COLORADOSCHOOLOFMINES
EARTH • ENERGY • ENVIRONMENT

# CSCI 370 Final Report

*Teamkeepers*

Chandler Carter
Connor Brown
Hayden Cooreman
Thomas Jensen
Wilhelm Northrop

Revised December 6, 2022

CSCI 370 Fall 2023

Dr. Christopher Painter-Wakefield

Table 1: Revision history

| Revision | Date | Comments |
|---|---|---|
| New | 20 September 2022 | Creation of requirements & design documents. |
| Rev – 2 | 10 October 2022 | Quality and Ethics section |
| Rev – 3 | 27 November 2022 | Results, Technical Design, etc. (final rough draft) |
| Rev – 4 | 6 December 2022 | Final draft |

# Table of Contents

# I. Introduction

Many of the courses at Colorado School of Mines (CSM) require students to work together and subsequently evaluate their peers. There are many half solutions that are currently used to fill the need for student feedback, but the most robust system currently in use is the Comprehensive Assessment of Team Member Effectiveness or CATME. CATME is a web-based peer evaluation interface developed by Purdue University. CATME is used by the Engineering, Design, & Society (EDS) department to poll students in the Capstone Program a minimum of twice a semester. Each poll using the service costs the department $2 per student, this cost quickly adds up because there are an average of 2,000 students in EDNS courses each year. Additionally, other courses such as CSCI 370 Advanced Software Engineering, use CATME for peer evaluation.

Due to the cost and the lack of flexibility in CATME, the EDS department proposed an internal web-based peer evaluation tool creating the first revision of this project over a year ago. The overall goal of the project is to provide a less expensive, more flexible, internal version of CATME. The project has gone through three teams with our team having been the fourth iteration. Through these earlier iterations a domain RamblinWreckReviews.com was established, a functional web interface was built, and an Amazon Web Service (AWS) Lightsail server instance was created. Our goal was to finish the hard work done by earlier teams, including but not limited to finishing the build out of core functionality, updating the software used, ensuring FERPA compliance, adding additional AWS infrastructure, and piloting a beta this semester.

As a result of the multiple iterations and rushed timelines, the original code base was lacking high software quality. Improving the quality of the current code base through increasing testing was important to ensure that future revisions of the project would have a more solid foundation. It also ensured that the current feature set was working as intended. In many cases, undocumented bugs and glitches prevented standard and basic functionality from working as intended. Although this was not part of the original project description, validating and fixing the current feature set became a large part of our group's iteration.

Some features, such as automated emails, needed to be completely redone to address the scale of the website. The original solution through Gmail could only support 500 emails per day, which was not enough for the average of 2,000 students in EDNS courses utilizing CATME. Additionally, due to mistakes in configuring the server’s security features, the instance had to be reset and reinitialized from scratch. These are just two examples of the limitations of the currently implemented feature set that needed to be addressed and improved.

# II. Requirements

## Functional Requirement

This section is adapted from the Ramblin Wreck Reviews version 3.0 project report. All new features are **bolded and underlined**.

*Website Application*

- Admins are owners of the product
- Professors and Students are users

**User roles:**

- Admin
    - Create categories of questions
    - Creates a collection of questions
    - Can specify required questions
    - **Specifies scoring schema for questions**
    - Many to Many relationships with **departments**
- Professors
    - Select questions from the different category buckets to create a survey
    - Post survey for students and specify dues dates
    - Reviews and grades surveys
    - Ensure professional compliance of comments
        - Without withholding results to all for a few comments
    - Releases the survey and comments to students
    - Creates survey viewing access for other professors and TA's
    - Create Course
    - Assign Survey
    - Assign Survey begin and end
    - Assign students to course
        - From CSV
        - **On individual basis**
    - **Add/remove students or move between groups**
    - Can be assigned as a professor (TA role), have to be student in their courses and professors in others
- Students
    - Takes surveys
    - Add comments for professors and students
    - Have the ability to take the survey multiple times within the open window
    - View survey history
    - Receives surveys and comments from other students

**Data:**

- Course
    - All students belong to the course.
    - The course belongs to the professor(s).
    - Professors can own multiple courses.
    - Students can be enrolled in multiple courses
    - List of students should be assignable via a CSV upload
- Groups (Link to set of users)
    - Need to be able to receive feedback from the other group members

        - **Individual join/remove without CSV**
    - Survey (Contains questions)
        - Should be sent out to the user
        - Needs start date and end date
        - Needs to have a preview for professors and students
        - **Refactoring - create survey when CSV is opened**
        - **Students should own previous surveys**
        - **Should have viewable template while editing**
        - Categories
            - Question grouping system
            - Questions
                - Questions are text fields selected from the different categories
    - **Users**
        - **Admins can be inactive or active**
        - **Professors can be active, inactive, or deleted (erase surveys when deleted)**
        - **Students should only exist in the context of a survey (when survey is deleted, student instance is deleted)**

