



**COLORADO SCHOOL OF MINES**  
EARTH • ENERGY • ENVIRONMENT

# CSCI 370 Final Report

*5<sup>th</sup> Time's the Charm*

Client: CSM Bodeau

Chloe Cadwell  
Brennen Denault  
Srijita Ghoshal  
Grant Weimer

Revised December 5<sup>th</sup>, 2023

CSCI 370 Fall 2023

Mr. Tree Michael

Table 1: Revision history

Revision	Date	Comments
New	September 3, 2023	Initial creation of requirements & design documentation.
Rev – 2	September 15, 2023	Filled sections I, IV, V, VI and XIII
Rev – 3	October 10, 2023	Revised based on advisor feedback, updated ERD
Rev - 4	October 19, 2023	Included information on Software Test and Quality, along with Ethical Considerations
Rev - 5	November 11, 2023	Included information on results, and lessons learned
Rev - 6	December 5, 2023	Grammatical fixes before peer review
Rev -7	December 10, 2023	Last paper revision before final submission

## Table of Contents

I. Introduction .....	3
II. Functional Requirements.....	3
III. Non-Functional Requirements.....	5
IV. Risks .....	5
V. Definition of Done .....	6
VI. System Architecture .....	6
VII. Technical Design .....	8
VIII. Quality Assurance .....	11
IX. Project Ethical Considerations .....	14
XI. Results.....	15
XII. Future Work.....	17
XIII. Lessons Learned.....	18
XIII. Acknowledgments .....	19
XIV. Team Profile.....	19
References .....	20
Appendix A – Key Terms .....	20

## I. Introduction

As a student at the Colorado School of Mines, there are many projects that require collaboration and teamwork. At the end of these projects, peer evaluations are crucial for determining student grades and feedback. Mines is currently utilizing Purdue University's Comprehensive Assessment of Team Member Effectiveness (CATME) software to host these peer evaluations. This application is hosted on the web and can be easily accessed by any Mines student who creates an account. Additionally, CATME has vast functionality including creation of surveys, comprehensive psychological evaluations, and detailed individual scores. Furthermore, CATME has preset categories that professors can choose to include questions from within their surveys. These categories include topics like teamwork, positive attitude, individual contributions, and more. The software is an efficient way for students to leave comments about their experience in a class or give individual feedback about their teammates.

However, the biggest downfall of this peer-evaluation software is the cost. In fact, CATME comes with an average cost of \$5000 each semester; a whopping price that Colorado School of Mines cannot continue to afford. Furthermore, CATME is crucial for the Engineering, Design, and Society (EDNS) Department, where the peer evaluation software is used for many courses. Therefore, the development of a web application supporting the same functionality as CATME is crucial.

In addition to cost, many professors are not completely satisfied with CATME's user experience. The end goal of this project is to provide a cost-effective long-term alternative to CATME along with software that meets client needs. This project will also have to be very stable with minimal maintenance, as there will not be dedicated staff for upkeeping the website. Donna Bodeau is an Engineering Design professor here at Mines, and she has been our client throughout this development process. Her vision is to have this software be utilized for any course here at Mines that requires a peer evaluation.

Originally, an alternative to CATME was developed by previous CSCI 370 groups starting in 2021, but there have been continuous issues with the stability of the current application. The main concern being unexpected and undefined crashing of the website which is hosted on Amazon Web Service (AWS) on a LightSail server. The main framework of the application is Django. Django is developed to be primarily used as a backend framework, but in this application, it is used as a full-stack framework in this project. Considering Django is a back-end framework, it is not best suited for front-end work. Therefore, a migration to a better-suited framework is essential.

Despite having four previous teams working on this project, progress seems to have reached a deadlock. There exists minimal to no documentation about the code and testing of the current application. Besides the little documentation, the current framework of this application is not well designed and could be the source for unexplained crashes. The vision is to redesign the current structure of the application, such that it has a stable foundation for future mutations.

## II. Functional Requirements

- User Roles
  - Admin/Department Head
    - Create questions
      - Can specify if question is mandatory
      - Determine bubble weight for each question
    - Add other admins and professors
    - All privileges of professor and student roles
  - Professor and Teaching Assistant (TA)
    - Select questions approved by admin
    - Create course
    - Add, edit, and remove students from course
      - From formatted CSV
      - Individual basis

- Create teams for students to be on
      - Add and remove students from teams
    - Create survey
    - Grant read access to TA's and possibly other professors
    - Assign surveys for students
    - Preview survey
  - Student
    - Takes surveys
    - Leave comments for professor and fellow team members (and possibly admins)
    - Can go back to revise survey responses any time before survey deadline
    - Receive grade from survey and comments from team members
- Data
  - Teams
    - Students belong to a team, many to one
  - Survey
    - Contains questions, does not own them
    - Start and end dates required
    - Expired surveys belong to students for viewing, many to one
    - Viewable template when editing and creating
  - Department
    - Admin belongs to department, one to one
    - Questions belong to department, many to one
  - Questions
    - Questions belong to departments, many to one
    - Questions used in surveys, no ownership
    - Admin creates questions
    - Questions have categories, set by admin
      - Categories have subcategories, also set by admin
- Website Technologies –
  - Front end
    - HTML/CSS for user interface
      - Mirrors professional Mines Websites
    - React/Express for functionality
      - Mainly calling the API
  - Back end
    - PostgreSQL server
    - Keep the ERD simple and easy to follow
  - API
    - Express API Framework
    - REST API to handle basic CRUD of database
    - Separates the front and back ends of the application
  - Email Server
    - Used to automatically send reminder emails to students
- Website Portals
  - Login page
    - Not under Mines' SSO, users will need their own account
  - Admin page
    - Admin should not see professor content
    - Should have separate professor dashboard if also a professor
    - Ability to add and remove admins for other departments
    - Question page

