Design Document

Team: 26

Gokul Yenugadhati

Vishaal Bommena

Harsha Lingareddy

Dhiman Swadia

Sanat Mouli

Vijay Viswan

Contents

1. Purpose

Students are required to use different platforms and services to find the right class to take. Our project aims to help students find the right class based on their interests and create an intuitive service that centralizes on a course recommendation system tailored to each user, peer comments on each individual class and a rating system to gauge public consensus on each course.

1.1 Functional Requirements

1. Creating a user account
    a. User will be able to create an account using email address and password.
    b. User will be able to use social accounts for login.
    c. User will be able to login with two-factor authentication

2. Profile
    a. User will be able to upload a profile picture for their account
    b. User will be able to view other students' profiles
    c. User will be able to choose what information in their profile is publicly shown

3. Input previous classes
    a. User will be shown an input form to add previous classes to get better recommendations.
    b. User will have an option to input multiple classes at a time.

4. Course List
    a. User will be shown multiple lists of courses which will be categorized by the degree requirements.
    b. User will be able to use a search function for the course list
    c. User will be able to sort the course list between multiple categories

d. The User will be shown a personalized “success score” for each class which will help the user determine if the user should take the class.

5. Up vote/Down Vote
    a. User will be shown the upvote count for each class which will help gauge the overall consensus for the class.
    b. User will have an option to down vote or up vote each class.
    c. User will be able to sort the courses by most/least up voted.

6. Course information
    a. User will be shown a brief description of the course.
    b. User will be shown timings for the course for each semester.

7. Professor Information
    a. User will be shown the recommended professors list for each course.

8. Comment
    a. User will be able to see the comments from other students.
    b. User will be able to interact with the comment to respond.
    c. User’s comment will be highlighted if they have taken that course

9. Comment generation
    a. User will be able to create individual comment for each course.

10. Support
    a. User will be able to report bugs to the webmaster

1.2 Non-Functional Requirements

1. Client Requirement
    a. The app will also use a REST API to query the database for data and reduce response time.

2. Design Requirement
    a. We intend our application to work with Google Chrome and other modern browsers.
    b. The frontend will be built with user experience in mind and will use React.js framework to build an interactive and stable UI.
    c. The app will also use a scalable implementation by hosting the front end using Python/Flask.
    d. The user interface should be well designed so the user can come gather information and analyze and makes decision regarding the courses that they take.
    e. The app shall use a relational database such as MySQL

3. Performance Requirement
    a. The application should be able to train and provide a rank for the courses the user is most likely to succeed in, rating them between least likely and highly likely to succeed.
    b. The recommendation system would improve as the user inputs more courses taken, and would over time be able to aptly classify and rank the available courses.
    c. We would track the performance of the application and log errors to ensure robust backend performance.

4. Security Requirement
    a. The app will use industry standard authentication practices like two factor authentications and sign up using social networks.

2. Design Outline

2.1 High Level Overview

Our project will use the client-server model. The front end server will host the client and will request data from the backend server. The backend server will respond to the client requests, by sending queries to a fully functional database and will provide a response to front end server, which will then render the data that it receive on the client. The backend will also handle a Machine Learning based micro-service. The micro-service is based on learn to rank model to rank the different courses available to CS students based on their opinion on their previous courses.

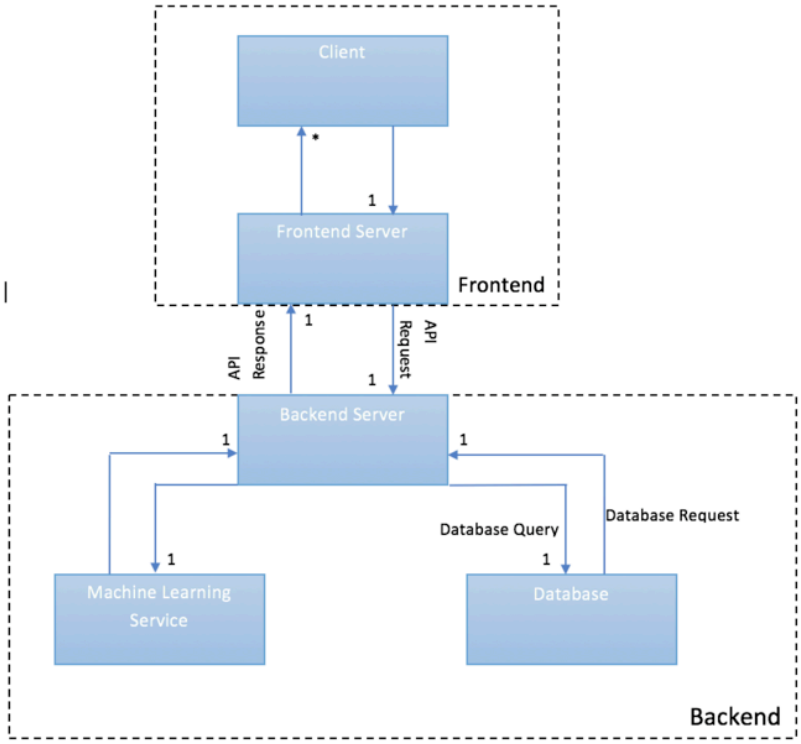

FIGURE 1: DIAGRAM OF HIGHER LEVEL ARCHITECTURE

2.1.1 Web Client

- Client sends AJAX requests to the server API
- Client receives an AJAX response from the server
- Response data is rendered to the user using React JS

2.1.2 Web Server

- Hosts the React application using Node.js
- Manage node dependencies and libraries for React.js

2.1.3 API Server

- Processes AJAX requests from the web client
- Retrieves data from the database using Python/Flask
- Returns an AJAX response to the web client

2.1.4 Database

- The database will store all of the user data in our Course Recommendation System.
- The database should be able to hold all the user data and course listings even if the API server or any other module crashes or does not function properly. The workings of the API server and other modules should have no effect on the user data which is to be stored.
- The database will primarily store relationships between the student table and the course table. The primary purpose of the database is to hold records of the students which take a particular course and the courses which a particular student will take

2.1.5 Machine Learning

- The Machine Learning feature would train on courses taken by the student in previous semesters and rank the courses available to the student.
- The Machine Learning feature would train and predict on the following student success features:
    - Critical Thinking
    - Math Skills

    - Memorization
    - Teamwork
    - Speech and Debate
- The Machine Learning feature would normalize the features based on the students comfort level in the courses taken so as to prevent overlapping.