*Website Technologies*

- Server
    - SSO via Shibboleth
        - Determine if the previous group's architecture can be reused or reimplemented
        - Determine if Mines SSO should be implemented
- Database
    - Django will largely handle database setup and upkeep. However, the team needs to design it and implement it through the model/view architecture
    - Database should ideally exist in a separate space from the site itself, such as an RDS
    - Mostly implemented by the previous group–should be functional, save for some modifications
    - **Automate data cleansing based upon configurations**
- **Bootstrap 5 Migration**
- **Email Server**
    - automate nag emails for students who have not completed
- **AWS timeout config changes**

*Website Portals and UI*

- Point of Entry: Login Page
    - Mines credentials
- Admin Portal Specifications
    - Must be able to be a superuser and an admin at the same time
        - Should not be able to see other professors' content
    - Needs to be able to upload CSV with students and CSV with admin/superusers
        - Needs to be able to add/delete professors and students
    - Needs to be able to pay for their usage
- Admin Portal UI
    - Template Page
        - Admins regulate the scoring criteria (no more than four)
        - Admins regulate the scoring range (no more than three per criteria)
    - Question Page

        - Admins can create questions under a specific category
        - Admins can make certain questions mandatory
        - Must be edit/delete/add questions
        - Must be able to make questions mandatory/optional
    - Navigation Bar
        - Contains a button that goes to the professor interface
        - Button that signs the user out
    - Manage Privileges Bar
        - Gives admin privileges to certain users

- Professor Portal
    - Ability to create a template for a survey that can be assigned to a class or classes
        - Template creation should be in an active edit style window, viewing the live version while editing
    - Dropdown of semesters that displays all courses managed by the professor
    - Should have classes and the ability to create a class
        - Ability to use templates for sheets so forms can be copied
        - Add or delete from list
        - **Ability to add or delete individual students**
        - Ability to add and delete classes
            - Preserve classes
        - Ability to preview forms before approving
    - Forms should not be linked uniquely to the class–needs to be transferable
    - Should see survey results and withhold results
        - Needs to hold individual questions
    - General user experience and accessibility improvements
        - Dropdown fields should reveal after click on any part of element, currently only working on click of down arrow
- Student Interface
    - Must be able to view list of classes
        - Must show survey for each class and due date
        - Submit question specific to prof
    - Must be able to exist in groups
    - Needs to save survey as they go and edit their responses
    - **Save completed surveys for viewing across teams (department and user surveys, export or share surveys across departments or a publicly available question)**
    - Classes should be able to have multiple surveys
    - Survey access
        - Creator, access results
        - Should be specified in the database

## Non-Functional Requirements

- FERPA Compliant
    - Must protect student data from data breaches and other possible data issues
    - No student should be able to view other student's data
- Must be hosted on a secure, public server
    - Hosted on LightSail Instance
    - Under the envelope of Mines AWS enterprise account

- Multi-Department Responsibility
    - ITS will not take responsibility for this, so it must allow multiple departments to
- Site design is accessible and inclusive
    - Ability to zoom and high contrast
- Compatibility with current browsers
    - Enforced by Bootstrap 5
- UX/UI should be intuitive, self-explanatory, and visually appealing
    - Mobile and Desktop Friendly
        - Enforced by Bootstrap 5
        - Layout should be flexible to work on all devices/orientations
    - Resources exist to explain site

## Risks

The primary risk of this project is maintaining FERPA compliance. If FERPA is violated in any way, the university may be subject to fines, sanctions, or litigation (US Department of Education). Protecting student information is critical to the success of this product. As with any web app, there are privacy best-practices which we must uphold. Our authentication system must be robust, we must prevent malicious actors from injecting SQL, our users must be structured so that only the appropriate parties see intended information, and the site must be protected against DDoS attacks if anyone tries to take down the survey system.

Another risk came in upgrading from Bootstrap 4 to Bootstrap 5. While most of Bootstrap 5 is backwards compatible, some libraries like jQuery are dropped for support. This upgrade was necessary to ensure site security, support long-term stability, and include new features. Minor changes in syntax and function names may cause website elements like modals and drop-down menus to break outright. The amount of work for this portion of the project will depend on whether there are dependencies that we cannot carry into the upgraded framework and how many function calls or tags are deprecated. Extensive testing and debugging will have to be done to verify the upgrade completes as intended.

A major portion of the work for the semester was feature refinement. Many existing features needed polishing or did not operate as intended, so the complexity of this work will depend on how many features need simple fixes or if major restructuring must be done (Shock et al.). Many of the features so far depend on the support of the tools it implements. The project as it exists relies heavily on web frameworks, so frameworks or libraries becoming unsupported or otherwise outdated is always a risk. Chosen frameworks/toolsets should be reasonably expected to offer a long support cycle to increase the lifespan of the product.

