CS CAPSTONE END OF TERM REPORT

MARCH 17, 2020

OSU CS APPLIED PLAN PORTAL

PREPARED FOR

OREGON STATE UNIVERSITY

DR. ROB HESS	Signature	Date

PREPARED BY

GROUP CS72

THE PORTAL TEAM

CLAIRE CAHILL	Signature	Date
JACKSON GOLLETZ	Signature	Date
PHI LUU	Signature	Date
ZACHARY THOMAS	Signature	Date

Abstract

Students at OSU who major in computer science may select the applied option. This option allows students to create a plan with a specific focus in an area of interest. Plan submissions must be reviewed and approved by OSU advisors. The current process is inconvenient and time consuming, both for students and advisors. Our application simplifies both the plan creation and review process. During winter term we worked on implementing features for the front-end and back-end of our application. By the end of the term, we had completed all of the functionality outlined in our requirements document. During winter term we ran into issues related to getting course data, securely transferring access tokens, and getting feedback from the advising team. During spring term we plan to focus on testing, improving the security of our application, making our application more accessible, and hosting our application.

CONTENTS

1 Introduction

Students at Oregon State University who major in computer science must choose between two routes: systems or applied. Those who decide to pursue the applied option may create a custom plan of courses to complete in order to graduate. The applied option allows students to select courses that total to at least thirty-two credits, that focus on some area of interest. The current system of creating and reviewing these custom applied plans can be confusing and time consuming.

Our team's goal is to produce a web application that simplifies both the process of creating and reviewing these custom applied plans for both students and advisors. We want to offer students a platform to easily explore and select courses for their applied plan, as well as create a better method to communicate with their advisors about submitted plans. For advisors, we want our application to offer them an easier solution to review and leave feedback on plans.

2 Progress

Throughout this term, our team has worked on developing this the CS applied plan portal while checking in weekly with our client to present the progress we had made. Currently, the project includes many features and fulfills the expectations for beta functionality. The front-end of the application was developed with React while the back-end uses Node.js, Express, and MySQL.

2.1 Brief Feature Overview

Before going into detail about the features we were able to implement this term, here is a brief overview for easy reference.

Front-end

- OSU Login Page
  - Users are automatically redirected to OSU's login page.
  - Prevents our application from needing to store passwords.
- Navigation Bar
  - Title links to homepage.
  - ”Notification” button that shows updates to plans in real time.
  - ”Log out” button to allow the user to log out of both the application and their OSU account.
  - Advisor-only ”History” button which, when clicked, displays the last five plans the advisor has viewed.
  - Head Advisor-only ”Manage Roles” button that opens the manage roles page.
- Student Home Page
  - Lists all plans that the student has created.
  - Link to create plan page.
- Create / Edit Plan Page
  - Allows students to search and filter courses.
  - Allows students to add courses to a plan.
  - Allows students to remove courses from a plan.

  - Allows students to name a plan.
  - Allows students to submit or update a plan.

- Advisor Home Page / Search Plans Page
  - Allows advisors to search for plans.
  - Allows advisors to sort and filter search results.
  - Clicking a plan redirects to the view plan page for that plan.
- Manage Roles Page
  - Allows the head advisor to search for users.
  - Allows the head advisor to change a users role (student, advisor, head advisor).
- View Plan Page
  - Allows users to view details about a plan.
  - Allows users to view the selected courses of a plan.
  - Allow users to see how many similar plans have been accepted and rejected.
  - Allows users to comment on a plan.
  - Allows advisors to change the status of a plan.
  - Button to print a plan.
  - Button to delete a plan.
  - Button to open the plan editor.

Back-end

- User Endpoints
  - GET user/:userId
  - GET user/:userId/plans
  - GET user/search/:text/:role/:cursorPrimary/:cursorSecondary
  - PATCH user/:userId
  - POST user
- Plan Endpoints
  - GET plan/:planId
  - GET plan/:planId/similar
  - GET plan/recent
  - GET plan/:planId/activity/:cursorPrimary/:cursorSecondary
  - GET plan/search/:text/:status/:sort/:order/:cursorPrimary/:cursorSecondary
  - PATCH plan/:planId
  - POST plan
  - DELETE plan/:planId
- Course Endpoints
  - GET course/:mode/:searchText

- Comment Endpoints
  - POST comment
- Review Endpoints
  - POST review
- Notification Endpoints
  - GET notification
  - PATCH notification/:notificationId
- Database
  - Plan Table
  - User Table
  - Course Table
  - Notification Table
  - Comment Table
  - Review Table
  - Selected Course Table
  - Recent Plan Table

2.2 OSU Login Page

The people who will use this application will fall into one of three categories: Student, Advisor, and Head Advisor. When users first start the application they are redirected to a login page that uses OSU's Central Authentication Service. By using this service, we are able to allow users to log in with their OSU credentials and get information about users without storing passwords or other sensitive information in our database. Once the user logs in, they are redirected to either the student or advisor homepage based on their role.

