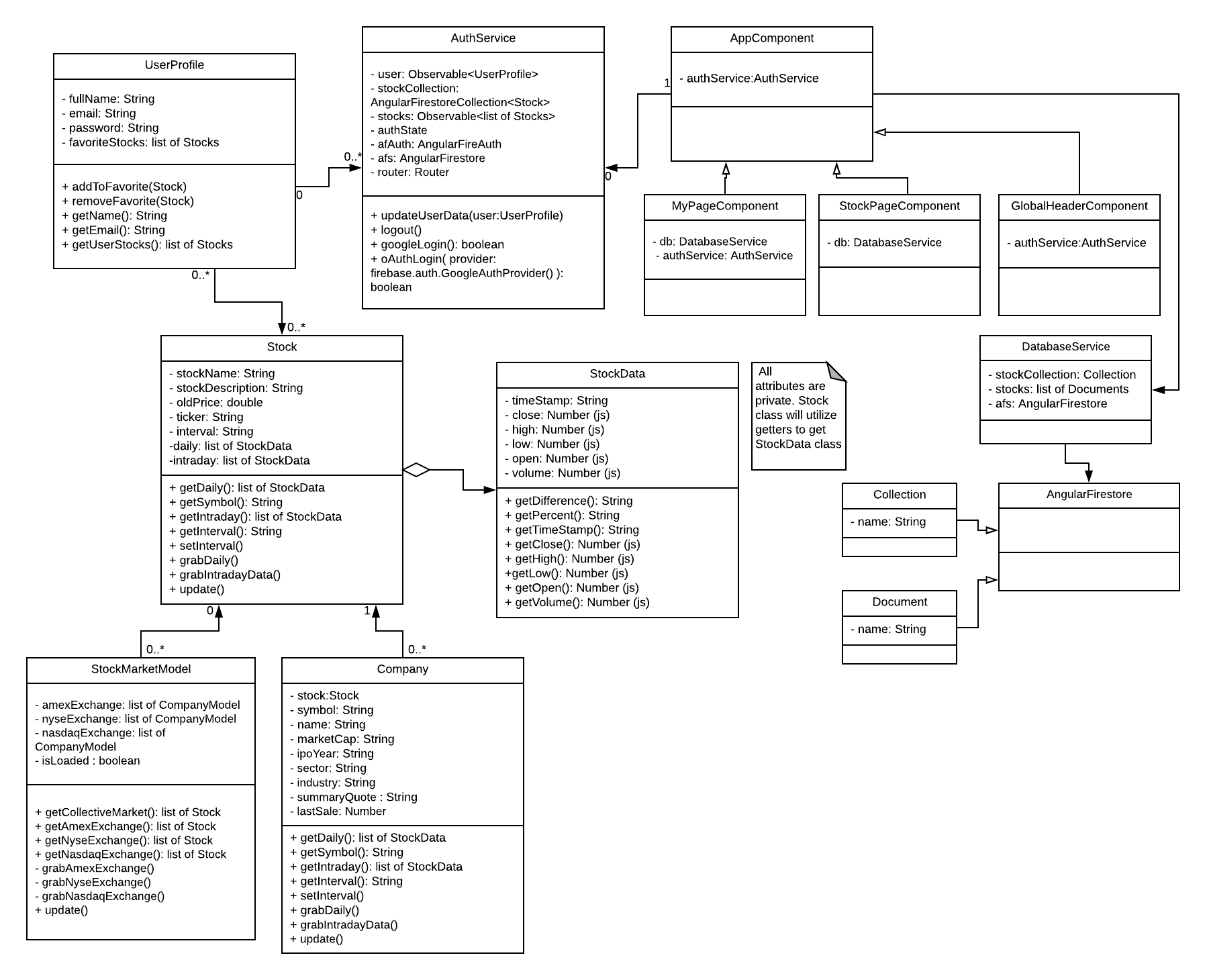
1. Architecture



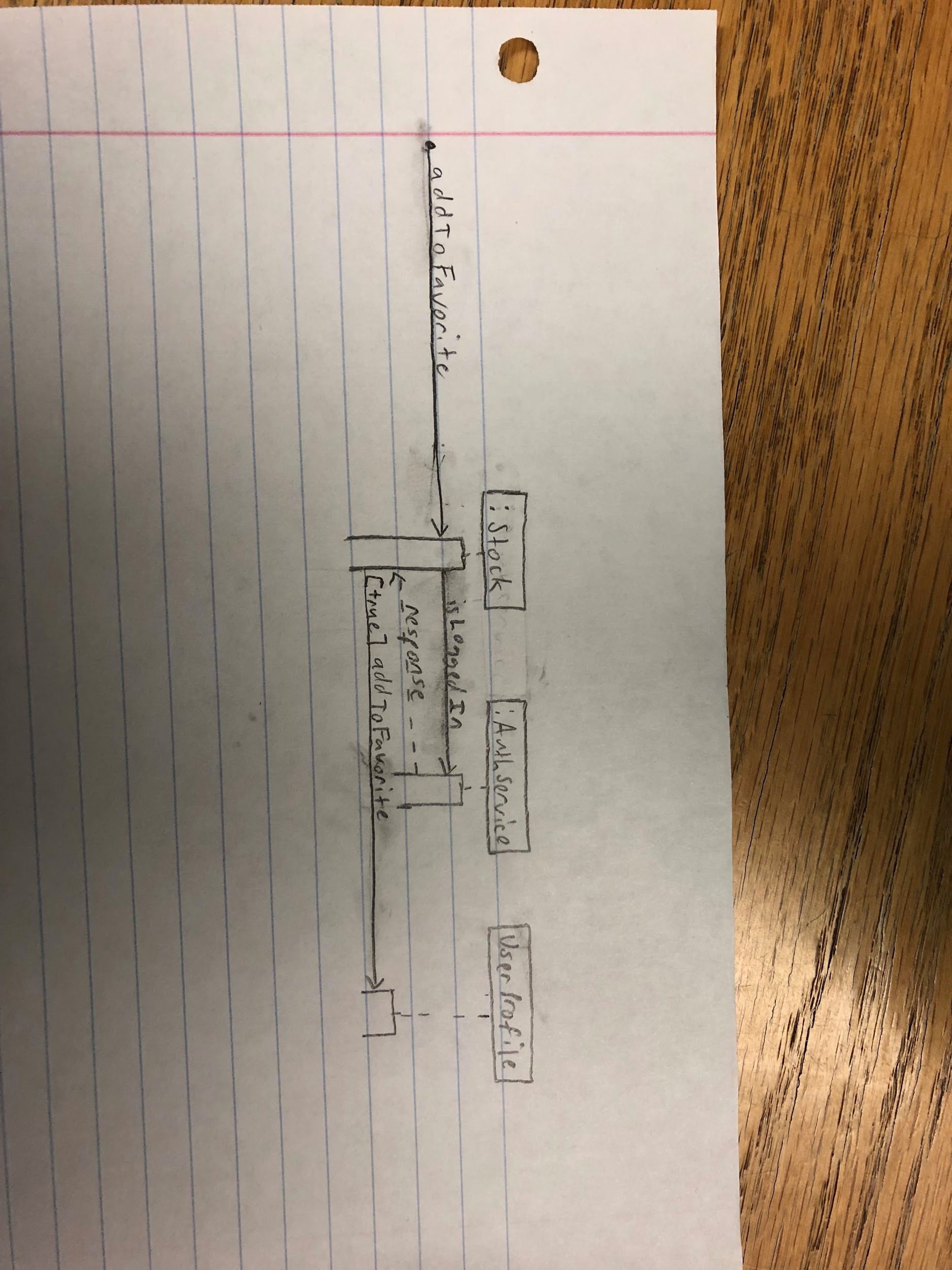
1. Class Diagram

(We split it into two different sections because it is too big to fit in one piece)





1. Sequence Diagram



Use Case Description:

**Name:** Favorite/Unfavorite Stock Trend Visualizations

**Brief Description:** This use case describes how the CAT user uses the CAT web application to save a specific stock analysis graph to their home page to easily re-access later. Or how to dismiss a graph from the same list.