Our final risk is in maintaining the site. Department administrators are responsible for sharing admin privileges with a single backdoor since ITS has no intention of maintaining the site. We need to design around this consideration and make sure that several current members have admin privileges or backdoor access at any given time. The solution must be able to be reliable after this team is finished with the project and require little work for upkeep and maintenance of the website or server.

## Definition of Done

- Have a completely functional prototype that is ready to be deployed to users via a web portal
    - Prototype must be testable on design class with students this semester.
    - Test a survey in a class of 25 or more
- Fulfill the above requirements and be a simple, easy-to-use interface for users familiar with CATME
    - Improvements to the interface must provide better feedback to users and be more intuitive for first-time users.
    - Each user type (e.g., admins, professors, students) must be able to complete all actions available for the user.
- Test cases pass

    - Test both database back-end and views to ensure that functionality is effective and not prone to crashing.
- Pass Quality Assurance Testing
    - Ask users who have not seen the site to test functionality to test for worst-case scenarios.
    - Have students and student workers take test surveys to ensure functionality.

## III. System Architecture

The figure below is an illustration of the system's architecture. Ramblin Wreck Reviews is hosted on an Amazon Web Services Lightsail web server instance owned by the Colorado School of Mines ITS Department. Content is served through the Nginx web server, which acts as a proxy server, handles the requests from a browser, and delivers content. Gunicorn is a Python-based web server gateway interface used to handle requests between Nginx and the web framework. Currently, the product uses Django as a full-stack framework for the site itself, which communicates with a MySQL database for user information and surveys.

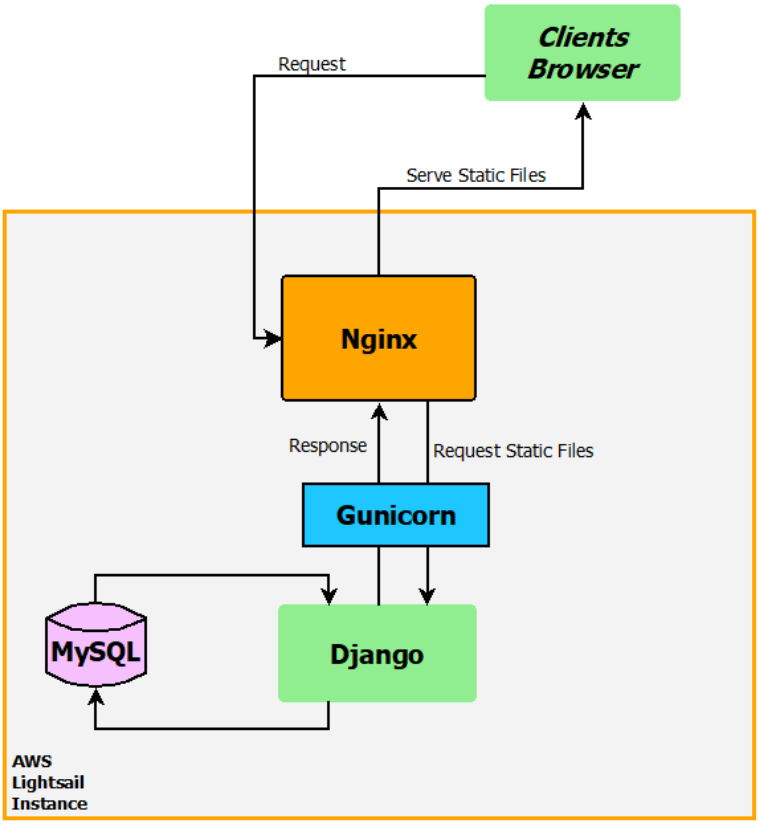


*Figure 1 System Architecture (Shock)*

Figure 2 illustrates the web pages and their functions in a website wireframe. The wireframe shows each user type and the actions that can be performed by each user type. The updated wireframe is refined for usability. By focusing on usability, user intuition about how to perform actions will be improved.

