“Scheduling Assistant”

DESIGN DOCUMENT

Deep Desai, Eric Stephens, James Kelly, Vincent Pedata, Josiah Bell, and Jayson Daughtry

Rowan University

Table of Contents

[1.        Purpose 3](#_Toc27303629)

[2.        Scope 3](#_Toc27303630)

[3.        Exclusions, Assumptions and Limitations 3](#_Toc27303631)

[4.        Design Overview 3](#_Toc27303632)

[4.1.        Description of Problem 3](#_Toc27303633)

[4.2.        Basic Solution Description 3](#_Toc27303634)

[4.3.        Functionality 4](#_Toc27303635)

[4.4.        Conflict of Ideas 4](#_Toc27303636)

[4.4.1.        Database structure 4](#_Toc27303637)

[4.4.2.        Registration phase for advisors 4](#_Toc27303638)

[4.5.        Tech Stack 4](#_Toc27303639)

[4.6.        Page Design, I/O, & Purpose 5](#_Toc27303640)

[4.7.        Page Flow 15](#_Toc27303641)

[4.8.        Database Design 17](#_Toc27303642)

[4.9.        Restful Endpoints 19](#_Toc27303643)

[4.10.        Node.js 22](#_Toc27303644)

## 1.        Purpose

This document presents the Solution Development Lifecycle (SDLC) Design / Installation Information for Scheduling Assistant. The goal of this document is to give readers more of a technical view of Scheduling Assistant and it’s intricacies.

## 2.        Scope

In this design document for Scheduling Assistant, the topics that will be covered are:

●      The ideology behind the creation of the program

●      The program’s tech-stack

●      Mock-up designs of the web-pages, their input/output, and overall flow of the

program

●      The philosophies and reasoning of conflicts in the design and decision-

making process

●      Database & Server design

## 3.        Exclusions, Assumptions and Limitations

Exclusively CS major only

●     Scheduling Assistant Only currently works with the Computer Science degree. This is mainly due to time and experience constraints

## 4.        Design Overview

## 4.1.        Description of Problem

            Throughout the years, Rowan’s CS curriculum has grown exponentially. Along with that has come with an influx of CS students. However, despite the large amount of students pouring in, the amount of advisors have stayed constant. Being able to manage all students with the same amount of time they had originally dedicated to each student is becoming more and more difficult as each year passes. How are students supposed to accommodate for this shortcoming? This is where Scheduling Assistant comes in.

## 4.2.        Basic Solution Description

Students will be able to use this web-based program to have their X years of college courses planned out for them based on different determiners. These determiners include options such as a financially-considerate plan, a four/three/five/etc… year plan, semesters planned on the grades they’ve achieved from past classes, and so on. Now, when students meet with their advisors, they will be able to meet them with their next schedule already planned out, and more time can be spent on guidance rather than simply class selection.

## 4.3.        Functionality

            This software will be able to:

●     Recommend students what classes to take for each semester

●     Be able to plan a schedule based around grades, financial situation, how long they’d like to attend rowan, etc…

●     Advisors will be able to keep track of what classes are and aren’t offered

●     Students and advisors will be able to communicate through chat logs

●     The ability for a student to change their recommended schedules

## 4.4.        Conflict of Ideas

## 4.4.1.        Database structure

Initially we were having a conflict over how requirements of classes could be stored. One side said only mysql database and the other saying a combination of mysql and JSON files. In the end, we found that the database structure is mostly stored in MySQL tabular format. However, the structure of the data is going to be represented as JSON using RESTful API. This is because it creates a loose coupling where multiple clients can access the data from the server independent of each other. Therefore we can leave it to just a MySql database

## 4.4.2.        Registration phase for advisors

            Initially for our login phase, there were two schools of thought. Either have the user go through a verification system, or just have the user select whether they are or aren’t an advisor. While both options aren’t very good, they would work. However, there was a potential third option we hadn’t considered at the moment, but discovered later in the designing process. This idea was to analyze the user’s email, and see if it were a @students or an @rowan. If it was @students, it’s a student user, and if it’s @rowan, they must be an employee, thus be an advisor. The discovery of this idea led to the resolution of this issue.

## 4.5.        Tech Stack

Below are all the different libraries and APIs we will use to create our program:

●     RESTful API

○     The webpage uses RESTful API to separate the client-server applications. RESTful API allows multiple users to pull information from the database without creating any constraints between the users. Most of the users, course, and degree information are Created, Read, Updated, and Deleted (CRUD) through RESTful API.

●     Node.js

○     This will be used in the form of the express.js framework. Node will be the web server as well as the home to the restful API’s

●     Passport.js

○     The website will need to have a form of authentication as well as session handling. Passport will be the javascript framework used for this.

●     Google OAuth

○     In combination with passport.js Google’s OAuth authentication is what will be used to authenticate users. The sign in process will use a google account sign in since Rowan’s student and faculty accounts are all google accounts.

●     Amazon Web Service Virtual Machine

○     The website will be created on an AWM VM. This allows the deployment of the website to be consistent. The same machine will run each iteration of the code base. This guarantees that testing is consistent for each release.

●     Socket.io

○     A library that enables real-time, bidirectional and event-based communication between the browser and the server. It will be running on node.js to create a chat system between the advisors and students.

●     MySQL

○      For this program, we’ll be using mysql for the backend database. This will keep track of several things such as:

■     A user’s username / password

■     All courses available as well as their attributes

●     courseID, course name, etc…

■     Each user’s chat logs

●     Vue.js

○     We will utilize Vue to develop the frontend of our website. We will use Vue.js tools such as:

■     Vuetify to design the webpage

■     VueX to aid in maintaining state across webpages

●     HTML

○     In addition to Vue we will be utilizing HTML for the frontend with:

■     CSS to aid in formatting and styling the webpage

■     JavaScript to make the webpage more interactive

## 4.6.        Page Design, I/O, & Purpose



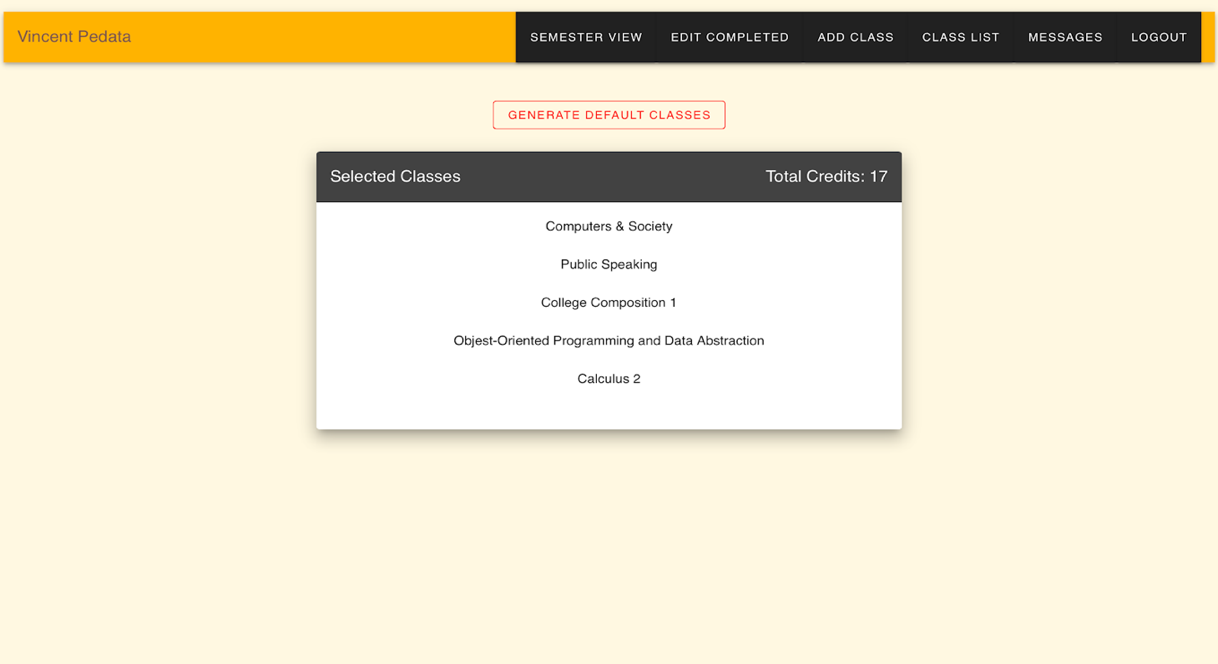
**Login Page**

[Login With Google] Button

* Opens a Google Account Login page.
* The account selected will be stored into the database if it does not already exist.
* Student users will be taken to the “Student Section” of the site and Advisor users will be taken to the “Advisor Section” of the site.

**Student Section:**

**Student View (Home)**

****

*This page serves as a hub for student navigation. It also displays currently selected courses for the student and allows the student to quickly generate a new schedule.*

[Generate Default Courses] Button

* This button generates a list of suggested courses using the data the user entered in the Student Edit Completed Courses page. This list of courses become ‘selected’ and these selected courses are visible to the advisor.

Selected Courses List:

* This list displays all ‘selected’ courses.
* Clicking one of these courses will open a pop-up displaying more information about the course.

*These following buttons are sometimes available on other student pages on the upper toolbar:*

[Semester View] Button

* This button takes you to the Semester View page.

[Edit Completed] Button

* This button takes you to the Student Edit Completed Courses page.

[Add Course ] Button

* This button takes you to the Student Add Course page.

[Course List] Button

* This button takes you to the Student Course List page.

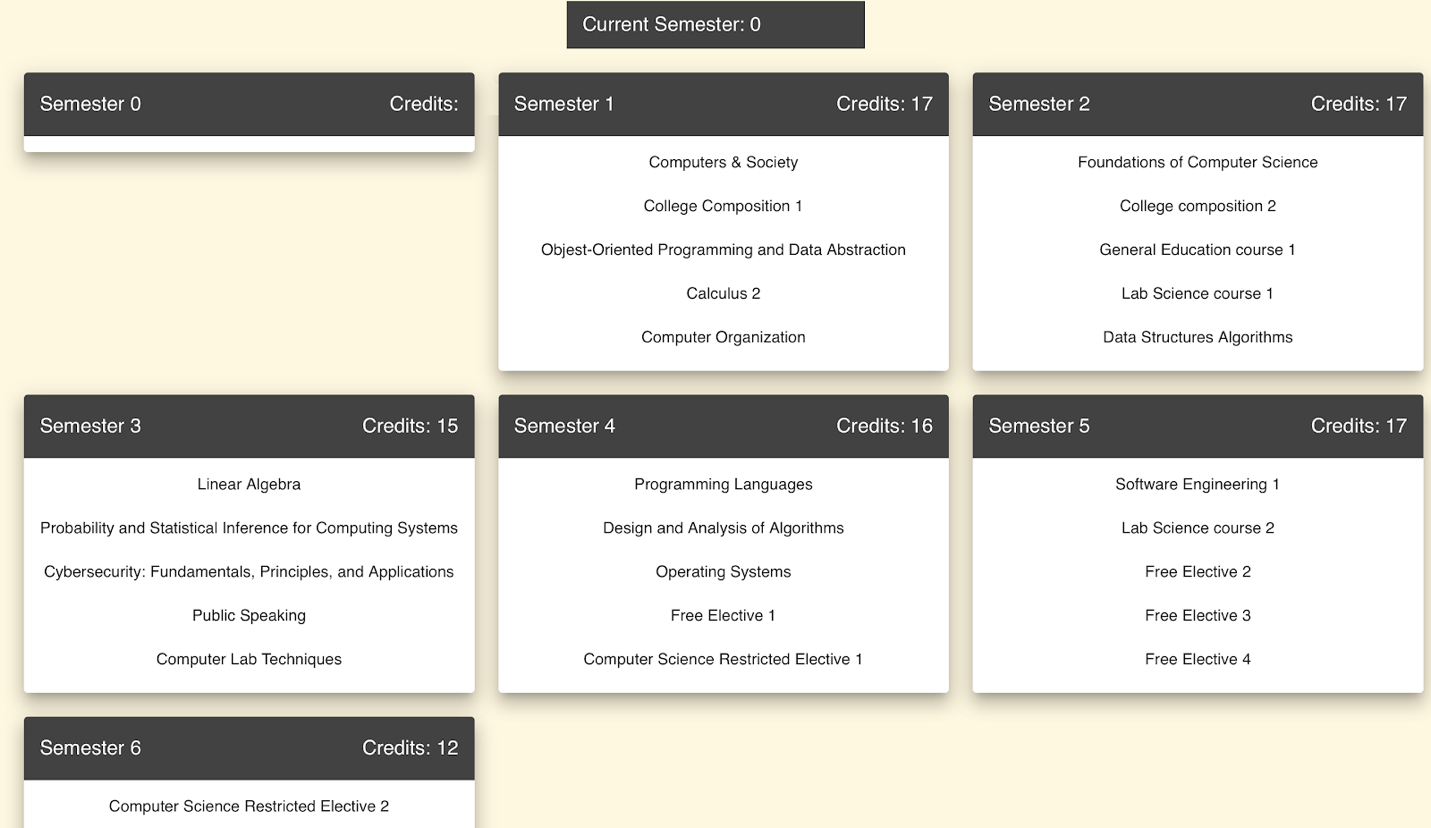
[Messages] Button

* This button takes you to the Student Messenger page.

[Logout] Button

* This button logs out the user and returns to the Login Page.

