# Mock Stock

Project Team: Luke Orr, Theodore Hecht, Sebastian Schwagerl, Jonathan Langston

# Table of Contents

**1. Project Definition (**100 - 200 words**)** – *Group responsibility*

*Some people want to learn about and practice stock trading without any financial commitment. Because of this, we have decided to create a mobile app that will allow people to learn about the stock market. Our users will be able to learn using a "hands on" approach, allowing them to trade "mock" stocks with virtual currency. This project will be created as a mobile app for the iOS operating system using Swift 4 and XCode, with a restful web API backend in .NET Core with a SQL database layer to hold user details, all created by the Mock Stock team.*

**2. Project Requirements** – *Group responsibility*

- Functional
    - The backend API must be able to query real-time financial data (stock prices) from a third party API.
    - The backend API must have middleware to authenticate users who make requests from the mobile app. (Our authentication solution may also have database requirements, like storing usernames/emails and hashed passwords.)
    - The backend API must be able to store and retrieve information about the user's virtual portfolio, exposed as a set of REST API endpoints that the frontend can access.
    - The frontend should let users buy/sell virtual stocks at real-world prices.
    - The frontend should let users examine their portfolio contents.
    - The frontend should allow users to display detailed information on specific companies.
    - The frontend should have a search/sort button to find specific stocks that the users wants.
    - The frontend should have a button to change the graphical timeline of the price history of the stock.
- Usability
    - User interface
        - Efficiency of use
        - Intuitive design
    - Performance
        - UI must be fast and responsive
        - API calls needs to be efficient

        - Database calls need to run quickly
- System
    - Hardware
        - Frontend: iOS device (phone), 6s and above
    - Software
        - Frontend: iOS current version
        - Backend: PaaS (Azure App Service)
    - Database
        - SQL database options: MySQL, PostgreSQL, MariaDB
- Security
    - Basic authentication middleware over HTTPS. Frontend sends username/password combo on each request encrypted over HTTPS, and backend middleware authenticates the request using this information.

**3. Project Specification** – *Group responsibility*

- Focus / Domain / Area
    - Stock market learning tool
- Libraries / Frameworks / Development Environment
    - Frontend: Swift4, UIKit (and other native iOS libraries), XCode
    - Backend: C#, .NET Core 2.X
- Platform (Mobile, Desktop, Gaming, Etc)
    - Mobile (iOS)
- Genre (Game, Application, etc)
    - Game/Education

**4. System – Design Perspective** – *Group responsibility*

- Identify subsystems – design point of view

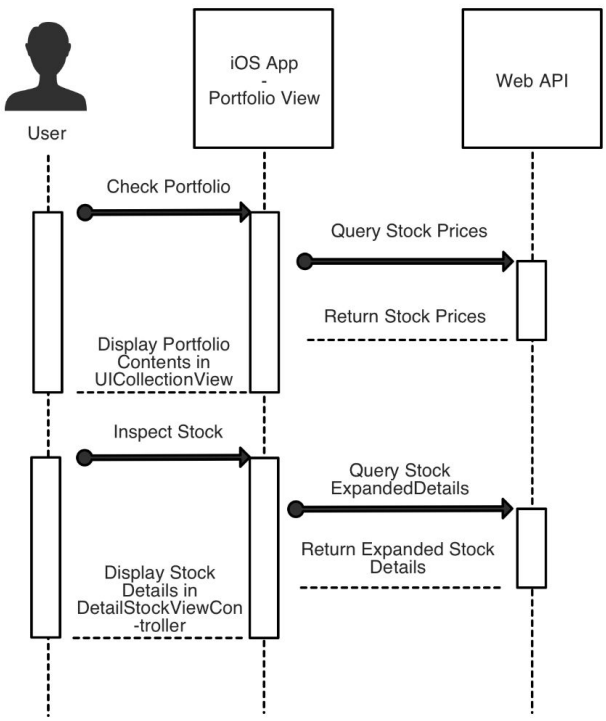
Front End Screens: (Portfolio, Leagues, Marketplace, Detailed Stock View).

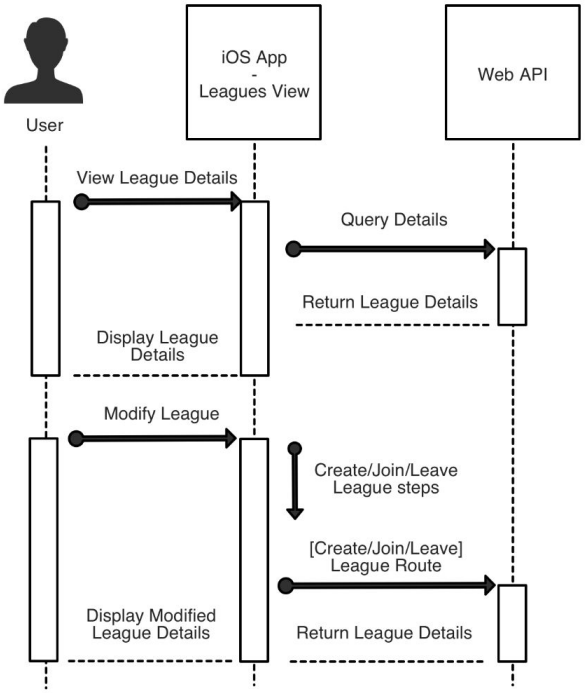
Portfolio Screen: This screen shows the user's portfolio. It contains a list of stocks the user owns, along with their current prices and the entire portfolio worth. It gets the current details for the portfolio from the backend API. The user can click a specific stock to see its expanded details (price history, etc.).