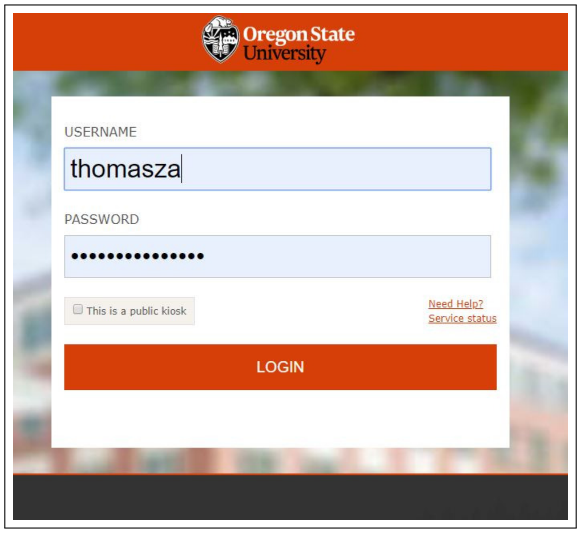


Fig. 1: Login Page

2.3 Navigation Bar

Each page includes a navigation bar which contains different items depending on the user's role. All users see the "Notifications" and "Log out" buttons, but only advisors see the "History" button that shows plans that were recently visited. Head advisors also can see the "Manage roles" button, which allows them to change a user's role. Notifications appear for students when an advisor has commented on or updated the status of a plan that they created. Advisors receive notifications when a student or another advisor has commented on or updated the status of a plan that they have reviewed.

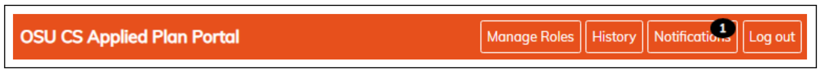
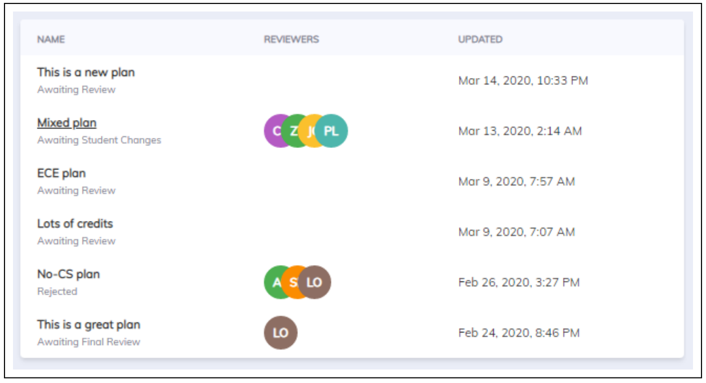


Fig. 2: Navigation Bar

2.4 Student Homepage

When a student first logs in, they are greeted with a home page that shows their submitted plans, if any. The table of plans includes the plan name, the time the plan was last updated, its current approval status, and icons indicating the advisors that have reviewed it. When the student first logs in, the application makes an API call to get all the plans that are stored under the user's public ID number. There is also a button in the bottom right corner that takes the user to the "Create plan" page.

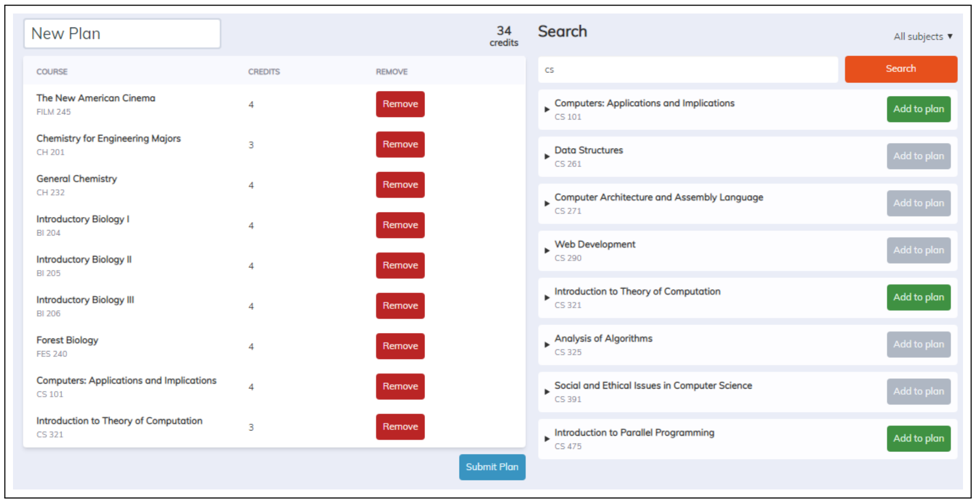


NAME	REVIEWERS	UPDATED
This is a new plan Awaiting Review		Mar 14, 2020, 10:33 PM
Mixed plan Awaiting Student Changes	C Z J PL	Mar 13, 2020, 2:14 AM
ECE plan Awaiting Review		Mar 9, 2020, 7:57 AM
Lots of credits Awaiting Review		Mar 9, 2020, 7:07 AM
No-CS plan Rejected	A S LO	Feb 26, 2020, 3:27 PM
This is a great plan Awaiting Final Review	LO	Feb 24, 2020, 8:46 PM