- Create questions under a specified category
  - Add, edit, and remove questions
- Professor Page
  - Ability to create templates for surveys
  - Dropdown of previous and current courses managed by professor
  - Ability to create a new class
    - Assign surveys to students
- Student Page
  - View enrolled classes
    - Show surveys for class with due date
    - Submit professor specific questions
  - See team members
  - Save survey as they progress

### III. Non-Functional Requirements

- FERPA Compliant
  - Must protect student data from breaches and attacks
  - Every student must only be allowed to see their own data
- Hosted on secure, public server
  - Hosted on LightSail
  - Under Mines AWS Enterprise account
- Each department is responsible for managing their own portion of the application
- Zoom, sharp contrast, and other accessibility functionality
- Compatibility with modern browsers
  - Primary emphasis on desktop browsers
  - Mobile phones are not the priority
- UI/UX should be intuitive, easy to follow, and visually appealing
  - Documentation to explain and help users

### IV. Risks

The Family Educational Rights and Privacy Act (FERPA) is something we must consider when developing a web application for faculty and student use. The goal of the application is to allow administrators and professors to survey students on participation in a group setting, thus we have several students commenting on, and evaluating the efforts of their peers. There should be no reason for students to be able to view other students' personal information. Mines is also legally obligated to protect students' information. According to the National Center for Education Statistics, "The penalty for non-compliance with the Family Educational Rights and Privacy Act (FERPA) and the Protection of Pupil Rights Amendment (PPRA) can be withdrawal of U.S. Department of Education funds from the institution or agency that has violated the law." (A.1). There are also fines associated with any school faculty involved with a FERPA violation. Therefore, it is pertinent that in the development of this application we remain FERPA compliant (A.2).

This site will not be maintained by ITS (Information and Technology Services) upon its completion and will need to be maintained by faculty. This would either be our client, or a Mines graduate student hired to maintain this site. There are potential risks with this because as faculty changes and administrative changes occur, there need to be transfers of access to the site and permissions for anyone who needs to maintain or update information on the site.

Security is another issue that comes to mind when thinking about the development of this application. We intended on making the authentication of users as secure as possible and have worked hard to ensure the protection of information. However, any work we have done on security for this application will still likely not be as secure as the Mines SSO.

SQL injection is a potential risk, because if the API is not secured then anyone may go in and identify themselves as a student, professor, or admin if their intentions were malicious.

## V. Definition of Done

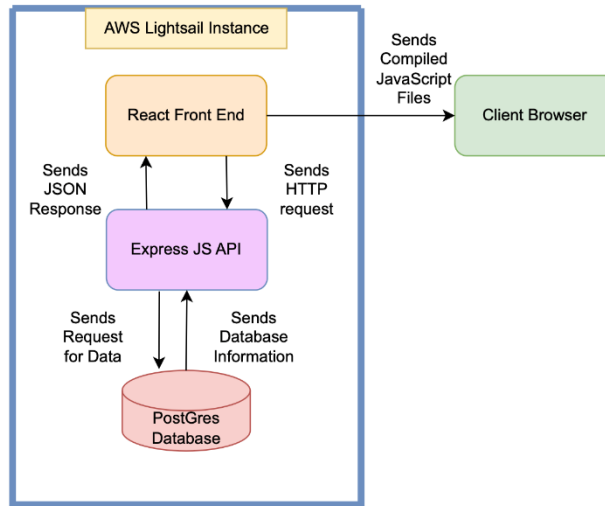
- Provide comprehensive documentation regarding our design decisions, indicating our motivation and reasoning for designing the tech stack the way we did.
- Ensure that the project is left in a state which is easy for the next team to pick up, including commented code.
- Have an Entity-Relationship-Diagram (ERD) to model the database in an intuitive and implementable way.
- Provide comprehensive tests to ensure the program does not unexpectedly crash and that the software functions properly.
- Provide a functional prototype of the program from the administrative perspective and some functionality from the superuser (professor) perspective.
- Instructions left to the client to ensure the project transitions smoothly to the next team.
- The technology stack for the project is well documented, justified, and accessible.
- The work we pass on to the next team is complete and thorough.

Due to the emphasis on thorough and maintainable work, we are unable to promise specific functionality. Rather than promising specific functionality and potentially rushing to fulfill our obligations, we ensure that the work we provided is thorough, documented and easily maintainable. We completed major work at a good pace, but we did not expect to complete the entire product.

## VI. System Architecture

The previous system architecture created by preceding teams had issues regarding the application and framework's stability. It had come to our attention that the old system architecture included Django as the main web framework; however, the previous teams were using the framework as a full-stack application. This led to issues with the stability of the application because it was hard to pinpoint where errors were occurring, since the whole application became one large conglomerate.