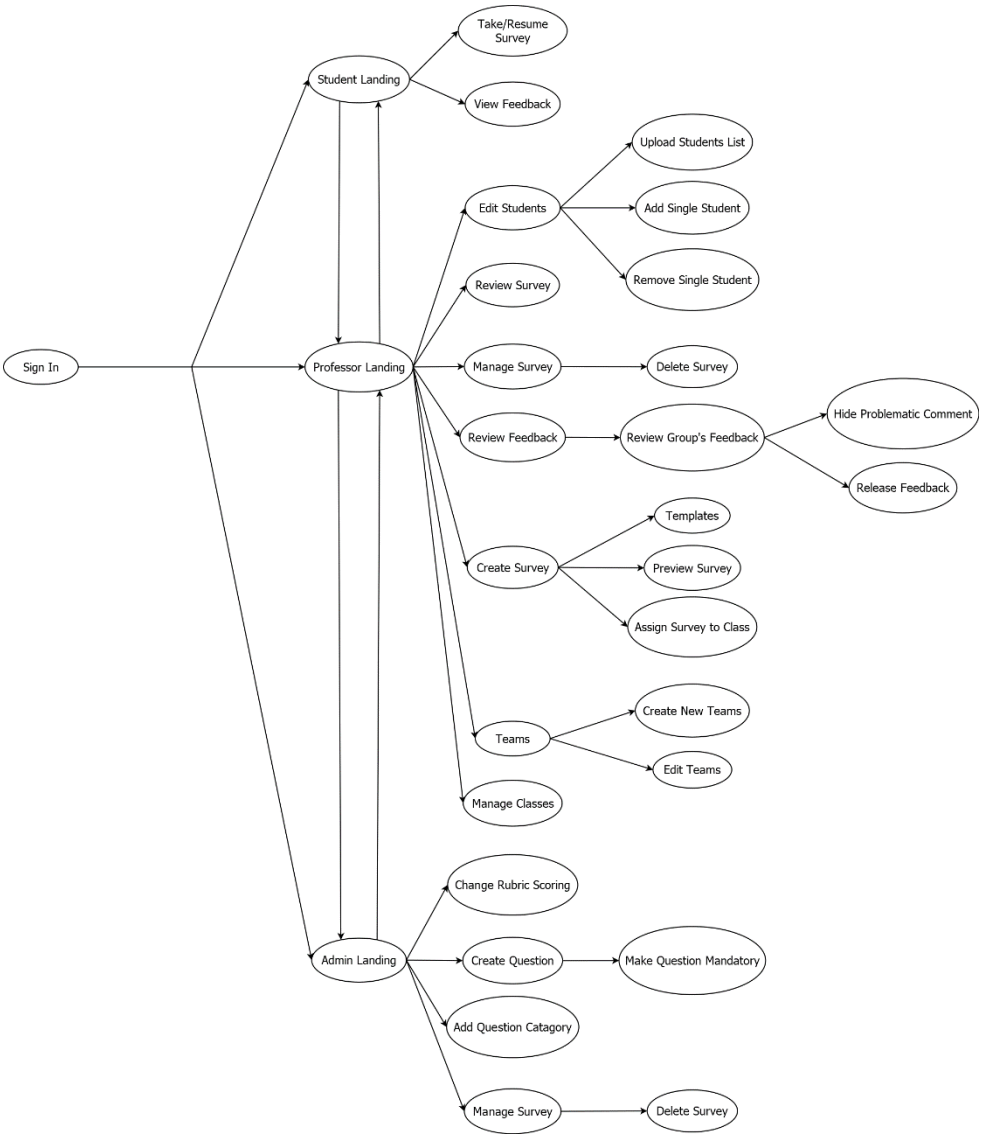


*Figure 2 Website Flowchart*

## IV. Technical Design

To ensure the quality of the current code base and the new features we intended on adding, Cypress was used to implement end to end tests for the website. While the original code base claimed 100% test coverage using Django's test coverage tool, this only tested the back-end features. This meant that using the website was a buggy and unpolished experience. To address this, end to end tests were implemented to find and address common bugs when using the website. These tests focused on the main user loop, creating questions as an admin, creating a survey as a professor, and taking the survey as a user. Testing this functionality without automated testing is very tedious. It requires a lengthy set up process involving setting up the database in a specific way and creating accounts for the test. In addition, creating each survey and answering all the required questions for every student takes time. This discourages comprehensive testing as it becomes the largest step in adding a new feature.

Automated testing interacts directly with the website through an automated browser. For Cypress, the automated browser identifies different elements of the webpage through html name, id, or other tags. Cypress can then simulate clicks, type into content boxes, or ensure that certain pages appear after a given action. This dramatically speeds up the testing process. It lowers the burden for quality software and makes it easier to validate old features and test new ones.

```
// Login to website
cy.get("input[name='username']").type("fakelarrysmith");
cy.get("input[name='password']").type("Dallas!!");
cy.get("button[name='login-submit']").click();

// Take survey for 123
cy.get("button[name='First Peer Eval for 123']").click();

// Fill out check boxes for first quesiton
cy.get("input[id='score_14_question_1_2']").click();
cy.get("input[id='score_16_question_2_2']").click();
cy.get("input[id='score_16_question_3_2']").click();
cy.get("input[id='score_15_question_4_2']").click();
```

*Figure 3 Automated Testing Example*

Figure 3 shows how the automated testing tools can find html input and automate the process of taking a survey. The only requirement is that a standard database is used that already contains the users, departments, and courses that are used for testing in the automated testing script.

Overall, the automated end to end testing allows for quicker validation of new features and improves development substantially.

While tests were an important part of the design process for this project, the database also presented difficulties as we added one of our functional requirements: automatic data cleansing. To help the team understand the database the previous team had set up and easily identify any needed refactoring, an Entity Relationship Diagram (ERD) was made.