Fig. 3: Student Plans Table

2.5 Create Plan Page

The student "Create plan" page consists of two panels where the right side contains a search and filter bar to explore courses and the left side displays the working plan, including courses that the user has selected. When a student searches for classes on the right hand side using the search bar, the React server makes an API call to the back-end to get courses in the database based on the course name or course code. Then the "Course container" on the right side will fill with the results of the API call. Each course shows the course code and name, with an "Add to plan" button and an arrow button which expands to show further details including prerequisites, a short course description, and credit hours. When the user clicks "Add to plan", the course will appear on the left hand side and the total credits will increase based on the credit hours of the added course. Graduate or required courses have a grayed out "Add to plan" button, and clicking on it will show an error message explaining why the user cannot add that course. The left hand side contains an input box for the plan name, the total credits of the plan, and the selected courses (each has a button to remove from plan), and a submit plan button. There are various checks that are in place on both the front and back ends of the application to make the entire process of approving plans smoother. There are restrictions for adding duplicate courses, graduate courses, required courses, maximum and minimum amount of credits, plan name length, and more. Once the user successfully submits a plan, it takes them to the ”View plan” page and automatically sets the plan status as ”Awaiting review”.



New Plan — 34 credits

COURSE	CREDITS	REMOVE
The New American Cinema FILM 245	4	Remove
Chemistry for Engineering Majors CH 201	3	Remove
General Chemistry CH 232	4	Remove
Introductory Biology I BI 204	4	Remove
Introductory Biology II BI 205	4	Remove
Introductory Biology III BI 206	4	Remove
Forest Biology FES 240	4	Remove
Computers: Applications and Implications CS 101	4	Remove
Introduction to Theory of Computation CS 321	3	Remove

Submit Plan

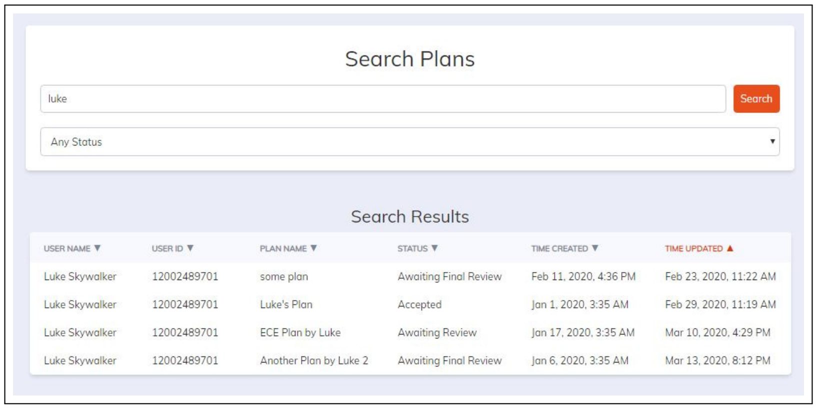
Search — All subjects — cs — Search

Computers: Applications and Implications CS 101	Add to plan
Data Structures CS 261	Add to plan
Computer Architecture and Assembly Language CS 271	Add to plan
Web Development CS 290	Add to plan
Introduction to Theory of Computation CS 321	Add to plan
Analysis of Algorithms CS 325	Add to plan
Social and Ethical Issues in Computer Science CS 391	Add to plan
Introduction to Parallel Programming CS 475	Add to plan

Fig. 4: Create Plan Page

2.6 Advisor Homepage / Search Plans Page

When an advisor first logs in, they will see a search box to search for plans. The advisor can search by student name, ID number, and plan name, as well as filter results by plan status. The table of plans includes the following columns: user name, user ID, plan name, status, time created, and time updated. This table can be sorted by clicking the title of any of these columns. Clicking on a plan brings the user to the same ”View plan” page that the student sees, except there is no option to edit or delete the plan. Instead, there is a drop down menu at the bottom of the table where the advisor can change the status of the plan.



