**ITCS 4155**

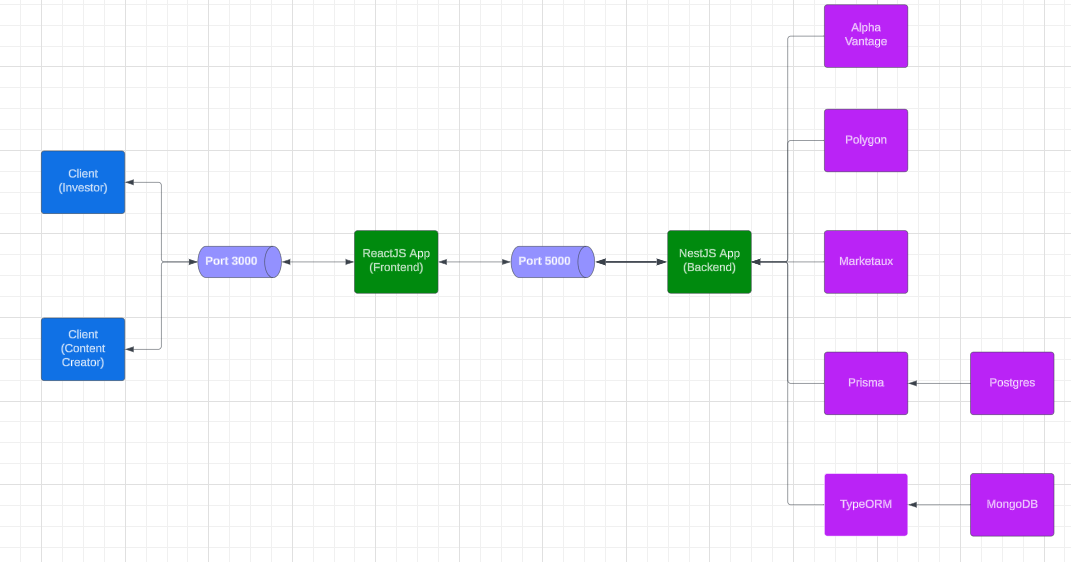
**SD2: Software Design Document**

The purpose of the software design document (SD2) is to provide an overview of the system and to help people understand the system. This document can be used by a programmer as a guideline for implementing the design without needing to make significant engineering design decisions, and will be provided to the customer as a final deliverable.

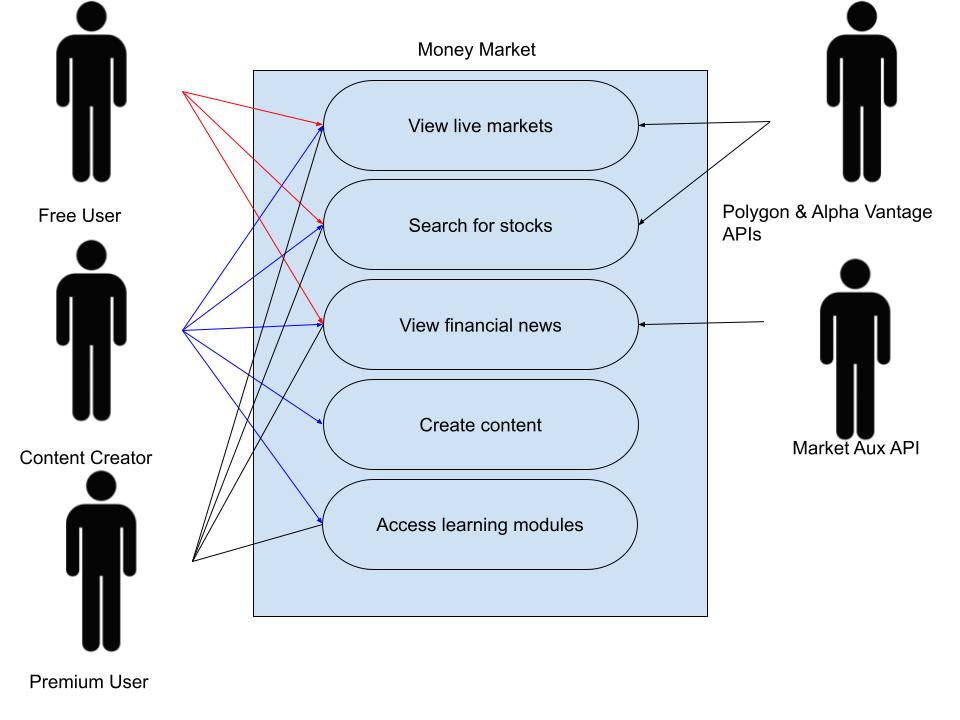
1. Project Overview

Money Market is created to address a lack of financial literacy by investors by providing a platform for investors to learn about investing and the stock market. Using Money Market, our users will be able to search for stocks to see stock data and graphs that document the stocks performance along with being able to access recent news articles related to investing. Our users will be able to access learning modules through a subscription that explain and demonstrate financial literacy aspects like how the stock market works, penny stocks, benefits of investment strategies and more. Users will be able to become content creators themselves to create learning modules for their fellow investors to benefit from. Our software will allow users to increase their financial literacy and become better investors.