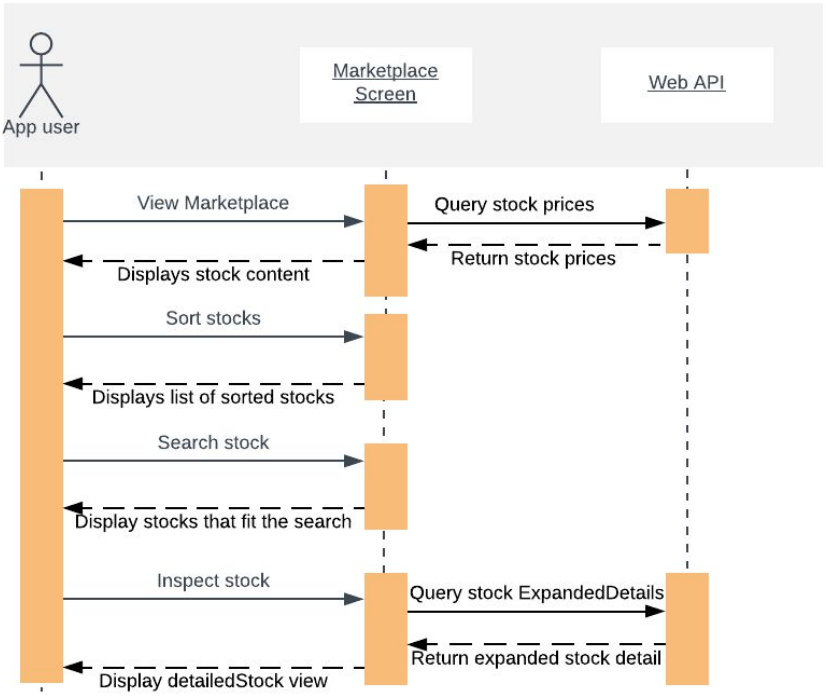
Leagues Screen: This screen shows all the leagues a user is a member of. A league is a competitive leaderboard with one or more users. Users can click on a league to see the current league details (a list of league members and their portfolio values.) Users can create, join, or leave leagues from this screen.

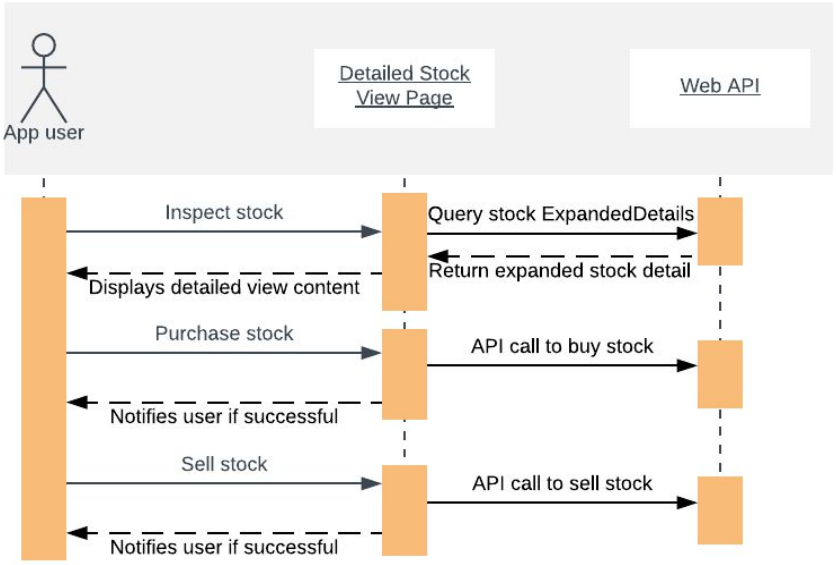
Marketplace Screen: This screen shows the stock marketplace. It contains a list of sorted stocks based on the users sort selection. The user will also be able to search stocks based off company name or ticker symbol. Current stock information will be obtained via the IEX api. The user can click a specific stock to see its expanded details (price history, etc.).

Detailed Stock View Screen: This screen shows the detailed view of a specific stock. The screen will display stock details such as price, today's change, change in the past year, as well as a graphical representation of the stocks price fluctuation. From this screen users will also be able to buy and sell stocks.


User
iOS App
-
Portfolio View
Web API
Check Portfolio
Query Stock Prices
Return Stock Prices
Display Portfolio
Contents in
UICollectionView
Inspect Stock
Query Stock
ExpandedDetails
Return Expanded Stock
Details
Display Stock
Details in
DetailStockViewCon
-troller


User
iOS App
-
Leagues View
Web API
View League Details
Query Details
Return League Details
Display League
Details
Modify League
Create/Join/Leave
League steps
[Create/Join/Leave]
League Route
Display Modified
League Details
Return League Details


App user
Marketplace Screen
Web API
View Marketplace
Query stock prices
Return stock prices
Displays stock content
Sort stocks
Displays list of sorted stocks
Search stock
Display stocks that fit the search
Inspect stock
Query stock ExpandedDetails
Return expanded stock detail
Display detailedStock view