Search Plans

luke — Search

Any Status

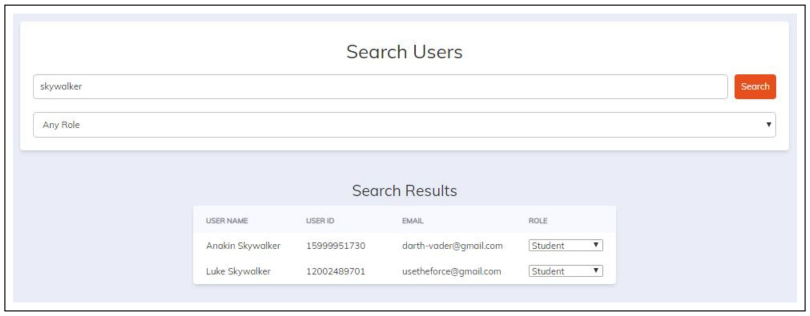
Search Results

USER NAME ▼	USER ID ▼	PLAN NAME ▼	STATUS ▼	TIME CREATED ▼	TIME UPDATED ▲
Luke Skywalker	12002489701	some plan	Awaiting Final Review	Feb 11, 2020, 4:36 PM	Feb 23, 2020, 11:22 AM
Luke Skywalker	12002489701	Luke's Plan	Accepted	Jan 1, 2020, 3:35 AM	Feb 29, 2020, 11:19 AM
Luke Skywalker	12002489701	ECE Plan by Luke	Awaiting Review	Jan 17, 2020, 3:35 AM	Mar 10, 2020, 4:29 PM
Luke Skywalker	12002489701	Another Plan by Luke 2	Awaiting Final Review	Jan 6, 2020, 3:35 AM	Mar 13, 2020, 8:12 PM

Fig. 5: Advisor Homepage / Search Plans Page

2.7 Manage Roles Page

The head advisor has access to an additional page in the application where they can change a user's role. Clicking the "Manage Roles" button in the navigation bar brings them to this page where they are greeted with a search bar to search for users and filter by role, similar to the search box on the advisor home page. When they input a search query, the search results table displays user name, ID, email, and a drop down menu to change the role to either student, advisor, or head advisor.

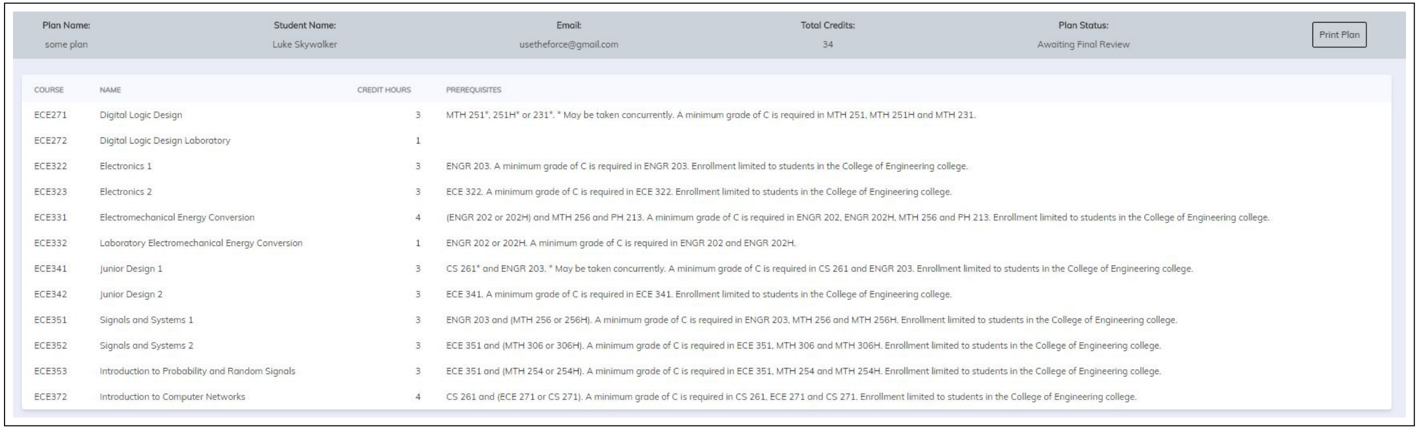


USER NAME	USER ID	EMAIL	ROLE
Anakin Skywalker	15999951730	darth-vader@gmail.com	Student
Luke Skywalker	12002489701	usetheforce@gmail.com	Student

Fig. 6: Manage Roles Page

2.8 View Plan Page

When a user clicks on an existing plan, it takes them to the "View plan" page, which consists of a header, a table of courses, and an activity feed. The header includes the plan name, student name, student email, total credits, plan status, and buttons to print, edit, or delete the plan. When the plan is accepted or rejected, the user cannot edit or delete the plan so these buttons are hidden. The activity feed shows all the comments on the plan, if any, and updates to the status. If a similar plans (one that includes all of the same courses as the current plan, but may have more) have been accepted or rejected then the number of each is listed on this page as well.



