Electronic Program of Study

Design Document

Team 15
Client: Tina Prouty
Academic Advisor: Md Maruf Ahamed

Saljooq Altaf (Coordinator/Full-Stack Developer)
Nathan Marquardt (Front-End Developer)
Noah Nickel (Front-End Developer)
William Hunt (Back-End Developer)
Carson Campbell (Back-End Developer)

sddec22-15@iastate.edu
http://sddec22-15.sd.ece.iastate.edu/

1. Overview

1.1 Acknowledgement

We want to give a special thanks to our client Tina Prouty and our academic advisor Md Maruf Ahamed for all the support and advice they have given our team throughout the project.

1.2 Problem and Project Statement

Incoming students are required to create a Program of Study which dictates what classes should be taken in the student's 4 years at ISU. The current method that Engineering intro classes use is a spreadsheet that must be manually updated and checked by both the student and advisor. This requires compiling resources from many different online sources that ISU provides which can be confusing and lead to significant problems that the advisors have to address. A better way of organizing this Program of Study would be to compile these resources into one application that can perform automatic checks for the validity of the plan. This would take a lot of pressure off of both the student and advisor as they would not have to compile and hand check the necessary resources.

1.3 Intended Users and Uses

The current iteration of our project will have students being the main users of this web application. Following are the core use-cases that have been focused on organized by the view:

- Student View:
    - Students will have the ability to see a list of current Programs of Study (POS) that they created.
    - After selecting a Program of Study, students will be able to use one of the 3 buttons on the screen.
    - Interactions with the selected POS can be creating a new Program of Study, opening the selected Program of Study, or deleting the selected Program of Study.
- Drag and Drop View:
    - Students will be able to use a drag and drop list to search for their desired courses. These courses can be searched with filters of course levels by their course number and department.
    - Once a student finds a class, they can add it to their Program of Study by clicking an "Add Class" button
    - Students should be able to add up to 12 semester containers in one Program of Study
    - Students should be able to delete whole semesters

- Once a class is in a Program of Study Semester, they should be able to move course items within the semester and to other semesters.
- Students should be able to click on a course item to gain more information about the course and to also delete a course item from their Program of Study via a button if they wish
- Students should be able to click a “Save POS” button to save the current iteration of their Program of Study
- Students should be able to save a copy of their Program of Study to their local machine by clicking on a Save to PDF button

1.4 Functional Requirements

- f1. A user should be able to create a new POS from their home page.
- f2. A user should be able to search and add classes to a POS.
- f3. A user should be able to save a POS to the server for later perusal or editing.
- f4. A user should be able to load and edit an already existing POS.
- f5. A user should be able to save their POS to a PDF format.

1.5 Non-Functional Requirements

- n1. The website should feature authentication to better protect users’ data.
- n2. The drag and drop interface should be intuitive and easy to use.
- n3. Course data should be comprehensive and up-to-date with the current catalog
- n4. Users should be able to access the website without needing a VPN

1.6 Standards

Work Competence (SE Code of Ethics 2.01) - Since most of the students working on the project are primarily Software Engineers - their technical expertise are suited to a large scale web-development software that uses complex frameworks like Angular.

Financial Responsibility (SE Code of Ethics 4.04) - We have made sure that we only make use of open source software and libraries to ensure that the costs to the client is minimal - and have made all financial decisions transparently.

Communication Honesty (SE Code of Ethics 1.04) - There are likely to be a lot of hurdles in implementing the new features that the client has in mind which may not be easily integrated into

the project and being honest and transparent about these hurdles is very important for our team to deliver a realistic product with certain limitations that our client understands.

Property Ownership (SE Code of Ethics 2.03) - Since we have been given access to virtual servers and resources for the purpose of creating this product for the client, we have restricted any use of such resources for personal use.

1.7 Engineering Constraints and Requirements

Constraints:

Scope - All work that entails creating an application that students will use to create POS and advisors will use to review them must be completed with a team of 5 engineers
Time - We would need to get the project finished up within one year i.e. before the second semester ended
Cost - As a senior design project, we had no source of funding.

Requirements:

See sections 1.4 and 1.5 for functional/non functional requirements.

2. Design

2.1 Previous Work and Literature

The previous senior design team (sddec21-04) previously designed a version of this software. However, their product was severely lacking in both functionality and maintainability. As a team, we came to a collective decision that it would be more fruitful to make a new software from scratch rather than updating the old design as was previously planned.

2.2 Design Evolution from SE491

The original design was for a POS - but we also had to implement a crawler and create our own database to get data for all the courses, prereqs, credits etc. from the ISU catalog. The difficulty of such a task was not fully appreciated in the original design.

Also, the original design didn't take into account difficulties with the ISU Okta team.
Other than two considerations, our implementation was consistent with the original design.

2.3 Security Concerns and Countermeasures

Physical Security

- There are no real physical security requirements - the physical servers are managed by Iowa State, so they will be hosting the website/backend/database/authentication on their premises.

Cyber Security

- There is an Okta service enforced throughout the app that mitigates DDOS attacks and provides authentication. Only people with an @iastate email address will be allowed to make use of most of the services once an OAuth2 Bearer token is obtained from the ISU Okta server.

3. Implementation

3.1 Features

Our finished product contains a core set of features. Users must log in to authenticate themselves, then they are able to utilize the POS creation and editing features. Users are able to create a new POS from the student landing page and add classes to it through a drag and drop interface. They are also able to edit an existing POS they've created by selecting it in the student landing page, upon which it will be loaded into the drag and drop interface for editing.

For more details on the software's features, please refer to Appendix i.

3.2 Frontend Design

The frontend of this project consists of an Angular framework. The project is written with multiple components that are composed of html, css, and typescript. Each component itself has typically 4 files within it. These are an html file, a css file, and two typescript files. The html and css files are used for the visualization of pages whereas the typescript files are used for the logic within those pages.

There is a component for each of the pages that we have for our website. Each component contains the necessary code for each of those pages to run successfully. The names of these components are advisor-view, drag-drop, home, messages, profile, search-page, and student-view. All of these components fall underneath the main component that uses them congruently. This main component is called app.

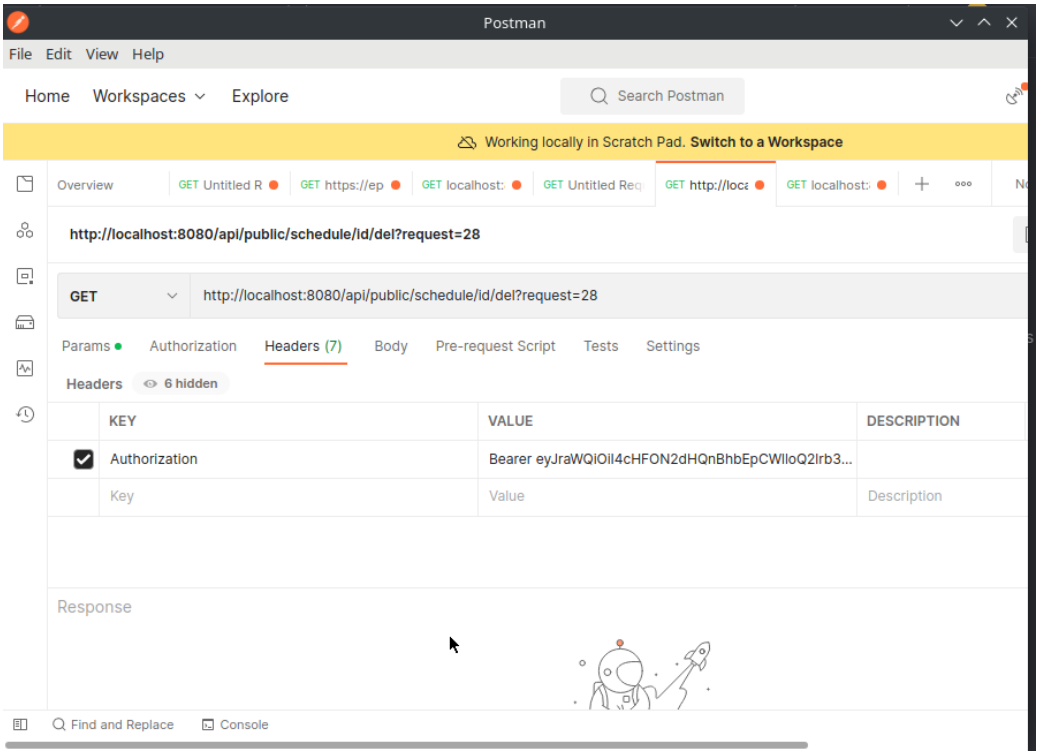
3.3 Backend Design

The backend server is a Spring Boot application written in Java. It uses various controller methods to communicate with a PostGreSQL database where all user and course data is stored. Controllers are the heart of the backend's functionality. Two main controllers serve the majority of the functions; the Course controller and Schedule controller. The Course controller provides a model for a Course and the ability to retrieve them when creating a POS. Each Course object in the database contains many descriptive fields such as title, department, description, and credit count. The course database is populated through the use of a standalone web scraper that gathers data from each ISU course catalog site. This was our solution for the problem of ISU having no easily accessible course database.

