Summary

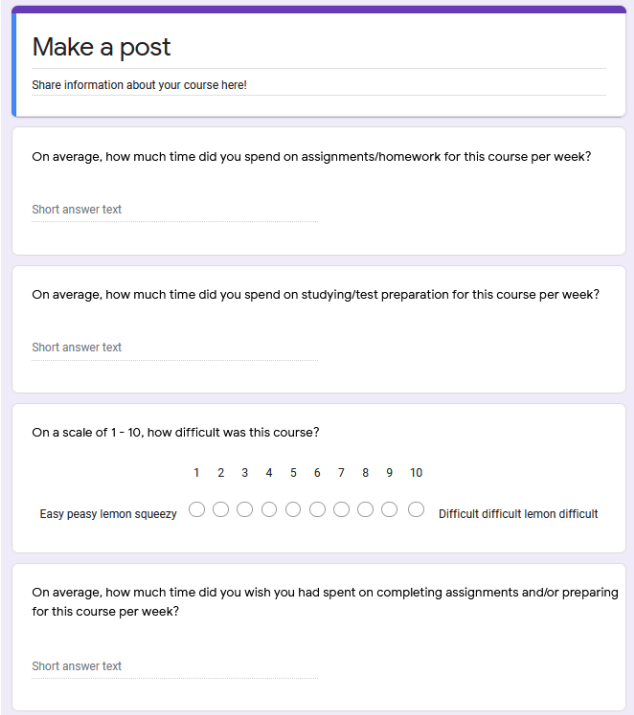
Our project name is *RateMyClass*. Our client is Professor Robert Lightfoot. The overall idea of this project is to allow students to get a sense of how each class compares to other classes based on difficulty and/or time consumption. Searching does not require an account and searching can be done by course prefix, number, professor name, and/or university name. To make reviews, an account must be created. Accounts can be created by both students and professors with admin accounts being created by special privilege with a special password.

Students are also able to provide ratings and comments in their reviews. Furthermore, both students and professors can then comment on each review. Comments and reviews can only be edited by the student/professor who created it. Inappropriate reviews and comments can be flagged by students and professors. Then, an admin can either unflag it or delete it. Professors can also pin certain reviews they deem to be of importance. Each of the ratings are also averaged at the top of the page.

Description of User Stories – Implemented

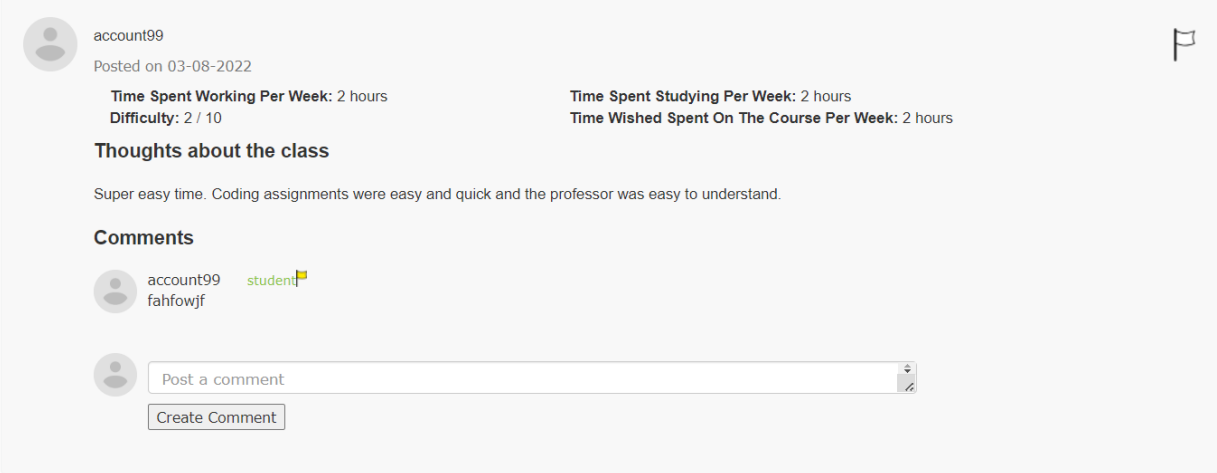
Make a post about a class: this feature allows the user to create a post for a course by entering a set of information into a form and then displaying the data collected on the website. We gave this story one point since it was a fairly simple extension of basic rails scaffolding for the reviews database. This story did not change significantly over the course of development; however, new fields were added on occasion to prompt the user for more information when creating a post.

