COLORADO SCHOOL OF MINES
EARTH • ENERGY • ENVIRONMENT

CSCI 370 Report

Ramblin Wreck Reviews

Rachel Castro
Jenelle Gilliland
Andrew Nicola
Monique Streetman

CSCI 370 Fall 2024

Advisor: Tree Michael

Client: Donna Bodeau

Table 1: Revision history

<table>
<tr><td>Revision</td><td>Date</td><td>Comments</td></tr>
<tr><td>New</td><td>9/1/2024</td><td>Filled in Sections
I. Introduction
II. Functional Requirements
III. Non-Functional Requirements
IV. Risks
V. Definition of Done</td></tr>
<tr><td>Rev – 2</td><td>9/15/2024</td><td>Filled in Section VI. System Architecture</td></tr>
<tr><td>Rev – 3</td><td>10/20/2024</td><td>Filled in Section VII. Software Test and Quality and VIII. Project Ethical Considerations</td></tr>
<tr><td>Rev – 4</td><td>11/10/2024</td><td>Filled in Sections
IX. Project Completion Status
X. Future Work
XI. Lessons Learned
XII. Acknowledgments
XIII. Team Profile</td></tr>
<tr><td>Rev - 5</td><td>11/21/2024</td><td>Updated Sections
VI. System Architecture
VII. Software Test and Quality
XII. Team Profile
Replaced future tense with past tense.</td></tr>
<tr><td>Rev - 6</td><td>12/4/2024</td><td>Split section VI. System architecture to sections
VI. System Architecture
VII. Technical Design.
Shifted later section numbers
Added page numbers
Implement future tense and grammar feedback
Clarify parts of sections
I. Introduction
II. Functional Requirements
III. Non-Functional Requirements
VI. System Architecture
VIII. Software Test and Quality
X. Project Completion Status
XI. Future Work
Added to Appendix A. Key Terms</td></tr>
</table>

Table of Contents

I. Introduction.......... 3
II. Functional Requirements.......... 3
III. Non-Functional Requirements.......... 4
IV. Risks.......... 4
V. Definition of Done.......... 5
VI. System Architecture.......... 5
VII. Technical Design.......... 7
VIII. Software Test and Quality.......... 9
IX. Project Ethical Considerations.......... 14
X. Project Completion Status.......... 16
XI. Future Work.......... 16
XII. Lessons Learned.......... 17
XIII. Acknowledgments.......... 18
XIV. Team Profile.......... 18
References.......... 18
Appendix A – Key Terms.......... 19

I. Introduction

Mines currently uses Purdue University's Comprehensive Assessment of Team Member Effectiveness (CATME) evaluation platform to generate surveys and collect student feedback. CATME is used in all the Design and Senior Capstone classes at Mines, as well as several other team-based classes, which leads to significant costs. At an average of about $2 per student per semester, or about $15,000 per year, its widespread use adds up fast. It also lacks crucial features and has an outdated user interface, which can be confusing. We are creating an up-to-date, in-house solution for peer evaluations that includes all the necessary features.

Our client Donna Bodeau, a professor in the Engineering Design and Society Department at Mines, wishes for additional features such as admin-assigned gradings for multiple choice questions and department-specific approved and mandatory questions. Another key motivation is making this service expandable past peer evaluations and able to support other forms of evaluation and feedback collection.

We are the 7th group to take on this task. Previous groups have set up Amazon Web Services (AWS) and deployed the page on the domain ramblinwreckreviews.com. Little to no testing has been done on the codebase, the page uses purely mock data, and it lacks many necessary features. Our group has remedied this by establishing a baseline of unit, end-to-end, and integration tests to make the service stable, fully integrating with the database, and implementing the needed functionality. On top of these goals, our group would like to use the Fall CSCI 370 class, or one of our client's EDNS classes, to run a pilot for the service and receive feedback.

II. Functional Requirements

- Administrators (Department heads)
    - Manage available and mandatory questions for departments
    - Manage authorized superusers (Professors, TAs)
    - Assign grade weightings to multiple-choice questions
    - Create and assign additional departments and admins
- Superusers (Professors, TAs)
    - Create surveys/forms from allotted questions
    - View up-to-date form completion rates/other stats
    - View completed forms and associated grades
    - Manage registered students on a per-class basis
- Users (Students, other evaluators)
    - Fill out surveys assigned to them
    - View results from previous surveys
    - View feedback
- Forms/Surveys
    - Collection of individual questions
    - Created per professor
    - Filled out by users
    - Saved at creation-time
- Questions
    - Created only by admins
    - Per department basis
    - Short answer/free response
    - Multiple choice/bubble select
        - Automatic grading (from assigned weights)

III. Non-Functional Requirements

- Must be Federal Educational Rights and Privacy Act (FERPA) compliant
    - Protect student educational information and Personal Identifiable Information (PII)
        - Encrypt stored data
    - Students are only allowed to see their own data
    - Surveys remain anonymous
    - Only admins and superusers see other user's data
- Must be secure
    - Hosted on AWS Amplify under the Mines Enterprise Account
    - Implements AWS Cognito for authentication
    - Different permissions for admin, professors, and students
    - Only verified accounts should have access
- UI
    - Visually appealing
    - Easy to navigate
    - Consistent with other Mines websites

IV. Risks

The main risk of our project is security and Family Educational Rights and Privacy Act (FERPA) compliance. FERPA exists to protect student records from misuse and governs who can access personally identifying information (PII) tied to educational records. This means any PII, including but not limited to addresses, phone numbers, dates related to attendance or enrollment, or educational information, including any records related to grades, course transcripts, financial or loan records, student assessments, assignments, or attendance, can only be accessed by authorized personnel with student consent.