Plan Name:	Student Name:	Email:	Total Credits:	Plan Status:
some plan	Luke Skywalker	usetheforce@gmail.com	34	Awaiting Final Review

COURSE	NAME	CREDIT HOURS	PREREQUISITES
ECE271	Digital Logic Design	3	MTH 251*, 251H* or 231*. * May be taken concurrently. A minimum grade of C is required in MTH 251, MTH 251H and MTH 231.
ECE272	Digital Logic Design Laboratory	1	
ECE322	Electronics 1	3	ENGR 203. A minimum grade of C is required in ENGR 203. Enrollment limited to students in the College of Engineering college.
ECE323	Electronics 2	3	ECE 322. A minimum grade of C is required in ECE 322. Enrollment limited to students in the College of Engineering college.
ECE331	Electromechanical Energy Conversion	4	(ENGR 202 or 202H) and MTH 256 and PH 213. A minimum grade of C is required in ENGR 202, ENGR 202H, MTH 256 and PH 213. Enrollment limited to students in the College of Engineering college.
ECE332	Laboratory Electromechanical Energy Conversion	1	ENGR 202 or 202H. A minimum grade of C is required in ENGR 202 and ENGR 202H.
ECE341	Junior Design 1	3	CS 261* and ENGR 203. * May be taken concurrently. A minimum grade of C is required in CS 261 and ENGR 203. Enrollment limited to students in the College of Engineering college.
ECE342	Junior Design 2	3	ECE 341. A minimum grade of C is required in ECE 341. Enrollment limited to students in the College of Engineering college.
ECE351	Signals and Systems 1	3	ENGR 203 and (MTH 256 or 256H). A minimum grade of C is required in ENGR 203, MTH 256 and MTH 256H. Enrollment limited to students in the College of Engineering college.
ECE352	Signals and Systems 2	3	ECE 351 and (MTH 306 or 306H). A minimum grade of C is required in ECE 351, MTH 306 and MTH 306H. Enrollment limited to students in the College of Engineering college.
ECE353	Introduction to Probability and Random Signals	3	ECE 351 and (MTH 254 or 254H). A minimum grade of C is required in ECE 351, MTH 254 and MTH 254H. Enrollment limited to students in the College of Engineering college.
ECE372	Introduction to Computer Networks	4	CS 261 and (ECE 271 or CS 271). A minimum grade of C is required in CS 261, ECE 271 and CS 271. Enrollment limited to students in the College of Engineering college.

Fig. 7: View Plan Table

2.9 API Endpoints

2.9.1 Users

- GET user/:userId

  This endpoint takes a user ID. It returns information about the specified user.

- GET user/:userId/plans

  This endpoint takes a user ID. It returns an array of all of the plans that the specified user created, ordered by the time that they were last updated.
- GET user/search/:text/:role/:cursorPrimary/:cursorSecondary

  This endpoint takes a string as the search query, a user role (any role, student, advisor, head advisor) and two cursors that are used for managing pagination.

  It returns an array of users in ordered by their name and a cursor that stores data about the position of the next element to return after the current page of results.

  This cursor pagination method ensures that if users are deleted or added to the database before getting the next page, the page will still return the correct data without duplicating or omitting any results.
- PATCH user/:userId

  This endpoint takes the user ID as a parameter and then a request body that contains one of the following fields: first name, last name, email, or role (student, advisor, head advisor). If the request passes validation then the selected user is updated. This endpoint returns the number of changed rows.
- POST user

  This endpoint takes a request body that contains a first name, last name, email, and role (student, advisor, head advisor). If the request passes validation then a new user is created. This endpoint returns the user ID of the newly created user.

2.9.2 Plans

- GET plan/:planId

  This endpoint takes a plan ID. It returns information about the specified plan.
- GET plan/:planId/similar

  This endpoint takes a plan ID. It returns the number of similar plans that have been accepted and rejected.
- GET plan/recent

  This endpoint uses the access token sent with the request to get the current user's ID. It returns the last five plans that the user has visited, ordered by the time that they were visited.
- GET plan/:planId/activity/:cursorPrimary/:cursorSecondary

  This endpoint takes a plan ID and and two cursors that are used for managing pagination.

  It returns an array of comments and reviews that exist for the given plan. The array is ordered by the time that the comment or review was created. It also returns a cursor that stores data about the position of the next element to return after the current page of results.

  This cursor pagination method ensures that if comments or reviews are deleted or added to the database before getting the next page, the page will still return the correct data without duplicating or omitting any results.
- GET plan/search/:text/:status/:sort/:order/:cursorPrimary/:cursorSecondary

  This endpoint takes text to use in a search, a plan status to filter by (any status, rejected, awaiting student changes, awaiting review, awaiting final review, accepted), a column value used to order the results, an order value (ascending, descending), and two cursors that are used for managing pagination.

  It returns an array of plans based on the search query. The array is ordered by the query parameters. It also returns a cursor that stores data about the position of the next element to return after the current page of results.