Unfortunately, the ERD revealed some puzzling database design decisions. For example, Questions and SurveyQuestions were entirely identical, but were represented as two different categories. In addition, teams were represented as projects in the course and made use of very odd linking. This confusing language halted our groups progress when we added the manage teams features.

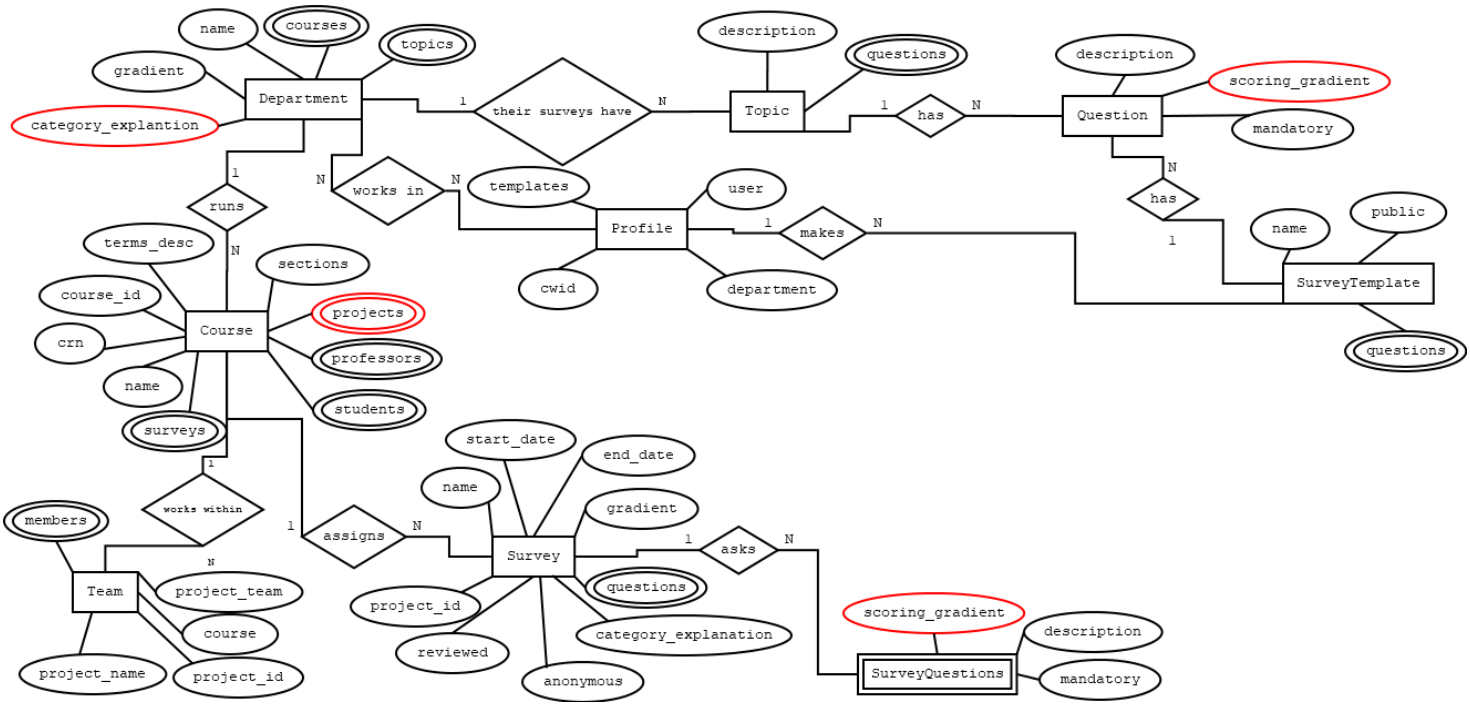


*Figure 4 Entity Relationship Diagram*

The attributes with a red border indicate that they must be initialized with a JSON file. After reviewing the existing database, modifications were made to simplify and condense the database, such as removing SurveyQuestions entirely and replacing them with Questions. This created more issues than our group originally anticipated due to how Django handles database migrations. Even though the SurveyQuestions table was identical to the questions table, the switch was difficult.

To set up automatic data cleansing, a separate command file was created to be run on the terminal that runs the website. The data_cleanse command looks through the database for any surveys that are at least seven years older than their end date and automatically deletes them and related items (such as their feedback), and this command can be set up to run daily at any desired time (i.e. every day at 1:00 AM).

Several additional AWS platforms are used to meet the needs of the web application. The Lightsail instance is configured to generate certificates for the domain. Route 53 manages the domain name for the website and emails. Amazon Simple Email Server (SES) is invoked via SMTP for account requests and survey notifications. Amazon WorkMail is used for handling bounce and complaint requests and forwarding them to site admins.

# V. Software Quality

## Testing Old Features:

The Problem:

While the current code base shows 100% coverage using Django's coverage tool, using the website is not a bug free experience. The bugs can appear as small text glitches on common fields or website crashes.