FERPA regulations hold that school officials who have a legitimate educational interest are exempt from requiring consent to view records. This includes professors, TAs, and other superusers and admins of our sites. Therefore, we do not need to worry about our superusers and admins viewing student data.

There have been multiple court cases regarding FERPA violations, resulting in cease-and-desist letters, fines, and additional penalties as state privacy laws apply.

In order to protect the institution and remain FERPA compliant, our project must protect student PII and educational records. This means primarily that students should not have access to the records of any other students. This can be accomplished with user permissions and should be a primary focus of our security.

Additionally, we should ensure that only verified administrators and superusers should be allowed to log in as such. If unauthorized users access these accounts, they could also gain access to PII and educational records. This means that we need to have a sufficient verification system to remain FERPA compliant. We will not be using a single sign-on (SSO) system to verify students at the request of the stakeholder, so password encryption must be strong. We are using AWS' Cognito service, which is FERPA compliant. AWS cannot see what kind of data is being stored or accessed, as it is all encrypted.

V. Definition of Done

- The website is in a usable state, with data pulled from and stored in our database.
  - All functional requirements from the list above are implemented
  - FERPA compliance is maintained
- Grades can be calculated based on student responses and administrator-designated weightings
- Emails are sent out automatically to students who still need to complete a survey
- Documentation is created to facilitate an easier transition for any future field session teams or other maintainers

Our software was delivered both as a functioning website (at ramblinwreckreviews.com) and as a GitHub repository containing the codebase. Our client will test our software by using it for peer review in one or more of her classes.

VI. System Architecture

Technical Issues

So far, our team has encountered only a few technical design issues. To produce a successful connection between API Gateway and the React app, we had to set a Cross-Origin Resource Sharing (CORS) policy to open up security. Future teams must address security access controls after completing the system.

System Architecture Components

- Frontend
  - Users interact with the web application.
  - Written with React.js.
  - User requests are sent via HTTP to the backend.
- Amazon S3
  - Simple storage service (S3) manages uploading/storage of small documents.
  - Contents of files are parsed by Lambda and sent to the RDS.
- AWS Lambda
  - Serverless computing that breaks the application into small functions that can be run independently.
  - Handles traffic to and from the API gateway, S3, Cognito, and other services.
  - Functions are mostly written in Python, some are auto generated in Node.js.
- Amazon API Gateway
  - Manages API endpoints, provides access to backend services.
  - REST API framework.
- Amazon RDS
  - Relational database service (RDS) houses our postgreSQL database for data storage.
  - Lambda functions interact with the database to perform database operations.
- Amazon Cognito
  - Provides user authentication services.
  - Manages user pools and associated privileges as well as sign-in service.
- AWS IAM
  - Identity and access management (IAM) controls permissions for accessing AWS services.
- Amazon SES
  - Simple email service (SES) sends emails from our web application.
  - Emails are triggered by Lambda functions or by Cognito.

Figure 1: Microservice Relationship Diagram

Design considerations

Several system requirements had significant impacts on our architecture decisions. Our client requires that we not use the Mines SSO for authentication, as she wants this to be a standalone system, not owned by Mines IT. Using the Mines SSO system would require an application update whenever the SSO was changed, and would restrict the platform to only Mines-affiliated users. Our client desires a system that does not need maintenance, as one using the Mines SSO system potentially would. Not using the Mines SSO also allows for potential future monetization via expansion to other universities. The previous group decided to use AWS Cognito for authentication, and that seems to work well with the rest of the architecture.

FERPA compliance is another big thing we had to take into account. Since this application handles PII, we have to ensure that only authorized users can access this information. This was accomplished mainly within the backend code; however, it was important to consider when determining what technology stack would work best. The entire AWS ecosystem has the ability to be FERPA compliant assuming it is set up right. In addition, all student data will be deleted after 7 years of inactivity (timeline specified by client), and surveys will be deleted 7 years after they close.

VII. Technical Design

API Structure

We decided to use a REST API, set up within AWS API Gateway. We have 2 top-level resources, departments, and courses. Off of departments, there are endpoints to modify faculty members, questions, and multiple-choice question weightings. For courses, there are endpoints to modify course faculty, students, and surveys, as well as survey responses. A diagram of the API structure is shown below.

- /courses
  - /:courseId
    - /faculty
      - /:facultyId
    - /students
      - /:studentId
    - /surveys
      - /:surveyId
        - /responses
          - /:responseId
- /departments
  - /:deptId
    - /faculty
      - /:facultyId
    - /mcq-weights
    - /questions
      - /:questionId
  - /inactive
- /users

Database Structure

The previous team set up a database using PostgreSQL, and we have decided to continue with that design. Users are stored with an ID, first name, last name, email, CWID, created time, and most recent login time. Users are connected to departments through the user_role table, which has an id, user id, resource id, resource type, user type, status, created and updated times, and who invited the user to the department. This table shows the relationship a user has to a department, whether they are an Admin or a Professor, and if they are active in the department.