Screenshot of Application

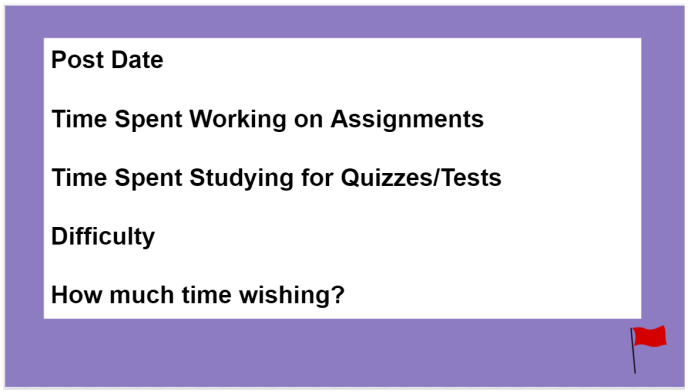


Lo-fi Sketch

Read posts: this feature allows for the user to view a post in the reviews database in a tabular format. We gave this story one point since it was also a relatively simple extension of rails scaffold for the reviews database. This story did not change significantly over the course of development; however, its role in displaying reviews to the average user did as it is reserved for admin usage at this point.

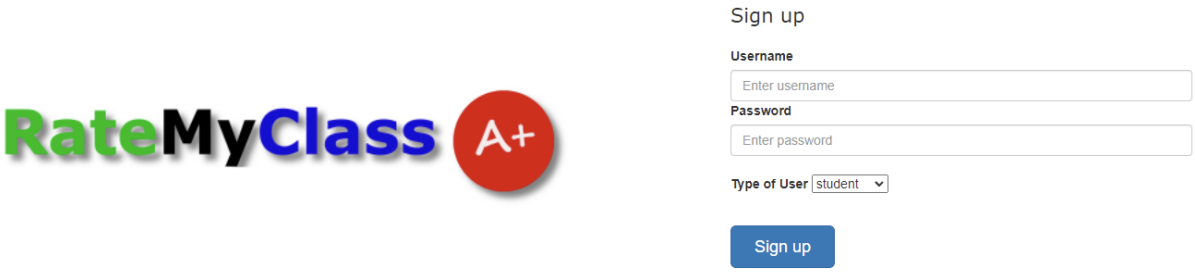


Screenshot of Application



Lo-Fi Sketch

Signup page: this feature allows for the user to create an account through a signup system and have that account added to the users database. We gave this story one point since it was also a relatively simple extension of rails scaffold for the users database. This story did not change significantly over the course of development; however, the type field was later added to facilitate the creation of different user types such as professors and admins, which also led to the modification of the backend to process these new user types.



Login page: this feature allows for the user to log in to an account they previously created which was stored in the users database. We gave this story one point since it was relatively simple to check if the user login credentials matched any in our user database. This story did not change significantly over the course of development.

Screenshot of Application

Lo-fi sketch

Delete user: this feature allows for the deletion of a user from the users database. We gave this story one point since it was also a relatively simple extension of rails scaffold for the users database. This story did not change significantly over the course of development; however, the ability to delete a user was given exclusively to admin users.

Remove post: this feature allows for the user to remove a post from the reviews database. We gave this story one point since it was also a relatively simple extension of rails scaffold for the reviews database. This story did not change significantly over the course of development; however, the ability for a user to delete a post was given exclusively to the account who created the post and an admin.

Search for a course: this feature allows for the user to locate a specific course and the attached reviews therein. We gave this story one point since it was a relatively simple search algorithm and not too difficult to implement. This story did not change significantly over the course of development; the search remained largely the same.

Delete comments: this feature allows for the user to delete comments on a review from the comments database. We gave this story one point since it was also a relatively simple extension of rails scaffold for the comments database. This story did not change significantly over the course of development; however, the ability of the user to delete a comment was given exclusively to the account that posted the comment and an admin.