The Schedule controller deals with saving and loading POS's from the server. Each Schedule contains the Courses selected in the POS, as well as the creator and time created. By managing Schedules populated with Courses, the backend can provide all saving and loading functionality needed for the creation and retrieval of POS's.
The backend also implements Okta authentication to provide authentication services and integrate with ISU's existing framework.

4. Testing

Testing routines have been developed separately for the Backend and Frontend.

- **Backend -**
  - We are using Postman to test all the endpoints, with an optional /public prepender to enable testing of protected endpoints as well - as well as leaving it there to be permanently accessible publicly
  - Once an endpoint is protected i.e. needs an authenticated user, we can add a Bearer token when we make the request in postman
  - 

  - How to get the Bearer token? Go to the profile tab -> https://epos.ece.iastate.edu/profile and there is a Bearer section in the table, the text next to it is the token
  - Testing is also rapidly available with Swagger, where go can run a request in a bash emulator. To run such simulations go to -> https://epos.ece.iastate.edu/api/swagger-ui/#/ , select any endpoint, and click 'Try it out' and the Execute

- **Frontend**
  - For testing frontend, we run the angular app locally in a dev environment, and with 'ng serve' we can see the effects of changes almost instantaneously under localhost:4200

- Since the api calls need to be made to a local server, we will need to run the Spring Boot backend locally, however we don’t need to run the database instance, we can access a dev database by simply turning on the VPN
- All other testing is done manually by opening a website in different browsers or seeing if features work as intended.

5. Closing Material

This section will include the hours worked for each team member during the implementation phase, challenges faced, and lessons learned during the project.

5.1 Project Hours Worked

Below is a table of all the team members and the amount of hours they have worked on this project during the implementation phase.

Team Member	**Hours Worked (Implementation Phase)**
Saljooq Altaf	169
Carson Campbell	100
William Hunt	102
Noah Nickel	88
Nathan Marquardt	119
Total Hours Worked	578

5.2 Challenges Faced

There were a few challenges that happened during this project. The first challenge that our team faced was not having a proper database where course information was held. To overcome this challenge, our team developed a web scraper that will grab course information from the Iowa State University Course Catalog website.

Another major challenge that occurred during this project was that our team needed to follow certain procedures to access certain resources. Some of the resources that we requested were never approved for our team or were placed into a process far longer than our team anticipated. This forced our team to change our approach on how we implement certain items within our project design. An example of this challenge was the communication between our team and the Iowa State Okta team. Due to the long process, we were unable to use some of the Okta integration that we initially planned to use and had to adjust our design accordingly.

5.3 Lessons Learned

Our team has definitely learned some lessons during this senior design project. One lesson learned during this project was the importance of planning and testing thoroughly in order to complete a successful project. Our team had a good project structure from our design phase that allowed us to plan accordingly for certain conditions. We felt that this allowed our project to run smoothly for the most part.

Another lesson learned was that we needed to figure out our specific resources we needed as soon as possible in order to request those resources as soon as possible. Some of the problems that we encountered during this project revolved around not having certain resources that we expected to have, due to the length of the process required to obtain those resources. If we were to go back, we would try to gain these resources as soon as possible in order to give our team the best chance to use those resources in our final application.

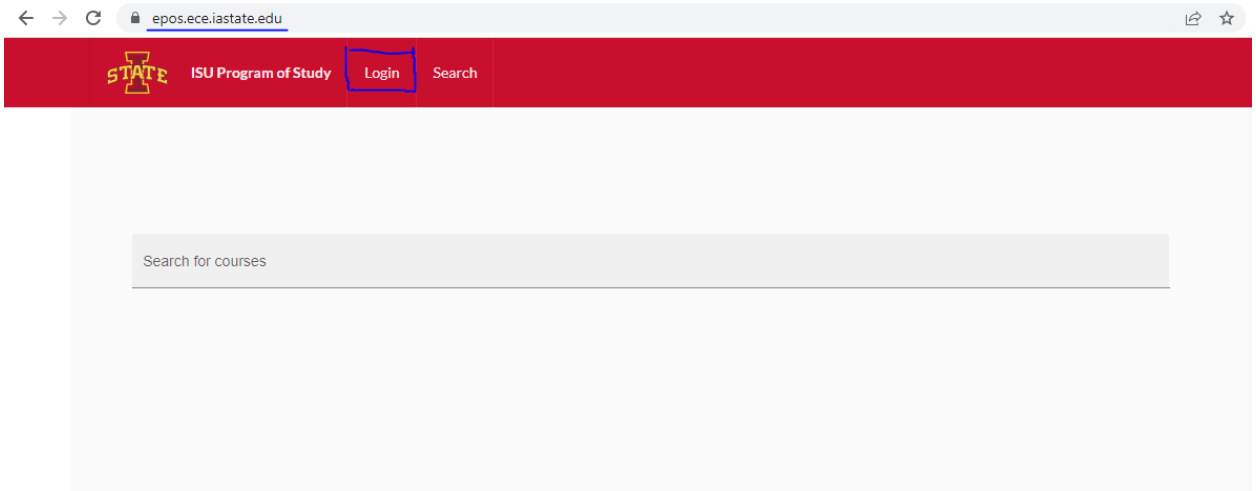
Finally, the last lesson we learned was to realize that the project plan will not be perfect and we must take into account that things will break or not be as we originally planned. Some of our plans were really strict and we needed to come up with specific alternatives to adjust for features breaking on the application. If we were to go back, we would develop a plan with a few alternatives for each feature to allow our project to be more fluid in its implementation.

6. Appendices

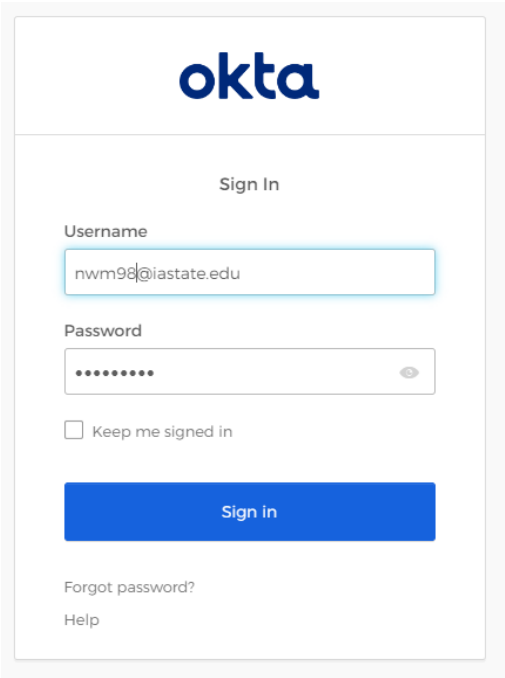
i. User Manual

How to use the website:

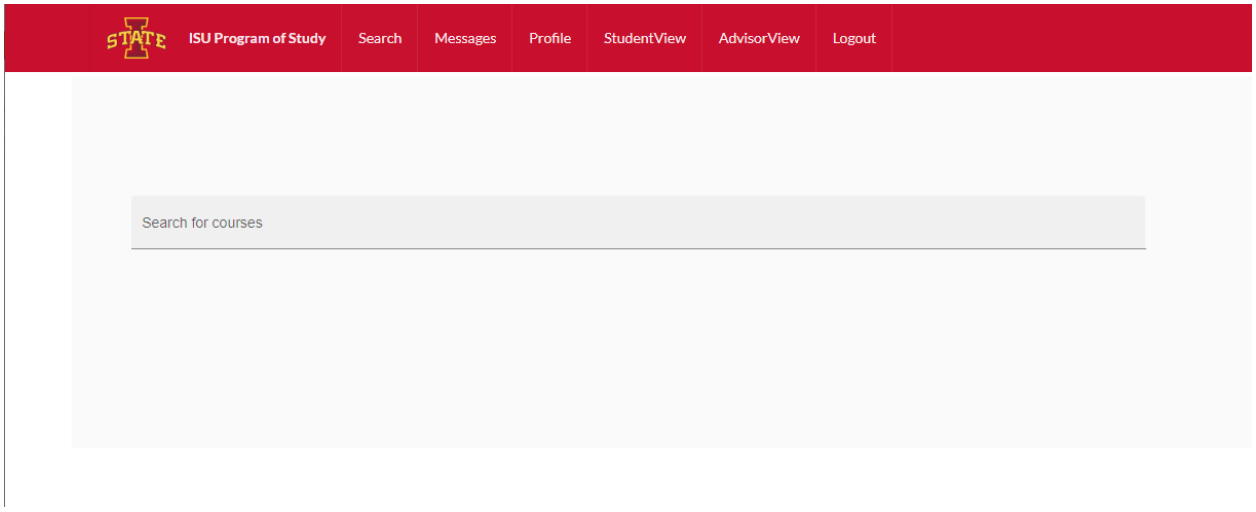
To use the website, the student will first have to head over to epos.ece.iastate.edu. Once they do, they will have to click on the login tab.