Departments are stored with an ID, multiple choice question weight ID, abbreviation, department name, time of activation, who activated the department, and status. All of the departments are stored in the table. When a new department wants to join RamblinWreckReviews, its status is changed from inactive to active.

The mc_config table stores all the information about the configurations for multiple-choice questions. It has an id, department id, a JSON structure of the question weights, status, time of creation, and who created the question weights. The JSON has category fields, the number of bubbles, and the grade weightings. Each department has one weight configuration for uniformity between classes.

Questions for surveys are stored in the question table with an ID, department ID, the question content, if the question is mandatory or not, status, the type of question (multiple choice or short answer), who created the question, question title, category, and when the question was modified. Questions have question weights applied based on which department that question is a part of.

The post_services_proc is a stored procedure that takes in a user's email, first name, last name, the user's role, and their CWID. This function inserts a user into the users table with their email, first name, last name, time of creation, and last login.

The create_question_weights stored procedure takes in a department ID, a JSON of question weights, and who created it. This function inserts a new question weight configuration into the mc_config table, including the department ID, the question weights JSON, status, the present time for creation, and who created the weights. This function also updates the mcq_weight_id in the department table.

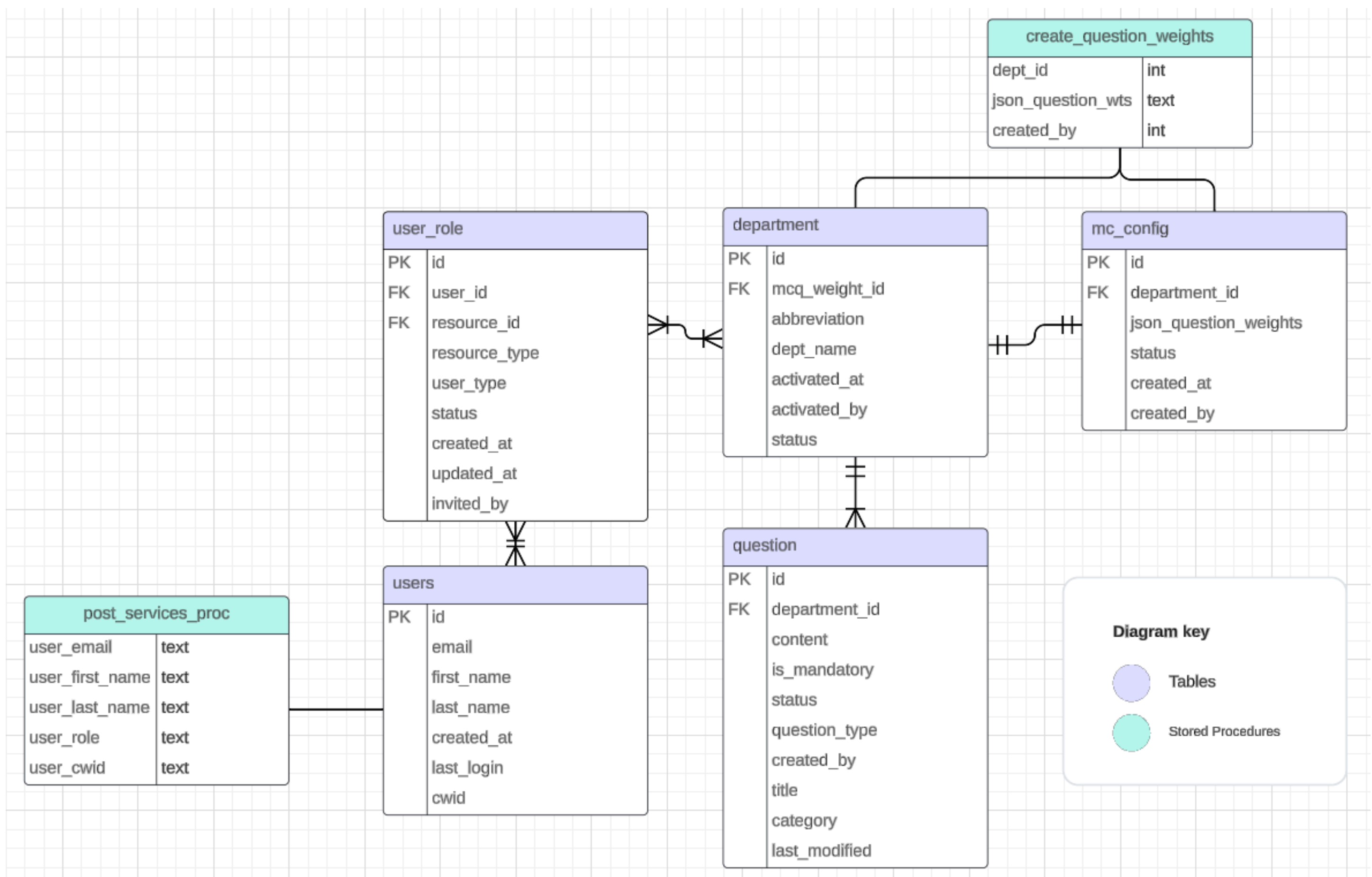

Figure 2: Entity Relationship Diagram (ERD)