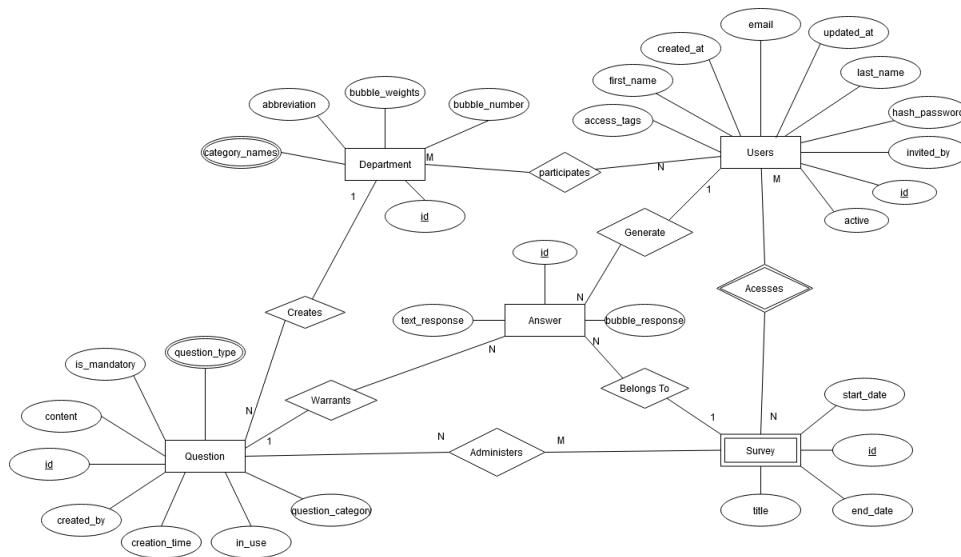
Instead of trying to build upon a broken foundation, our team worked to redesign the foundation so that it would be stable for the following teams to come. Therefore, we split our web application into the front-end framework and the backend framework. This allows the application to be more maintainable overall because if one side of the application breaks, it is much easier to track down the issue. Lastly, we chose our front-end framework to be React and our backend framework to be Express because it is industry standard to use these light frameworks. Additionally, React and Express are taught in the Web Applications course at Mines, so following field session teams will feel more comfortable using these frameworks. Additionally, React and Express run effortlessly on any operating system, whether it be MacOS, Windows, or Linux. Lastly, our database is Postgres because it is also commonly taught here at Mines in the Database Management course, and it is often used to model relational databases. In Figure 1, below we show how each side of the application interacts with each other.



**Figure 1 – Application Design**

The previous teams’ database designs were incoherent, and not intuitive to incoming teams who needed to be able to understand the database design. As a team, we tried to eliminate a cyclical entity relationship diagram (ERD). But due to the relationships in the database, a cyclical ERD was the result.

The questions in the database are created by administrators and belong to the department. The users who participate in the department could access the surveys in some capacity, whether they can create or take a survey depends on their user type. For a survey to be created, it must have access to a list of questions to administer which is how we reach a cyclical ERD. All our relationships within the database can be referenced in Figure 2 below.



**Figure 2 - Database Design Diagram**

It is important to be able to see the interactions each type of user has within the realm of our surveys because each user will have distinct roles. As shown in the User entity in our database diagram, there exists a field titled “access\_tags.” This field notes which roles each user has. Here are the roles and permissions we have outlined for each user given their access tag:

- **Administrator** – the head of any department at Mines



- Has the ability to create, delete, and update questions for their department question bank
- Can add new professors and administrators to the application
- Can reactivate/deactivate other users
- **Superuser** – any professor at Mines
  - Can create surveys
  - Chooses questions to include in their survey from the department question bank
  - Add/Remove students to a survey
  - Can see and release student grades
  - Can see student responses
- **TA (Teaching Assistant)** - Student aid to the professor
  - Can create surveys
  - Chooses questions to include in their survey from the department question bank
  - Can see and release student grades
- **Student** – Regular student taking any class
  - Can take surveys
  - Views feedback

Our goal in the project is to make everything we do and accomplish understandable for future teams so that when they need to understand an aspect of what we were doing, they can do so with ease. Therefore, we have set up our entity relationship diagram and system architecture diagram in a matter that is clean and simple.

## VII. Technical Design

In this project we utilized React and Express to develop this web application. We had the opportunity to utilize a CSS framework, however we did not take that route so that we could design the application to the client's specifications. We used HTML and CSS to design and implement the screens that looked professional, and like other Mines websites/ applications. As shown below, we created sliding scale abilities to choose the number of categories and the number of bubble selections for each category. There is also a range for which to auto scale the bubble weight values, as well as allowing the user to change the individual values should they want. Figure 3 below shows what the page looks like.

### Set Department Grading Scale

Department:

**Number of Question Fields: 4**  
For example, fields can be the names of categories such as: Excellent, Good, Unsatisfactory, etc.

**Category Titles:**

Category 1:

Category 2:

Category 3:

Category 4:

**Number of Bubbles Under Each Category: 4**

**Category Grades Autoscaling:**  
Entering a min and max will automatically set the grading scale for all the bubbles under each category.

Enter minimum grade value:  Enter maximum grade value:

---


**Preview**

This is how a question looks like on a survey. The numbers represent the grading scale. You can click the values and edit them if you want to override the autoscaled grading.