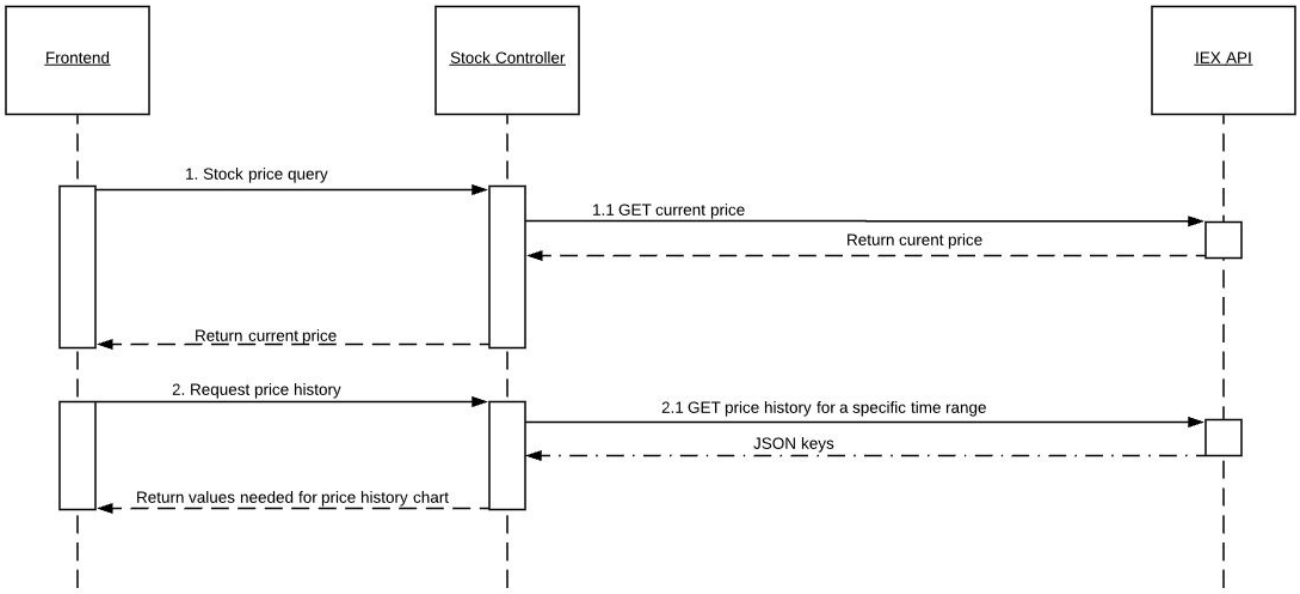

App user
Detailed Stock View Page
Web API
Inspect stock
Query stock ExpandedDetails
Return expanded stock detail
Displays detailed view content
Purchase stock
API call to buy stock
Notifies user if successful
Sell stock
API call to sell stock
Notifies user if successful

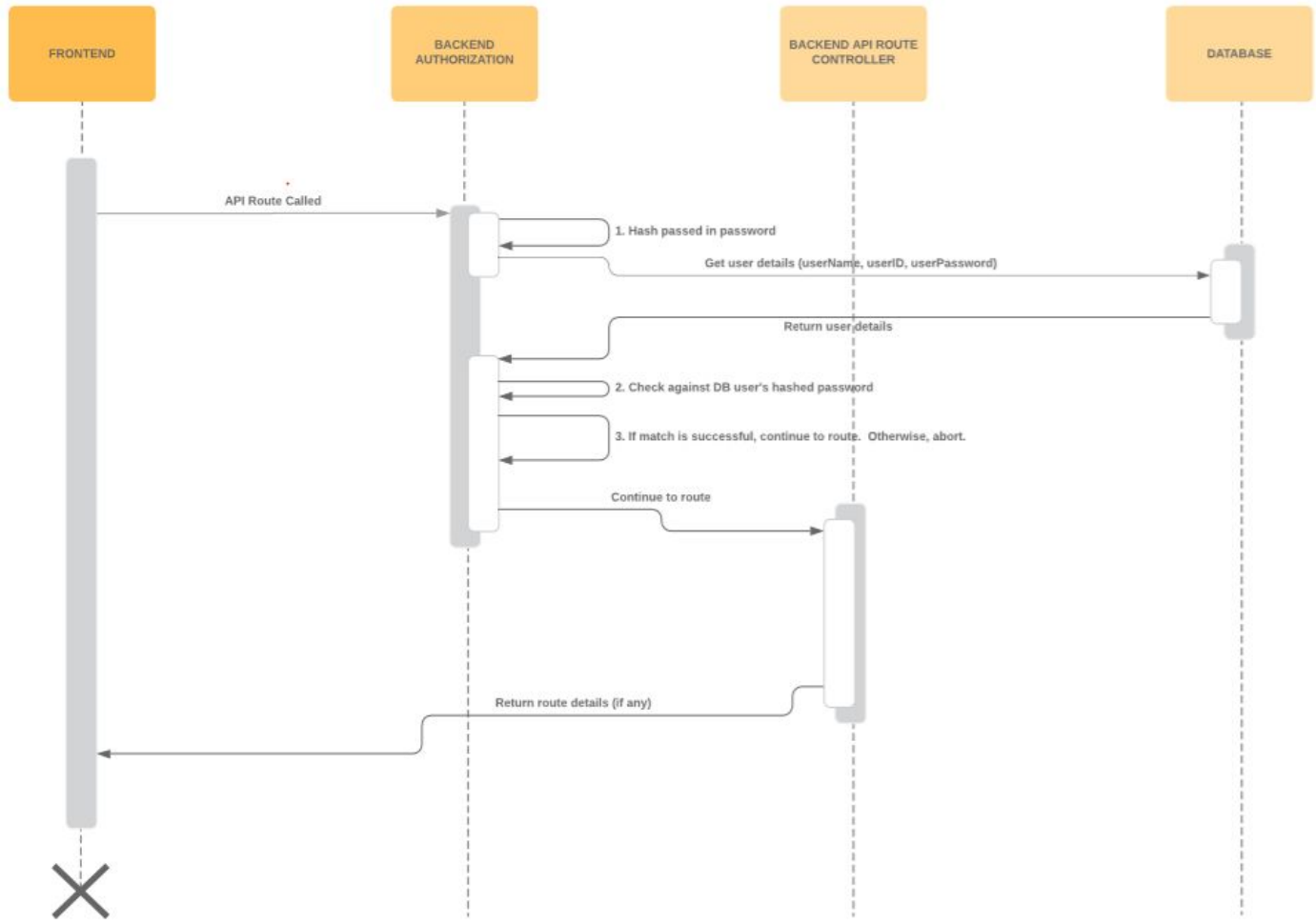
Backend Subsystems: (Authentication Middleware, Database Layer, 3rd Party API)