Create secondary comments table: this feature allows for comments attached to a review to be stored in the comments database. We gave this story one point since it was relatively simple to create the comments database and link it to the reviews database. This story did not change significantly over the course of development.

Add more information to the reviews database: this feature added more fields to the reviews database and by extension to the create a post form (course code, department, professor, etc.). We gave this story one point since it was relatively simple to add additional column entries to the reviews database. This story did not change significantly over the course of development.

Make a comment on a post: this feature allows for the user to create a comment attached to a specific review and add it to the comments database. We gave this story one point since it was also a relatively simple extension of rails scaffold for the comments database. This story did not change significantly over the course of development; however, the ability of the user to create a comment was given exclusively to students and professors with an account who are signed in.

Make reviews more readable: this feature made it easier to tell what page the user was on and what they were looking at. We gave this story two points since it involved a great deal of database manipulation and sorting in order to get the reviews grouped together

correctly and to show that information in a way that was easy to understand. This story changed somewhat over the course of development, in that, the way in which we displayed the reviews to the user evolved as new features and styling came in. Additionally, with the inclusion of new user types, features available only to professors or admins were also added (pinning, unflagging, etc.).

Edit post: this feature allows for the user who created a post to edit their post and have that change reflected in the reviews database. We gave this story one point since it was also a relatively simple extension of rails scaffold for the reviews database. This story did not change significantly over the course of development.

Delete comments: this feature allows for the user to delete a comment attached to a specific review. We gave this story three points since it was difficult to find and delete the comment with the way in which our comments database was set up. This story did not change significantly over the course of development; however, the ability for a user to delete a comment was given exclusively to the account who created it and an admin.

Flag a post: this feature allows for the user to flag a post or a comment and have that reflected in the view. We gave this story one point since it was relatively simple to add a flag field in the comments and reviews database and set it accordingly. This story did not change significantly over the course of development; however, the ability for a user to unflag a post was given exclusively to an admin to simulate the review process for flagged content.

Professor pin reviews: this feature allows for a professor user to pin certain reviews to the top of a review grouping for a course in order to highlight or showcase reviews they think are good. We gave this story one point since it was relatively simple to reorder the reviews grouping for that course to place pinned reviews at the top. This story did not change significantly over the course of development.

Add grade distribution link: this feature allows for the user to see the grade distribution for the course they are viewing if it is a Texas A&M course. We gave this story one point since it was relatively simple to call this grade distribution website as a link and pass it the course data to show. This story did not change significantly over the course of development.

Create and standardize navigation bar: this feature allows for the user to navigate the website from any page with the navigation bar at the top, allowing for easy access to each function of the site. We gave this story three points since, although it was simple to add functionality to it, it was difficult to create the front end of the navbar using bootstrap and css. This story did not change significantly over the course of development; however, new links were added on the navbar according to what type of user was signed in.

Add ability to see other courses of that type with different professors: this feature allows for the user to view all professors teaching the course the user is viewing. We gave this story one point since it was relatively simple to reorganize the reviews grouping in this format. This story did not change significantly over the course of development.

Add ability to see other courses taught by that professor: this feature allows for the user to view all courses taught by the professor the user is currently viewing. We gave this story one point since it was relatively simple to reorganize the reviews grouping in this format. This story did not change significantly over the course of development.

Finalize styling: this feature finalized all styling and formatting for each page on the website into a presentable application. We gave this story three points since it was challenging to create a good looking and easy to navigate front end with bootstrap and css. This story did not change significantly over the course of development.

A Note on Legacy Code

The project is not considered a legacy project as the project idea and development was implemented from scratch. No legacy code was used in this project.

Team Positions

During the initial stages of the team formation, Osric Nagle was selected to be the Product Owner and Jacob Smith was selected to be the Scrum Master. As development of the project progressed, the team's dynamics changed and Jason Bernal was selected to be the Product Owner. The Product Owner was responsible for scheduling meetings with the client, ensured the team met the client's needs, and oversaw the development of the project. The Scrum Master was responsible for facilitating standup meetings, document iteration reports, refine and maintain the product backlog. Although the responsibilities of the roles are clear, the team would interchange responsibilities as needed.