VIII. Software Test and Quality

We did two methods of testing our software: Automated unit tests and manual integration tests. Unit testing was done on the front end with Jest and the react-testing library, and on the backend using the testing features present in AWS Lambda. All of these tests were run and thorough integration testing was done with all new features and any modified features before any deployment to production. We used the ESLint linter with react presets on the front end to help keep a consistent code style and catch bad coding practices. We also implemented code reviews to ensure maintainability and consistency between different components. Unit tests checked for correct functionality, and integration tests checked to ensure everything works together as expected.

In order to integrate and automate the front-end unit testing we used Jest, a simplistic JavaScript testing framework, as well as the react-testing library, which helps test frontend components. These tools allowed us to ensure that our components behaved as expected by simulating user interactions, testing component rendering, and validating outputs under various conditions.

The tests we ran for major components include:

- Question Rendering
  - **Purpose:** Ensure the Question component renders the correct type of question.
  - **Description:** When admins add a question, test if it renders a bubble or text question depending on their input.
  - **Tools/Environment:** Manual testing, Jest, react-testing
  - **Aspect of Code Quality:** Validation
  - **Results:** Implemented, passing
- Menu Bar Rendering
  - **Purpose:** Ensure the MenuBar component renders the menu for the permissions of the user.
  - **Description:** When any page is loaded, the MenuBar should load with the specific options for the three roles: user, superuser, and admin.
  - **Tools/Environment:** Manual testing, Jest, react-testing
  - **Aspect of Code Quality:** Validation
  - **Results:** Implemented, passing
- Menu Bar Chevron
  - **Purpose:** Ensure the MenuBar component renders the chevron, and that when the chevron is clicked or hovered over it displays or hides the menu.
  - **Description:** When any page is loaded, the MenuBar should load with a chevron to display or hide the menu. It should display and hide as necessary.
  - **Tools/Environment:** Manual testing, Jest, react-testing
  - **Aspect of Code Quality:** Validation, consistency
  - **Results:** Implemented, passing
- Nav Card Rendering
  - **Purpose:** Ensure the NavCard component renders the correct options for navigation given test props.
  - **Description:** When any page is loaded, the NavCard should be visible and the user should be able to interact with it.
  - **Tools/Environment:** Manual testing, Jest, react-testing
  - **Aspect of Code Quality:** Validation, consistency
  - **Results:** Implemented, passing
- Drop Zone
  - **Purpose:** Ensure the DropZone component uses the given DropZone hooks when called.
  - **Description:** When the DropZone component is interacted with by the user, the DropZone should use the DropZone hooks.

    - **Tools/Environment:** Manual testing, Jest, react-testing
    - **Aspect of Code Quality:** Validation, consistency
    - **Results:** Some implemented, passing - Further tests on the DropZone component should be implemented.

We did not have time to complete as much testing as we wanted. These are some tests that should be implemented at some point.

The tests that could be run for Student privileges and interactions include:

- Student Home
    - **Purpose:** Ensure student users can navigate to pages like "Take Survey," "View Results," and "Update Profile."
    - **Description:** When students log in, they should see a dashboard with cards linking to different sections. Clicking on each card should route them to the corresponding page.
    - **Tools/Environment:** Manual testing, Jest
    - **Aspect of Code Quality:** Coding standards, consistency
    - **Results:** Test not implemented
- Student Take Survey
    - **Purpose**: Ensure students can take surveys correctly.
    - **Description**: Student users can select and submit answers to survey questions. The page should dynamically load the questions, allow for answer selection, and submit responses properly.
    - **Tools/Environment**: Manual testing, Jest
    - **Edge Cases**: If the student tries to submit without answering all required questions, they should be prompted to complete the survey.
    - **Aspect of Code Quality**: User experience, consistency, validation
    - **Results:** Test not implemented

The tests that could be run for Professor privileges and interactions include:

- Professor Manage Students
    - **Purpose:** Ensure professors can manage student enrollment and status.
    - **Description:** Professors can view and edit the list of students enrolled in their courses, including changing their status or course information.
    - **Tools/Environment:** Manual testing, Jest
    - **Edge Cases**: Professors should not be able to remove students who have already completed or submitted work.
    - **Aspect of Code** Quality: Coding standards, verification
    - **Results:** Test not implemented
- Professor Manage Survey
    - **Purpose:** Ensure professors can create, edit, and delete surveys for their courses.
    - **Description:** Professors can create new surveys, modify existing surveys, and delete surveys from the system.
    - **Tools/Environment:** Manual testing, Jest
    - **Edge Cases:** If a professor tries to delete a survey that has already been taken by students, a warning should appear, and deletion should be restricted.
    - **Aspect of Code Quality:** Validation, coding standards
    - **Results:** Test not implemented
- Professor Student Responses
    - **Purpose:** Ensure professors can view the responses from students to surveys.