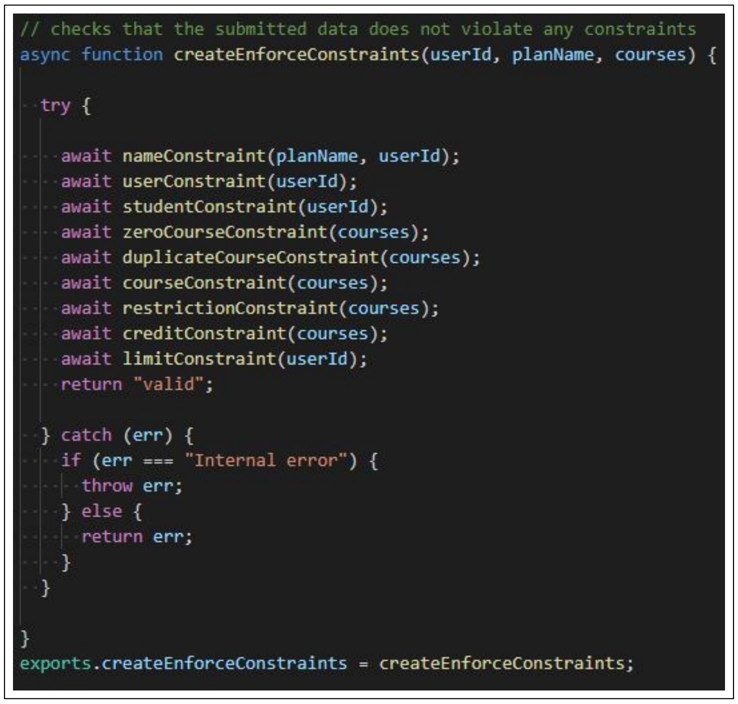
This cursor pagination method ensures that if plans are deleted or added to the database before getting the next page, the page will still return the correct data without duplicating or omitting any results.

- PATCH plan/:planId

  This endpoint takes the plan ID as a parameter and then a request body that contains an array of the selected courses and/or a plan name. If the request passes validation then the selected plan has its name and/or list of selected courses updated. This endpoint returns a Boolean value for if the plan name was changed and a boolean value for it the selected courses were changed.
- POST plan

  This endpoint takes a request body that contains an array of courses and a plan name. If the request passes validation then a new plan is created. This endpoint returns the plan ID of the newly created plan.
- DELETE plan/:planId

  This endpoint takes a plan ID. If the request passes validation then it deletes the selected plan in addition to all of the plans selected courses, comments, and reviews. This endpoint returns the number of table rows that were deleted by this action.



```
// checks that the submitted data does not violate any constraints
async function createEnforceConstraints(userId, planName, courses) {

  try {

    await nameConstraint(planName, userId);
    await userConstraint(userId);
    await studentConstraint(userId);
    await zeroCourseConstraint(courses);
    await duplicateCourseConstraint(courses);
    await courseConstraint(courses);
    await restrictionConstraint(courses);
    await creditConstraint(courses);
    await limitConstraint(userId);
    return "valid";

  } catch (err) {
    if (err === "Internal error") {
      throw err;
    } else {
      return err;
    }
  }

}
exports.createEnforceConstraints = createEnforceConstraints;
```

Fig. 8: Function that handles validating a new post plan request

2.9.3 Courses

- GET course/:mode/:searchText

  This endpoint takes a search mode (course ID, course code, course name), and text to use in a search. This endpoint returns an array of courses ordered by course code.