2.2 Flow of Events

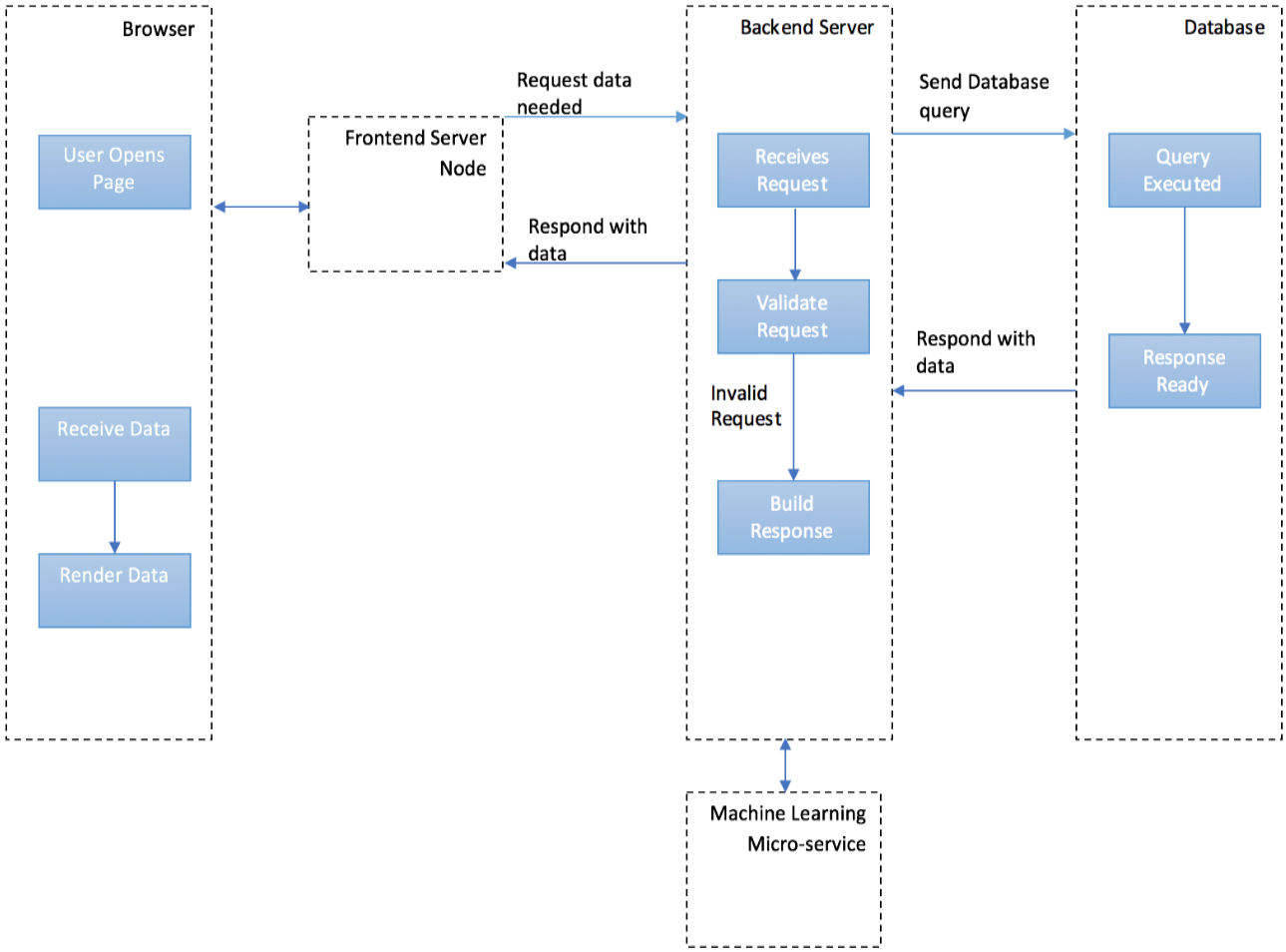

FIGURE 2: FLOW OF EVENTS

The diagram above shows the flow of events. The flow starts with the user opening the application through their browser. The frontend server hosts the frontend client and manages the dependencies required to run the frontend. This helps us to have an independent architecture which aids the frontend team to develop faster and be independent of the backend server. This also means that in an off chance that the backend server errors out, the frontend server will still be functional.

The backend server will first receive and validate the request from the frontend server. If the request is valid it will query the database, where the query will be executed and then generate a response to send back to the server. If the request is invalid, no query will be made and an error response will be sent. The data will then be sent back to the frontend server where it will render the data onto the page for the user to view it. The machine learning micro-service requests data (student course traits and course traits) from the database for classification and ranking. The student course traits are normalized based on the student's opinion of the course and used for training. The micro-service classifies and predicts ranks upon completion of the training phase.