    - **Description**: Professors can view individual and aggregated responses from students to survey questions, including filtering options to view specific student data.
    - **Tools/Environment**: Manual testing, Jest
    - **Edge Cases:** Responses should be displayed in chronological order, and professors should not be able to alter or delete students' responses.
    - **Aspect of Code Quality:** Data integrity, verification
    - **Results:** Test not implemented

The tests that could be run for Admin privileges and interactions include:

- Admin Home
    - **Purpose:** Ensure admin users can navigate to the pages "Edit Question Weights", "Manage Faculty", "Create Survey Questions", and "Create Department"
    - **Description:** When admins are on their homepage, they have a selection of cards to navigate to other pages, clicking on them should route them to the intended page
    - **Tools/Environment:** Manual testing, Jest
    - **Aspect of Code Quality:** Coding standards, consistency
    - **Results:** Test not implemented
- Admin Edit Question Weights
    - **Purpose:** Ensure administrators are able to configure the bubble choice questions
    - **Description:** Admin user adjusts the category sliders and it dynamically updates the fields
    - **Tools/Environment:** Manual testing, Jest
    - **Aspect of Code Quality:** Consistency
    - **Results:** Test not implemented
- Admin Edit Question Weights Error Detection
    - **Purpose:** Verify that user input is valid
    - **Description:** Admin user needs to input category names for all selected categories, assigned grades are strictly increasing and between 0-100
    - **Tools/Environment:** Manual testing, Jest
    - **Edge Cases:** If a user has an empty category before a filled one it should still be marked as invalid
    - **Aspect of Code Quality:** Validation
    - **Results:** Test not implemented
- Admin Manage Faculty Add
    - **Purpose:** Ensure admin users can add the other admin/professors in their department
    - **Description:** Admins can add other faculty to their department with either the professor or admin role, updating the invited by field to be the current admin
    - **Tools/Environment:** Manual testing, Jest
    - **Edge Cases:** If a user that is an admin of a different department, they cannot be added as an admin to the current department
    - **Aspect of Code Quality:** Coding standards, verification
    - **Results:** Test not implemented
- Admin Manage Faculty Edit
    - **Purpose:** Ensure admin users can add the other admin/professors in their department
    - **Description:** Admins can change pertinent information which updates the "Last Modified" field with the current date
    - **Tools/Environment:** Manual testing, Jest
    - **Edge Cases:** Promoting a user to admin should only be possible if they are not already an admin of a separate department, admins should not be able
    - **Aspect of Code Quality:** Coding standards, verification
    - **Results:** Test not implemented
- Admin Manage Faculty Filtering

    - **Purpose:** Ensure admin users can add/edit/disable the other admin/professors in their department
    - **Description:** When admins view the faculty table, they should be able to create and use filters
    - **Tools/Environment:** Manual testing, Jest
    - **Edge Cases:** By default we sort by enabled users first, so unless a filter directly checks enabled status, disabled users should all be concentrated towards the end
    - **Aspect of Code Quality:** Coding standards, verification
    - **Results:** Test not implemented

For our back-end AWS Lambda testing, we used the test events provided in the Lambda Console and automated with AWS CodePipeline. We did not have time to implement all of these. These tests include:

- Verify Departments Retrieval
    - **Purpose:** Ensure that data pulled from this Lambda function is an accurate list of the active departments
    - **Description:** Test event in Lambda calling the getDepartments function, should take no parameters
    - **Tools/Environment:** AWS Lambda, CodePipeline
    - **Results:** Implemented, passing
- Department ID Retrieval
    - **Purpose:** Verify that department objects are properly pulled by ID
    - **Description:** Call the getDepartmentByID function with valid and invalid IDs, error messages should be returned with invalid IDs
    - **Tools/Environment:** AWS Lambda, CodePipeline
    - **Results:** Implemented, passing
- Test Users are Properly Stored
    - **Purpose:** Check that users are properly added to all necessary tables
    - **Description:** Call the addUser function, creating user stubs in the DB if they don't already exist
    - **Tools/Environment:** AWS Lambda, CodePipeline
    - **Edge Cases:** If a user does not already have an account, they are sent an email by AWS SES inviting them to create an account
    - **Results:** Test not implemented
- Test Add Student (already in class)
    - **Purpose:** Verify that no work is done adding students to courses when they are already in them
    - **Description:** Add a student to a class who is already in the class, already in the DB tables, and already in the Cognito tables
    - **Tools/Environment:** AWS Lambda, CodePipeline
    - **Results:** Test not implemented
- Test Add Student to Survey
    - **Purpose:** Verify that existing student is properly added to the survey, also pushing an email notification
    - **Description:** Add a student to a survey that is not already in the survey, but is in the DB tables, and in the Cognito tables
    - **Tools/Environment:** AWS Lambda, CodePipeline
    - **Results:** Test not implemented
- Test Add Student New
    - **Purpose:** Verify that new student is properly added to the survey, and are sent an invitation email notification
    - **Description:** Add a student to a class who is not in the survey, not in the DB tables, and not in the Cognito tables
    - **Tools/Environment:** AWS Lambda, CodePipeline
    - **Edge Cases:** If a student has been invited before but did not make an account they would be in the DB tables and we should not make additional entries
    - **Results:** Test not implemented
- Test Add Professor