2.9.4 Comments

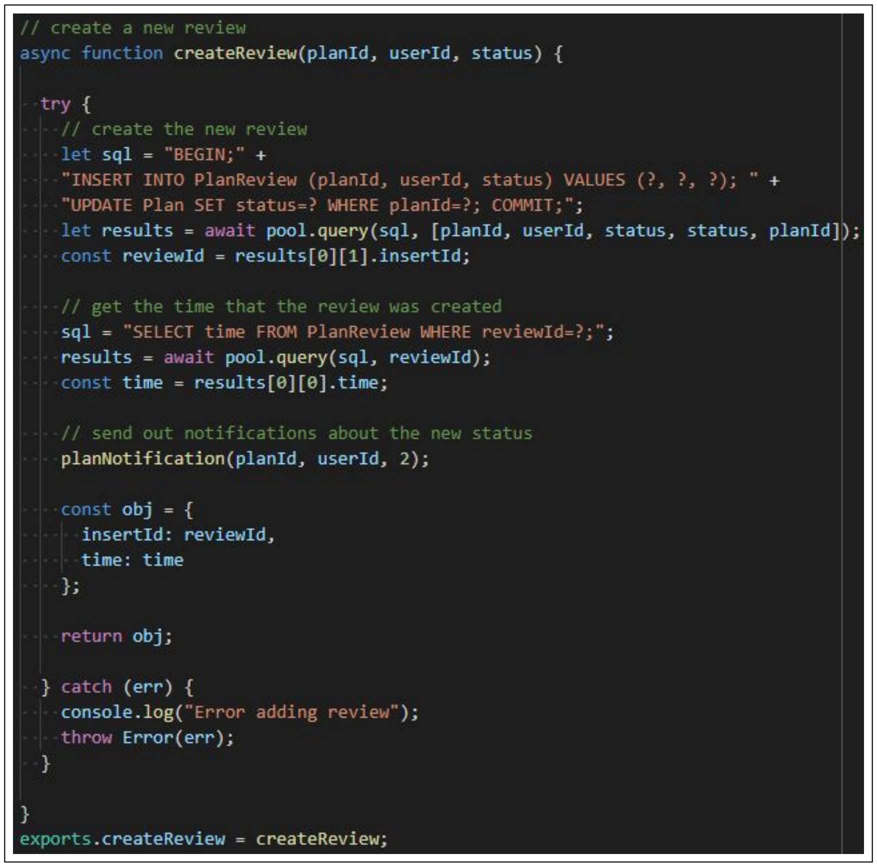
- POST comment

  This endpoint takes a request body that contains the plan ID of the plan to leave the comment on, the user ID of the commenter, and the comment's text. If the request passes validation then a new comment is created. This endpoint returns the comment ID of the newly created comment and the time that the comment was created.

2.9.5 Reviews

- POST review

  This endpoint takes a request body that contains the review ID of the plan being reviewed, the user ID of the reviewer, and the new status being assigned by the review. If the request passes validation then a new review is created. This endpoint returns the review ID of the newly created review and the time that the review was created.



```
// create a new review
async function createReview(planId, userId, status) {

  try {
    // create the new review
    let sql = "BEGIN;" +
    "INSERT INTO PlanReview (planId, userId, status) VALUES (?, ?, ?); " +
    "UPDATE Plan SET status=? WHERE planId=?; COMMIT;";
    let results = await pool.query(sql, [planId, userId, status, status, planId]);
    const reviewId = results[0][1].insertId;

    // get the time that the review was created
    sql = "SELECT time FROM PlanReview WHERE reviewId=?;";
    results = await pool.query(sql, reviewId);
    const time = results[0][0].time;

    // send out notifications about the new status
    planNotification(planId, userId, 2);

    const obj = {
      insertId: reviewId,
      time: time
    };

    return obj;

  } catch (err) {
    console.log("Error adding review");
    throw Error(err);
  }

}
exports.createReview = createReview;
```

Fig. 9: Function that creates a new review

2.9.6 Notifications

- GET notification

  This endpoint uses the access token sent with the request to get the current user's ID. This endpoint returns an array of all of the notifications that the user has yet to check.

- PATCH notification/:notificationId

  This endpoint takes the notification ID. If the request passes validation then the selected notification is marked as checked / seen. This endpoint returns the status code.

2.10 Database

We had decided that a relational database would make the most sense for our application, so we created a MySQL database. Our database has eight tables.

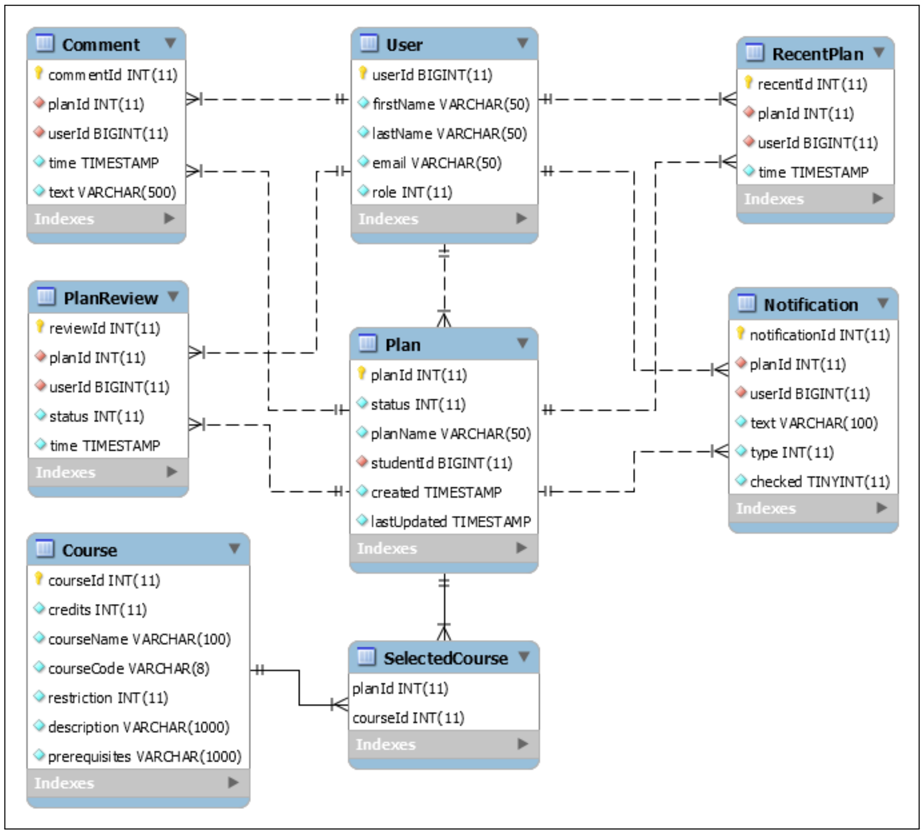


Fig. 10: EER Diagram of Database

2.10.1 Plan Table

The plan table stores general data about plans. It stores the plan ID, status, plan name, the user ID of the student that created the plan, the time when the plan was created, and the time when the plan was last updated.