**Semester View**

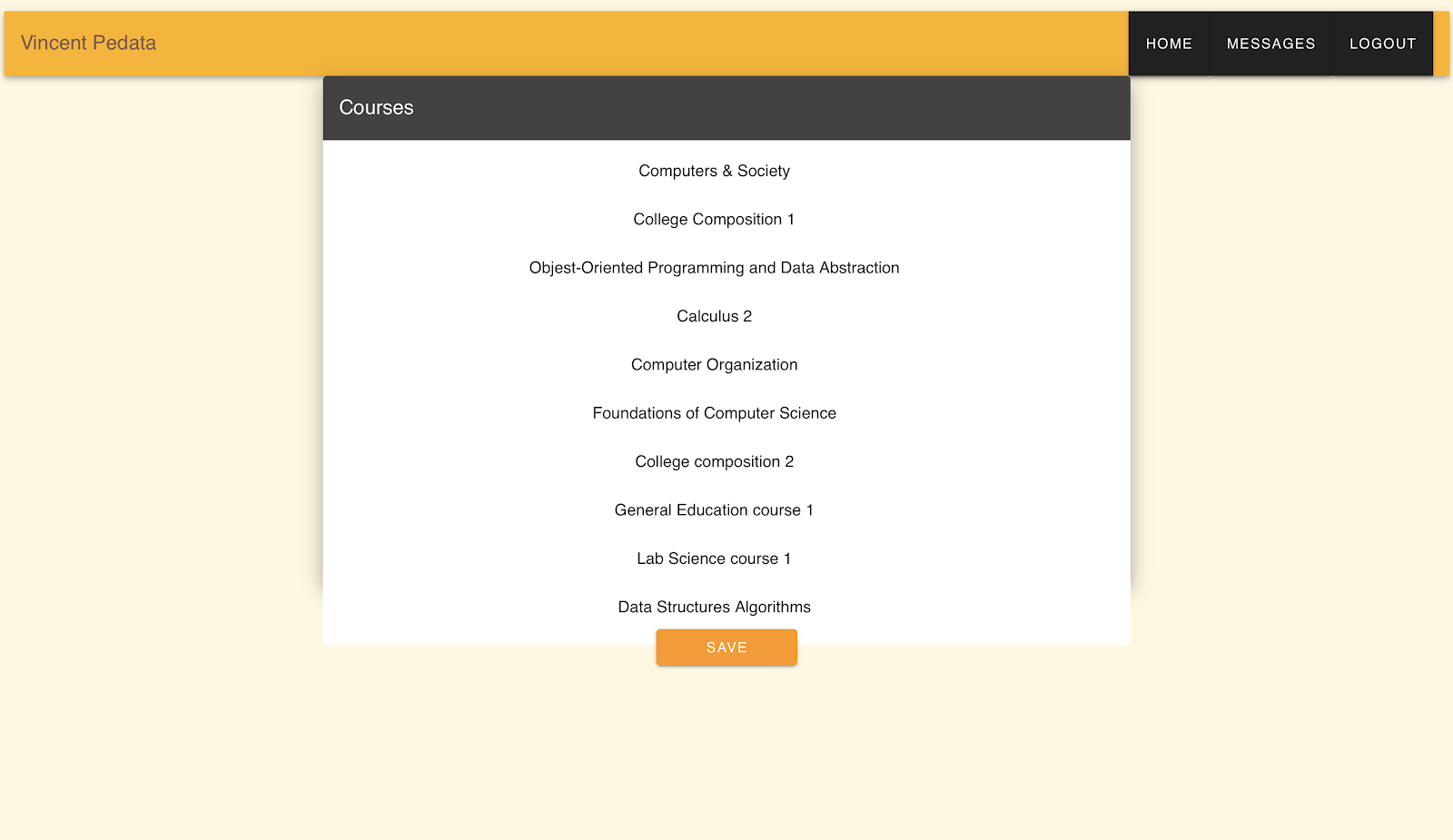


*The purpose of this page is to display an anticipated schedule for a student’s education.*

Semester Lists:

* These lists show an anticipated course list for a given semester.
* Courses in the list are selectable but not modifiable directly.
* Changing completed courses in Student Edit Completed Courses will change the semester list results.

**Edit Completed Courses**



*The purpose of this page is to allow a student to modify what courses they had already completed.*

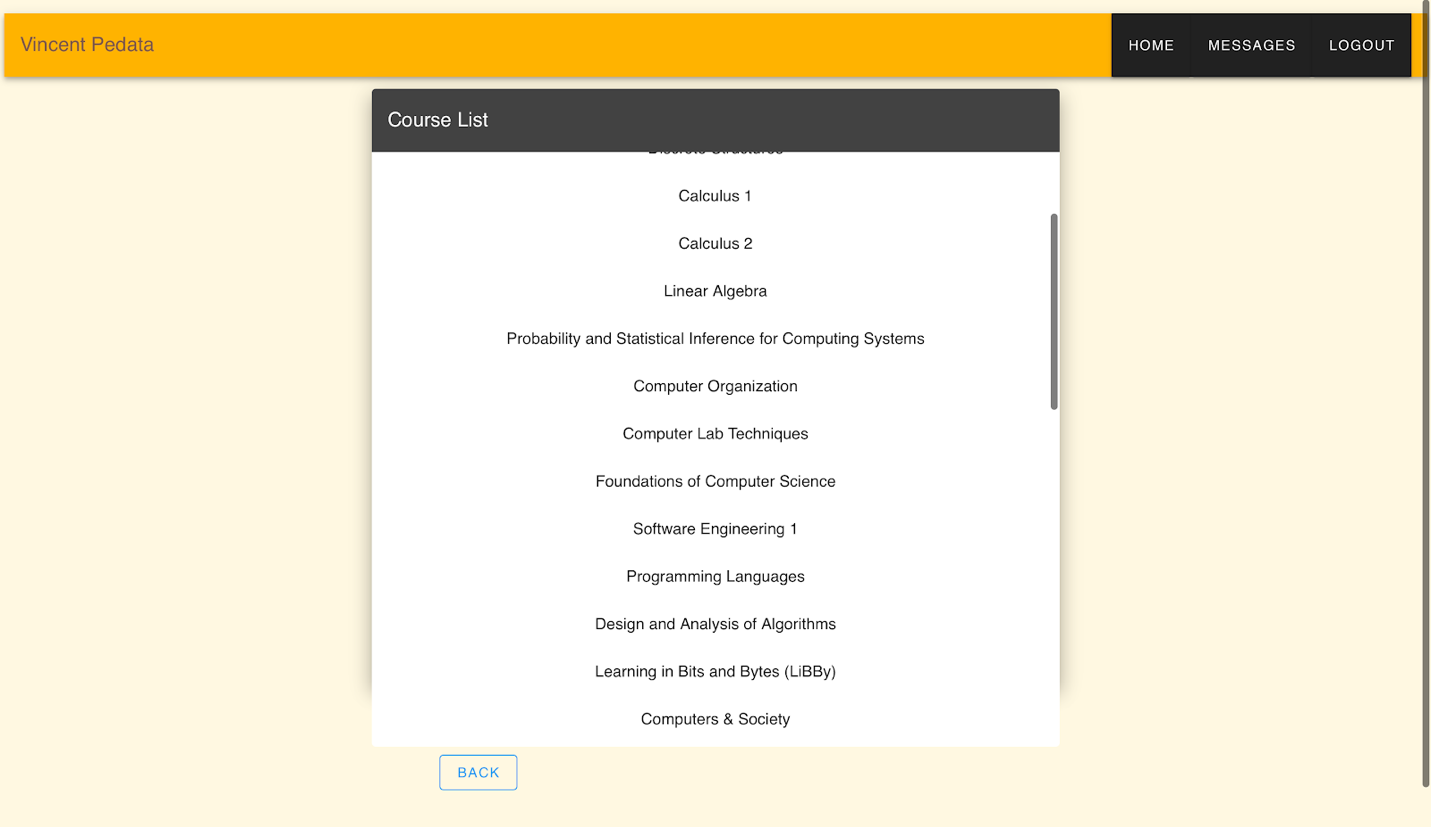
Courses List:

* This list displays every course available.
* Selecting a course displays more information and allows the student to select what grade they had received in the course.