Our Solution:

Due to the varying severity and abundance of the bugs, focusing on the most severe bugs first is our main priority. To accomplish this, three steps must be taken.

- First, we must label and categorize the bugs that we find. We will accomplish this through QA testing.
- Second, if the bug is unacceptable, we must write tests to show where code is failing. This will ensure that the bug does not reappear and that we understand what logic is failing.
- Third, we must remove the bug and retest the functionality.

Policy:

- If any bug is encountered while working on the website, it must be added to the backlog along with the severity of the bug and instructions on how to replicate said bug. Approaches to verifying the bug fix will vary on a case-by-case basis determined in sprint planning.

## Testing New Features:

The Problem:

When new features are added, we will write unit tests in the tests.py file to maintain 100% coverage using Django's coverage tool. That said, 100% coverage using Django's coverage tool is not enough. To ensure that our new features do not repeat the mistakes of the previous project, our code needs to be thoroughly tested with QA testing.

Our Solution:

To ensure that new features only add the intended functionality, each new feature must maintain 100% coverage using Django's coverage tool. In addition, each feature must be thoroughly QA tested by the developer that wrote the new feature AND one additional developer on the team.

Policy:

- All new features must maintain 100% coverage using Django's coverage tool by writing new tests in the tests.py file.
- When a new feature is complete the developer of the new feature and one additional developer must QA test the feature by using the feature while attempting to break the feature for a minimum of 20 minutes (This may vary by feature).

## Maintenance:

The Problem:

To ensure that the project is maintainable, our group needs to write our code in a similar way to the previous groups. In Django, there are multiple ways to accomplish similar tasks. If our group chooses to use alternate methods to implement our own functionality, it will make the code base more confusing for future maintenance.

Our Solution:

Due to the scope of the project and the scale of the current code base, we cannot refactor the existing code base for the remainder of the semester. For that reason, our primary objective is to improve the code base where we interact or make changes.

Additionally, to ensure our additions are consistent with the existing code base, we will reference and expand the Django Tech Notes document to reflect the practices used in the existing code base.

Finally, continuing the high standard of quality comments that are already in the code base is important. The comments will ensure that future developers will have some context for the project if other documentation is lost. If other documentation is not lost, comments will still remind developers of the context of the code.

Policy:

- When implementing basic Django functionality, a developer should ensure that it aligns with the examples provided in the Django Tech Notes to ensure a consistent code base.
- When a new feature or change is completed, a developer should review the code for good coding practices or confusing uncommented code.

## Project Ethical Considerations

This project has two main ethical concerns as the website goes live. The first is the importance of keeping anonymous comments anonymous. In our project this looks like ensuring that there are no information leaks and that a student cannot determine who has sent them a specific comment. This relates to ACM Code of Ethics 1.6 - Respect Privacy and ACM Code of Ethics 1.7 - Honor Confidentiality. The second is protecting student data student data. In this project this looks like ensuring that sensitive academic data, such as the grade a student receives on a peer review is not unjustly disclosed. This relates to ACM Code of Ethics 1.2 - Avoid Harm.

A general concern relates to ACM Code of Ethics 2.1 - be honest about limitations. The initial code base was riddled with bugs and broken features. We had to disclose these issues and discuss how they impacted the ability to deliver new functionality. This included lost SSH keys resulting in lost work on the server, misrepresented features, and unstable releases. Having to disclose this to the client was difficult as we knew we weren't meeting their hopes.

# VI. Results

In the end, several functional and non-functional requirements have been met, but the project overall still needs more work. Requested features such as better student/team management (add/remove students without csv files, etc.), automatic data cleansing, the ability to specify and edit scoring schema, and the migration to Bootstrap 5 have all been completed, but some features from previous iterations of the project exhibit issues.

Features such as the improved student/team management improve the user experience for the professor. Implementing the team management features was more difficult than anticipated. The previous group used the terms 'project' and 'team' interchangeably in some definitions, but differently in other definitions. This made reviewing and improving the previous code for anything related to projects and teams very difficult. While we were eventually able to implement the features, we were unable to completely remove or redefine the two terms. We determined that the work would be more challenging than it was worth. Additionally, removing or redefining projects in the code could have caused more issues than would have prevented.

To address the issues with already existing features and to make future work on the project easier, the team has created an extensive set of tests that can be used to ensure quality for the entire website (not just the database and back-end). Issues and bugs that still exist have been catalogued for future work. The Cypress tests provide a minimum standard of stability for the website and ensure that the website can function. The tests do not cover all edge cases, but do cover expected use cases for admins, professors, and students. As detailed in the technical design, testing the website thoroughly is a tedious and difficult task. Creating questions as an admin, creating surveys as a professor, taking the surveys as a student, and reviewing the surveys as a professor takes a couple of minutes. Doing this every time you make a change to the website is not sustainable. Thus, the Cypress tests promote software quality by making it substantially easier to validate that the website is functioning as intended.