After clicking on the Login tab, a student will be redirected to an OKTA login prompt where a student will enter their login information and be authenticated.

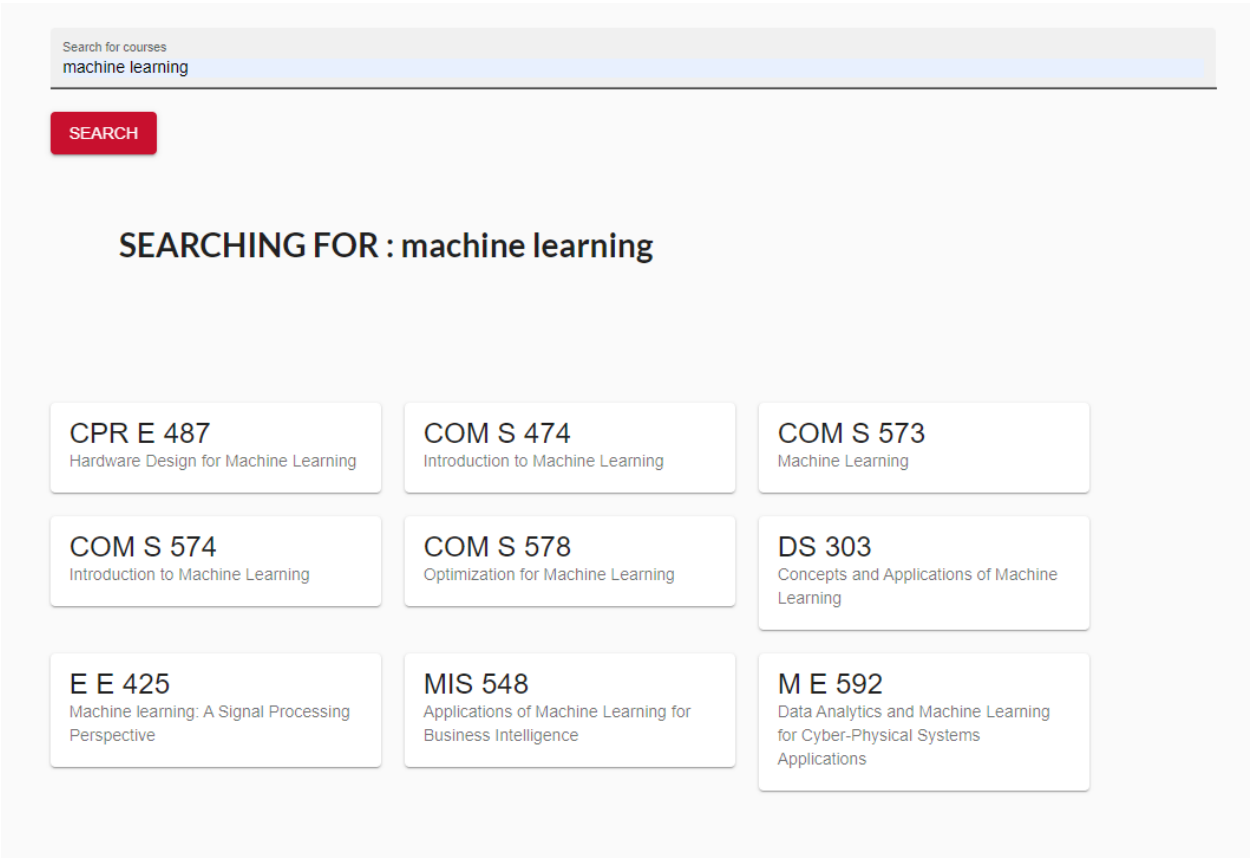


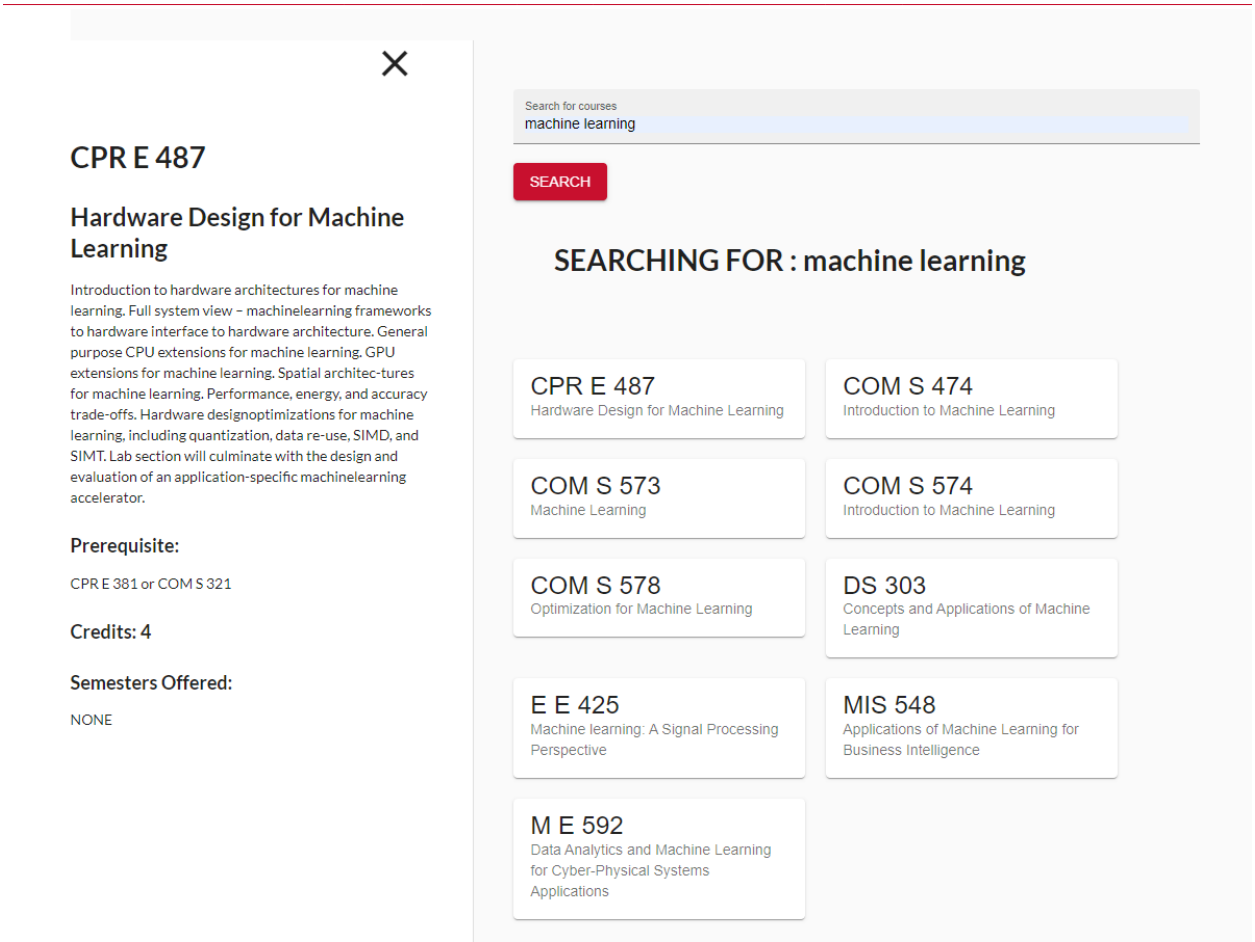
After logging in, a student will then be redirected to the search courses page. As one can notice, there are more tabs on top now for more pages that a student can utilize. We will now break up the manual into how to use each of the features on the website.