[Save] Button:

* This button saves the data stored about completed courses to the database.

**Student Course List**

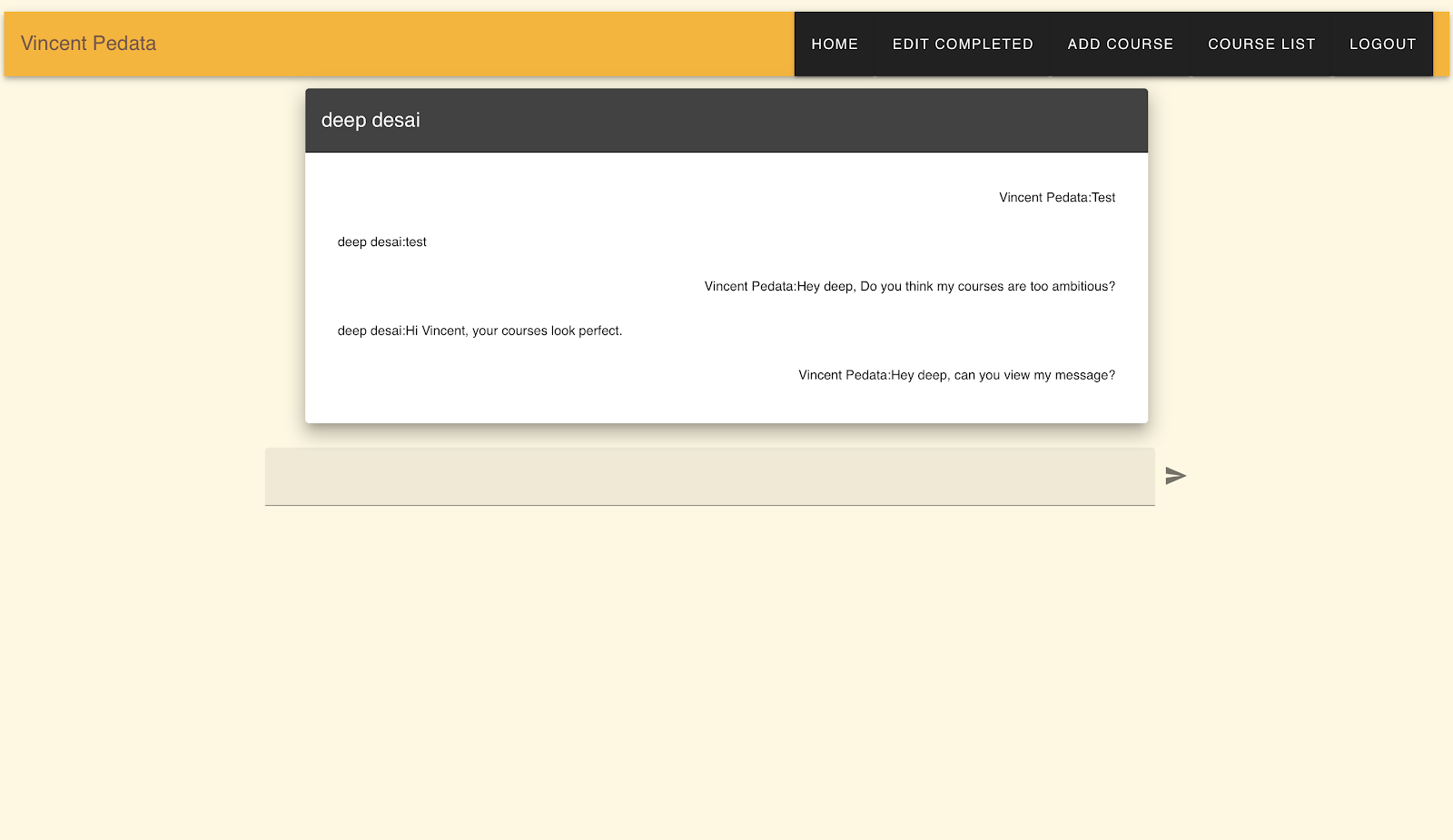


*The purpose of this page is to allow students to view every course in the database.*

Course List:

* This is a list of all courses.
* Clicking a course will open a pop-up displaying more information about the course.