2.3 Activity State Diagram

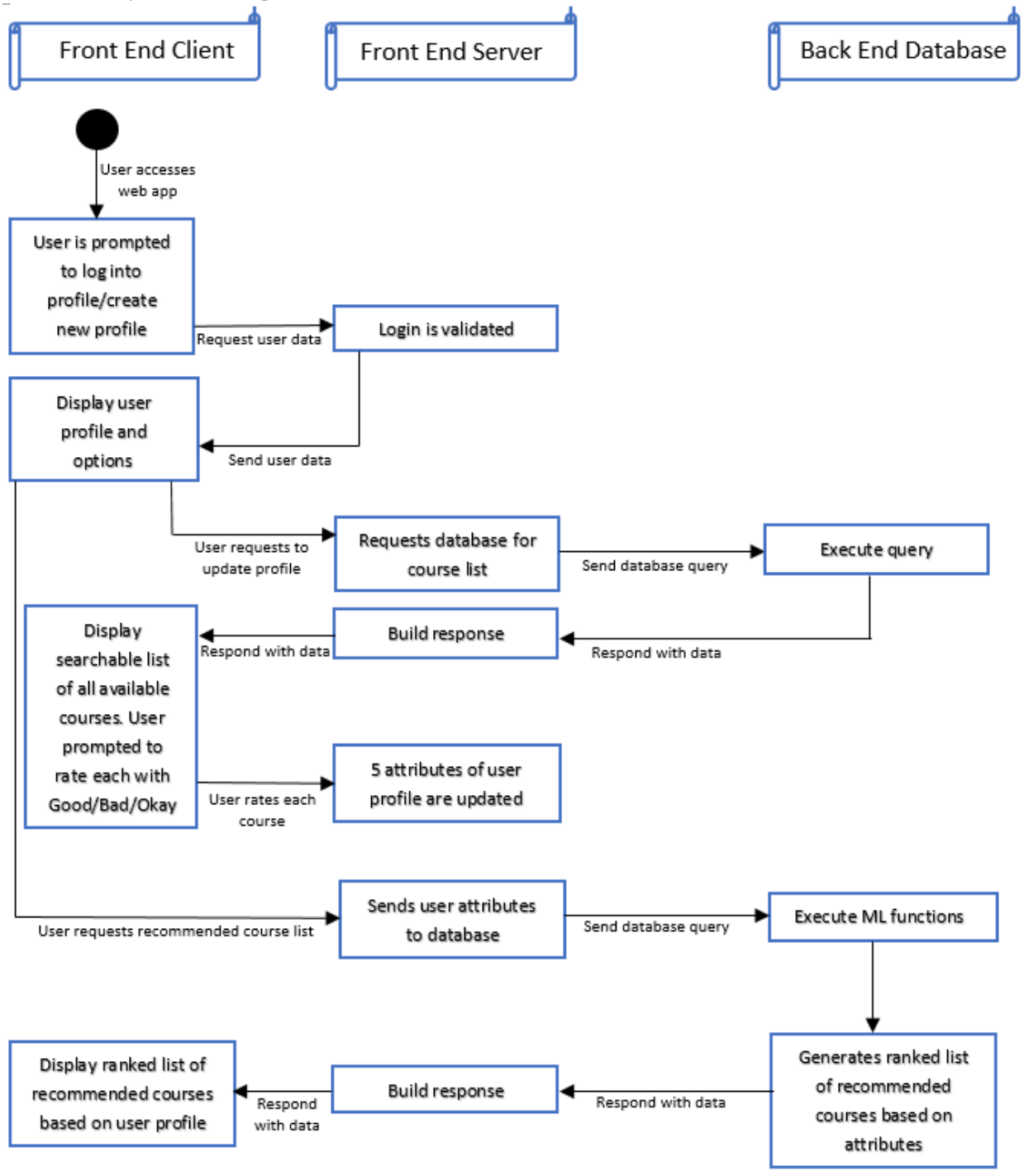

FIGURE 3: ACTIVITY STATE DIAGRAM

3. Design Issue

3.1 Functional Issues

Issue 1: What kind of metric to use for recommendation score?

Option 1: Yes or No

Option 2: Point based metric

Decision: The point based metric was chosen over a Yes or No metric since this would help generate a better score for the user, while aiding in the ordering process for the courses. Based on the point based metric, the user would be able to understand how successful they would be in the course better.

Issue 2: How should users navigate through their course list?

Option 1: Display all course lists on the central page.

Option 2: Individual pages for each course list sub-section.

Decision: We opted to display all course lists on a centralized page that allows the user to navigate to the specific course because the user can access the information on a centralized page without having to look for it.

3.2 Non-Functional Issues

Issue 1: What type of front end framework to use.

Option 1: React.js

Option 2: Angular.js

Decision: We choose React.js as we can build intuitive user interfaces easily and quickly. React renders the DOM by utilizing a virtual DOM and generates a diff between the new state and the old state. This means that React will make the most optimum changes to the DOM which generates a better performance for the frontend elements.

Issue 2: Whether to use a specialized server to host front end

Option 1: Node.js

Option 2: Flask

Decision: We decided to use an individual server for the frontend so we can parallelize development between frontend and backend teams. Node also gives us access to various packages and libraries which will help us speed front end development.

Issue 3: What type of back end framework is most appropriate for our data?

Option 1: Node.js

Option 2: Flask

Decision: We chose to use Flask because it is a microservice framework written in Python. Python is also the language in which we are writing the Machine Learning micro-service. Therefore, it is easier to integrate it into the back-end framework.

Issue 4: Which API to use for the Machine Learning Service?

Option 1: TensorFlow

Option 2: Scikit

Decision: Scikit because it is better for "off the shelf" algorithms like Learn to Rank (LTR) and Simple Vector Machines (SVM). While Tensorflow allows for a good implementation of deep-learning, it is not a necessity for the scope of this project.

Issue 5: Which HTTP connector should we use to connect the frontend to the backend?

Option 1: AJAX

Option 2: WebSocket

Option 3: SSE

Decision: We chose to use AJAX to connect the frontend to the backend since it allows for asynchronous exchanging small amounts of data with the server without affecting the remaining of the page.

Issue 6: Which languages are appropriate for implementing our backend service?