Search for Courses Page

On the Search for Courses page, a user can enter keywords into the search bar to find courses that match that description. After the search, cards of course that match are placed underneath the bar. A user can then click on one of the course cards to gain more information about that specific course.





Messages Page

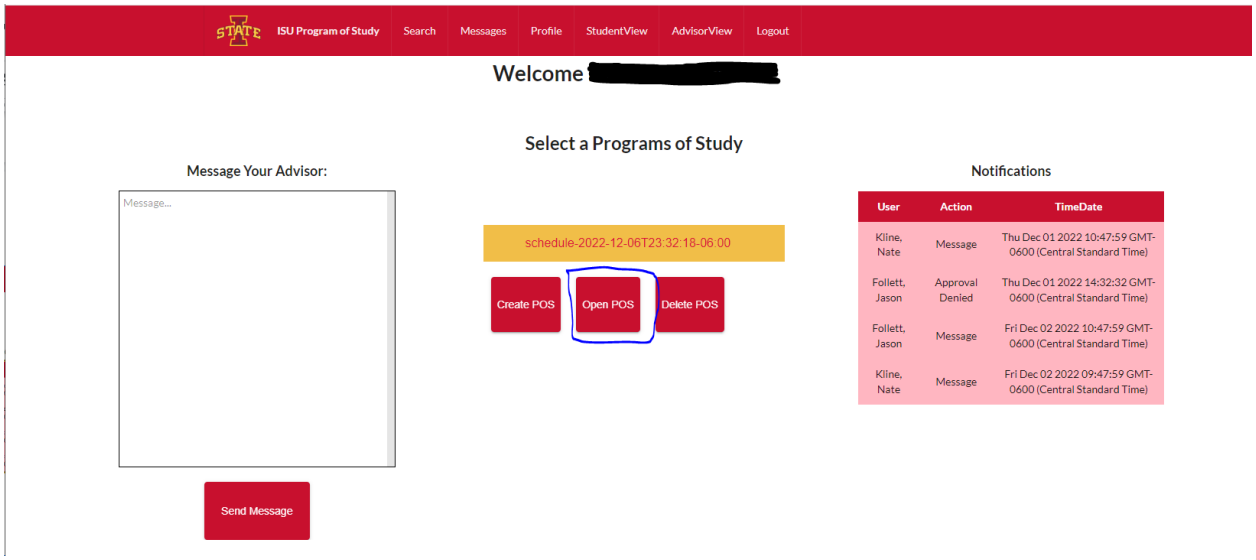
This page is to show that the frontend and backend are talking with one another and that there is a connection between the frontend and backend. There is no use for this page other than that.

Profile Page

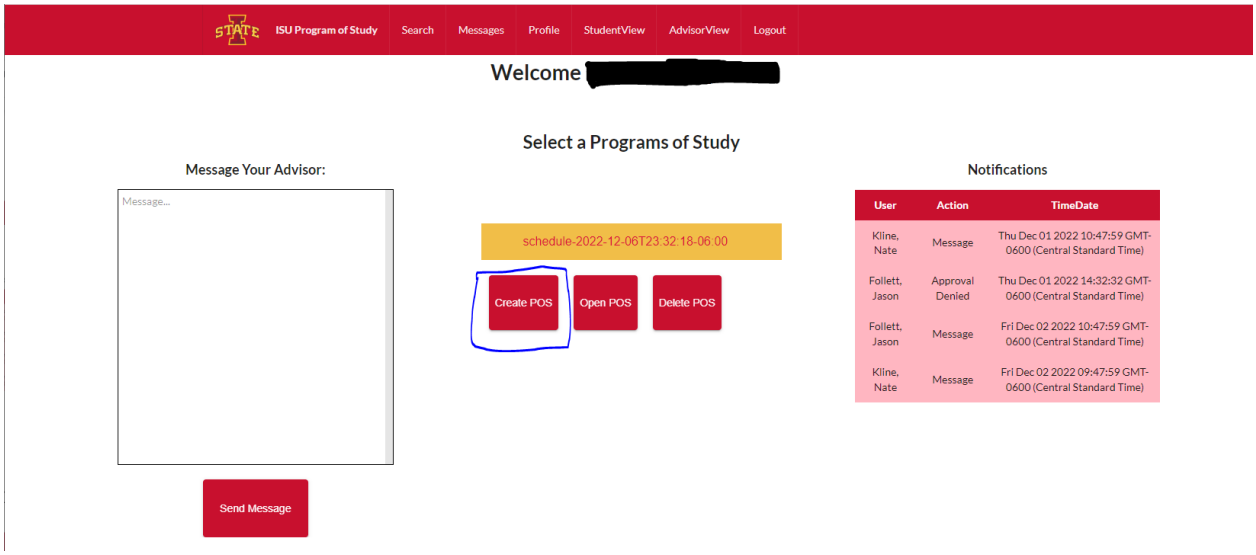
This page shows the information of the user based upon their bearer token. There is no usage for this page other than confirmation that the correct user is logged in.

Student View Page

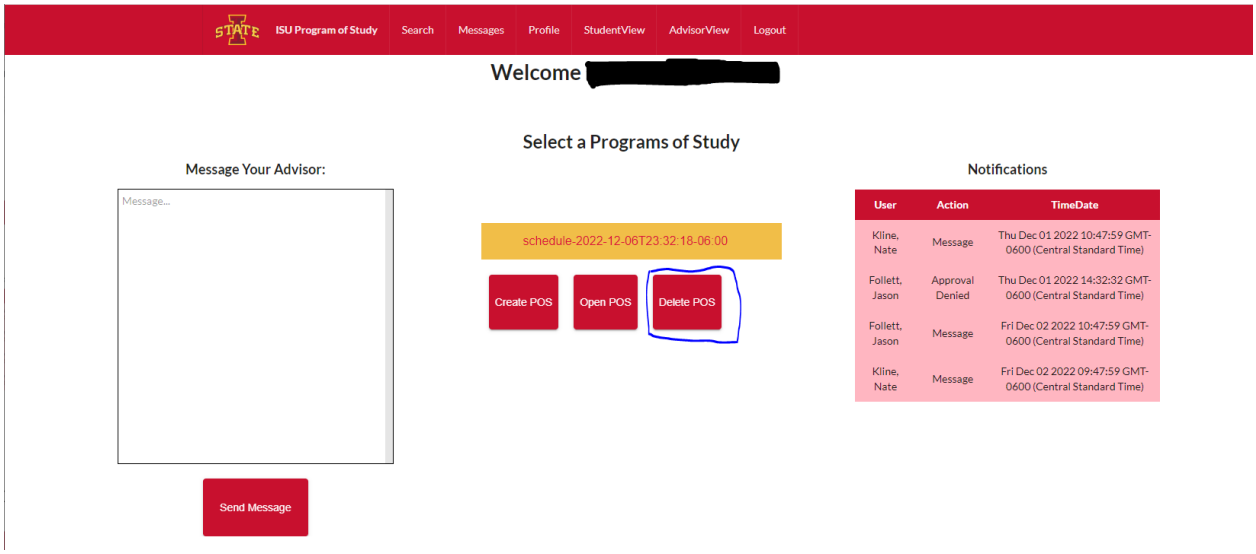
This page will be a great hub for a user to access their already created Programs of Study. A user can click on a unique Program of Study and then Click on the Open POS button to then open up the created Program of Study on the POS Creation Page. A user can then view and/or update that Program of Study.