**Student Messenger**

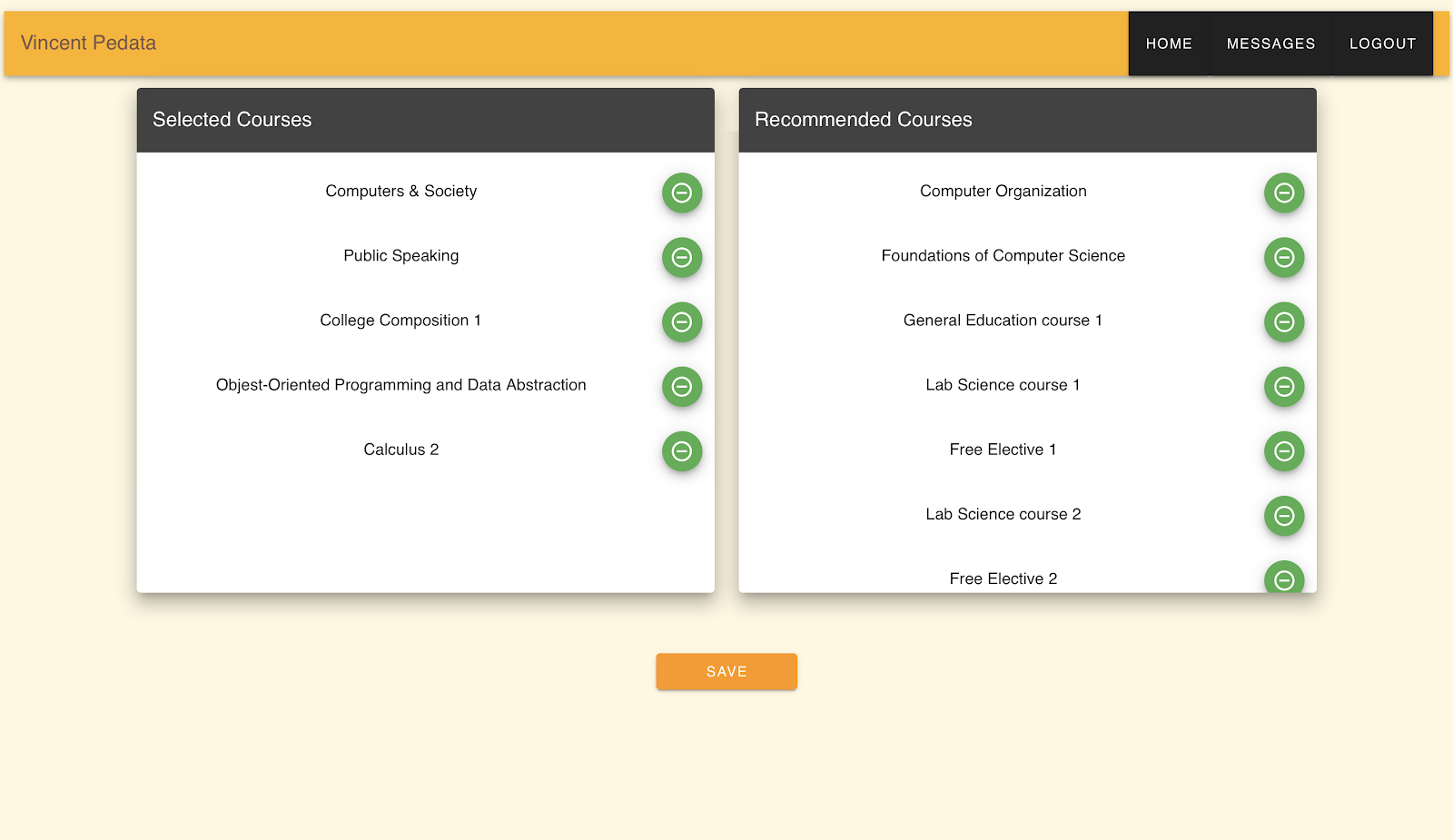


*The purpose of this page is to give students live communication with their advisor.*

Chat:

* Select the text box to begin to write a message.
* Click the arrow button to send a message.

**Student Add Course**



*The purpose of this page is to allow students to manually add/remove courses from their currently selected courses.*

Selected Courses List:

* This is a list of currently selected courses.
* Clicking the symbol beside the name will transfer this course to the “Recommended Courses” list and it will no longer be ‘selected’.
* Clicking the title of the course will open a pop-up displaying more information on the course.

Recommended Courses List:

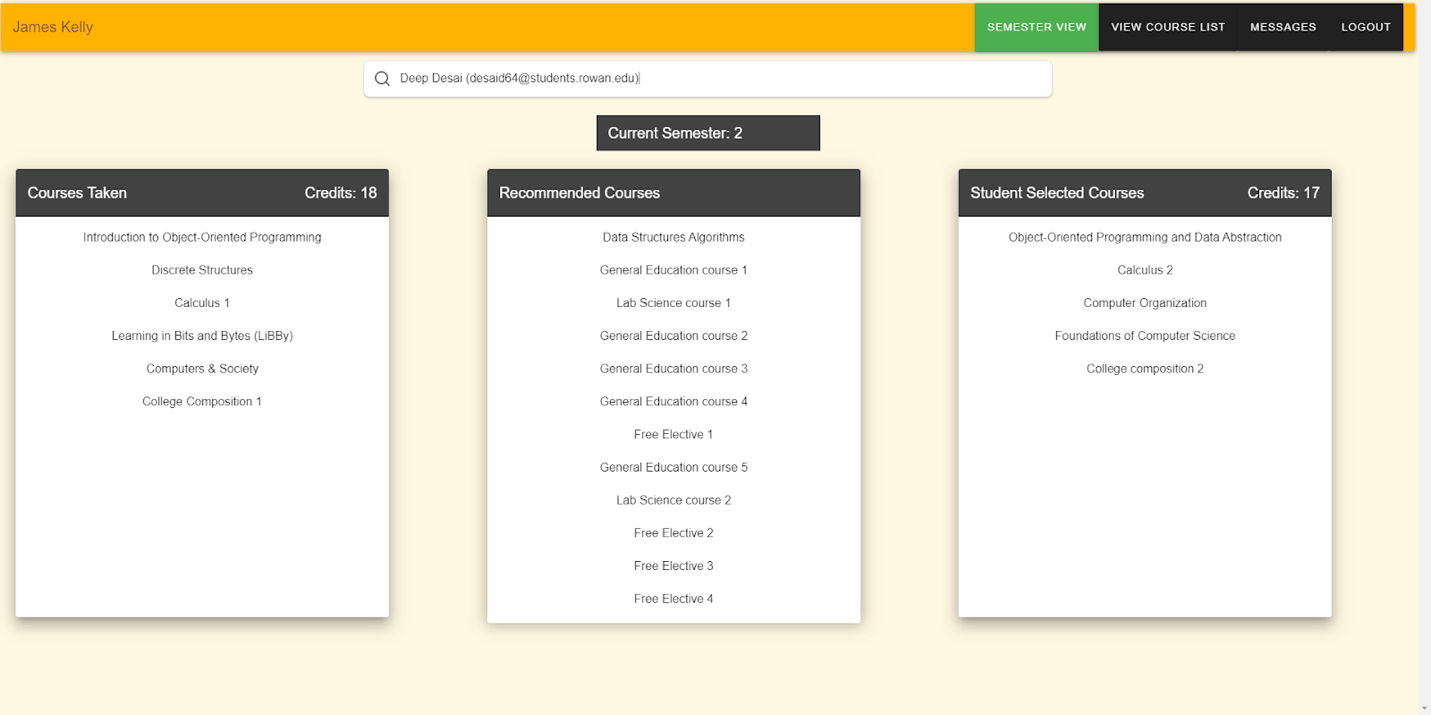
* This is a list of currently unselected but still recommended courses.
* Clicking the symbol beside the name will transfer this course to the “Selected Courses” list and it will become ‘selected’.
* Clicking the title of the course will open a pop-up displaying more information on the course.

[Save] Button:

* This button saves the data stored about selected courses to the database.

**Advisor Section:**

**Advisor View (Home)**



*This page serves as a hub for advisor navigation. It also allows the advisor to display selected courses for any student.*

Student Search Bar

* This search bar allows the advisor to look up students by name.
* Searched students also have their email displayed to ensure the correct student is being found.
* Finding and selecting a student through the search bar will fill the lower three lists with data about that student’s courses.

Courses Taken List

* This is a list of courses that the searched student has already completed.
* Selecting a course will display more information about that course.

Recommended Courses List

* This is a list of courses that the student has not yet taken but are recommended.
* Selecting a course will display more information about that course.

Student Selected Course List

* This is a list of courses that the searched student has selected for the upcoming semester.
* An advisor should consider whether or not these selected courses would be a good choice for the given student.
* Selecting a course will display more information about that course.

*These following buttons are sometimes available on other advisor pages on the upper toolbar:*

[View Course List] Button

* This button takes you to the Advisor Course List page.