**Context Diagram**

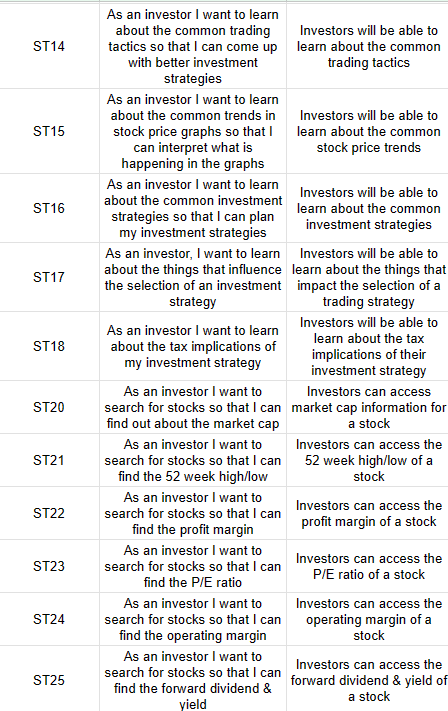
****

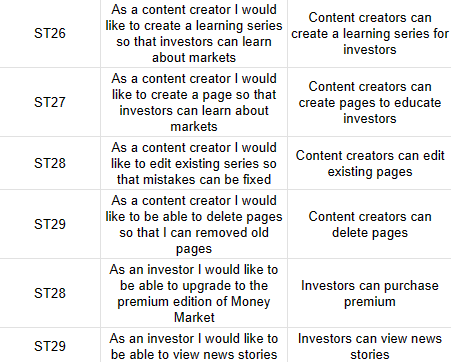
**Use Case**



**User Stories**

****

****

****

2. Architectural Overview

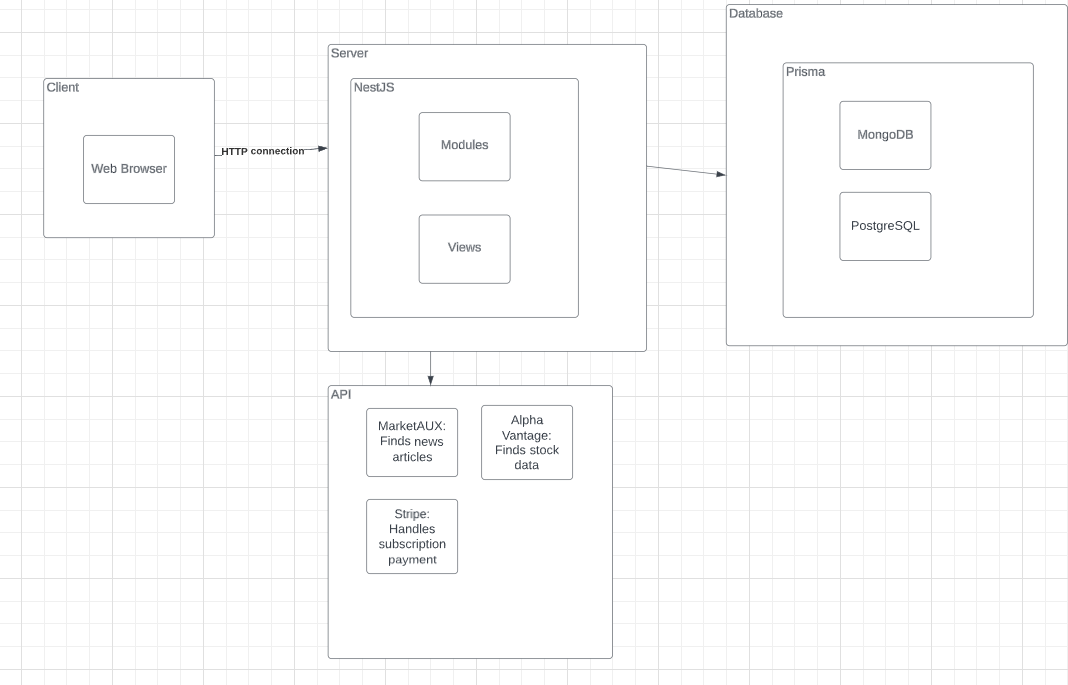
We didn’t discuss many alternative designs for this application. We decided to choose the most realistic architecture we could. This was done so the team could learn more about actual industry-used tools like React and Nest.js. The other architectures we discussed were less used in industry thus leading to our choosing of other technologies.