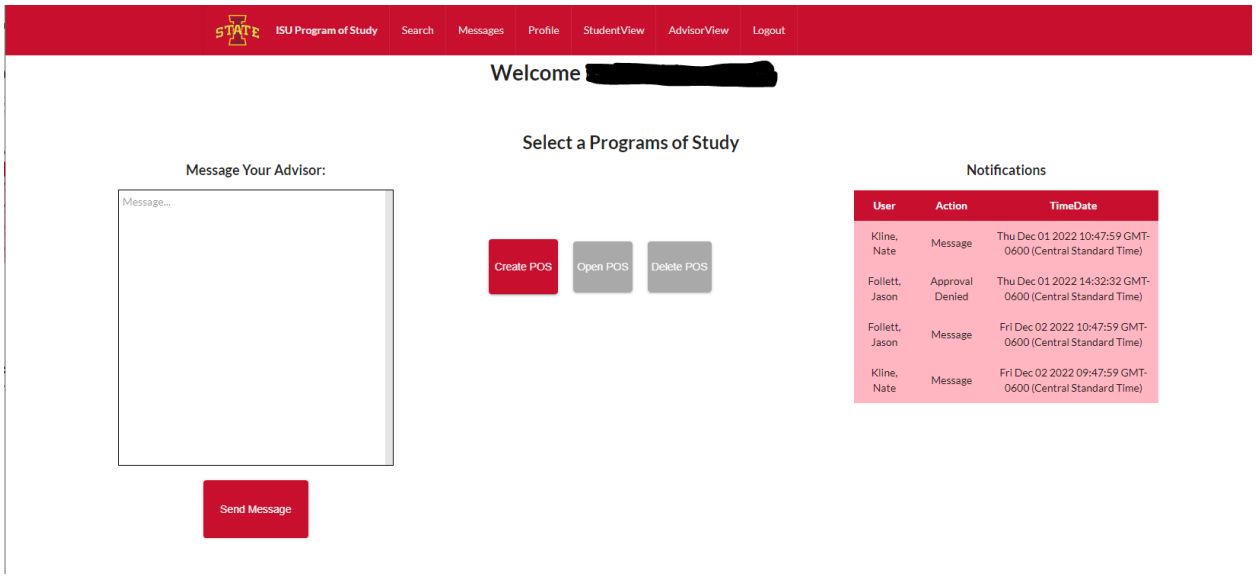


A user can also click on the Create POS button to be redirected to the POS Creation Page with a Program of Study that is empty.



Finally, A user can click on a Program of Study and then Click on the Delete POS button to delete the selected Program of Study from their Student View.

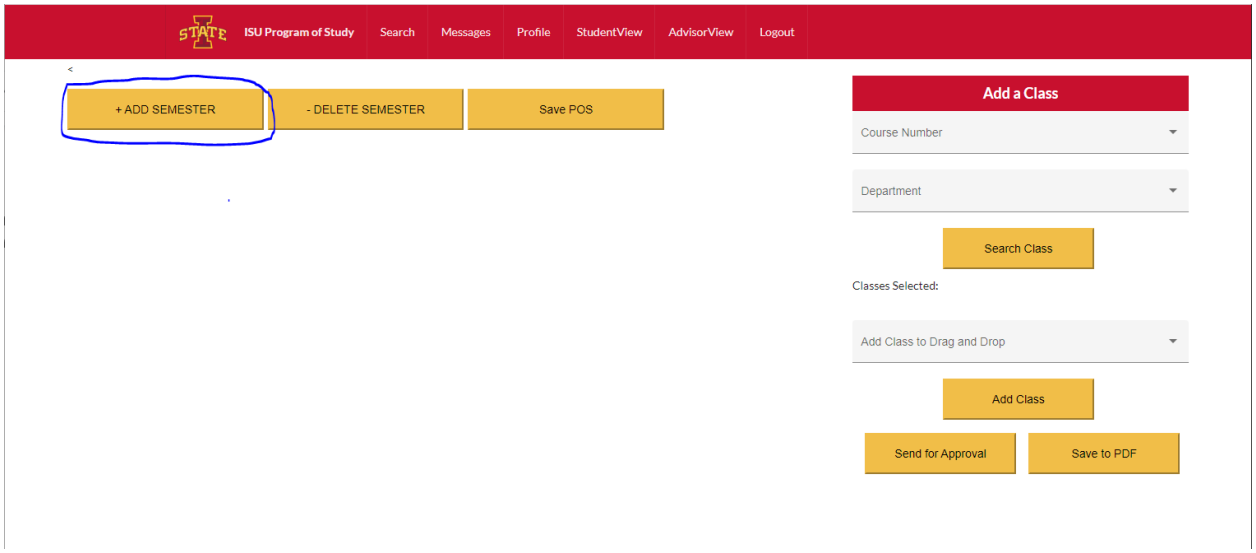


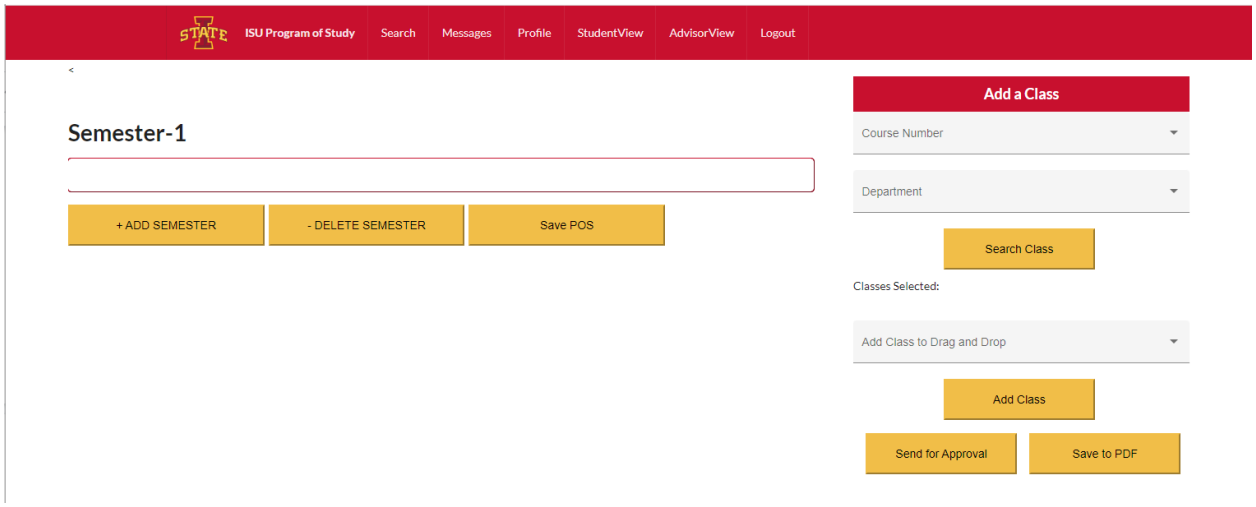


Note: There is a message input field on the left with a button and a notification table on the right of the page. These two features are currently unavailable, but are shown to be concepts for a later team to develop if the project continues.