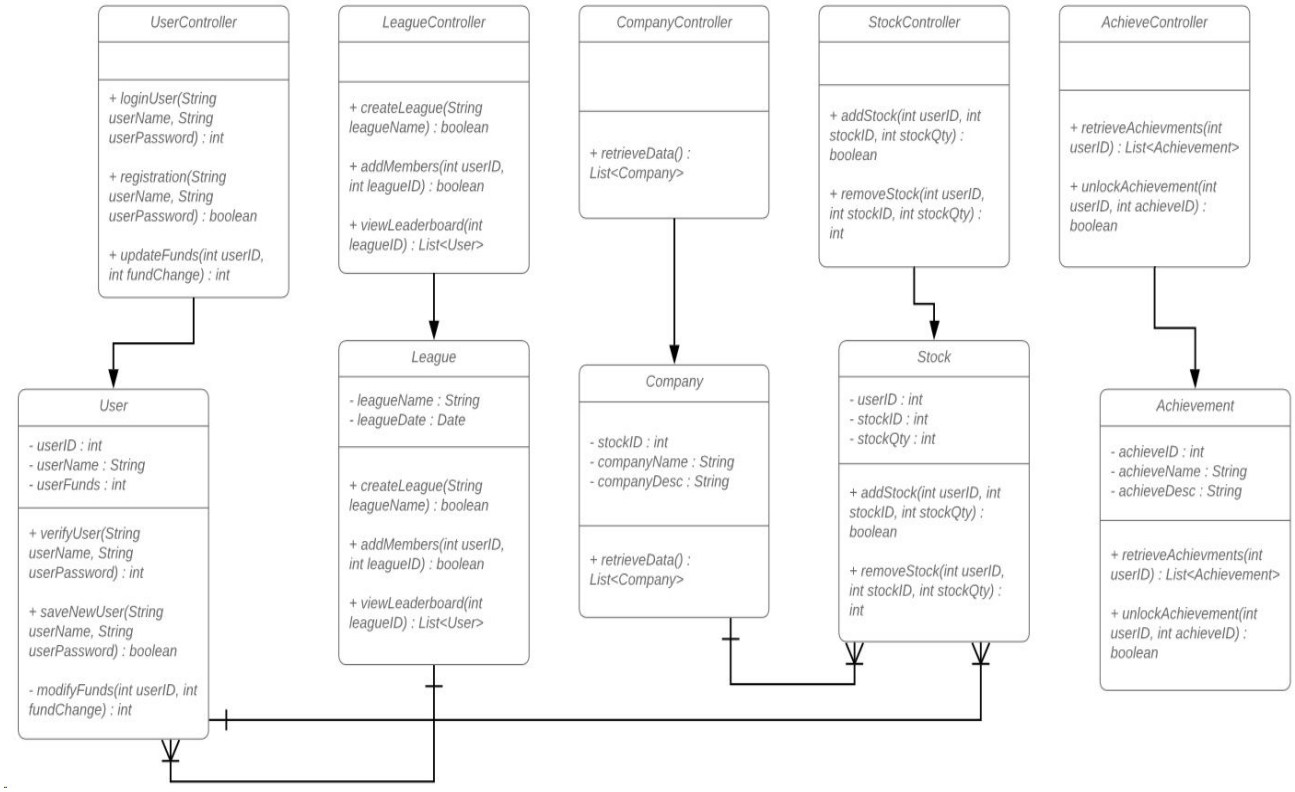
3rd Party API: A 3rd party API is needed in order to gather the information needed on stock prices and their details. The IEX API is used for this purpose since it can provide real-time data for free. This will be called from the backend API when the frontend needs the stock information.

Authentication Middleware: Authentication is performed before any call to the backend API route controllers, to verify that the user is who it is. It will hash the password provided, and compare it to the one that is stored on the database. If they match, it will continue to its intended destination on the backend, though if it fails, it will abort and throw an error.