Option 1: Python

Option 2: Javascript

Decision: Since we would like to have our ML micro-service communicate with the backend with ease, and since most of the Machine Learning APIs which we use are implemented for Python, we have decided to use a Python based micro-service (Flask) to implement our backend service.

Issue 7: What database software should we use for our data?

Option 1: MySQL

Option 2: MongoDB

Option 3: PostgreSQL

Decision: We decided to go with a traditional RDBMS such as MySQL. We decided against a Document Store such as MongoDB because a Document Store prefers high insert rate over transaction safety. For CourseRec, we do not want to risk losing clients data. Therefore we decided against MongoDB. Both MySQL and PostgreSQL are RDBMS and similar in performance. Since we are more familiar with MySQL tools associated with MySQL database management, we decided to use MySQL.

4. Design Details

4.1 Data Class Level Design

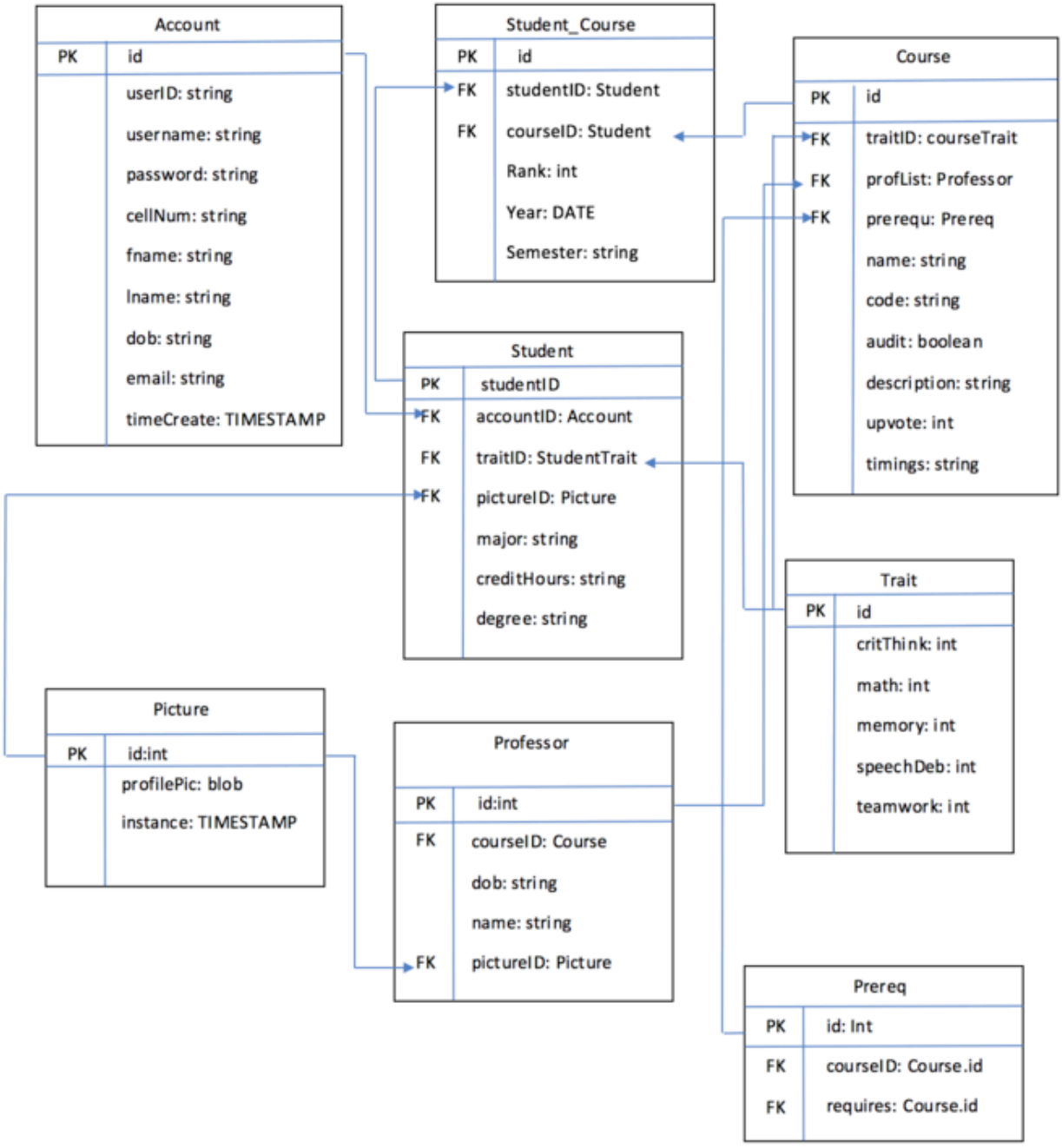

FIGURE 4: DATA CLASS LEVEL DESIGN

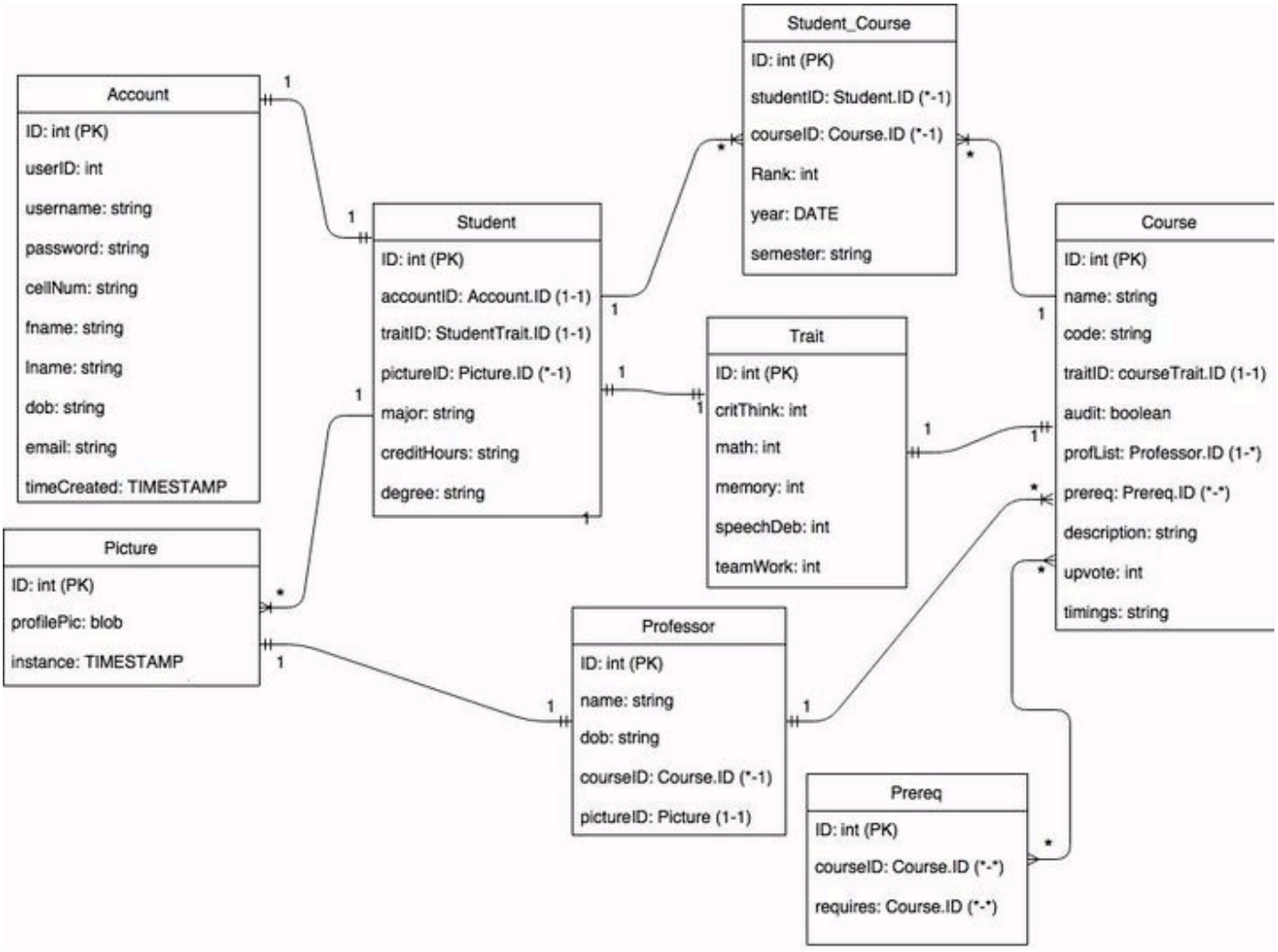

FIGURE 5: CLASS LEVEL DESIGN

Description of Data Class Level Design and Class Level Diagram: Above is a mockup of our database schema. This diagram may change a bit in the future as some columns may be moved around and tables may be added to support our features.

1. **Account:**

   This table handles each User's account information. Our users are only students. All of the data related to the Account such as the account handle, email and other personal student information are stored in the account table.

2. **Student:**

   The student table contains all the information which is stored in the Account table as well as all the information stored in the student trait table. The credit hours taken as well as the student's major is contained in the table.

3. **Course:**

   The Course table contains information related to a course such as the course code (CS 30700) and the course description. It holds a link to the Trait table such that for each course, there is a row in the trait table linking it. The professor teaching the course is also displayed as well.

4. **Professor:**