Excellent	Good	Ok	Unacceptable
<input type="radio"/> 100 <input type="radio"/> 95 <input type="radio"/> 89 <input type="radio"/> 84	<input type="radio"/> 78 <input type="radio"/> 72 <input type="radio"/> 67 <input type="radio"/> 61	<input type="radio"/> 55 <input type="radio"/> 50 <input type="radio"/> 44 <input type="radio"/> 39	<input type="radio"/> 33 <input type="radio"/> 27 <input type="radio"/> 22 <input type="radio"/> 16

Figure 3 - Admin Survey Settings Page

As part of making the application user friendly, we wanted to be able to display current and past admin and superusers in a system. Only admins have access to this page but because they also manage superuser access and privileges, they need to be able to see who is active/ inactive. To do this we utilized tables in HTML to display users and their privileges and access. The toggle status (when populated) has a button that admins can use to activate or deactivate a user's privileges, which updates the API with that information changing the user's status. Figure 4 below shows the webpage.

 Ramblin' Wreck Reviews

---

**Invite Faculty Member**

First:  Last:

Email:

Department:

Access:  Professor  Admin

---

**Manage Faculty Member Access**

Department:

**Admins**

First	Last	Email	Current Status	Toggle Status
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>

**Professors**

First	Last	Email	Current Status	Toggle Status
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>

Figure 4 – Manage Faculty Page

To facilitate the testing of our application, we utilized the Jest Testing Framework within both the React frontend and Express backend. Our main reasoning for this was because these two frameworks are both written in JavaScript and Jest is one of the most popular JavaScript testing frameworks. Additionally, using the same testing framework made all the tests within our application quite organized because of the partitioning through the testing suites. Another benefit was being able to run all the tests with a singular command such as “npm test.” Below is an image of what the testing suites look like when they are run. Jest outlines the success and failure of these tests in a user readable way by giving which suites pass, the number of tests that passed within a suite, and any errors that may have occurred. Figure 5 underneath shows how the terminal looks for these tests.

```
PASS src/__test__/ManageFaculty.test.js
● Console

console.log
  Forced render 0

  at src/ManageFaculty.js:20:13

console.log
  Forced render 0

  at src/ManageFaculty.js:20:13

PASS src/__test__/CreateContent.test.js

Test Suites: 2 passed, 2 total
Tests:       4 passed, 4 total
Snapshots:  0 total
Time:        2.846 s
Ran all test suites.
```

Figure 5 – Jest Test Suites

We used AWS LightSail to deploy our software. LightSail is a product which essentially runs a virtual machine in the cloud which can be accessed through SSH. As a base Ubuntu installation, deployment would seem simple by emulating the WSL environment that two of our group members ran the project on. However, the Ubuntu image was changed by Amazon and had strange characteristics which made the deployment fail.

First, operating system updates were unstable (which is very contrary to the design philosophy of Ubuntu and Debian), and when we attempted either to upgrade the packages, or upgrade the operating system, the entire instance would be lost and inaccessible.

At that point we had to restore one of the backups which leads to the second problem with LightSail. To back up a snapshot of the instance, a new instance must be created, which leads to billing confusion and unintuitive instance management.

Third, LightSail has very minimal usage and documentation. So, when something goes wrong it is very time-consuming and difficult to fix, if it is even possible. Outdated Stack Overflow pages and tangentially related Amazon documentation are not intuitive debugging resources.

Fourth, since the entire software is hosted on one instance, it would be very vulnerable to break ins and errors, if the instance goes down, the entire application fails entirely which does not align with the client priority of

maintainability. If some malicious actors were to break into the LightSail, they would have access to everything, which would certainly be a FERPA violation.

Fifth, files are difficult to get on and off the LightSail, we cloned our GitHub repository to the machine, but that requires a GitHub account. At first, we used a team member's personal account, but after recreating the instance we created a GitHub account for the project, which feels like a bad solution to a problem which should not exist.

Lastly, the software simply would not work on the instance, it worked at one point but after the system update and multiple blank slate starts, it would simply not work. NPM would not install the necessary packages to run the program, even with docker containerization it would still not start. NPM was the same version on my WSL environment and the LightSail as all the necessary building tools; however, despite working on my WSL environment, it would not run on the LightSail instance.

LightSail was the one part of the software architecture that we did not look at replacing due to the amount of architecture that we would be re-designing and replacing. It is our team's opinion that LightSail should be replaced with a different AWS hosting solution. Should the proceeding team decide to give LightSail another shot, they can access it through the Mines AWS SSO (which requires an IT ticket), then through the console home after receiving access to the LightSail instance (which requires another IT ticket). The Mines AWS SSO can be accessed through Office 365 on the web when signed into a mines email address once access is granted. We recommend AWS App Runner as an AWS deployment tool for this project. Below Figure 6 is a screenshot of the LightSail dashboard.

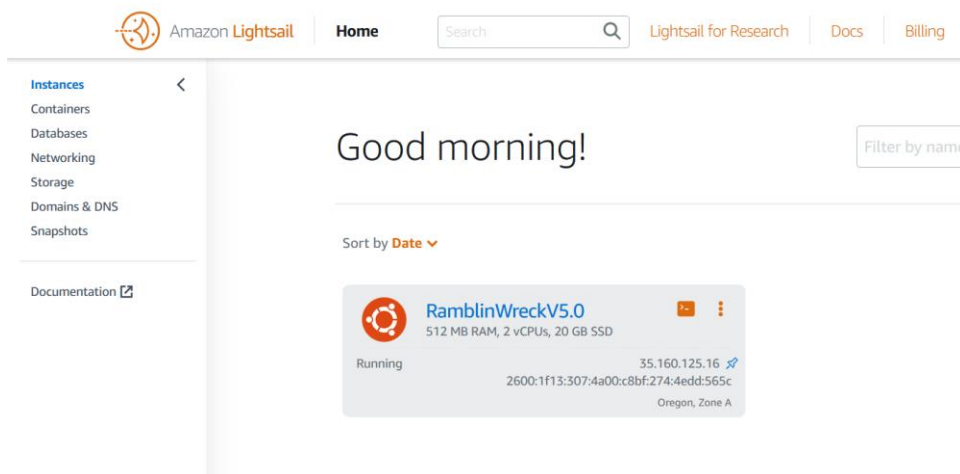


Figure 6 - AWS LightSail Dashboard

It is worth noting that LightSail does have some perks, like configurable firewalls, website domain services and other networking services. It also has database functionality as seen on the left side of the dashboard, however it costs money, so our team ran the database on the instance.