- **Purpose:** Verify that the professor is properly added to the department
- **Description:** Add a user as a professor to departments, test with both non-existing and existing user info
- **Tools/Environment:** AWS Lambda, CodePipeline
- **Edge Cases:** If professors don't have an account they are invited by email, otherwise they are notified of being added to a course
- **Results:** Test not implemented

- Test Add TA
  - **Purpose:** Verify that the student is added as a TA, given professor read permissions only for that course
  - **Description:** Adding an existing student as a TA to a particular course, should update roles and permissions
  - **Tools/Environment:** AWS Lambda, CodePipeline
  - **Edge Cases:** If a student has been invited before but did not make an account they would be in the DB tables and we should not make additional entries
  - **Results:** Test not implemented

Alongside the unit testing via Jest and react-testing we conducted End-To-End testing with Cypress. AWS Amplify has the most support for the Cypress test framework, allowing us to create a copy of our cloud configurations locally without interfering with our production database instances. These tests run on the assumption that our UI, front-end, and back-end unit tests are passing as having duplicate tests in different frameworks is redundant and cumbersome to rewrite when business logic has been adjusted. We did not have time to implement all of these. Our tests are as follows:

- Admin Create Department
  - **Purpose:** Ensure admins can initialize departments with a specific user as the admin
  - **Description:** We test form submission with a variety of inputs following different scenarios, some successfully create the department and some fail to create.
  - **Tools/Environment:** Cypress
  - **Edge Cases:** If the user does not have an account, ask the admin creating the department if they are sure they want to add this user. If the user is already an admin of another department, show a descriptive error popup. If the user is somehow able to input an email in the Mines domain, show an error popup. If the department has already been created, show an error popup.
  - **Results:** Test not implemented
- Admin Manage Questions - Load
  - **Purpose:** Ensure admins can view their department questions
  - **Description:** We test the loading and parsing of questions and questions categories
  - **Tools/Environment:** Cypress
  - **Edge Cases:** Categories without questions should not be displayed
  - **Results:** Implemented, passing
- Admin Manage Questions - Create
  - **Purpose:** Ensure admins can create questions in existing categories
  - **Description:** Add questions to already existing categories, and verify they are persistent between reloads
  - **Tools/Environment:** Cypress
  - **Edge Cases:** Categories without questions should not be displayed
  - **Results:** Test not implemented
- Admin Manage Questions - Create Category
  - **Purpose:** Verify admins can create a new category, populating it with a question alongside
  - **Description:** Fill out the add category form, check that a new category is added if it doesn't exist
  - **Tools/Environment:** Cypress
  - **Edge Cases:** If the category already exists, the question should go into that category, rather than making a new one
  - **Results:** Test not implemented

- Admin Manage Question Weights
  - **Purpose:** Verify admins see the current department multiple choice question weightings before editing them
  - **Description:** Load onto the manage question weights page and verify the current weightings are received
  - **Tools/Environment:** Cypress
  - **Edge Cases:** If there is no mcConfig associated with the department, it should use the page should display a default one
  - **Results:** Implemented, passing

We also have Non-Functional Requirements Testing

- FERPA Compliance
  - **Purpose:** Ensure student data is protected according to FERPA guidelines
  - **Description:** User permissions should be closely managed and users should not be able to see more than they are able to, there should be authorization checks for all DB data retrieval
  - **Tools/Environment:** AWS Lambda, manual testing
  - **Edge Cases:** Students as TAs should only be able to see what a professor would for that particular course, permissions should be managed entirely from the back end to prevent role escalation attacks
- Secure Connections (HTTPS)
  - **Purpose:** Ensure user connects are securely managed via HTTPS
  - **Description:** The website is hosted via AWS, providing secure connections for users
  - **Tools/Environment:** Web browser, AWS Hosting
  - **Edge Cases:**
  - **Results:** All attempts to connect to HTTP version of site are automatically upgraded to HTTPS.
- Accessibility
  - **Purpose:** The website must be accessible to all users
  - **Description:** The website should work for various screen sizes and should be ADA compliant, meaning using alt text for images, providing clear instructions and error indicators
  - **Tools/Environment:** Web browser, screen reading software (narration)
  - **Edge Cases:** Landscape/portrait displays, mobile phones
- UI
  - **Purpose:** Ensure the UI is approachable and user-friendly
  - **Description:** Collect user feedback from potential users with various levels of experience on the application for all roles
  - **Tools/Environment:** Manual testing, feedback

IX. Project Ethical Considerations

Ethics Principles Pertinent to Development

IEEE: 2.03. Use the property of a client or employer only in ways properly authorized, and with the client's or employer's knowledge and consent [1].

This principle is especially relevant because our product handles sensitive information. The data provided by clients and employers should be used ethically and responsibly with proper authorization. Our product needs to ensure data privacy and security, following the FERPA guidelines. Also, access to Mines' AWS account must be used strictly within the permissions set by the client. Precautions have been taken to prevent unauthorized access or misuse of the AWS account.

ACM: 3.6 Use care when modifying or retiring systems [2].