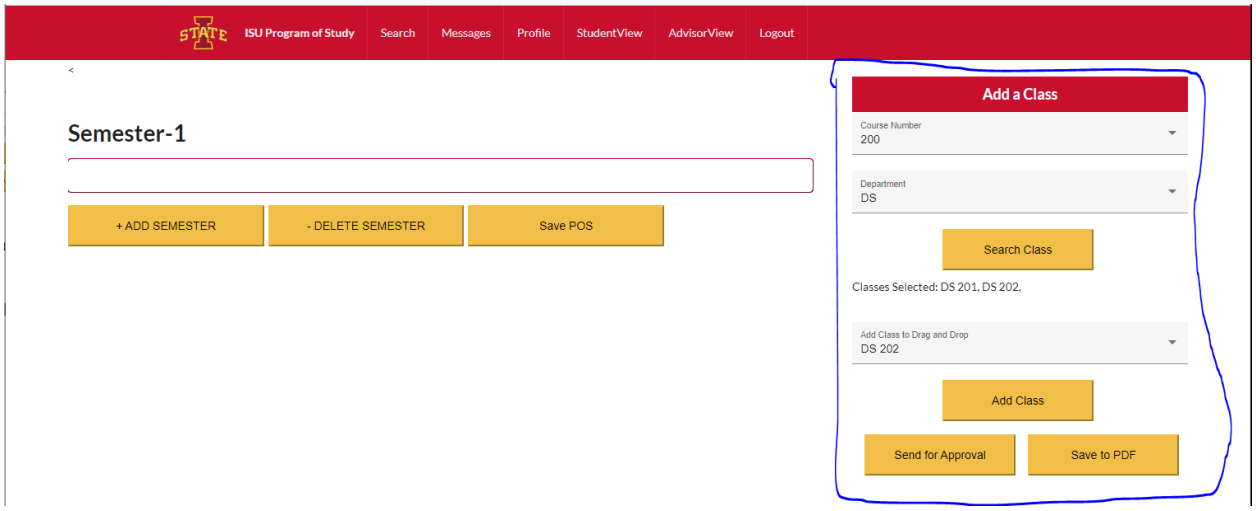
POS Creation Page

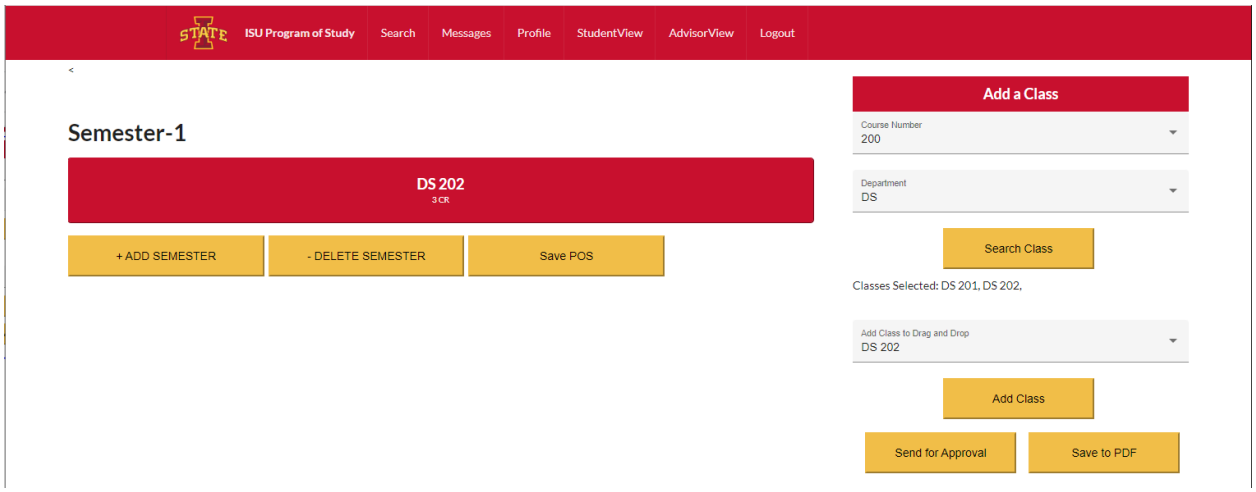
This page allows users to create a Program of Study. Users can first click on the Add Semester button to create an empty semester container.



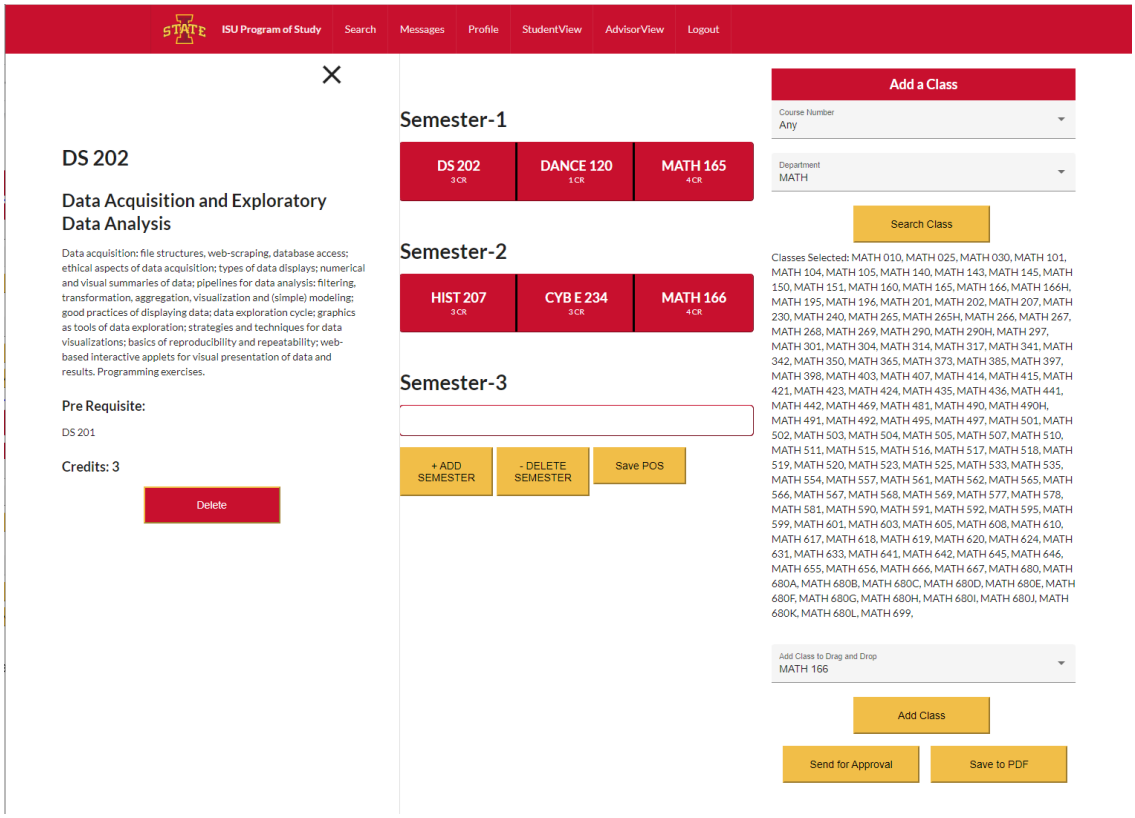


A user can then add a class item to that semester with the “Add a Class” Search on the right of the screen. A user can select a course level and department as filters for their search via the drop down menus. After selecting the filters, a user can select the class to be added to the semester and click on the Add Class button. Once the click happens, then that class will be added to the semester.

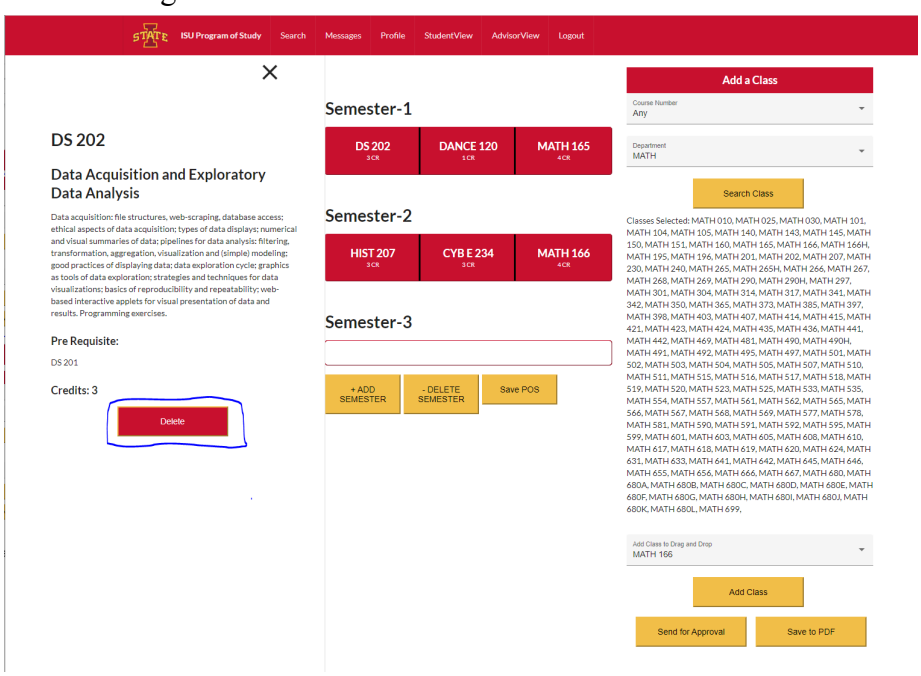


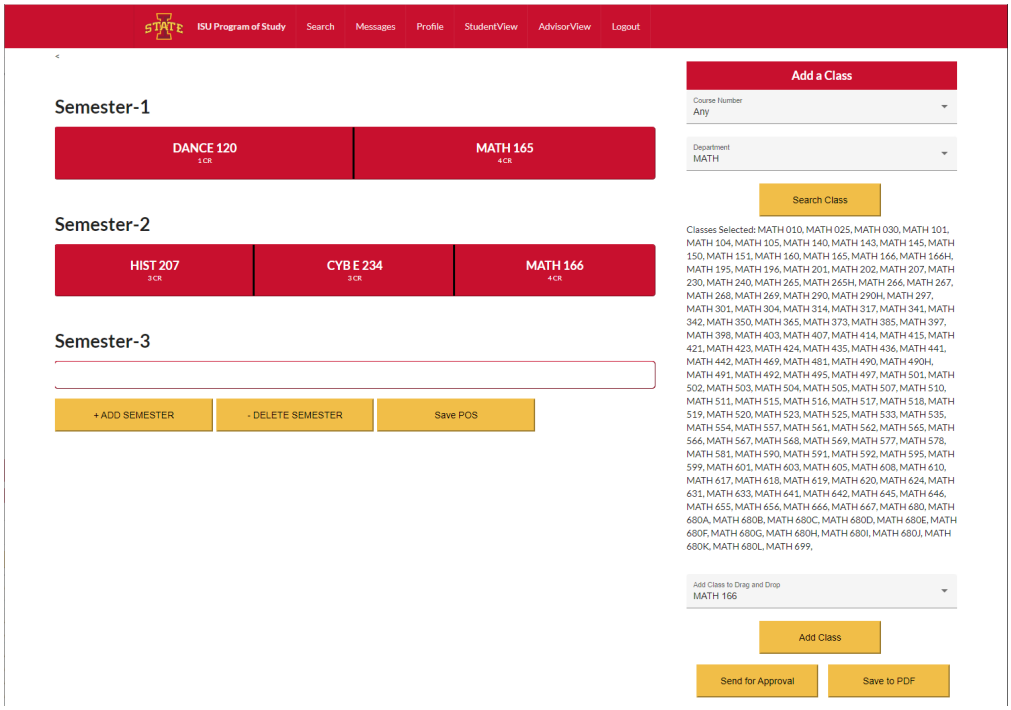


Once a class item has been added to a semester, it can then be moved from one semester container to another. Also, a user can get more information on a class if they click on the class item object.