## VIII. Quality Assurance

Considering that our application deals with the handling of sensitive student data and confidential survey information, it is important that complete and thorough testing is implemented during development. The minimal maintenance that this software will eventually receive also motivates thorough testing such that the software will be easy to maintain and ensure the safety of the data stored in our application. Therefore, our team has devised a testing plan given the requirements of our application.

### Functional Requirements Testing

- User Roles

- Purpose of Test
  - This test needs to ensure that only certain user views are available based on the permissions of that user. For example, only Administrators should see the application's admin portal. The same goes for professors and students only being able to see their respective views.
- Description of Test:
  - This test would require unit testing in the React side (user interface) of the application. We would ensure that these unit tests check to see if the proper views are displayed on the screen given the role of the user that is logged in. This would be done through HTML validation testing.
- Tools required for Test:
  - Jest Testing Framework – A framework quite easy to install and configure from our application.
- Threshold for Acceptance:
  - All unit tests need to pass for the security of our application.
- Edge Cases:
  - We have scenarios where a user could be a professor in a separate department, but an administrator in another department. For these users, our application needs to ensure that the user logs in as only of these roles (either: professor or admin), and they can switch to the other role if they need to. Another edge case is teaching assistants. These are students who aid in a classroom, so these students would need to choose whether they are logging in as a student or TA.
- Results of Testing:
  - Considering that this side of testing requires user authentication, we were not able to test this yet. Given the timing, we were not able to implement authentication.
- **Application Data**
  - Purpose of Test:
    - These tests need to ensure that data vulnerable to leaks is secure and stored. This includes user passwords, survey responses, and student grades.
  - Description of Test:
    - Utilize unit tests and end to end tests where there are checks to see that data is encrypted and not able to be intercepted by malicious software. Our focus is to ensure that our hashing code passes unit tests that ensure the security of encrypted data. Unit tests are placed in both the API and frontend to make sure data is always encrypted.
  - Tools required for Test:
    - Jest Testing Framework
  - Threshold for Acceptance:
    - All unit tests and end to end tests must pass.
  - Edge Cases:
    - If a user decides to change or update their password, that process must also be encrypted and secure.
  - Results of Testing:
    - Our team was not able to implement a login system in the time given, so we were unable to test the security of this type of sensitive data.
- **Website Technologies**
  - Purpose of Test:
    - To ensure that React (our frontend), Express (our backend), and Postgres (our database) are all running properly.
  - Description of Test:

- This can be done with regular User Interface testing. If a user accesses the webpage and finds that items are not loading or updating, then, the supporting backend and database are not currently running. To verify if the database is the issue, one can launch the API and see if it is mutating the database.
  - Tools required for Test:
    - Any developer on our team.
  - Threshold for Acceptance:
    - The User Interface is making API calls, and the database is saving/updating these changes.
  - Edge Cases:
    - The application may be passing this end-to-end testing in the local development environment; however, it may fail on the AWS server. So, developers need to verify it is working in both cases.
  - Results of Testing:
    - The User Interface testing was conducted, and we found that the application was passing all the expected behavior it should have.
- **Website Portals**
  - Website portals apply to the different views each type of user should see. Therefore, the testing covered under User Roles verifies this functional requirement. This could be covered by a sophisticated, automated front end test suite, but that was not implemented due to the time limit and cost, manual interaction deemed most effective.

## Non – Functional Requirement Testing

- FERPA Compliant
  - Our tests require data to be secure and not available to users without the authorization to view sensitive data. Therefore, our User Roles section outlines the concern for authorized users. Additionally, our Application Data section targets the security of data. Therefore, these sections above meet the requirements of FERPA compliance.
- Hosted on secure, public server
  - Considering our application is hosted on a LightSail instance on Amazon Web Services, the security of the server is maintained by Amazon. Amazon does a lot of inside testing to make sure that our application is not vulnerable when it comes to placing the item on their platform. As developers, we cannot test their security; however, we can test how our data is secure and encrypted. We address this in the Application Data section above.
- Each department is responsible for managing their own portion of the application
  - This is similar to our User Roles section outlined above. Only users that have administration access should be able to make changes to their departments, and only professors can manage surveys for their class. Therefore, the testing covered in that section would meet this requirement.
- Compatibility with modern browsers
  - Purpose of Test:
    - To ensure that the application is following all functionality and is maintaining the same visual styling across different web browsers: Google Chrome, Microsoft Edge, and Safari.
  - Description of Test:
    - A user will open the same web domain on Google Chrome, Microsoft Edge, and Safari and will ensure that the styling looks the same. They will also ensure that functionality has not changed across the different platforms.
  - Tools required for Test:
    - Any regular user who can view the application and use it on different web browsers.
  - Threshold for Acceptance:
    - The website looks and functions like intended on Google Chrome, Microsoft Edge, and Safari.