**Flow of Events:**

1. The use case begins when the CAT user clicks on a specific stock graph’s “favorite” function.
2. Use case: Validate User is Signed In is performed
3. The graph is saved to a user-sensitive home-page list
4. The use case ends successfully

*Alternative Flows*

* User not signed in:

If step 2 use case does not complete successfully, the use case ends with a failure condition (web app may prompt user to sign in)

**Key Scenarios:**

* No internet connection at time of “favorite”

**Pre-conditions:**

* Active internet access

**Post-conditions:**

* Successful Completion:

User successfully saves or discards chosen graph from user-sensitive home-page list

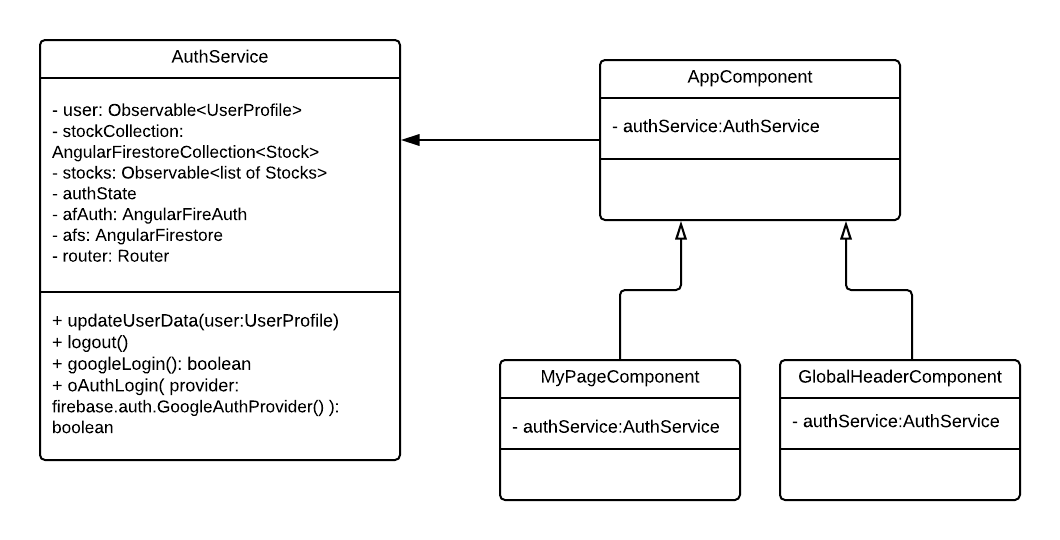
* Failure Condition:

Error, if unanticipated, will be saved to a ticket

1. Design Patterns

Observer Pattern:

Our AuthService class uses Firebase to authenticate a user. When a user successfully signs in, the AuthService class has a variable called authState that gets updated with the user’s information. The AppComponent, the main component of Angular that knows about all other components, is notified about the authState through an authService variable. Other components are then updated, which updates our user profile page and the global navbar. The global navbar is updated when the user is authenticated by switching a sign-in button to a dropdown menu that contains a mypage link and sign-out button.