This principle is pertinent to the development of our product because we have inherited it from a previous team. Before changing existing systems, we needed to understand the design and architecture thoroughly. Hasty modifications could break the current functionality and hinder new progress. Many systems had skeletons sketched out by the previous team, so we built our systems on top to maximize the useability provided by our predecessors. For future teams, our work was documented so development can continue without confusion and setbacks.

Ethics Principles in Danger of Being Violated

ACM: 2.9. Design and implement systems that are robustly and usably secure [2].

Currently, our design has limited security in place provided by the Mines AWS account. There is strong account authentication that was provided by the previous team, but less security for the data. For coding and testing purposes, security has been lessened. When the architecture is complete and ready to be used, stronger security will be put in place. If this principle is violated, there is the risk of sensitive data being exposed. Information that is shared in confidence could be used to harm clients. Users would no longer trust our product, so all operations would have to stop.

Ethics Tests

Common Practice Test: This test asks whether our model aligns with standard industry practices. In software development, the widely accepted ethical practices regarding private data usually include encryption, access control, and data minimization. Our team has implemented these during our development process. Data is naturally encrypted server-side in all AWS microservices, which we are exclusively using for our project. Our web app is built around access control; we have three separate roles that can only access exactly the information they need to see. Between departments and the roles, users do not see any data beyond their own (if they're students), their class (if they're professors), or their departments (if they are administrators). We have also minimized the data we need to store regarding users. Instead of using CWIDs as a unique identifier, we are using student emails. Our database stores only their name, emails, and any survey results connected to them. All of these data safety practices are also in line with FERPA compliance, which we are required to implement in our project as we are dealing with educational data. Additionally, we have implemented AWS Cognito as an authentication service, which is a secure microservice that ensures unauthorized users are not accessing our app.

Legality Test: This test examines whether the development of our app complies with relevant laws and regulations. The main law that applies to our project is FERPA (Family Educational Rights and Privacy Act), briefly discussed above. Our web app collects and manages student data which is protected by FERPA. Our project is designed with FERPA compliance as a fundamental principle, ensuring that students can only access their data and that PII is protected. Additionally, our app includes a user permissions system and data handling that satisfy the legal requirements of FERPA. The decision to not use a single sign-on system also aligns with the need for maintaining control over access to sensitive data. Any unauthorized access could lead to fines from the government or from individual legal access from students whose data is exposed.

Ethical Considerations

If our software quality plan is not correctly implemented, we are mainly at risk for a customer data breach. Data breaches in the past have had far-reaching consequences. The first one is a loss of trust of users of our site; students and schools may choose not to use our project if they cannot be sure their data is safe. Additionally, in the event of a data breach, the owners of our project could potentially be exposed to liability. As discussed in the previous section, adherence to FERPA is a primary concern of our project. Violation of FERPA requirements could result in fines, bans, or individual lawsuits from students whose data is exposed. Unauthorized disclosure of students' PII can harm students' reputations and sense of privacy.

A second ethical concern is a lack of transparency and accountability. Student grades may be based on the results of the survey we provide, so it is important to ensure results are accurate. If our web app does not display results and provide transparency in how they are obtained, we could be at risk of displaying inaccurate data and potentially affecting student grades. We need to ensure our project is accurate.

X. Project Completion Status

Despite our optimism and enthusiasm for this project, we were unable to achieve the goals we set out at the beginning of the semester. We made significant progress on the project, but there is still much to accomplish. When we started, we were handed a bare-bones website with no functionality. We implemented all of the major pieces of functionality for one of our three principal user groups and started on the other two. In addition, we set up a REST API with authentication workflows to control communication between the website and the AWS backend services, and migrated the frontend into a repository controlled by our client instead of the previous group. We implemented some basic testing, but not as much as we hoped to accomplish.

We were able to complete the functionality of the Admin page. This included four components: managing questions, managing faculty, editing question weights, and creating departments, as well as a minimal homepage.

The Manage Questions page displays the questions by category, showing the title, question type, mandatory status, and activated status. The questions can be filtered by title, type, content, mandatory status, category, activated status, who created the question, when the question was created, and when the question was last modified. These descriptions are also displayed underneath each question. New questions can be added by inputting a question title, checking if the question is mandatory, inputting the question itself, and choosing the question type: multiple choice or short answer.

The Manage Faculty page displays the names, access levels, and status of all of the users who are either Admins or Professors. New users can be added by inputting their first and last name with their email and their role, whether Admin or Professor. This process will grant an existing account the proper role or create an account for the user if none exists. If users are no longer an Admin or Professor, their status can be switched to Disabled. Users can be filtered by name, email, access level, status, who they were invited by, when they were invited, and last modified date.

The Manage Question Weights page allows Admins to change how multiple-choice survey questions are structured. Admins can change the number and title of broad categories (ex. Excellent, Good, Poor), and the number of subdivisions in each category. The grading of these questions can be set manually or autoscaled with a minimum and maximum grade value. All of these settings are saved to a department. Below the settings is a preview that shows what the students see when taking a survey.

The Create Department page allows Admins to activate departments that want to use RamblinWreckReviews. It has a dropdown of every department at CSM that is not active. Choosing a department will activate it by creating an Admin for that department, via the same process used on the Manage Faculty page.