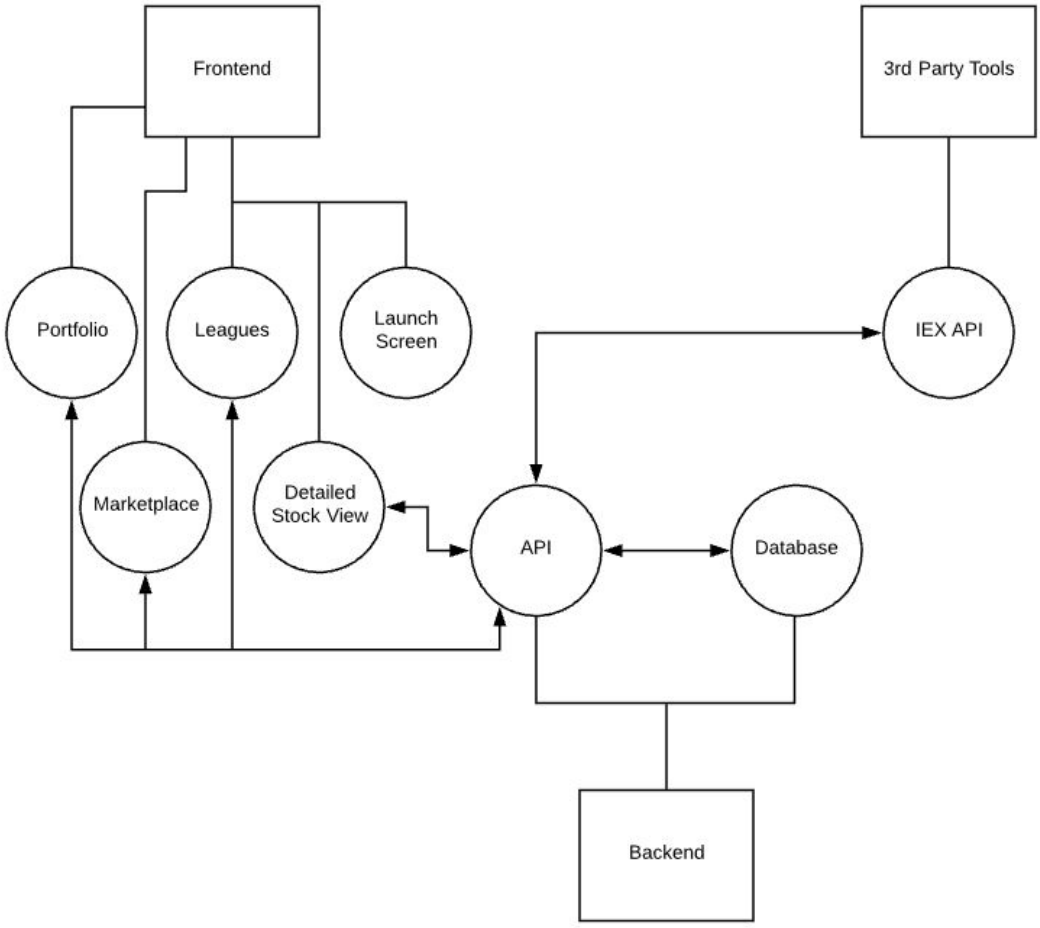
Database Layer: The database classes are split into 5 categories: User, League, Company, Stock, and Achievements. They will be reachable through their respective controllers, depending on the actions on the view. There may likely be a model that handles SQL connections that all 5 other models will connect to, though it will depend on the complexity of the SQL formulas.




FRONTEND
BACKEND AUTHORIZATION
BACKEND API ROUTE CONTROLLER
DATABASE
API Route Called
1. Hash passed in password
Get user details (userName, userID, userPassword)
Return user details
2. Check against DB user's hashed password
3. If match is successful, continue to route. Otherwise, abort.
Continue to route
Return route details (if any)


UserController
+ loginUser(String userName, String userPassword) : int
+ registration(String userName, String userPassword) : boolean
+ updateFunds(int userID, int fundChange) : int
LeagueController
+ createLeague(String leagueName) : boolean
+ addMembers(int userID, int leagueID) : boolean
+ viewLeaderboard(int leagueID) : List<User>
CompanyController
+ retrieveData() : List<Company>
StockController
+ addStock(int userID, int stockID, int stockQty) : boolean
+ removeStock(int userID, int stockID, int stockQty) : int
AchieveController
+ retrieveAchievments(int userID) : List<Achievement>
+ unlockAchievement(int userID, int achieveID) : boolean
User
- userID : int
- userName : String
- userFunds : int
+ verifyUser(String userName, String userPassword) : int
+ saveNewUser(String userName, String userPassword) : boolean
- modifyFunds(int userID, int fundChange) : int
League
- leagueName : String
- leagueDate : Date
+ createLeague(String leagueName) : boolean
+ addMembers(int userID, int leagueID) : boolean
+ viewLeaderboard(int leagueID) : List<User>
Company
- stockID : int
- companyName : String
- companyDesc : String
+ retrieveData() : List<Company>
Stock
- userID : int
- stockID : int
- stockQty : int
+ addStock(int userID, int stockID, int stockQty) : boolean
+ removeStock(int userID, int stockID, int stockQty) : int
Achievement
- achieveID : int
- achieveName : String
- achieveDesc : String
+ retrieveAchievments(int userID) : List<Achievement>
+ unlockAchievement(int userID, int achieveID) : boolean