- Edge Cases:
  - The website is not designed for mobile use, so the styling and functionality may differ on mobile devices. However, our client has specified that the focus of the application is good functionality on a laptop/desktop.
- Results of Testing:
  - This was tested and it was found that the website looks and runs as expected on Google Chrome, Microsoft Edge, and Safari.
- UI/UX should be intuitive, easy to follow, and visually appealing
  - Purpose of Test:
    - To ensure that the user interface is easy to navigate through and is a pleasant and smooth experience. We want to ensure that users would prefer the user interface of our application compared to CATME, the software we are replacing.
  - Description of Test:
    - Allow the client to navigate through the website performing distinct functions within the Admin, Professor, and Student sections. After they go through the different sections, they give their feedback on the experience and look of the web page.
  - Tools required for Test:
    - Client testing of the webpage
  - Threshold for Acceptance:
    - Client approves of the user interface. Make any changes necessary given the feedback.
  - Edge Cases:
    - The client mentions that the user interface is not as they had envisioned. In this case, we would go ahead and change the UI, so that the client is satisfied.
  - Results of Testing:
    - This was tested through client demos, and our client has approved the UI/UX design.

## IX. Project Ethical Considerations

There are several ACM/IEEE principles that we considered in developing the Ramblin' Wreck Reviews. We observed these principles for both their value in the application and the possible dangers if they are neglected. The ACM (Association of Computing Machinery) and the IEEE (Institute of Electrical and Electronics Engineers) explain how computing professionals need to consider how their products and actions affect the greater world around them. The goal of these ethical practices is to minimize harm, damage, and dissatisfaction with the work done by computing professionals. Another goal is to maximize diversity/inclusion, product quality, and benefit to humanity. With these practices in mind, we have closely considered which principles of each ethical organization are most pertinent to this project and which are most in danger.

### **Important ACM/IEEE Principles for Ramblin' Wreck Reviews:**

- (IEEE) 3.07 Strive to fully understand the specifications for software on which they work.
- (IEEE) 3.13 Be careful to use only accurate data derived by ethical and lawful means and use it only in properly authorized ways.
- (ACM) 1.7 Honor confidentiality.

These principles are of utmost importance in the security and stability of the website. The most important principles are those of security and protection of data for our users. This is because of a previously explained constraint in IV. Risks, where our client can face federal penalties and fines of up to \$500 dollars per student if FERPA is not complied with. Therefore, IEEE 3.07 is a crucial ethical principle because our application needs to meet FERPA compliance, and our application will be developed such that FERPA is never violated. IEEE 3.13 is necessary because our

responsibility as developers is to ensure that sensitive student data is not visible to unauthorized users. In the same vein, ACM 1.7 requires us to honor the confidentiality of student survey responses and their grades.

#### **ACM/IEEE Principles in Danger for Ramblin' Wreck Reviews:**

- (IEEE) 1.03 Only approve software if it meets specifications, passes appropriate tests, and respects privacy
- (ACM) 2.9 Design and implement systems that are robust and useably secure.
- (ACM) 1.2 Avoid harm.

These principles are in danger due to the application we are creating for our client. Our completed project will need to adhere to these principles when ready for release. In development, however, applying these principles can prove challenging and thus are in danger of being neglected. To avoid the damage that would be caused by a lack in these principles, we must keep security and privacy at the forefront of our minds to be sure that we are constantly testing for security concerns, or leaks of personal information. For example, IEEE 1.03 requires us to add unit tests that ensure sensitive data is not vulnerable to leaks. Additionally, ACM 2.9 encourages us to develop an application that uses secure hashing to store user passwords and data. Lastly, ACM 1.2 requires us to prevent any data leaks during development because the consequences would be profoundly serious for the Colorado School of Mines and its stakeholders.

## **XI. Results**

### **Testing:**

#### **Usability Testing:**

- Toggling the activity status of faculty members
  - This functionality is within the admin section of the application. In here, an admin can either deactivate or activate a faculty member.
  - The functionality of this code was tested using the Jest testing framework. Unit tests were written to see if the activity status of test users were toggled from active to deactivated. The unit tests pass and validate that this user functionality is working properly.
- Adding a new faculty member under a department
  - This functional element within the admin section allows administrators to be able to add new faculty members to their department or add another administrator under a different department.
  - This functionality was tested using Jest unit testing and the test cases ensured a user was added to their corresponding department specified by the HTML form. The tests pass and all functionality works as intended.
- Adding a new question to a department
  - This portion of our application allows administrators to be able to post a new question under a given department at Mines.
  - This section's functionality was tested using Jest unit tests, validated if test questions were posted to the API, and then these questions were displayed on the UI. The tests also validated that the questions were posted to their corresponding department.
  - All our unit tests pass with expected results.
- User input handling
  - Utilizing user testing we were able to handle form input errors. We tested this by writing down the different possible invalid text inputs. And we tested all these inputs and noted if it broke our application or not.



- If it broke our application, we added more form error handling. We repeated this until we found no use cases that broke our application.
- Although we did not have enough time to extract this into unit tests, we recommend that invalid user input be handled in unit tests in the future.
- HTML validation
  - To ensure buttons on the UI have the expected behavior when clicked, we utilized HTML validation testing.
  - These unit tests were written in Jest and were made to verify if the expected HTML elements appeared on the screen given a button press.
  - We found that all unit tests passed, and the buttons were updating like we envisioned them to.
- API Call Testing
  - This type of testing validates if API calls on the frontend are only made a certain number of times, and that unnecessary API calls are not repeated.
  - Tests to handle this situation were written in Jest and all tests currently pass. In the end, we verified that the frontend is not making more API calls than expected.