Scrum Iteration Summary

Iteration 0: Our team was formed and a project idea was generated. We brainstormed our first user stories, made our first lo-fi mockups and created an initial UML diagram.

Iteration 1: Our first iteration consisted of setting up the database with tables to represent a user and post/review. We then began creating the front end login page and front end signup page for the users.

Iteration 2: In this iteration, we went ahead and created the backend logic for create, read, update, and destroy for users (linked with the front end pages created in iteration

1) and posts/reviews. We also began to write cucumber tests to demonstrate that the actions above work for both users and posts.

Iteration 3: This iteration consisted of adding the comments portion for our project, which consisted of being able to create, read, update, and destroy comments. We also made each comment dependent on a review. We also added more fields to the reviews, such as writing comments and which user created it. We were thus required to add more fields on the new reviews page. Finally, we implemented the search feature. We also wrote cucumber tests to ensure that the backend logic was working properly for comments, search, and reviews pages.

Iteration 4: Here in iteration four, we implemented flagging reviews and comments by professors and students. We then also created different views for students, professors, and administrators. For example, only administrators can destroy other users,reviews, and comments but can't comment or make reviews.

Iteration 5: We had most of the functionality down by the time this iteration started. We simply added a feature that allowed students to view other courses taught by a particular professor and other professors who teach the same course. The main idea at this iteration was to finalize the styling of the application to make it look more presentable and appealing.

- List of customer meeting dates, and description of what happened at the meetings, e.g. what software/stories did you demo.

Customer Meeting Dates:

- Iteration 0: 12/29/2021 at 1:00pm EET

Project idea was proposed and approved/finalized. Team roles were assigned, client meetings were planned, and initial user stories, logo, and lo-fi mockup were created.

- Iteration 1: 01/06/2022 at 1:00pm EET

We met at a cafe across the street from the hotel. Login page and signup page was demoed. We discussed when our meeting would be and how our code will be integrated with the user stories.

- Iteration 2: 02/08/2022 at 12:15pm CDT

Create User, Delete User, Edit User, Login as User, Sign up as User, Create Review/post, Edit Review/post, Delete Review/post, View Review/post demoed. We discussed how we started to implement cucumber and rspec testing for backend code logic.

- Iteration 3: 2/22/2022 at 12:15pm CDT

Search A Course, Comment on a Review/Post, Add More Information to Course, Remove a Post, Create secondary comments database, Edit Comment, and Delete Comment demoed. We demonstrated our cucumber tests and how much code coverage we had. We also discussed how we can implement moderation and grant special actions to certain users.

- Iteration 4: 3/8/2022 at 12:15pm CDT

Flag a post and comment (student, professor, admin), make a comment on a post (student, professor), add views for student, professor and admin, pin a review (professor) demoed. We discussed that we could also sort reviews by professor and view other courses taught by a particular professor.

- Iteration 5: 4/14/2022 at 10:30am CDT

Add ability to see other courses of that type with different professors, Add ability to see other courses taught by that professor, Finalize styling demoed. We also demonstrated that we had achieved about 95% of line coverage for our cucumber tests and wrote over 50 rspec tests.

- Explain your BDD/TDD process, and any benefits/problems from it.

Behavior Driven Design: Our first step is to look at the user stories we generated. Then, we write a cucumber test using capybara to and web steps to perform a certain task we want it to perform. Then, we may test to ensure that we are on the correct page and/or that we see the correct output. We also coincided with the coverage report to ensure that we were covering necessary lines.

If the test failed, we would often look at the error message. Typically, the error is that the content or button is not there. Sometimes, it is due to a routing issue as a result of not reaching the correct route due to an error. Another common error is that a certain field does not exist. This is because we might not have typed the correct field into the step definition.

One key problem was that when we had to edit front end layouts and add other features, we often changed the button name or path of a certain button. As a result, this may cause many tests to fail due to certain buttons not existing on the new page and/or the result of being on a different page.