2.1 Subsystem Architecture

This is the link to the backend diagram: [MoneyMarket Flow Chart: Lucidchart](https://lucid.app/lucidchart/a32e7d72-e46b-4ae9-93d6-a82b629aef91/edit?invitationId=inv_acd54ccd-c933-47d6-8378-132d0c3028e2&page=WNVaKl5m5hUJ#)

The stock data section of the backend program is responsible for getting all the stock data and sending it to the front end for display. The news module is responsible for the news data being sent to the front end. The auth module is responsible for account creation and logins. The main architectural styles followed are mostly what is done by many companies. Each component or module is modularized making the program easier to use and read. Another reason for this is to address the issue of people possibly not understanding how the software works.

2.2 Deployment Architecture



Clients can connect to the Money Market server using an HTTP connection. Our server runs a nestJS framework which allows the client to make API calls when needed to access specific pages and have their payment handled to upgrade their account. Our server connects to our databases of MongoDB and PostgreSQL which are handled by Prisma.

2.3 **Data Model**

Our application stores user and subscription data and our learning modules. Storage regarding users will consist of when the account was created, first name, last name, username, their role in the application, userID, subscription status, and learning modules. Subscription will hold an ID, the time the subscription was created, wether the subscription is valid and the userID associated with it. Learning modules will contain an id, when it was created and store the userID for the user who created it.

2.4 Global Control Flow

Money Market is an event-driven application. Searching for a stock and visiting the news page causes our application to populate the page using real time information about the stock or recent news articles. The home page, news page, stock page, and llearning page operate as modules independent of each other and do not require the user to follow a linear path. Users can generate content in whatever order they want as our pages operate asynchronously. Our application will operate under one time dependency for subscribed users as a subscription to Money Market is only valid for 1 month before needing to be renewed again.

3 Detailed System Design

Money Market was built using a Nest.js backend and a React frontend. The application uses a PostgresSQL database along with a Mongodb database. The modules that make up the backend include, auth, user, stockdata, news, and profile. The auth module connects to the login database and authenticates users when they login in with the correct credentials. The user module stores all the user data. The stockdata module uses the polygon api to get real stock data. The news module uses marketaux to get real news stories. Finally, the profile module provides all the profile information.

Front End Repository:[ljohnston00/ITSC4155\_MDSp24\_Group7 (github.com)](https://github.com/ljohnston00/ITSC4155_MDSp24_Group7)

Back End Repository: [amf5214/money-market-api at learning-framework-creation (github.com)](https://github.com/amf5214/money-market-api/tree/learning-framework-creation)

3.1 Static view

You must include UML class diagrams showing further decomposition of the major modules and the relationships among these classes. In each class in the UML class diagram, you must show:

● Important attributes, their type, and their visibility

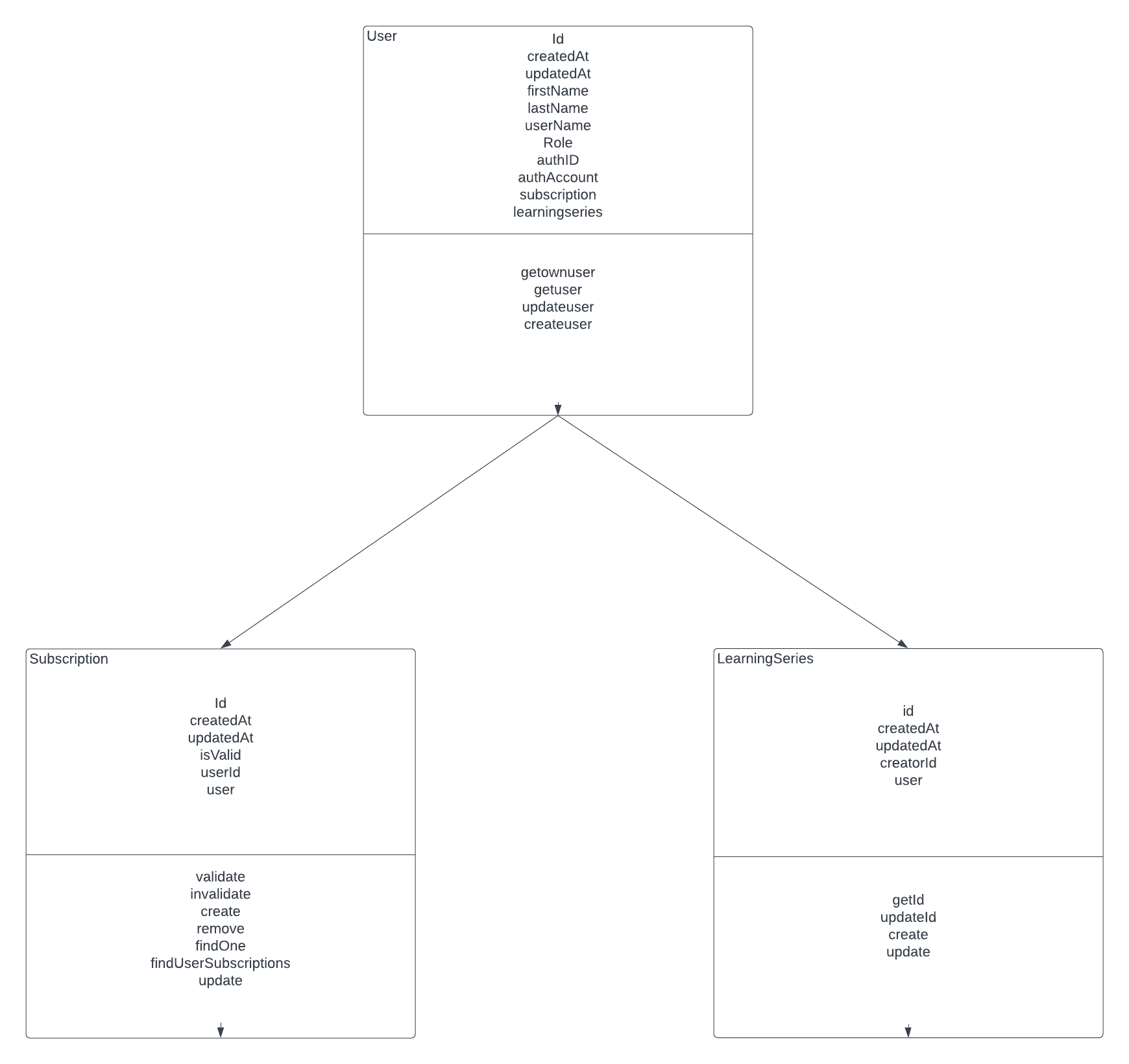
● Important operation/method names, their parameters, return types, and their visibility (public, private, protected, package)

● Associations between classes and multiplicity constraints

In addition, you will also need to write support text that justifies your decomposition of your modules into the classes shown in your UML class diagram. This justification should discuss the responsibilities of each class. The justification should describe other alternative designs, if any, and why your design is better.

Finally, you should describe any design patterns that are included in this design and why you’ve applied them.

*For reference material, see* [*Robert Martin’s article on UML class diagrams*](https://www.khoury.northeastern.edu/home/riccardo/courses/csu370-fa07/umlClassDiagrams.pdf)*, the Sparx tutorial on UML* [*package*](https://sparxsystems.com/resources/tutorials/uml2/package-diagram.html) *and* [*class*](https://sparxsystems.com/resources/tutorials/uml2/class-diagram.html) *diagrams, or the optional UML textbook recommended on the syllabus. For reference material on design patterns, check out the lecture slides, Ch. 5 in your textbook, the POSA and GoF books on reserve in the library, or this very handy* [*online catalog of design patterns*](https://refactoring.guru/design-patterns)or this website on [common patterns](https://www.martinfowler.com/articles/writingPatterns.html).