- Sub-System Communication

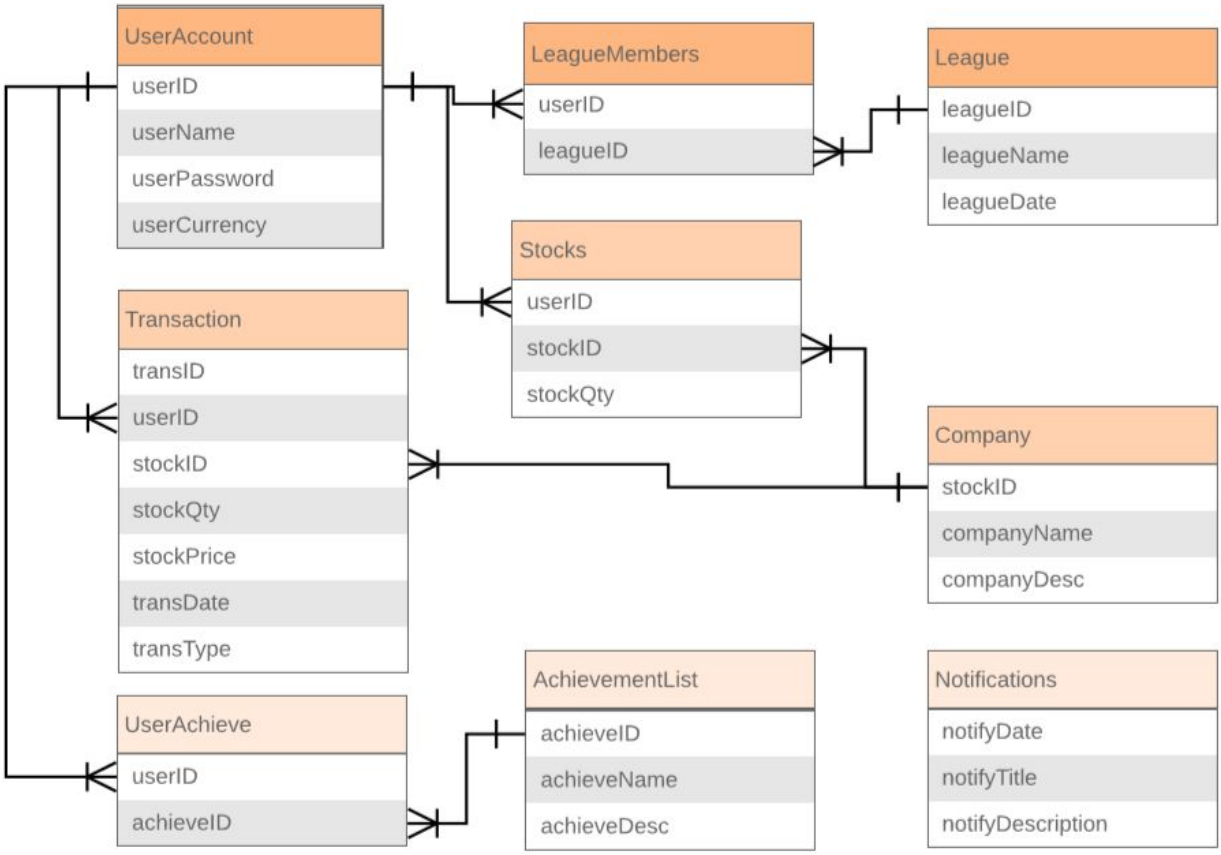



- The frontend is made up of multiple screens that interact with the backend API after being verified through the middleware authentication. They call the API when some kind of information is needed, which it will then communicate with the database or the 3rd party stock API (IEX) to get that information and supply it to the frontend. The Portfolio and League screens will be sent information on the user from the database. The Detailed Stock View and Marketplace screens will be sent information on current stock prices from the IEX API. When stocks are bought or sold, the API will need to interact with the database to update information on the user's ownership of the stocks and IEX to get the current price to update the user's currency.

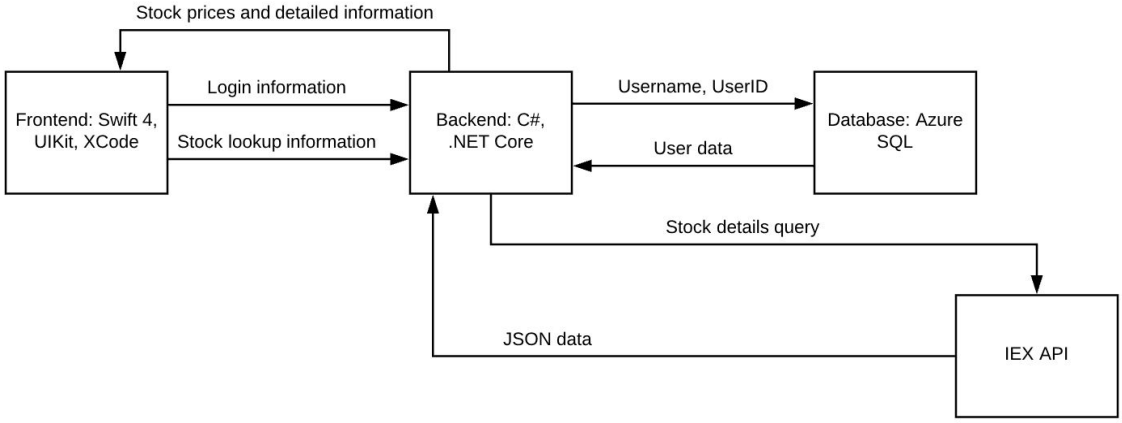
API Routes (Front to Back end communication. All data passed is in JSON.)