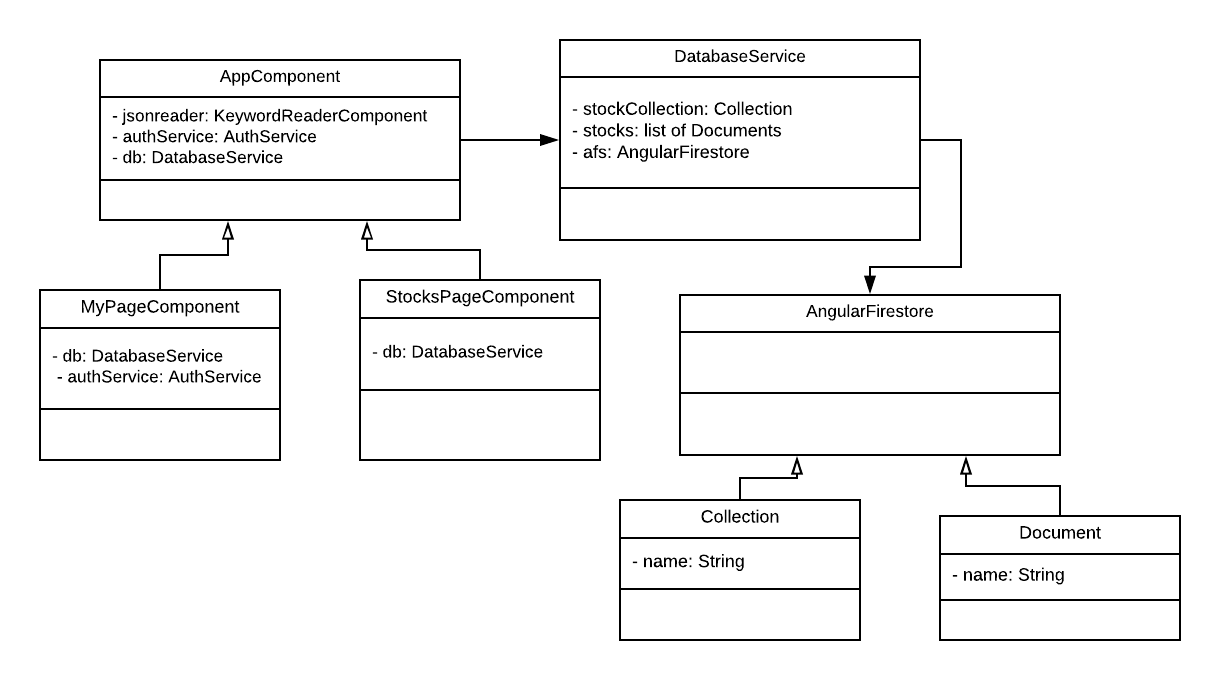
AuthService: <https://github.com/kdl222/CAT/blob/dev/project/CAT/src/app/auth.service.ts>

AppComponent: <https://github.com/kdl222/CAT/blob/dev/project/CAT/src/app/app.component.ts>

MyPageComponent: <https://github.com/kdl222/CAT/blob/dev/project/CAT/src/app/mypage/mypage.component.ts>

GlobalHeaderComponent: <https://github.com/kdl222/CAT/blob/dev/project/CAT/src/app/global-header/global-header.component.ts>

Bridge Pattern:

AngularFirestore is not a class that we have created, but a service that we have used that stores our user and stock information. AngularFirestore uses collections and documents to store information. We then have a class called databaseService that acts as our database and grabs information from AngularFirestore. Our application uses a “users” collection that contains many documents that store info on a certain user.

Note: The user and stock data stored from AngularFirestore is displayed in the html files, while the database is in the ts files.

AppComponent: <https://github.com/kdl222/CAT/blob/signIn/project/CAT/src/app/app.component.ts>

MyPageComponent:

.ts file - <https://github.com/kdl222/CAT/blob/signIn/project/CAT/src/app/mypage/mypage.component.ts>

.html file - <https://github.com/kdl222/CAT/blob/signIn/project/CAT/src/app/mypage/mypage.component.html>

StockPageComponent:

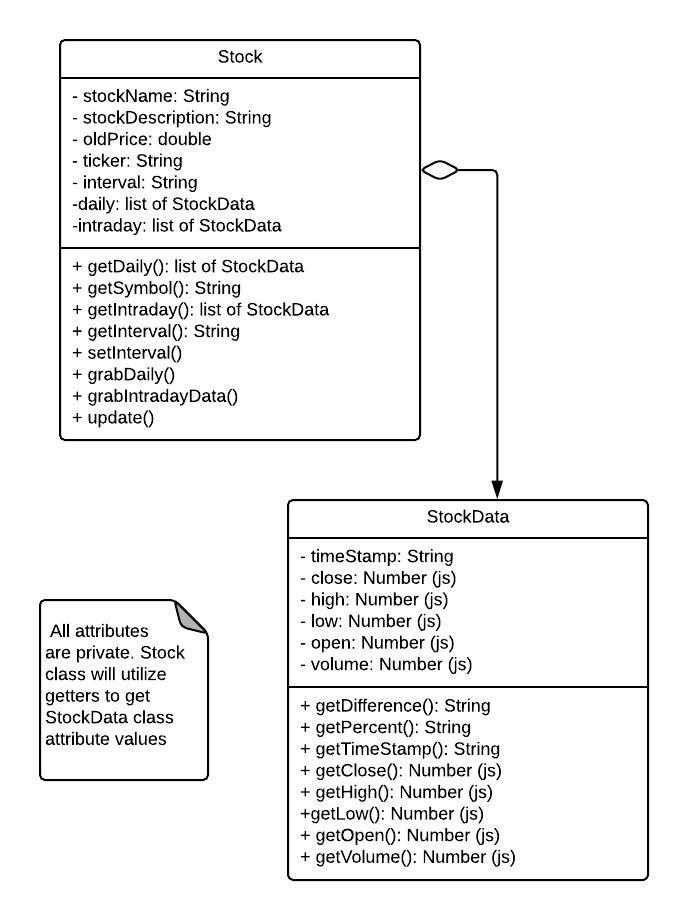
.ts file - <https://github.com/kdl222/CAT/blob/signIn/project/CAT/src/app/stocks-page/stocks-page.component.html>

.html file - <https://github.com/kdl222/CAT/blob/signIn/project/CAT/src/app/stocks-page/stocks-page.component.html>

databaseService: <https://github.com/kdl222/CAT/blob/signIn/project/CAT/src/app/database.service.ts>

Private Class Data Pattern:

Stock has private instance of StockData and uses getters to access the attributes of Stock Data.



Stock: <https://github.com/kdl222/CAT/blob/dev/project/CAT/src/app/Stock.model.ts>

StockData: <https://github.com/kdl222/CAT/blob/dev/project/CAT/src/app/Stock.model.ts>

Stock and StockData are in the same file, but are different classes in Stock.model.ts.

1. Design Principles (SOLID)

* Single Responsibility Principle:

Most, if not all, of our classes are designed to accomplish a single given responsibility. An example of this would be our AuthService. AuthService handles the authentication of a user regarding logging in and signing out.

* Open/Closed:

In the Stocks and News Feed page, new Stocks and Articles are continuously being added to the third-party APIs that we are using. In this way, the software entities are constantly being extended on their own, but the functionality is not being changed - the way the APIs work is not being modified.

* Liskov Substitution:

Using Angular 5, all web pages of our web application are the same page, only loaded differently based on the choices the user makes when interacting with the web application. In this way the web pages that the user activates overshadow the ‘main’ page of the web application according to their needs.

* Interface Segregation:

The main homepage is connected to every other page of the web application. Users can navigate to the pages that they want and are not forced to use or observe the pages that they do not want. In that way, the interfaces connected to the pages that the user does not wish to view are not forced upon the user.

* Dependency Inversion:

The Stocks page and the News Feed page are not updated with information until the user searches for what they want; all data is abstracted data. Once the user searches for what they want, the system will populate the Stock page and/or the New Feed page depending on which page the user is currently viewing.

1. Group Participation

Savannah Fischer - 40% Wrote description for section 1, created the design pattern for the private class data, and wrote section 6.

Kaitlyn Lee - 40% Wrote sections 3-4, created the observer and bridge design patterns, and worked on section 6.

Clayton Williams - 10% Section 2 Architecture

Brandon Horner - 10% Section 2 Architecture