   The Professor table holds the details of a particular professor such as the name, date of birth, Picture and most importantly, all the courses which he is teaching.

5. **Student_Course:**

   The Student_Course table allows for many students and many courses to be in the table. This means that 1 student can be associated with many courses in this table and 1 course can have many students in this table. The table also keeps track of the year and semester in which 1 student took a particular course. The table serves as a way to model the many to many relationship between the Student table and Course table.

6. **Prerequisite:**

   The Prerequisite table holds information about a particular Course and a prerequisite for that Course. If a particular Course has 2 prerequisites, then we store each prerequisite individually on a separate row in this table. For example, if CS 30700 has CS

18000 and CS 25100 as prerequisites, the entries will follow as {(CS 30700, CS18000), (CS 30700, CS25100)}. This models a many to many reference between the Course table and Prerequisite table where 1 course can have many prerequisites, where a prerequisite in itself is in a Course table.

7. **Trait:**

   The Trait table contains the 5 variables which are used to give attributes to a particular course or a particular student. The trait table holds data for a Student as well as data for a Course. Each record can hold either student traits or course traits.

8. **Picture:**

   The picture table contains the student's profile picture and Time Stamp. One Student can have many pictures associated with it. This association allows us to provide a choice for a student to revert back to an old picture, if required. One Professor can have only one picture associated with it.

4.2 Sequence Diagram

4.2.1 Navigation Sequence Diagram

Our navigation emphasizes ease of data access. The course list page gives users rich data about the courses as well as the upvote count of the course. The top bar gives one click access user profile info where they can update their account settings and take surveys to get better recommendations. Course page gives easy access to various data points like grade trends and gives access to comment sub module where users can post and view comments.