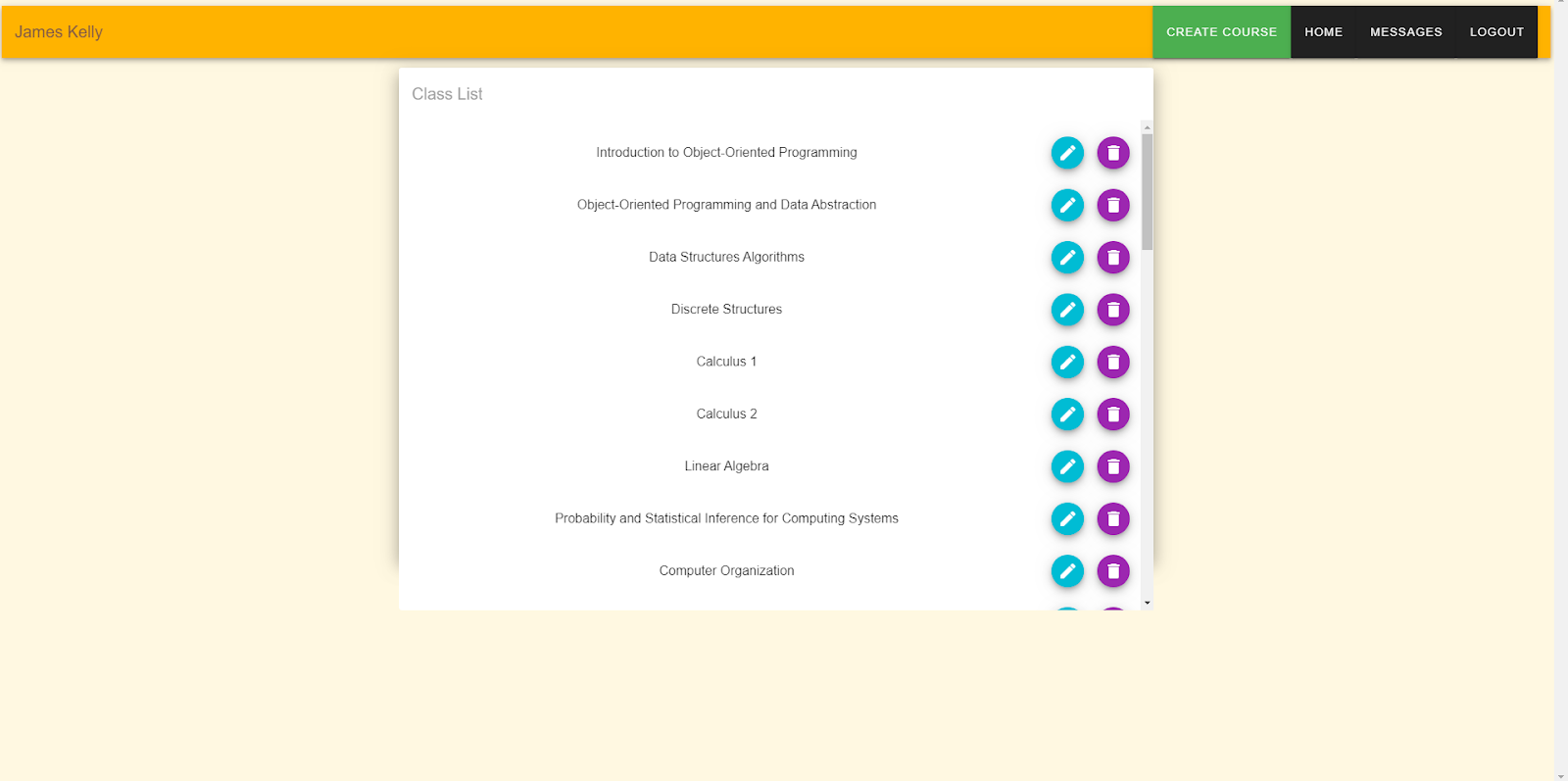
[Messages] Button

* This button takes you to the Student Messenger page.

[Logout] Button

* This button logs out the user and returns to the Login Page.

**Advisor Course List**



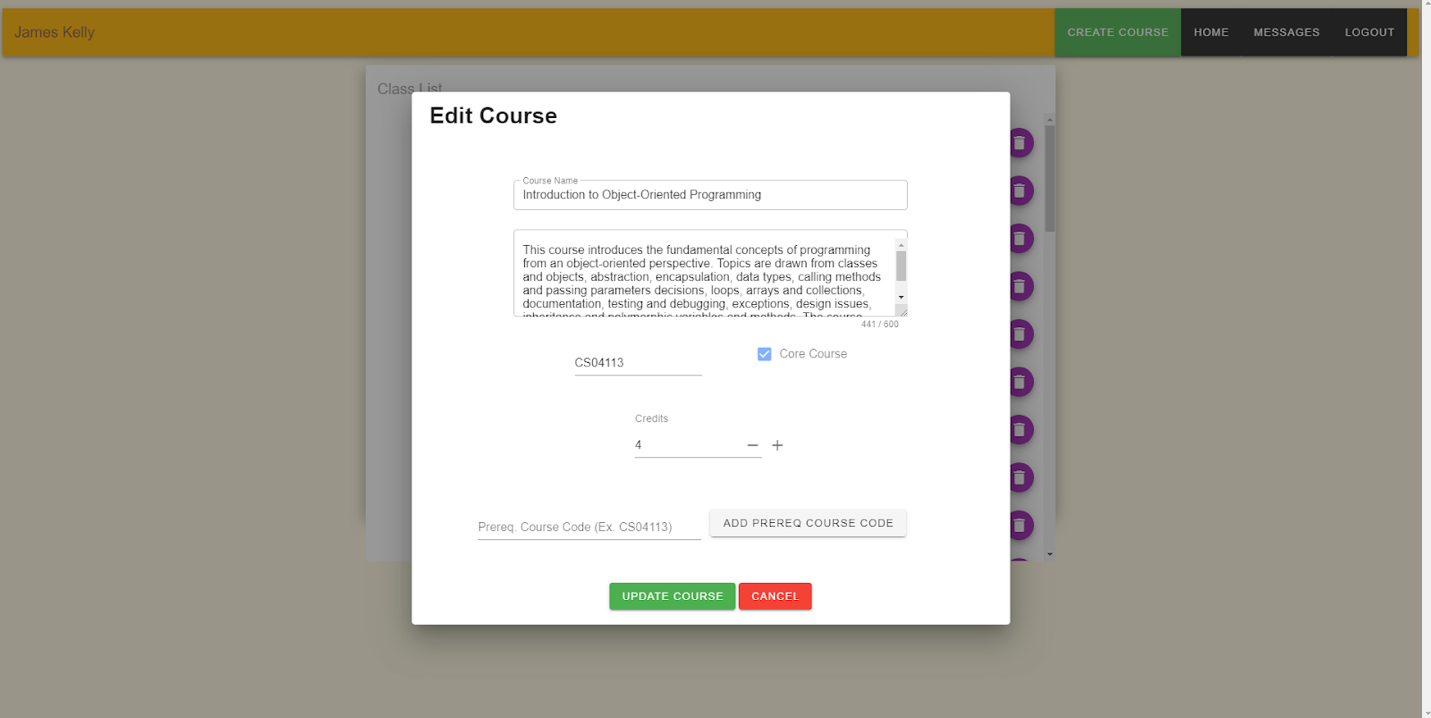
*The purpose of this page is to allow advisors to view every course in the database and modify it if necessary.*