### Performance Testing:

- API Validation Testing
  - Several testing suites have been added to ensure proper functionality for the backend API.
  - All current tests are using Jest for the framework, with several different calls for different use cases. Not only were good calls tested, but badly formatted calls, forbidden actions, and others are in place to ensure that errors are handled correctly.
- Browser Testing for the Frontend
  - To ensure compatibility of the webpage between different browsers, we tested this by running the application in the development environment on Safari, Chrome, and Microsoft Edge. All buttons and functionality were tested through user testing, where users performed functionality on the webpage to see if everything was working as expected. We found that functionality and visual display remained the same across all browsers.
- API Mock Testing
  - To pinpoint which side of the web application is broken during an issue, it is important to write API Mock Tests. These tests check to see that the frontend side of the application has full functionality despite if the API is not running. Instead, Mock API data is simulated in the frontend to function as the replacement for the API. We wrote these unit tests in the Jest testing framework and found that all tests passed. This indicates that the UI is performing the proper functionality without being reliant on the API.

### Testing Summary:

- Backend
  - Currently, all tests are properly formatted and run without error. There are some still not passing, but that will be for the next team. As mentioned above, both good and bad cases are tested to ensure correct functionality despite the existence of errors in the request. While the testing suites are minimal, they cover all the code currently written, and fringe edge cases are left to the developer stepping through the application.
- Frontend
  - Currently, there are test suites written for the functionality of admin users of the application. These tests check for usability, functionality, and even disconnects between the frontend and the backend side of the application. Note that functionality for other types of users has not been implemented

yet. Therefore, the test cases overlook professor and student user scenarios. However, the current test cases for admin functionality pass and are complete.

## XII. Future Work

### Features We Did Not Have Time to Implement:

- User Authorization
  - We were unable to implement user authorizations due to a lack of time, and we also knew that this would take a large amount of time to complete. We believed that if we could get most of the functionality for the site up and running, this would leave any following teams with enough time to manage user authorization.
- Separate user views
  - Due to lack of user authorization, we could not grant individual permissions to corresponding user groups. For example, without authentication any user can create questions under a department although only admins should have this special ability.
- Allow users to take/ preview surveys
  - We ran out of time to implement this functionality; however, it is our hope that the work we put forward for the backend and frontend so far will help the next team to fulfill the required functionalities with ease.
- Create student view (just surveys to take)
  - Student view should be the easiest screen to implement as it should only consist of surveys assigned to the student for them to take. They should have no other functionality. The reason we did not get to this is simply because we ran out of time, and we were building functionality from the top down, meaning we began with admin, then professor. Professor functionality still has some work that needs to be done, and admin is complete.
- Implement CSV upload
  - The client requested that professors be able to upload a CSV file of students in classes for the assignment of surveys. This functionality was not implemented due to running out of time, however there is a file "FileUpload.js" that has some code from GeeksforGeeks.com explaining how to implement a file upload functionality using JavaScript.
- Email reminders
  - Email reminders were a similar consideration to user authorization. We knew this would be time-consuming and we wanted to accomplish more with the actual functionality of the website. We knew that if we could manage to get most of the functionality for the site then any following team(s) would be able to spend more time on this feature.
- Launching on AWS
  - NPM is not installing all packages / installing improper packages that are not supported on Ubuntu, and thus it is not currently possible to run on the Ubuntu LightSail instance. We also tried to run the docker image, but that ran into the same problem. Our recommendation is either make a new LightSail instance, or to find a more suitable hosting solution on AWS for a react frontend, express backend, and Postgres DB.
- Domain is still not functional
  - The website domain [www.ramblinwreckreviews.com](http://www.ramblinwreckreviews.com) was given to our team in a broken state, and we attempted to resolve this issue through the AWS LightSail instance. However, due to the issues launching on LightSail, the domain is still not functional.
- Create Surveys
  - Frontend page exists for survey creation; however, not all API calls do not exist.

### Frontend Future Work:

- Authorization of individual users
  - Allowing individual users to log in and have their own functionalities available to them based on their role(s)
- Create surveys
  - The survey creation screen already exists; however, it is still not able to create a new survey. It shows required questions that will/ need to be added to the survey, allows the user to pick a start and end date, and set the survey a title. This is still not properly hooked up to an API call yet so there is still work to be done in the code to connect it to the database so that they survey saves properly and can be administered.
- Take / Administer surveys
  - Users need to be able to take surveys that have been assigned to them. Admins and professors need the ability to assign surveys, and students need the ability to take them. Professors can also take surveys administered to them by admins. Professors should also be able to take their own surveys to preview the look and feel of the surveys they create.
- Launch on AWS
  - Currently there are issues launching [www.ramblinwreckreviews.com](http://www.ramblinwreckreviews.com) on AWS. Using LightSail may not be the best approach for launching the website.
- Email reminder
  - Admins and Professors (and TA's) need to be able to initiate email reminders to their classes with just a click of a button. These should also be automated for anyone who has not completed the survey and tell them how much time they have left to complete it.
- Management of CSV files
  - Professors and TAs should be able to upload CSV files for each survey creation. The CSV will contain student data and potentially any teams they are a part of (like CATME). The goal will be for these surveys to be administered either to individuals or teams depending on how the CSV file reads.