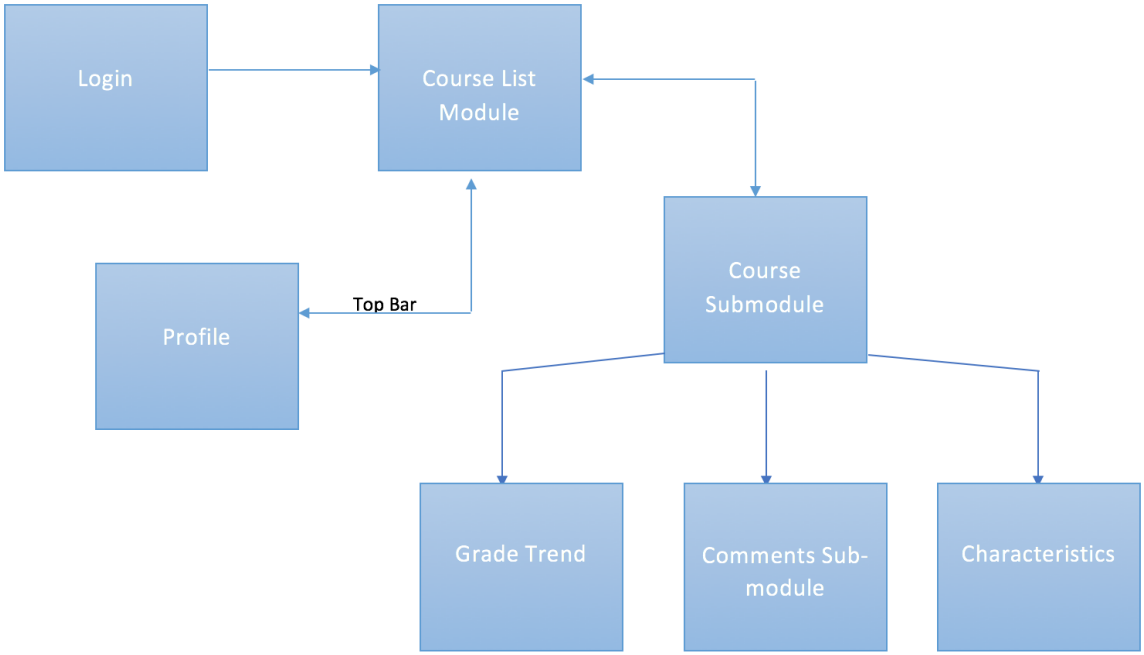

FIGURE 6: SEQUENCE DIAGRAM FOR OVERALL DIAGRAM

4.2.2 Add New Course Sequence Diagram

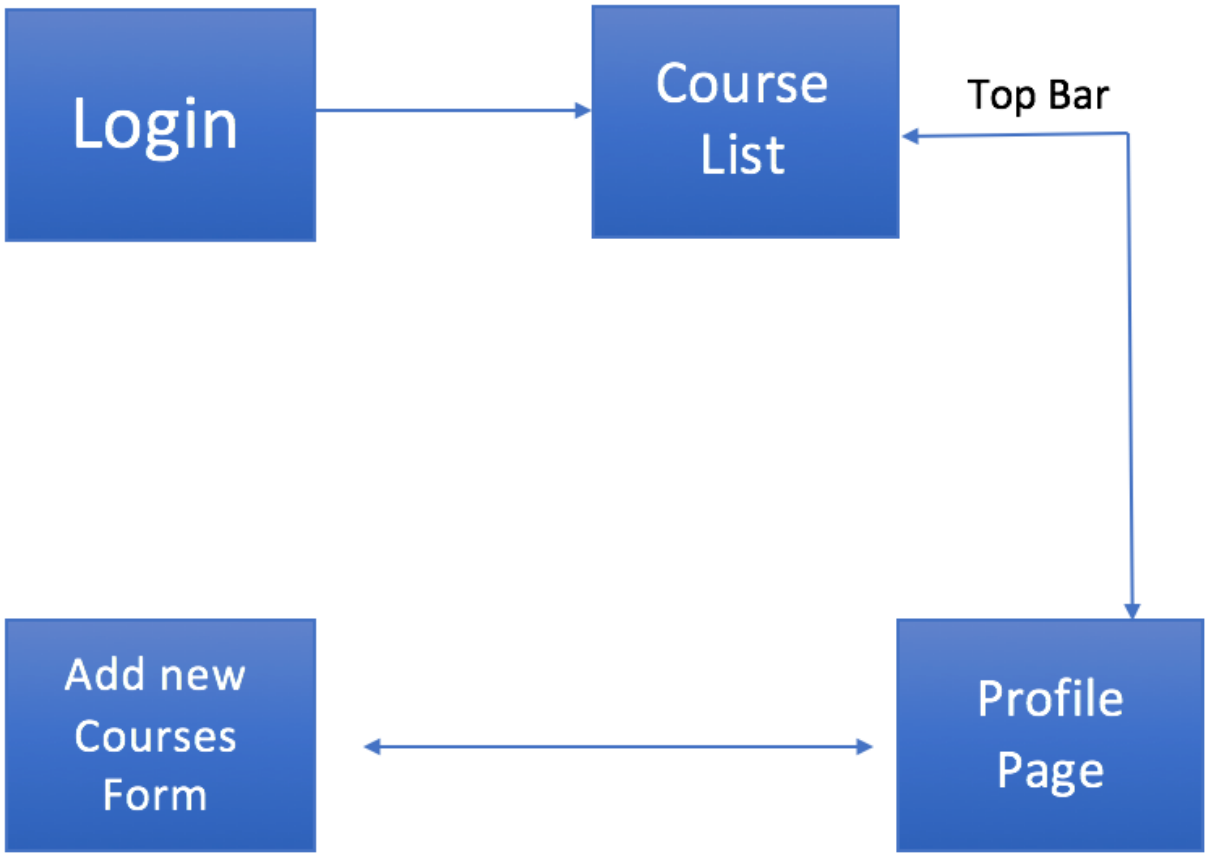

FIGURE 7: SEQUENCE DIAGRAM FOR NEW COURSE INPUTS

The figure 7 shows how to add new courses to our system. Based on the input courses and the survey data, the training and prediction model will create a tailored list.