XI. Future Work

Our team has implemented the Admin functionality, with the ability to manage questions, manage faculty, edit question weights, and create departments. We were able to get started on the Professor Functionality, but more development is needed. Below is a list of features that need to be implemented in the future:

- **The Professor Homepage:** List the surveys and show how many students have completed them. Have the option to edit the surveys. Be able to view the results of an individual survey.

- **Create Course:** Create a new course in a department, and add surveys and view-only professors/TAs to that course.
- **Manage Surveys:** Create a new survey with a title, start and end date, required questions, optional questions, and a list of students with their associated teams. Student information should be uploadable via CSV or entered manually. The CSV functionality should partially exist from the previous group. Page should have a mechanism to release survey results after they are reviewed. This mechanism should include an option to release feedback comments with names attached rather than anonymously.
- **Manage Students:** Edit student information and remove students from a survey. Manually add students by first name, last name, and email. If a survey has finished, allow re-entry for specific students.
- **Student Responses:** List team groups and the students' responses for each group. Have the option to suppress individual answers from being included in the feedback students can access. List the grades of each student with and without personal evaluation.
- **Email Notifications**: Send email notifications to students to remind them to take the survey.
- **Student Homepage:** List all of the surveys a student has to take, as well as completed surveys.
- **Surveys:** Allow student access to surveys within the specified time window. Surveys must have the ability to be saved, and students should be able to go back in and complete the survey and edit their answers at any point during that window.

By focusing on these areas, future teams can build on our foundation and move closer to delivering a fully functional, scalable, and user-friendly peer review platform for the Colorado School of Mines. The Admin functionality has been provided, in addition to the previous team's skeleton for the Professor and Student functionality. All that needs to be done is to fill in the missing features.

We estimate that a minimum viable product (MVP) could be completed by the end of the next field session, with the possibility for several more semesters of work in improving features and adding on other surveys.

In addition, several security concerns need to be addressed before this is rolled out. The CORS policy is currently completely open for testing purposes, and that should be tightened and validated. SQL injection should already be handled with the postgres tools we use, but this should be verified as well.

XII. Lessons Learned

Documentation is a key part of development. We learned that being able to read documentation when looking into new packages, libraries, or services, and reviewing code with a lack of documentation are crucial skills for software developers to have. We took on a codebase consisting of several previous groups worth of code in various conditions, with little to no comments in code and a brief overview of what each page does. Like any project, we are not the only ones working on code so it is pertinent that we document as much as we can.

Writing maintainable and scalable code is difficult and takes much longer to develop, but React provides a good framework to build on. Refactoring pages into compositions of components, using modular designs, and following general React paradigms are key to writing maintainable code. Following the DRY principle also leads to maintainable code, especially as project parameters change or when business logic needs to be revisited, as things need only be changed in one place.

The AWS Suite provides many useful features but also comes with several challenges. We learned how to switch from the previous group's public repository to a new private repository managed by our client, how to configure a secure API utilizing AWS Authorization services, and how to support the bulk of our application functionality.

Incorporating procedures into our databases ensures the consistency of our data and enables more complex actions to be taken during typical usage.

The AGILE development process allows us to approximate the scope of what we can get done. For example, if we have team members struggling to complete work, we can communicate with our client and adjust team plans and project scope accordingly. For effective AGILE usage, we had to be able to accurately estimate how much effort a component would take, which is something we struggled with at the beginning and got much better with later on.

XIII. Acknowledgments

We want to thank our client, Donna Bodeau, for being extremely gracious with us and listening to our feedback and questions throughout the semester. We would also like to thank our advisor, Tree Michael, for sticking with us along the way and offering advice as things came up and life happened.

XIV. Team Profile

Monique Streetman

Monique Streetman is a Junior Majoring in Computer Science on the Data Science track. She is a member of the Mines Climbing Team, and mentors an FRC team. She was responsible for the authentication workflow, dynamic table, and some backend functionality.

Andrew Nicola

Andrew Nicola is a senior majoring in Computer Science in the Data Science track. From Castle Rock, Colorado, he is passionate about music and video games. He was in charge of UI development and some backend functionality.

Jenelle Gilliland

Jenelle Gilliland is a senior majoring in general computer science. She is from Littleton, Colorado. She enjoys reading and playing video games. She was responsible for connecting the website to the database through PostgreSQL and AWS Lambda.

Rachel Castro

Rachel Castro is a senior majoring in Computer Science + Data Science. She is from Boulder, CO and enjoys skiing and playing Factorio. She was responsible for some AWS Lambda functionality and unit testing.

References

[1] IEEE, “Code of Ethics,” IEEE Computer Society, https://www.computer.org/education/code-of-ethics (accessed Nov. 8, 2024).

[2] ACM, “ACM Code of Ethics and Professional Conduct,” Association for Computing Machinery, https://www.acm.org/code-of-ethics (accessed Nov. 8, 2024).

Appendix A – Key Terms

Term	Definition
AWS	*Amazon Web Service*
CATME	*Comprehensive Assessment of Team Member Effectiveness*
CORS	*Cross-Origin Resource Sharing*
FERPA	*Family Educational Rights and Privacy Act*
MVP	*Minumum Viable Product*
PII	*Personal Identifying Information*