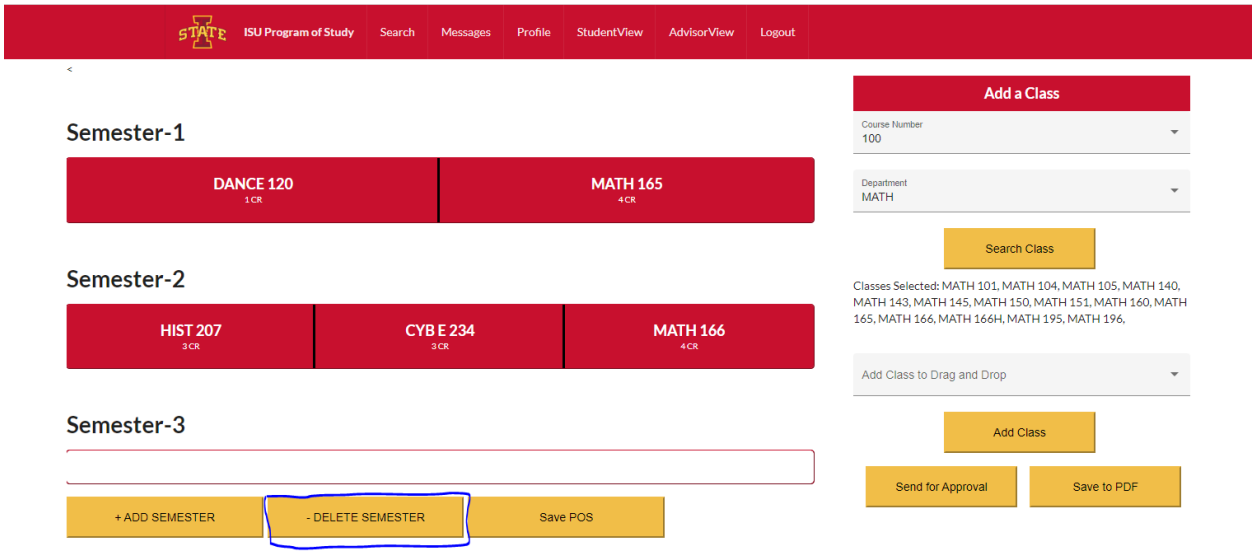


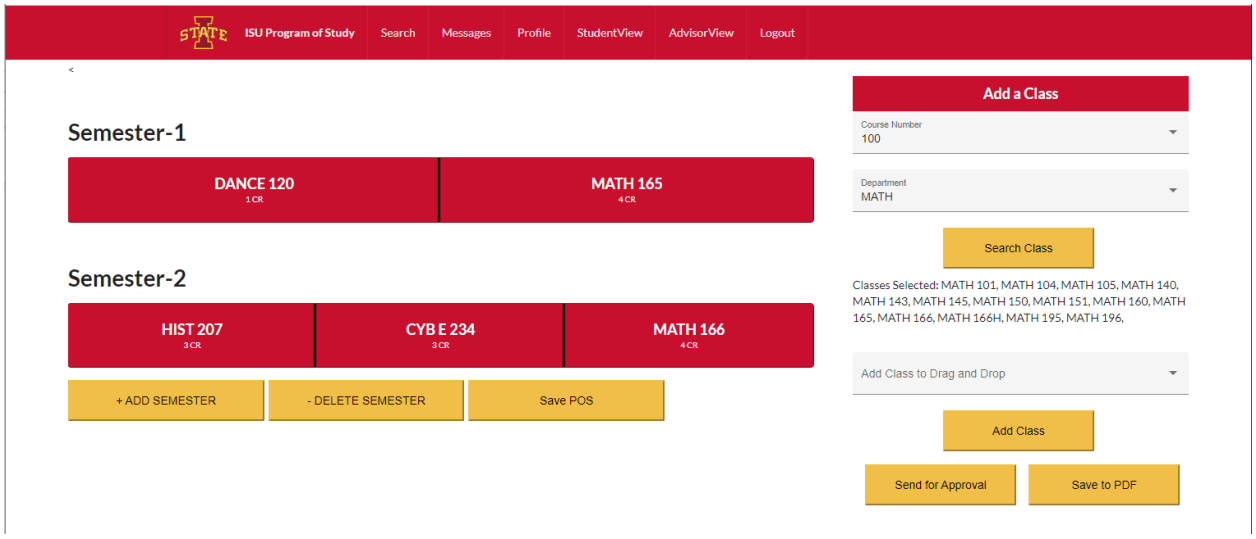
A user can also delete the class item object from the semester by clicking on the class item and then selecting the Delete button from the information drawer.



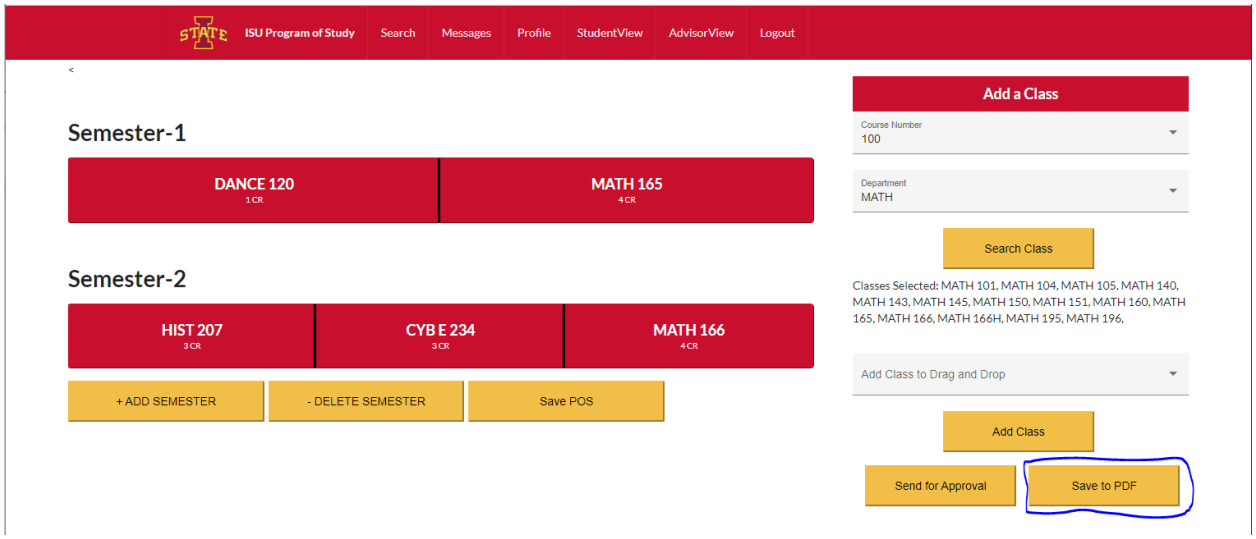


A user can also remove a whole semester if they wish via the "- Delete Semester Button" on the bottom of the Program of Study.