4.4 UI Mockups

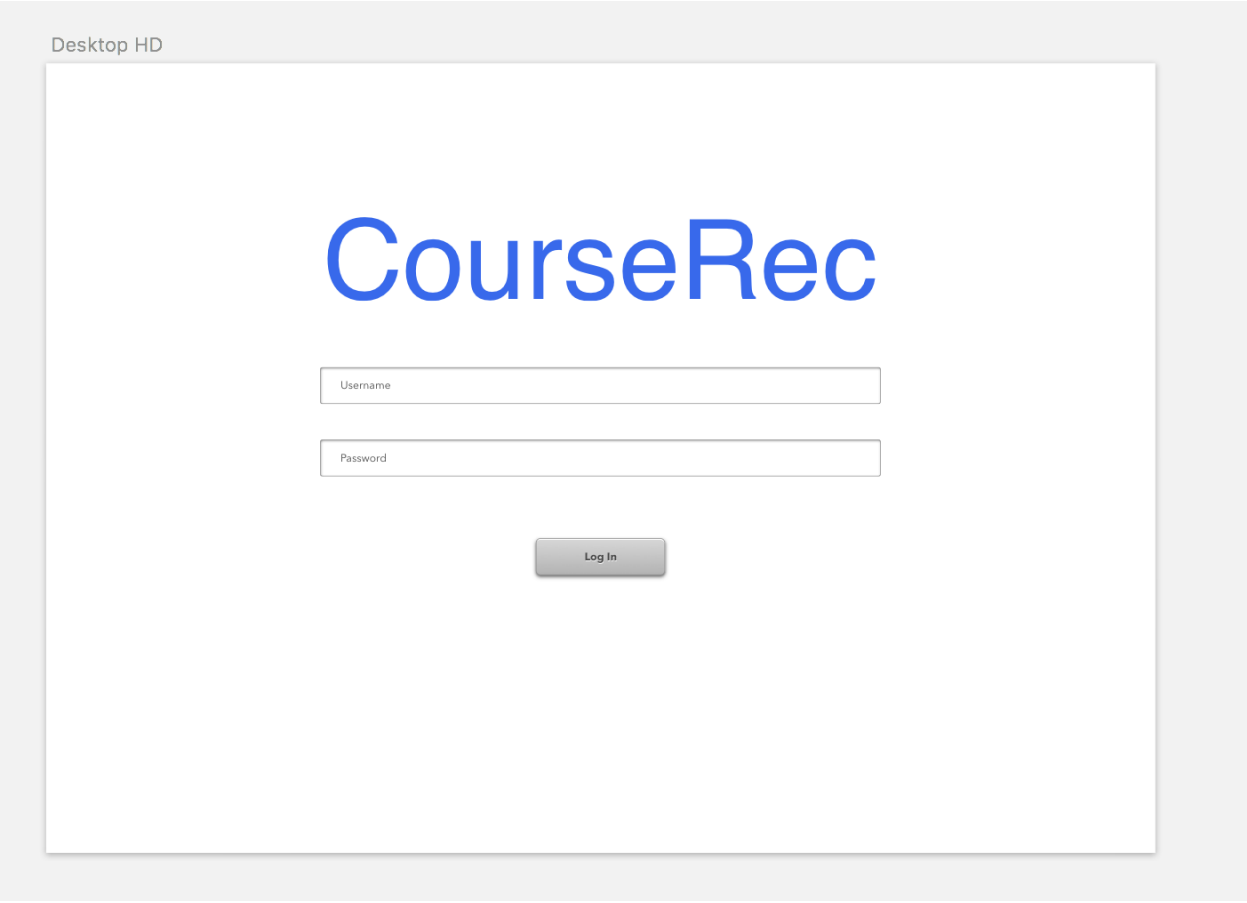

FIGURE 8: WEB LOGIN PAGE

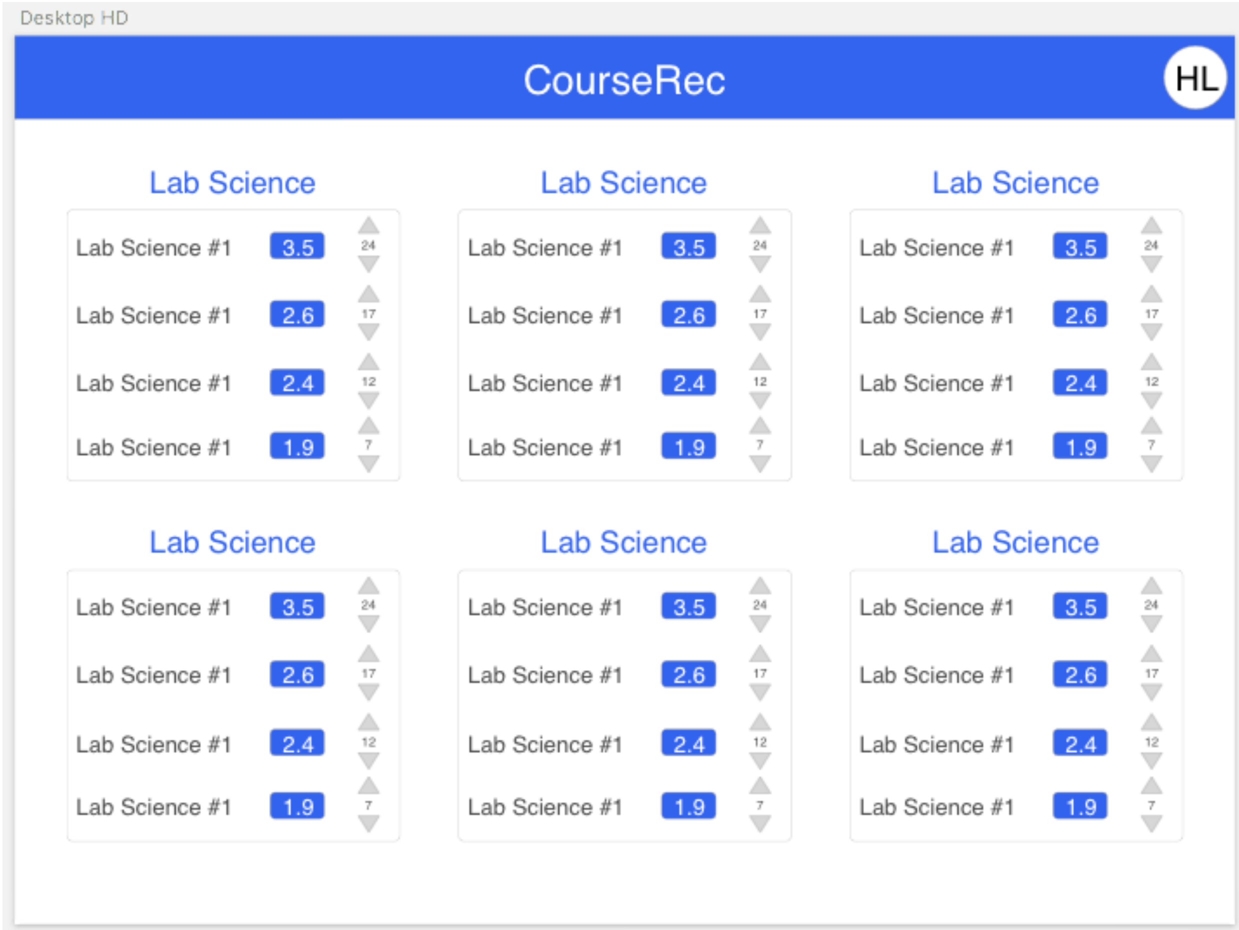

FIGURE 9: COURSELIST PAGE

This is the course list page, as well as it show various data points to the user. For each class, a user can down vote or up vote as well click to go towards a detailed course page.