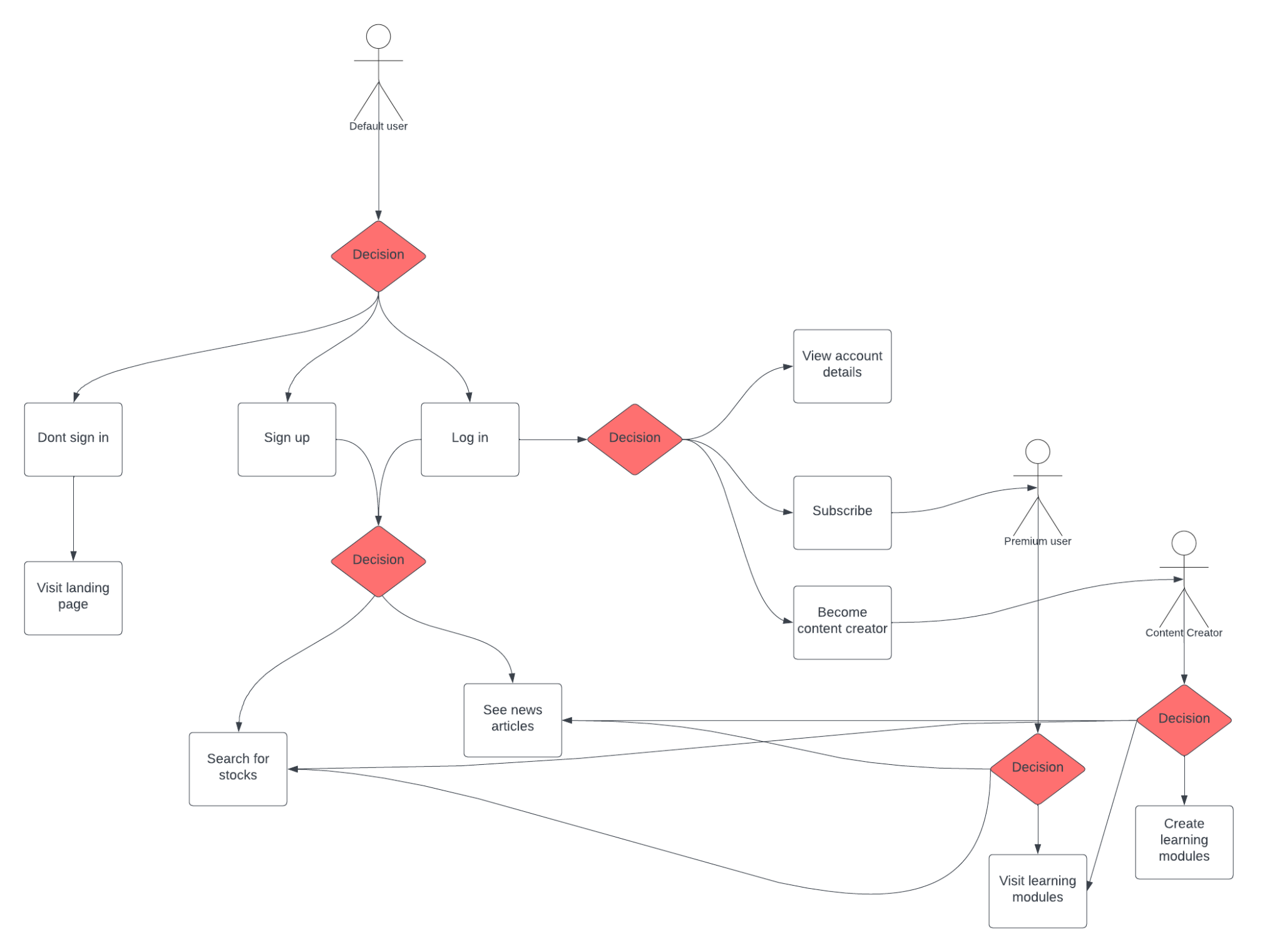


The user table holds user information like when the user created and updated their account, their first name, last name, username, role, subscription, and learning series. Users store an array that holds any subscriptions that have been assigned to them along with any learning series that they have been responsible for creating. Subscription has an ID, stores the time created and updated, along with references to the user whom it is assigned to with userId and the user class. The learning series has an ID along with the time it was created and updated as well as the user class from the user who created the series.

3.2 Dynamic view

You must show the design of your system’s behavior using UML sequence diagrams. These sequence diagrams should show the time-ordered sequence of interactions among classes to support an important system function. Your sequence diagrams should be consistent with the class diagrams given in Section 3.1. In other words, you should not have participating objects in an interaction that do not appear in a class diagram; if you find that this is the case, you should go back and revise your class diagram to include the new element. You may supplement your sequence diagrams with state-transition diagrams or activity diagrams (useful for describing algorithms), but these are not required.

Make sure to add your Testplans, Previous and current sprints and brief sprint reviews as well as any additional items you feel are helpful you feel are helpful at any point. Remember, it’s not necessary to add documents that are not providing useful information. Only necessary items. If you don’t have them ready, you can add them later using version control in github.



The basic user interface allows you to log in or sign up or not sign in and only visit the landing page. When signed in, you can visit the account details page, subscribe, or become a content creator. The basic user can search for stocks and see recent news articles but if the user chooses to upgrade their account it will give additional features. Subscribing allows a premium user to visit learning modules as well as the ability to search for stocks and see news articles. Users can apply to become a content creator which will give them the ability to create and update learning modules.