#### Backend Future Work:

- Launching in AWS (same as frontend)
- Professor functionality separate from surveys
  - Reading survey results
  - Survey exporting
  - Email functionality
  - Uploading students via CSV file
- All student functionality
  - Taking surveys
  - Accessing survey results once posted by the professor
  - Populating the answers table in the database
- Authentication and authorization
  - Login endpoint
  - Account creation

### XIII. Lessons Learned

- AWS LightSail might not be the best solution for hosting the application. Our team redesigned the entire tech stack, barring the hosting and deployment. LightSail can have snapshots which are great, but a new instance must be created to use the snapshots, which is not an intuitive process.
- Instead of trying to mend the unstable foundation from the previous team's project, our team decided to start from scratch. Instead of making our goal: "We need to implement as much functionality as we can," we decided

that the objective should be to build a solid foundation, as previous teams have struggled with stability in the software. This successfully aligned with our client goals as well.

- The database design we created had several iterations, each adding functionality. We created the database design before we started programming, and as such, we uncovered areas of functionality that the database did not support that we had overlooked in the original design process. Having flexibility with the database and valuing stable solutions rather than shoe-in solutions has led to the database we currently have.
- Creating storyboards illustrating the front-end design before implementation saved us a lot of time with our client. We showed our client the storyboards and received feedback on the preliminary designs before iterating and implementing the storyboards in React.

### XIII. Acknowledgments

Our team would like to give a big shoutout to our client, Donna Bodeau. Her willingness to be flexible and support our development throughout the semester has been a wonderful experience. Being able to brainstorm ideas together through small sketches and seeing the ideas come to fruition is an experience our team cherished. Also, we appreciate Donna's cheerful and positive attitude that made our team look forward to our client meetings. Once again, we appreciate all the hard work and time she has given us. Her leadership is something we are in debt to for this project. Her creativity is something we will miss after this semester.

We would also like to acknowledge our advisor, Tree Michael. He has provided so much guidance and expertise to our project that has contributed to our success this semester. We would like to thank him for all his technical knowledge that has allowed us to build a solid framework for this application. His guidance has allowed us to pivot and make changes quickly when programming issues have risen without sacrificing programming efficiency. His advice in web applications has really facilitated our development throughout this semester. Lastly, all the time he has set aside for meetings and giving us feedback is much appreciated. We recognize that his time is valuable and are so thankful to have had such a great portion of it to be mentored throughout this semester.

### XIV. Team Profile

- Chloe Cadwell
  - Chloe Cadwell is a senior studying Computer Science. She is not from Colorado and moved around the West a lot before coming to Mines. She focuses most of her time on being a diligent student, but when she has free time, she enjoys playing games on her PC.
- Brennen Denault
  - Brennen Denault is a senior (and a half) studying Computer Science with an emphasis in Cybersecurity. Originally from Illinois, he moved out west to finish his degree surrounded by mountains instead of cornfields. Brennen has had experience working on backend API's as well as all the fun projects assigned at Mines. He can often be found outside fishing or hiking in his spare time.
- Srijita Ghoshal
  - Srijita Ghoshal is a junior studying Computer Science and Business. She is a Colorado native and is often seen painting portraits in her free time. She has decent exposure to Web Development, as her last two internships involved writing full-stack applications. Other than coding, she loves spending her time with her two cats.
- Grant Weimer
  - Grant Weimer is a senior in Computer Science, originally from the western slope of Colorado. He is a high-grade contributor and spends his time doing homework for the various classes here at Mines. He

has interned at CACI, HP and TheTradeDesk and will work at TheTradeDesk once he graduates in December 2023. His hobbies include Mariokart Wii, Mariokart 8 Deluxe and watching Wes Andersen movies.

## References

- A.1 : [1] “Section 6: Releasing Information Outside an Agency,” National Center for Education Statistics, [https://nces.ed.gov/pubs97/p97527/Sec6\\_txt.asp](https://nces.ed.gov/pubs97/p97527/Sec6_txt.asp) (accessed Sep. 10, 2023).
- A.2 : [2] O. Muscad, “The Ultimate Guide to FERPA: Everything to know,” DATAMYTE, <https://datamyte.com/guide-to-ferpa/#:~:text=Schools%20that%20fail%20to%20comply,FERPA’s%20provisions%20are%20reasonably%20straight%20forward.> (accessed Sep. 10, 2023).
- A.3: [3] “Code of Ethics,” ACM, <https://www.acm.org/code-of-ethics> (accessed Sep. 26, 2023).
- A.4: [4] “Code of Ethics,” IEEE, <https://www.computer.org/education/code-of-ethics> (accessed Sep. 26, 2023)

## Appendix A – Key Terms

Include descriptions of technical terms, abbreviations and acronyms

<b>Term</b>	<b>Definition</b>
<i>Entity Relationship Diagram</i>	<i>A concept architecture diagram used to model database relations between different objects: such as People, Places, Items, etc.</i>
<i>FERPA</i>	<i>The Family Educational Rights and Privacy Act which is U.S legislation which imposes restrictions upon personal data privacy.</i>
<i>TA</i>	<i>Teaching Assistant – A student that aids in grading and teaching of a course. They help the professor.</i>
<i>AWS</i>	<i>Amazon Web Services</i>
<i>ACM</i>	<i>Association for Computing Machinery</i>
<i>IEEE</i>	<i>Institute of Electrical and Electronics Engineers</i>