| Route | Description |
| --- | --- |
| GET /api/portfolio | Returns the authenticated user's portfolio details. (User details, portfolio contents and current prices.) |
| GET /api/stock/symbols | Returns a list of stock details (prices, etc.) for the given symbols. |
| POST /api/stock/buy | Buys a stock. Expects the ticker symbol of the stock to buy and the amount of stock to buy. |
| POST /api/stock/sell | Sells a stock. Expects the ticker symbol of the stock to sell and the amount of stock to sell. |
| GET /api/leagues/id | Returns league details. (Members and portfolio values.) |
| POST /api/leagues/create | Creates a new league. Returns the uniquely created league id, and registers the authenticated user as the league owner. |
| POST /api/leagues/delete | Deletes a league, if the authenticated user is the owner. Expects the league id. |
| POST /api/leagues/join | Joins a league with the authenticated user. Expects the league id. |

- Entity Relationship Model (E-R Model)




- Overall operation - System Model



**5. System – Analysis Perspective** – *Group responsibility*

- Identify subsystems – analysis point of view
- Backend: Web API + Database
    - C# and .NET Core were chosen because of a combination of developer familiarity (with C#) and cross platform ability. We can run a .NET Core backend on Windows and Mac, meaning our front end iOS team can develop with the backend code running for development/testing purposes. (This wouldn't be possible with older versions of .NET, which are Windows only.)
    - There are several different options for hosting our web application, the primary ones being Microsoft's Azure and Amazon's Web Services. We ended up choosing Azure, due to strong integration with Microsoft's .NET ecosystem and its pricing. Azure App Services allows 10 free applications, with no need to micromanage virtual machines, and allows us to scale those applications out horizontally with increased instances if we need them. Azure also provides a month of free credits to be used for cloud PostgreSQL database services, which we can activate near the delivery date of the project. However, beyond the free month, PostgreSQL would cost money to host on Azure, whereas Azure SQL could be hosted free for a year. But since our frontend development team will develop with MacOS for iOS development, it would be ideal to use a database capable of running locally on those computers, which Azure SQL was not made to do.
- Frontend:
    - Swift4 and UIkit were chosen for the front end for a couple of reasons. One is that overall the development team is more familiar with IOS development compared to Android. Another reason is because IOS beats out android when it comes to development complexity. Android has more devices and OS versions to worry about compared to IOS. In 2018 over 50% of IOS devices were running on IOS 12 (the most current version) compared to android where only .1% of users were running on Android 9 Pie and 14.6% on Android 8 and 8.1 Oreo. Because of those reasons, developing for IOS would result in a less complex app building process.

- System (Tables and Description)
    - Data analysis
        - Data dictionary (Table - Name, Data Type, Description)

| Name | Data Type | Description |
|---|---|---|
| userID | integer | Used for tracking the user with one value. |
| userName | string | The entered username. Paired with password to access. |
| userPassword | string | Will be hashed before being stored. Entered password is compared to stored hashed password. |
| userCurrency | double | The current unused funds on a user's account. |
| leagueID | integer | Used for tracking league. |
| leagueName | string | Name assigned to league upon creation. |
| leagueDate | DateTime | Date and time league was created. For use in record keeping, or potential deadlines. |
| stockID | integer | Used for tracking companies and stocks. |
| stockQty | integer | The number of stocks owned by a user for a specific company. |
| companyName | string | Name of Company |
| companyDesc | string | Short description of company, like what they sell, or what services they provide. |
| transID | integer | Used for tracking a transaction; for when a user buys or sells a stock. Transaction History. |
| stockPrice | double | Price paid or earned from buying/selling a stock. |
| transDate | DateTime | Time at which the exchange occurred. |
| transType | string | Determine if the stocks were bought, sold, or any other options. |

| | | |
|---|---|---|
| achieveID | int | Used for tracking various achievements. |
| achieveName | string | Title of Achievement. |
| achieveDesc | string | Description of Achievement Requirements |
| notifyDate | DateTime | Date at which notification was created. May last a certain amount of time beyond creation. |
| notifyTitle | string | Header of notification. |
| notifyDesc | string | Body of notification. For updates to the product, or for ongoing events. |

    - Process models
      The process model we will be using is Agile and Scrum. Agile was chosen because of its emphasis on less documentation and more developing. This process model will us to focus more on productive work compared to other more plan heavy models such as waterfall. We will also be using the Scrum process framework which is a subset of Agile. Scrum was chosen because of the benefits it provides, such as providing estimates on how long certain features will take to implement with the product backlog. It will also allow us to be in control of the project schedule with the use of a burn-down chart and product backlog.
- Algorithm Analysis
    - Search: When searching, we will have a list of stocks in a users portfolio or a list of stocks in the market stored on the device. Searching through this list would take at most the entire length of the list, meaning the time complexity of searching will be O(N).
    - Sort: Sometimes users will want to sort stock entries (for example, by current value, or by their growth over a certain period of time.) Merge Sort would give us a worst case of O(n*log(n)) for sorting on current price. Sorting on the growth over a given period of time would be the same complexity, since computing growth-over-time for each sorted stock could be done within the sorting algorithm and would just add extra constant-time computations, which we could ignore. But if we computed the growth-over-time outside of the sort, it would be O(n + (n * log(n))), since we would need to iterate over the list first before sorting.