[Create Course] Button

* This button creates a pop-up that allows the advisor to create a new course.
* Creating a course requires the following information:
* Course Name
* Course Description
* Course Code
* Core (Whether or not the course is core/required)
* Credit Hours
* Prerequisite Course Codes (Which must exist already in the course list)

Course List:

* This is a list of all courses.
* Clicking the pen button will open an ‘Edit Course popup. This is in identical format to the Create Course popup, but previously-input data will be auto-filled.
* Clicking the trash-can button will delete the course. Be careful not to delete courses that are prerequisites for other courses.
* Clicking a course will open a pop-up displaying more information about the course.

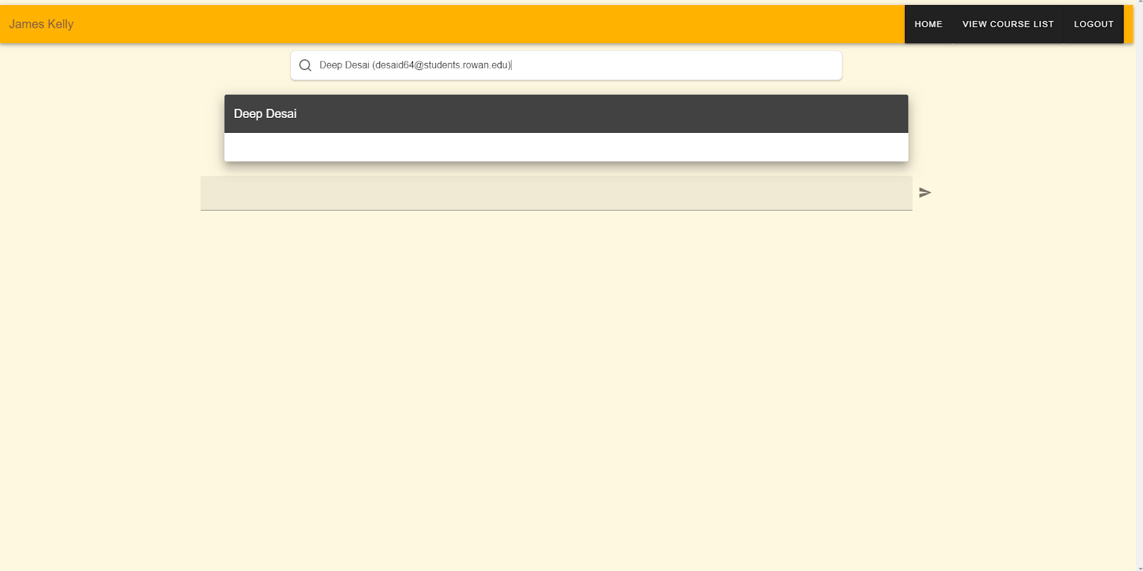


**Advisor Messenger**

*The purpose of this page is to give advisors live communication with their students.*

Student Search Bar

* This search bar allows the advisor to look up students by name.
* Searched students also have their email displayed to ensure the correct student is being found.

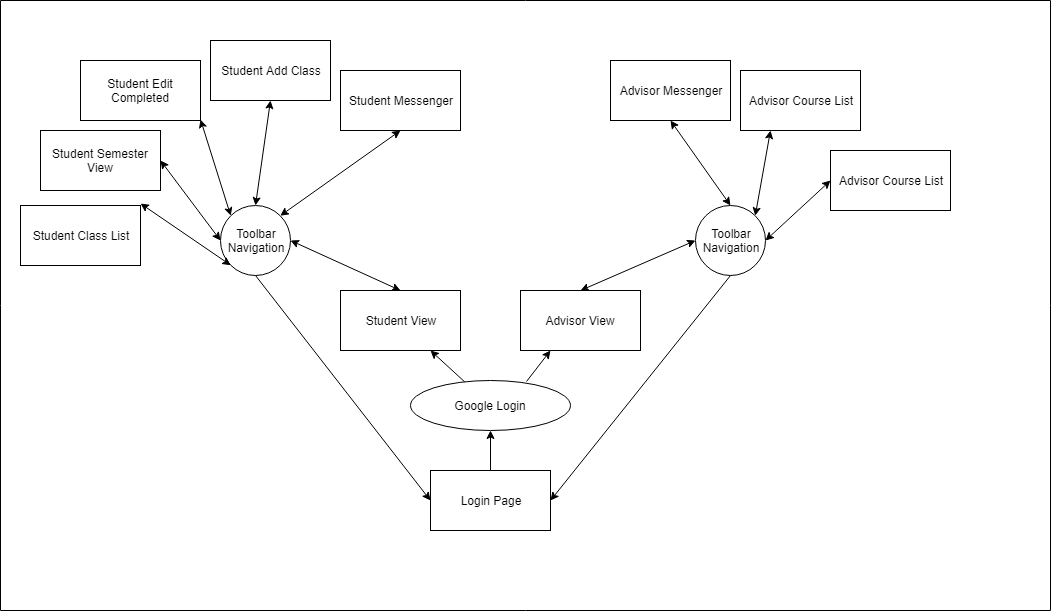


* Finding and selected a student through the search bar will open a chat with the student.

Chat

* Select the text box to begin to write a message.
* Click the arrow button to send a message.