Additionally, there were issues with the LightSail instance the website was hosted on. Notably, the SSH key for the webserver had been lost and no auto renewals for the certificates had been set up. To address these issues, the team set up a new LightSail instance. The LightSail instance was set up in the same fashion as the original server, but the SSH key was accounted for and not lost. Auto renewals were also set up for the certificates to prevent the website from losing its domain access again.

The temporary Gmail service used to inform users of their new accounts was replaced with an Amazon SES email server. This means that rather than coming from a Gmail account, the email originates from noreply@ramblinwreckreviews.com. Not only is the email address more professional, but it allows for more emails to be sent out than the temporary Gmail service. This is important as it allows for the surveys to not be limited by the number of students in EDNS classes.

The database has also been improved to be more concise and manageable. This will decrease the confusion for future groups working on the project. In addition, our group created an ERD to better explain the relationships between the different elements in the database.

The migration to Bootstrap 5 allowed the team to improve the UI somewhat as well, but the website now no longer works on older browsers, such as Internet Explorer. Current versions of Google Chrome, Mozilla Firefox, and Microsoft Edge are all able to use the website.

# VII. Future Work

As mentioned in the results section, some of the already existing features for the project do not work entirely properly, and so future work on the project will include addressing these issues (which can be identified with the bug report and tests the team has generated). The team has also implemented an automatic data cleansing system to keep the database for the website manageable, but the system only deletes surveys if they are at least 7 years older than their closing date. Although this system is still FERPA compliant and helps keep the database from growing too large, a more customizable system with a dedicated webpage that allows professors to set when they want their surveys to be deleted will allow for a more user-friendly experience and allow the school to decide how long surveys should be kept.

Additional cleansing of old users and courses would also help reduce the database size and increase security. The current database cleansing script only removes surveys, which have the most data associated with them, but a future improvement could expand the cleansing script to remove inactive users.

Overall, the largest issues that must be addressed in the future are the UI/UX and the consistency of existing features. The UI of the website can at times be clunky and in need of improvement, especially for department admins and professors. Actions such as creating a survey need to be streamlined. When a user is familiar with the website, the templates are a helpful feature, but they confuse new users. Nowhere on the professor page is there a 'create a survey button'. The button does not exist because templates are used to create and save surveys for assigning to multiple classes, but this is not immediately clear to the professor.

Additionally, the UI/UX could be improved to be more inviting and modern. The website doesn't look bad, but it certainly lacks a modern look. Given this is a tool created for the School of Mines, it also represents the quality of the School of Mines. If the website were to be expanded and used in another university, the presentation would need to be improved. The website is less cumbersome and clunky than CATME, but there is still room for improvement.

While the test coverage ensures that core features are not broken, a more comprehensive test suite would catch more bugs and glitches. Tests that focus on edge cases, or deliberately nonsensical user actions would greatly improve the test coverage. In addition, it may increase security as it could catch unintended information leaks.

Additionally, email bounces and complaints are currently handled by the admin manually. Combined with Amazon SNS, bounce complaints can be handled automatically. A unique page set to handle API calls can be used to handle these notifications. Automating the process would reduce the demand on administrators. Django is capable of handling these notifications and implementation would not be difficult.

Finally, there are currently acceptable bugs still not fixed in our backlog. These bugs have been well documented but were not nearly as high a priority as others. Examples include uploading class spreadsheets in the form of 'filename.csv.csv'. While this bug would probably never occur, an unknowing user could accidentally upload a spreadsheet of this form and be confused as to why the website does not respond. This bug, and others like it, were deemed acceptable due to their minimal impact and low likelihood. Future groups may address these bugs to provide a more stable website.

## VIII. Lessons Learned

- Django is a powerful tool for connecting the back-end and front-end of a website. One does not need extensive knowledge in SQL to create a database with it, and Django makes creating unit tests for the website database easy. There is also plenty of documentation for Django, making it an easy to learn and use framework.
- Cyprus is a fantastic tool for testing front-end features and website UI as well as their integration with the back-end. It's an easy to learn, free program that allowed us to automate the process of testing the website UI. This makes tests faster to create and execute.
- If you're working with an existing database, making an ERD of it can help quickly reveal any issues with the structure of the database, and it can help the team quickly understand how the website interacts with the database.
- Tests ensure smooth development. Even if a feature was made by a previous team, it is crucial that it is extensively tested before any other features depending on it are made. Tests should also cover every end and feature of the website, regardless of how little code goes into it.
- Agile development helps make an overwhelming amount of work manageable. New features were planned out on a sprint-by-sprint basis, but often, team members would stumble upon bugs and issues while trying to complete their tasks. Agile development encouraged us to communicate as a team frequently, and so these issues would always be made present immediately and could be addressed in later sprints. By enforcing communication and organization, agile development helped us as a team stay on track, even when the workload grew as we worked.
- Quality of work is more over quantity. The previous group did a large quantity of work that gave the website a foundation. Nearly all features were accounted for, however, the implementation was often buggy and confusing. This constantly prevented our group from gaining forward moment as we had to address the shortcomings of the previous codebase. Ensuring quality code would have made the transition from project to project much smoother, even if that meant limiting the scope.
- Address problems early. Our group was originally tasked with a different project, but the vague and unclear definition made us pivot to this project. Our entire group is very grateful that we decided to take action and address the problem early rather than waiting for it to become a much more challenging problem.