**6. Project Scrum Report -** *Group Responsibility*

- Product Backlog (Table / Diagram)
- Sprint Backlog (Table / Diagram)
- Burndown Chart

**7. Subsystems**

**7.1 Subsystem 1** – Name 1 - *Individual responsibility*

- Initial design and model
  - Illustrate with class, use-case, UML, sequence ..... diagrams
  - Design choices
- Data dictionary
- If refined (changed over the course of project)
  - Reason for refinement (Pro versus Con)
  - Changes from initial model
  - Refined model analysis
  - Refined design (Diagram and Description)
- Scrum Backlog (Product and Sprint - Link to Section 6)
- Coding
  - Approach (Functional, OOP)
  - Language
- User training
  - Training / User manual (needed for final report)
- Testing

**7.2 Subsystem 2** – Name 2 - *Individual responsibility*

- Initial design and model
  - Illustrate with class, use-case, UML, sequence ..... diagrams
  - Design choices
- Data dictionary
- If refined (changed over the course of project)
  - Reason for refinement (Pro versus Con)
  - Changes from initial model
  - Refined model analysis
  - Refined design (Diagram and Description)
- Scrum Backlog (Product and Sprint - Link to Section 6)

- Coding
    - Approach (Functional, OOP)
    - Language
- User training
    - Training / User manual (needed for final report)
- Testing

**7.3 Subsystem 3** – Name 3 - *Individual responsibility*

- Initial design and model
    - Illustrate with class, use-case, UML, sequence ..... diagrams
    - Design choices
- Data dictionary
- If refined (changed over the course of project)
    - Reason for refinement (Pro versus Con)
    - Changes from initial model
    - Refined model analysis
    - Refined design (Diagram and Description)
- Scrum Backlog (Product and Sprint - Link to Section 6)
- Coding
    - Approach (Functional, OOP)
    - Language
- User training
    - Training / User manual (needed for final report)
- Testing

**7.4 Subsystem 4** – Name 4 - *Individual responsibility*

- Initial design and model
    - Illustrate with class, use-case, UML, sequence ..... diagrams
    - Design choices
- Data dictionary
- If refined (changed over the course of project)
    - Reason for refinement (Pro versus Con)
    - Changes from initial model
    - Refined model analysis
    - Refined design (Diagram and Description)
- Scrum Backlog (Product and Sprint - Link to Section 6)
- Coding
    - Approach (Functional, OOP)

    - Language
- User training
    - Training / User manual (needed for final report)
- Testing

**8. Complete System** – *Group responsibility*

- Final software/hardware product
- Source code and user manual – screenshots as needed - Technical report
    - Github Link
- Evaluation by client and instructor
- Team Member Descriptions

***This is just a guide, and use it to create/improve your report. Feel free to add sections. You are responsible for your own subsystem/s, not other members. You have to contribute to the team's goals and objectives, and develop your subsystem/s, write your documents and slides.***

**Things to Consider:**

What to do if player has dropped to very low $ in the Stock game? Give chance to reset game? Have them deal with it like real life?

- That's a good question. We certainly don't want them to keep making new accounts. Perhaps we have weekly achievements & milestones that are completable even with low $, which will give them more virtual currency. Perhaps some virtual currency can be given each day they log in, to incentivize them using the app often. We can come up with a list of ways to address this issue in the next week or two as we plan the app.

Shall we have a chart of stock prices available for every company, if possible to import?

- We might could keep up-to-date data on a certain number of high profile companies (like those in the S&P 500) from whatever financial API we use, and record those regularly. If a user searches for a specific ticker name not in the list, we can make another API call to grab that data in real time (or if they want to see the historical price of the stock, to display in a chart, perhaps there is a way to grab that from the financial API. Or maybe there are open databases out there with historical stock price information? In that case we could import it and store in our database.)

Complete Before Progress Report #2

**1 - DESIGN:**

Subsystems:

- ❏ Use-case, sequence, & UML Diagrams w/ descriptions if necessary. **(Everyone)**
- ~~❏ Subsystem Communication + Overall System Diagrams. **(Tyler)**~~
- ~~❏ API Route list (GET, POST, etc.) **(Trey)**~~
- ❏ ER Model **(Sebastian)**

**2 - ANALYSIS:**

- ~~❏ Backend Systems Analysis **(Trey)**~~
- ❏ Frontend Systems Analysis **(Luke)**
- ❏ Data Dictionary **(Sebastian)**
- ❏ Process Models **(Luke)**
- ~~❏ Algorithm Analysis - Search & Sort. **(Luke & Trey)**~~

**3 - Update 1-3 on Doc (Low Priority):**

- ❏ In the introduction, remove the tech information, replace with general functionality information.
- ❏ In the Project Requirements, go into more detail for “System.”
- ❏ In the Project Specifications, explain in more detail what each chosen piece of tech actually does for us and why (briefly) we chose it.