## 4.7.        Page Flow

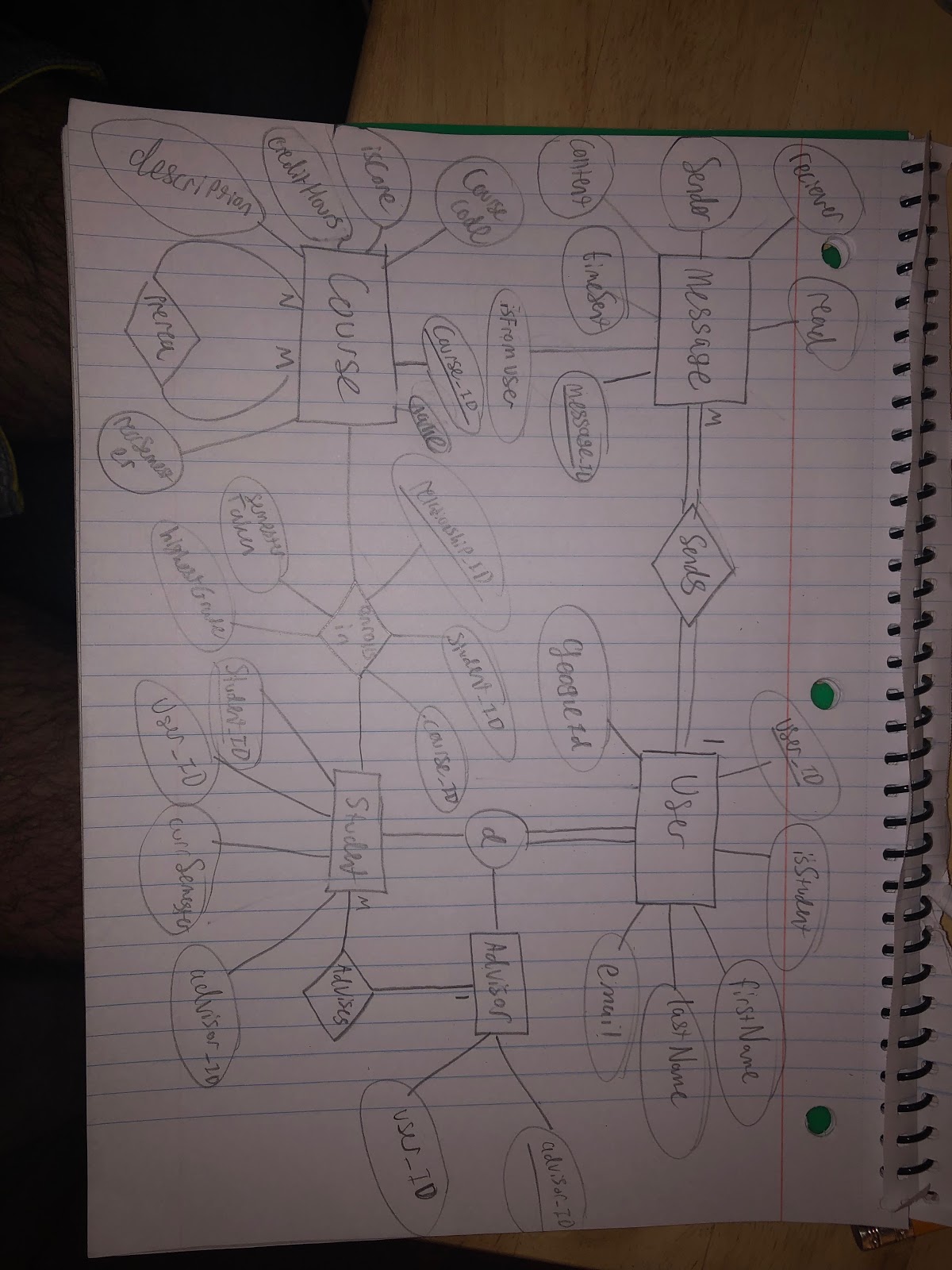


## 

## 

## 

## 4.8.        Database Design



## 4.9.        Restful Endpoints

The endpoints are labeled in the form of REQUEST TYPE /endpoint/url/. Inputs to an endpoint are denoted by brackets. For example, {user\_id} expects an integer, {subject\_name} expects a string. Additional parameters will be handled in through query strings and/or form data (for post requests).

* GET /course/all
  + Returns a table of all the courses available in the database as JSON.
* GET /course/number/{course\_code}
  + Returns the information about the course corresponding to course\_code in form of JSON.
* GET /course/{course\_id}
  + Returns the information about the course corresponding to course\_id in form of JSON.
* GET /user/name
  + Returns the user name given by the google profile.
* GET /user/{user\_id}
  + Returns the user’s information corresponding to the user\_id in form of JSON. In addition, the type\_id is a hidden command which tells if the user is advisor or student and it automatically generated based on the user’s information after the user has log-in.
* GET /user/all
  + Returns all user’s information stored in database as JSON.
* GET /user/getmyid
  + Returns the user ID of the current user in session in the form of JSON.

**For Students Only**

* GET /user/student/info
  + Returns the info of the current student logged in as JSON.
* GET /user/student/getmyadivsor
  + Returns the advisor of current student logged in as JSON.
* GET /user/student/getmyCurrSem
  + Returns the current semester for the student logged in as JSON.
* POST /user/student/courses/taken
  + Add courses completed by a student as JSON to the database.
* GET /user/student/courses/taken
  + Returns the student's courses completed in form of JSON.
* GET /user/student/courses/recommended
  + Returns a list of courses recommend in form of JSON.
* GET /user/student/courses/getUserRecommed
  + Returns a list of courses selected by the student from recommend courses in form of JSON
* DELETE /user/student/courses/deleteNegOnes
  + Delete the courses selected by the student to take for next semester. Returns false if the list is empty.
* GET /user/student/courses/yetToTake
  + Returns a list of courses that are left to take in form of JSON.
* GET /user/student/courses/yetToTakePreReqs
  + Returns a list of courses that are left to take with prerequisites in form of JSON.
* POST /user/student/messages/new
  + Create a new message corresponding to the specified student. Returns the message information as JSON.
* GET /user/student/messages/all
  + Returns a list of messages corresponding to the student in form of JSON.

**For Advisors Only**

* GET /user/advisor/info
  + Returns the info of the current advisor logged in as JSON
* POST /user/advisor/course/
  + Create a new course in the database by the advisor.
* PUT /user/advisor/course/
  + Updates an existing course in the database by the advisor.
* DELETE /user/advisor/course/{course\_id}
  + Delete a course from the database with the corresponding course\_id. Returns false if none exists.
* GET /user/advisor/student/all
  + Return a list of all the student under the current advisor  in form of JSON.
* GET /user/advisor/student/curSem/{student\_ID}
  + Return the current semester by the student\_ID for the advisor in form of JSON.
* GET /user/advisor/student/{student\_ID}/taken
  + Returns the student's courses completed by the student\_ID for the current advisor in form of JSON.
* GET /user/advisor/student/{student\_ID}/recommended
  + Returns a list of courses recommended by the student\_ID for the current advisor in form of JSON.
* GET /user/advisor/student/{student\_ID}/studentRecommeded
  + Returns a list of courses selected by the student from recommend courses by the student\_ID for the current advisor in form of JSON
* GET /user/advisor/student/takenCredits/{student\_ID}
  + Returns the total credits completed by a student\_ID in form of JSON.
* GET /user/advisor/student/{student\_ID}/yetToTake
  + Returns a list of courses that are left to take by the student\_ID for the current advisor in form of JSON.
* GET /user/advisor/student/{student\_ID}/yetToTakePreReqs
  + Returns a list of courses that are left to take with prerequisites by the student\_ID for the current advisor in form of JSON.
* POST /user/advisor/messages/new
  + Create a new message corresponding to the specified student. Returns the message information as JSON.
* GET /user/advisor/messages/{student\_ID}
  + Returns a list of messages corresponding to the student\_ID in form of JSON.

## 4.10.        Node.js

Node.js will be implemented using Express.js and passport.js. Express.js will handle the web server as well as the RESTful APIs and database connection. Passport.js will be utilized in order to authenticate users using Google’s OAuth authentication as well as session handling.