## IX. Team Profile

- Connor Brown
    - Connor is a first-generation college student originally from Austin, Texas. Connor is avid about health and wellness, enjoying hikes with his dog, Java, strength training, and biking. During his time at Mines, Connor has worked for Mines assisting with the transition to remote as an ITS assistant and maintaining the Undergraduate Research and Honors website. Outside of school, Connor served as treasurer of the Interfraternity Council and Vice President of Finance of his fraternity, and he has spent time working in the Computer Science industry interning at Helmerich & Payne, prototyping legacy product solutions transitions to the cloud, and as a primary developer for a new web product. Connor will return to Helmerich & Payne after this fall to continue his work developing, maintaining, and testing web apps.
- Chandler Carter
    - Chandler is a Computer Science major who is graduating Fall 2023. He is a TA for the Computer Science department and is currently considering graduate school. His main hobbies are video games, music, and cooking.
- Hayden Cooreman
    - Hayden is a current senior majoring in Computer Science with a minor in McBride Public Affairs. He is a member of Blue Key Honor Society, Mines' Student Representative to the Board of Trustees, a Software/Systems Engineering Intern at Northrop Grumman Space Sector, and an undergraduate researcher at NREL. When not working or studying, Hayden is an avid mountain biker, reader, and nature photographer. Hayden is motivated to stay at Mines for a Master's Degree in Computer Science. After Mines, he intends to attend law school and practice in consumer privacy rights and consumer advocacy.
- Thomas Jensen
    - Thomas is a senior Computer Science major who is graduating in December of 2022. On campus he is a tight end on the Mines football team as well as a member on the executive board of the Mines chapter of the National Society of Black Engineers. Over the last few summers he has worked at Procter and Gamble as an IT Management intern.
- Wilhelm Northrop
    - Wilhelm is a senior majoring in Computer Science on the Computer Engineering track. Last summer Wilhelm worked as a Patent Engineer Intern at Holland & Hart in Boulder. He is pursuing a career in patents and will continue his work at Holland & Hart in Boise, Idaho after his graduation in May. Besides professional goals, Wilhelm is currently training to hike the 567-mile Colorado Trail next July.

# References

Shock, C., Graham, T., Onwudiachi, O. K., & Wattanasupt, P. I. (2022). (rep.). *CSM EDNS 1 Final Report* (3rd ed., pp. 1–77). Golden, CO.

US Department of Education (ED). (2021, August 25). *Family educational rights and privacy act (FERPA)*. Retrieved September 17, 2022, from https://www2.ed.gov/policy/gen/guid/fpco/ferpa/index.html

# Appendix A – Key Terms

Include descriptions of technical terms, abbreviations and acronyms

| Term | Definition |
|---|---|
| *AWS* | *Acronym for "Amazon Web Services".* |
| *CATME* | *Acronym for "Comprehensive Assessment of Team Member Effectiveness".*<br>*A web-based peer evaluation interface developed by Purdue University.* |
| *CSV* | *Acronym for "Comma-Separated Values".* |
| *DDoS* | *Acronym for "Distributed Denial of Service".*<br>*A type of attack that makes resource unavailable to intended users.* |
| *Django* | *A high-level Python web framework .* |
| *EDNS* | *Acronym for "Engineering, Design, & Society".*<br>*A department for the school and the clients of this project.* |
| *ERD* | *Acronym for "Entity Relationship* Diagram".<br>*A flowchart that illustrates how entities relate to each other within a system.* |
| *JSON* | *Acronym for "JavaScript Object Notation".*<br>*A lightweight data-interchange format, easily read and written by humans and machines.* |
| *LightSail* | *A virtual private server through Amazon that offers management of cloud resources* |
| *FERPA* | *Acronym for "Family Educational Rights and Privacy Act".*<br>*A Federal law that protects the privacy of student educational records.* |
| *RDS* | *Acronym for "*Relational Database Service".<br>A *managed, open-source cloud database service through Amazon.* |
| *Schema* | *A Representation of a plan, in this case grades, as a model.* |
| *SES* | *Acronym for "Simple Email Service".*<br>*a cloud email service provider through Amazon* |
| *Shibboleth* | *A web-based tool that supports SSO between two organizations or applications.* |
| *SSO* | *Acronym for "*Single sign-on" |