2.10.2 User Table

The user table stores data about users. It stores the user ID, first name, last name, email, and the users role (student, advisor, head advisor).

2.10.3 Course Table

The course table stores data about courses offered at OSU. It stores the course ID, credit hours, course name, course code (ex: CS101), a restriction code (required course, graduate course), a description of the course, and any prerequisites.

2.10.4 *Notification Table*

The notification table stores data about notification that users have received. It stores the notification ID, the plan ID of the plan that the notification is referencing, the user ID of the user that the notification is for, the notification message, the type of notification (comment, status update), and if the notification has been viewed or not.

2.10.5 *Comment Table*

The comment table stores data about comments made on plans. It stores the comment ID, the ID of the plan that the comment was made on, the user ID of the user who made the comment, the time when the comment was created, and the comment's text.

2.10.6 *Review Table*

The review table stores data about reviews made to plans. It stores the review ID, the ID of the plan that was reviewed, the user ID of the user who made the review, the status that the plan was set to (rejected, awaiting student changes, awaiting review, awaiting final review, accepted), and the time the review was made.

2.10.7 *Selected Course Table*

The selected course table stores the courses that were selected for specific plans. It stores a plan ID and the course ID of a course that was selected for that plan.

2.10.8 *Recent Plan Table*

The recent plan table stores data about the plans that an advisor has visited. This data is used to allow advisors to see the last five plans they have visited. It stores the recent ID, the plan ID, the user ID of the advisor who visited the plan, and the time that the plan was visited.

3 Problems

3.1 Course Data

One of the earliest problems we faced was getting access to course data to use in our application. After researching possible solutions we invited Miguel Fernandez, who works on the API development team at OSU, to one of our client meetings to discuss the possibility of getting access to course data through an API. Miguel's team is working on a courses API but it will not be done by the time that we complete our application. We discussed with our client possible solutions and settled on developing the application to be easily modifiable to work with the course API, so that it can switched over to after we leave the project, while manually entering course data from the OSU course catalog for the time being.

3.2 Access Token Security

One of the requirements of our application was to allow users to sign in using their OSU credentials. Our client suggested using OSU's central authentication service to help achieve this goal. While integrating the central authentication service with our application we opted to use a simple but insecure method of sending the access token between our API server and the user. This was done with the intention of getting the basic functionality working so that the progress of other features would not be halted and with the plan of improving the security of this token transfer in the following term. Our solution in the following term will be to start using cookies to store access tokens as that method is more secure.

3.3 Meeting with the Advising Team

At the start of our project our client had informed us that Nick Malos, who is an advisor for the the School of EECS, would be joining us for our meetings and would be giving feedback on our project as it evolved. Roughly midway through our first term Nick stopped showing up for meetings and in the second term Nick shared with our client that he would be transferring to a different department. This unfortunately means that we have had less access advisor feedback then we would have liked. We have talked with our client about this issue and he has been in touch with Tyler DeAdder who is the Head EECS Advisor. Our client and Tyler have agreed that we will hold a meeting during this final term to focus on demoing our application so that we can get advisor feedback.

4 Remaining Work

Fortunately we were able to complete all of the core functionality that we were originally tasked with. With the core features out of the way we want to focus more on creating a polished final product, this means through testing, improving accessibility, and improving security.

4.1 Testing

For testing, we hope to focus on conducting informal user studies to assess the usability of the entire application and we would also like to develop unit tests to help find and fix bugs. We have been considering using Jest for creating our unit tests as it is a reliable JavaScript testing framework.

4.2 Security

One of our greatest concerns is the security of our application. Currently our application uses a convenient but insecure method to send user access tokens between the API server and the user. Our goal for this final term is to switch to using cookies for storing the authentication token. In this case the cookies would be set by the API server. This would mean that the user would be unable to read the contents of the cookie, but would still be able to send the cookie to the server with each request. This would fix a major vulnerability in our system, specifically by ensuring that the access token cannot be intercepted by a third party.

4.3 Accessibility

We want as many users as possible to be able to use our application. In that regard we are interested in accommodating users with disabilities. In addition to accommodating a variety of users we also want to ensure that our application works on as many devices as possible. To that end we will be incorporating responsive design into our application so that our application is viewable on a variety of screen sizes and orientations.

4.4 Hosting

We will also need to set aside some time to setup hosting for our application. Heroku is the platform we are planing to use for hosting as we can build, deploy, and scale our application.