Test Driven Design: Our process regarding test driven design involved creating Rspec test cases for modules, or features, and ensuring that there is proper functionality within

those modules. The difference between behavior driven design and test driven design is the scope of coverage the tests would cover. The behavior driven design would cover a high level approach to our application whereas test driven design would cover a low level approach to our functionality of the application.

The benefits of this approach allowed for proper functionality to be implemented within the application. The logic behind the application was thoroughly tested to ensure that there are no/minimal defects in the code.

The problem with this approach is that it limited ideas in the implementation of the application. It's possible that the development team focused primarily on ensuring the test cases passed and did not expand on the functionality of the application.

Configuration Management Approach

As a team, we mainly worked off of Amazon Web Services Cloud 9 platform which allowed us to code simultaneously on the same project. We did not need to do any spikes. We technically have two branches. However, we only use the master branch as the main branch was our initial attempt at setting up the project, which was not successful. We pushed to Github and Heroku after making a few changes. There were not any formal releases although there are tags for each iteration.

Issues We Had in the Production Release Process to Heroku

Throughout the development of the application, new releases of the application were deployed and thoroughly tested to see if the deployed version of the application was working properly. There would be instances where the application would work locally through the EC2 instance but not be working on Heroku. This would typically occur when integrating a new feature. The development team would debug the codebase, redeploy the application, and retest the deployed application until the application performed as expected. The major cause was due to the db:migrate not being ran on the production side when a particular portion of the code needed to access some portion of the database. Another issue was certain functions, paths, or buttons not compatible so that needed adjustment.

Issues We Had with AWS Cloud9 and GitHub

The standard version of AWS Cloud9 did not have enough computational power to handle the development of the project. The service would occasionally crash/freeze due to too many developers making changes at once to the codebase. It would often crash

when all three members of the team were on the codebase. An upgraded version of AWS Cloud9 was used to mitigate this issue. The upgraded version of AWS Cloud9 does come with a premium charge of $0.0464 per instance hour, which is an increase from the standard version which was free. Due to how inexpensive the cost was, this was within budget for the development team. From there onwards, the issue did not arise again.

The integration between AWS Cloud9 and Github was not straightforward for the development team as the introduction to AWS Cloud9 was a new tool to the team. To ease development, Jacob's Github credentials were used as the default user in Cloud9's environment and thus every time a commit was made by anyone in the development team, the commit was made under Jacob's user credentials. This is typically not standard practice as the case is usually each user should have their Github credentials connected to the codebase and each commit should be credited to the individual user but this was deemed acceptable to facilitate development early on.

Other Tools Used

CodeClimate's Quality feature was used to rate the maintainability of the application throughout its development. During every iteration of the application, the quality feature was utilized to help the development team ensure that the codebase was written in a manner that will ease future development and maintainability.

SimpleCov was used to determine the code coverage of the team's RSpec and Cucumber tests. SimpleCov would determine the relevant lines of code to test and generate a report that the team can view to see what lines were tested. This was essential in testing the codebase and ensuring that proper functionality was implemented.

Development:

Use 'rbenv' to switch the rails version.

Use 'rake db:migrate' to set up the database. Use 'rake cucumber' to run cucumber tests. Use 'rake spec' to run spec. 'rake' also has other commands. Run 'rake –tasks' to see other tasks.

`bundle install` command to install the gems used in the rails application.

`rails s -b 0.0.0.0 -p 8080` command was used to run the rails app through the EC2 instance.

Deployment:

`git add <file>` command was used to stage the files to be committed.

`git commit -m “<message here>”` command was used to leave a commit message to the files.

`git push heroku HEAD:master` command was used to deploy the application.

- Links to your Pivotal Tracker, public GitHub repo, and Heroku deployment, as appropriate. Make sure these are up-to-date.

Link to Pivotal Tracker: https://www.pivotaltracker.com/n/projects/2547059

Github Repo: https://github.com/RateMyClassBeta/RateMyClass

Heroku Deployment: https://evening-wave-53546.herokuapp.com/

- Links to your poster video and demo video.

Link to Slides and Demo Video:
~~This will be included by the deadline stated on Canvas which is May 10, 2022 11:59PM Central.~~
https://www.youtube.com/watch?v=dmUKYPPEubA