FIGURE 10: COURSEDESCRIPTION PAGE

This is the course description page, as well as it show various data points to the user. For each class, it will show the course info, prerequisites, description and also a comments section for each class.

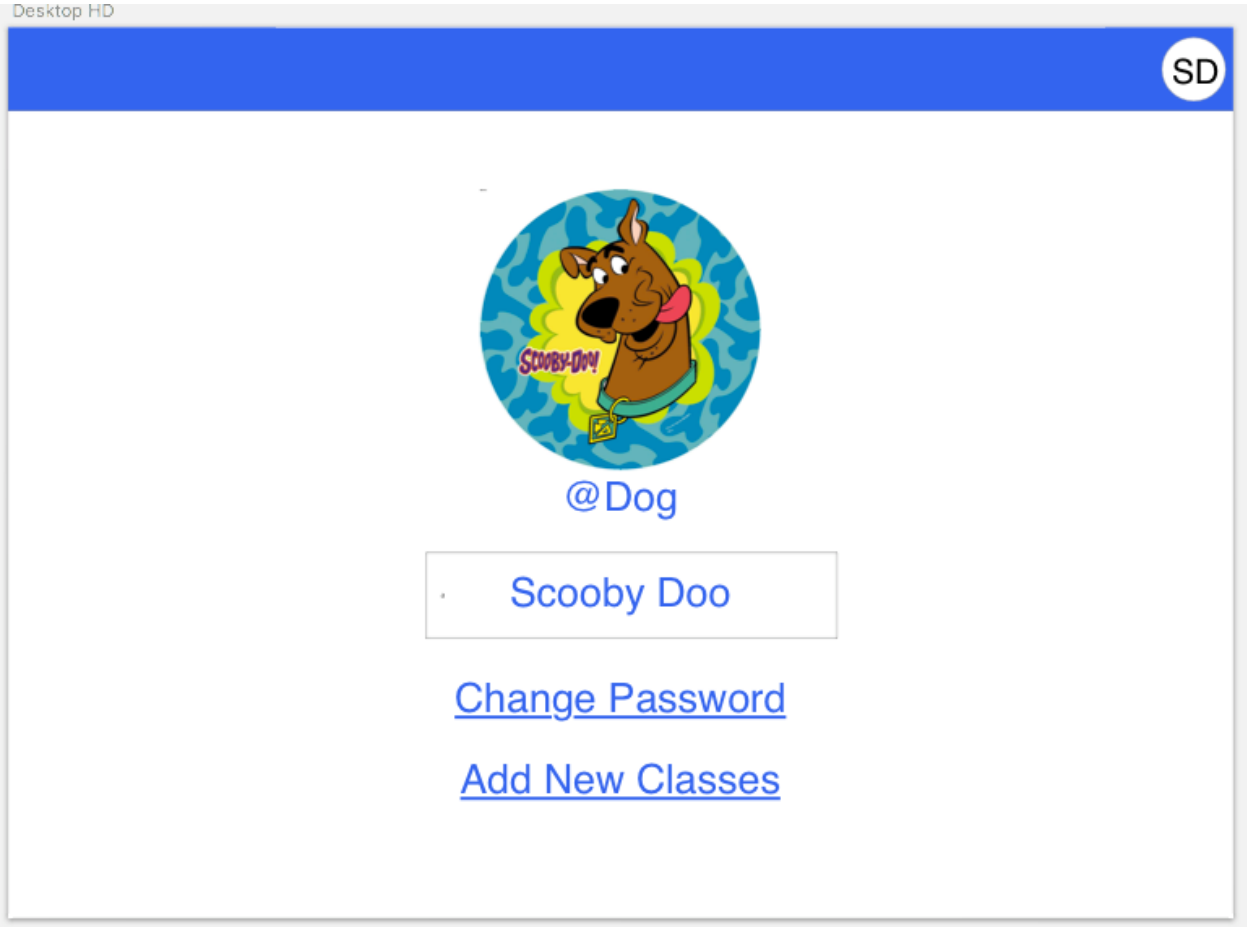

FIGURE 11: PROFILE PAGE