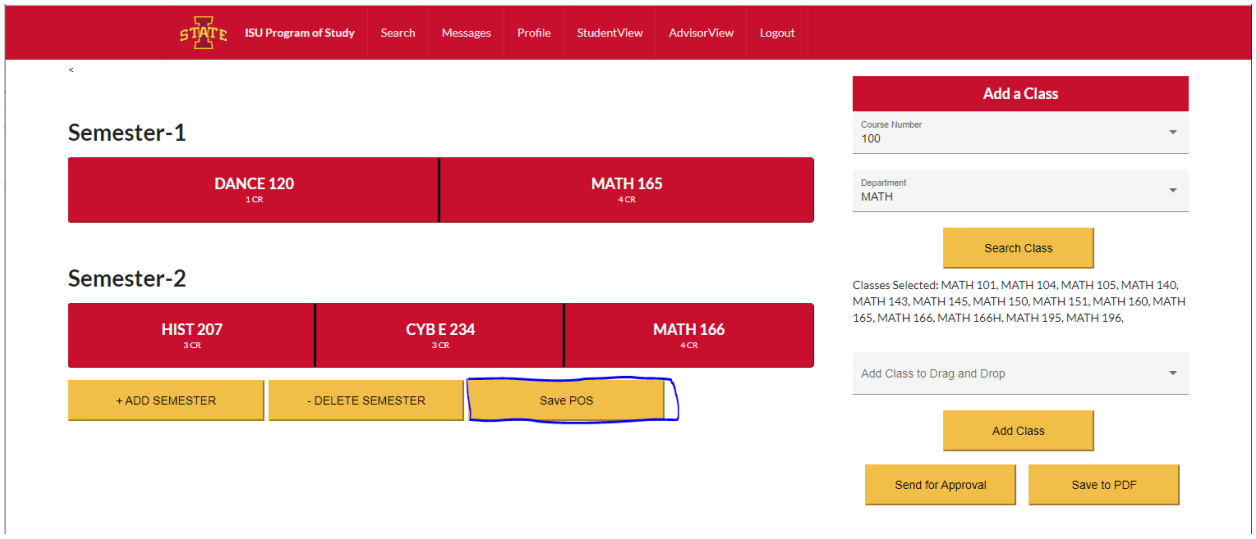




If a user has completed their Program of Study, then they can click on the "Save to PDF" button to save their created Program of Study in a PDF format.



Finally, a user can click on the "Save POS" button at the bottom of the Program of Study to save their Program of Study for later use or for reference. Once this button is clicked, the user will then be redirected to the Student View page with a new Program of Study added to their list in the middle of the screen.

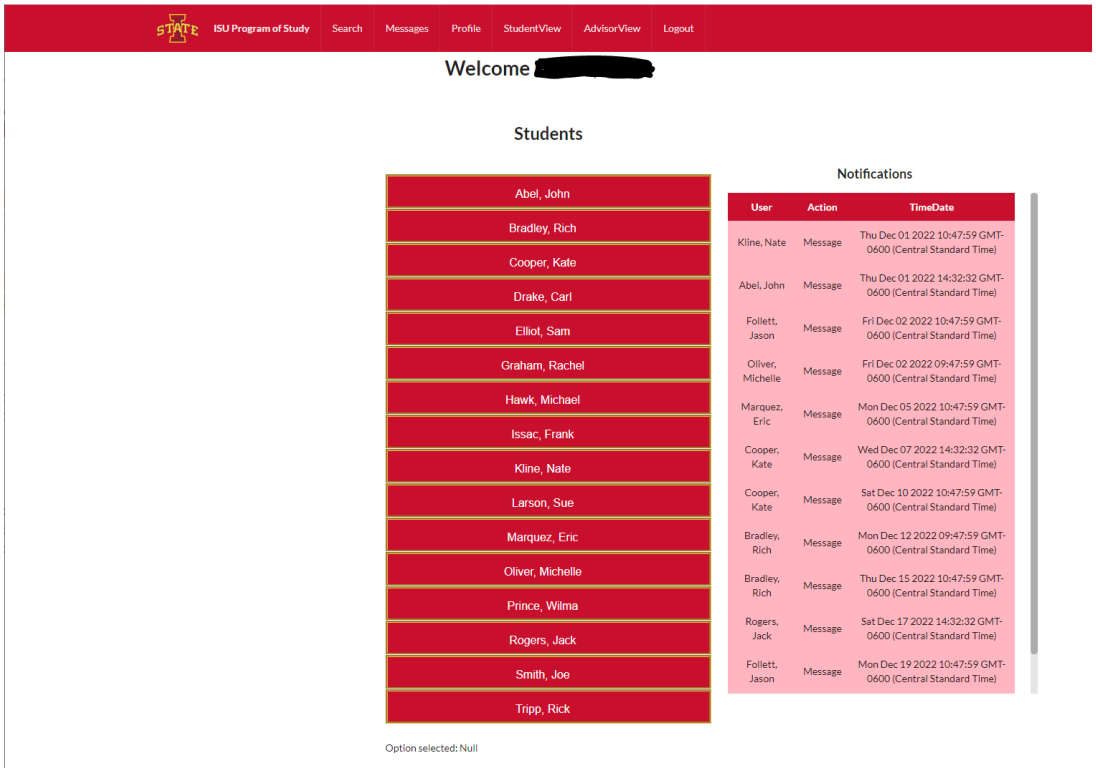


Note: The current iteration of this page also includes a “Send for Approval” button on the right. This feature has not been created by our team and the button is for a web page format concept. It can be implemented by a later team if they wish to include Advisor users and have students send their Programs of Study to their advisor for approval.

Advisor View Page

The current iteration of this project does not have advisor-type users in mind. However, our team has developed a concept for an Advisor View which includes a list of the students the Advisor has in the middle of the page as well as a notification table on the right of the page. This view could be implemented by a later team if the project were to continue.

Note: The buttons for each student and the data in the notification table are all static “dummy” data. This is only for visual purposes to fill the page.



ii. Alternative Design

Since one of the deciding factors to go for Single Page Application (SPA) was the interactivity, there are new Server Side Rendering (SSR) frameworks out there like Sveltekit or SolidJs that would've allowed for a lower initial load time of the web page while allowing us to have very similar interactivity. However, most of such frameworks are too new (Sveltekit still hasn't reached 1.0 while SolidJs is too new to have any ecosystem). This means that the choice of Angular for such interactive applications was quite correct - especially with so many UI and authentication libraries to rely on. The only other library/framework with a similar ecosystem is React.

iii. Other Considerations

There are three more ‘glue codes’ that are involved in deploying the code:

- apache httpd conf files that essentially are responsible for forwarding the relevant (backend/frontend) requests to the relevant ports. Also frontend content may be static/dynamic, leading to more specific forwarding rules

```
root@epos-web:/etc/apache2/sites-enabled# sudo systemctl restart apache2.service
root@epos-web:/etc/apache2/sites-enabled# sudo systemctl restart apache2.service
root@epos-web:/etc/apache2/sites-enabled# sudo systemctl restart apache2.service
root@epos-web:/etc/apache2/sites-enabled#
```

```
!/bin/bash [root@epos-web: /etc/apache2/sites-enabled]
21                          AllowOverride All
22                          Require all granted
23                          Allow from all
24                  </Directory>
25
26                  ProxyPass         /api/  http://localhost:8080/api/
27                  ProxyPassReverse  /api/  http://localhost:8080/api/
28
29                  # ProxyPass         /  http://localhost:4200/
30                  # ProxyPassReverse  /  http://localhost:4200/
31
32
33                  # DocumentRoot /var/www/html
34                  DocumentRoot /home/maintainers/sddec22-15/sddec22-15/Frontend/frontend/dist
35
36                  RewriteEngine On
37
38                          RewriteRule "^/api" - [L]
39
40                          # If an existing asset or directory is requested go to it as it is
41                          RewriteCond %{DOCUMENT_ROOT}%{REQUEST_URI} -f [OR]
42                          RewriteCond %{DOCUMENT_ROOT}%{REQUEST_URI} -d
43                          # RewriteRule "^/api" - [L]
44                          RewriteRule ^ - [L]
45
46                          # If the requested resource doesn't exist, use index.html
47                          RewriteRule ^ /index.html
```

- there are three bash files - used to automate the code at different levels - all three are called ‘run_silently.sh’. These essentially provide the necessary logic to deploy the repository in any linux kernel with the forwarding rules that we have defined
- a yml file ‘gitlab-ci.yml’ at the root of the project that is used to automate the deployment process (Continuous Deployment) as well as to monitor the correct build (Continuous Integration). This makes use of docker to ensure proper connection and build - all the relevant secrets used for deployment are stored under:
  SETTINGS -> CI/CD -> VARIABLES

iv. Code

All project code can be found in the sddec22-15 Gitlab repository at https://git.ece.iastate.edu/sd/sddec22-15.