UDiscuss

Team Not Piazza

Lucas Katsanevas, Mark Patterson, Ryan Howell, Kaden Hendrickson

September 9, 2022

Table of Contents

1. Executive Summary

With the School of Computing paying roughly $12,000 per year on Piazza, a replacement product is clearly needed for online classroom discussions. However, all of the alternatives lack key features or are difficult to use. The purpose of UDiscuss is to provide an online classroom forum application that is free to the university and satisfies both students' and professors' discussional needs in a straightforward manner. It will be able to be connected to students' university accounts for roster setup and ensuring each student has exactly one account, and used in Canvas to keep all LMS data in one place.

The intended users of UDiscuss are students, professors, and TAs at the University of Utah. Although UDiscuss is being created with the School of Computing in mind, it will be beneficial to any college in the university and is being made as such. This is because every student needs a central place to ask questions and have discussions outside of the classroom, yet emailing instructors typically results in longer response times, less collaboration, and more stress for instructors. Furthermore, with COVID safety being a must, it is important for students who miss a class or participate over Zoom to still have a voice among their colleagues.

UDiscuss' main functionalities will be similar to those already found in Piazza. It will allow students and instructors to create questions and notes, while also letting them write questions and comments on existing posts. Students will be able to post anonymously, send private messages to instructors, and edit posts using a rich text editor. Despite these similarities, UDiscuss will have additional features as well. This includes sorting through questions more easily with filters, allowing students to delete posts, letting instructors mark a solution as finalized, and creating nested replies in comment threads. UDiscuss will also aim to have a simpler and more beautiful UI than Piazza and its alternatives.

One thing that makes this project interesting to us is being able to leave our mark on the university. All of our group members have had interactions with UDiscuss' alternatives and feel that our version is going to be something that helps students and instructors alike. With this idea in mind, we are also taking extra care in making this a well-documented and versioned code base given the high probability of it living past our class. Another thing that interests us is designing the UI. We plan on incorporating many nice features, such as collapsable comment threads, that make UDiscuss easy and enjoyable to use. While alternatives have made decent UIs, it is more difficult than it needs to be to read and create posts. As a result, we plan to implement flags on posts and other things that are easily recognized by the user to minimize confusion and time spent on the website.

2. Background

Overview of Project

The reason that our project is so needed is because it allows students to get quick help on questions in a nicely formatted forum. It also allows for students to refer back to questions they may have asked already or questions that their peers have already asked. This allows for more students to be helped and learn from one question, as opposed to many questions being asked in class and being forgotten shortly afterwards. This benefits professors for the same reason, saving them precious time during their busy days. In addition, people besides the course staff can answer questions and make follow ups, leading to a more collaborative environment.

The main example of a similar program and the one we are basing UDiscuss off of is Piazza. Piazza is currently used by the School of Computing and is a pretty good solution to the problem we are trying to solve. The issue with it is that it was initially free, but now costs the School of Computing $12,000 per year. It also has its fair share of flaws. One of the main flaws that detracts from its effectiveness is the overall bloat of the UI. The UI has many features that are never touched by the majority of users, such as its stats pages and networking tools. These items create distractions for students and instructors. Piazza also has minor flaws like the structure of the post threads and how posts are flagged. All of these deter students from using the resource as it can easily become overwhelming. On the other hand, Piazza does a lot of things right. Our goal for UDiscuss is to port these features and improve upon the things mentioned that they did wrong, while also making it free.

There are other alternatives to UDiscuss besides Piazza, such as Canvas Discussions and School Status. Canvas Discussions is useful in that it is already in Canvas and has all of the basic discussion functionality implemented. However, it lacks anonymous posting, has no support for voting, and makes it difficult to skim through posts. We plan to make UDiscuss better than it by supporting votes for posts and comments, giving instructors a way to mark solutions as verified, and allowing users to filter and search through posts more easily. On the other hand, School Status has many of the features Canvas Discussions is missing. It is mainly focused on LMS features and has an overwhelming number of functionalities. There is no free version and it's difficult for instructors to set up a class for the first time. UDiscuss will be better because it will be free, its UI will be less cluttered, and class setup will be simpler.

Technologies Used

We are leveraging many technologies in our project. The main thing that we wanted to focus on when choosing them was how well we knew them, as well as how well the people that will possibly be maintaining UDiscuss know them. With these things in mind, we created an ASP .NET Core 6.0 project with a React frontend and a MariaDB database.

Our reasoning behind choosing React was a bit different from that of the other technologies. With React's popularity being at an all time high, we have all been interested in learning it and believe that it will be beneficial for future jobs in industry. We also thought that it would be perfect for our forum application with how it can keep track of a component's state and its ability to swap components in and out. For instance, swapping a component containing the selected post and its comments for a create post component is much easier in React than in something like an ASP.NET MVC project. In addition, React works well with an ASP .NET Core application. We are using other frontend frameworks as well to make it easier to develop, such as Bootstrap and various NPM packages.

As mentioned, the reason behind choosing the rest of our technology was familiarity. Lucas and Mark already have experience with ASP.NET Core from the web software class. We have also all worked on projects using C# and the instructors that would be maintaining UDiscuss do as well. These instructors, Professor Kopta and Professor de St. Germain, have expressed interest in our project and believe they would primarily be responsible for it unless anyone from our team wants to help maintain it. We chose the latest version (6.0) of ASP.NET Core because we want to have the most up-to-date version available so that the university has as little work to do as possible in the future. It also may allow for expansion that would not be available in lower versions of ASP .NET Core. As for MariaDB, our comfort level with it is high as we have all taken the databases course. This also means that Professor Kopta's familiarity with it will be more than enough for maintenance in the future. Another plus is that it's completely free, as opposed to some of the other database options.

Software/Hardware Requirements

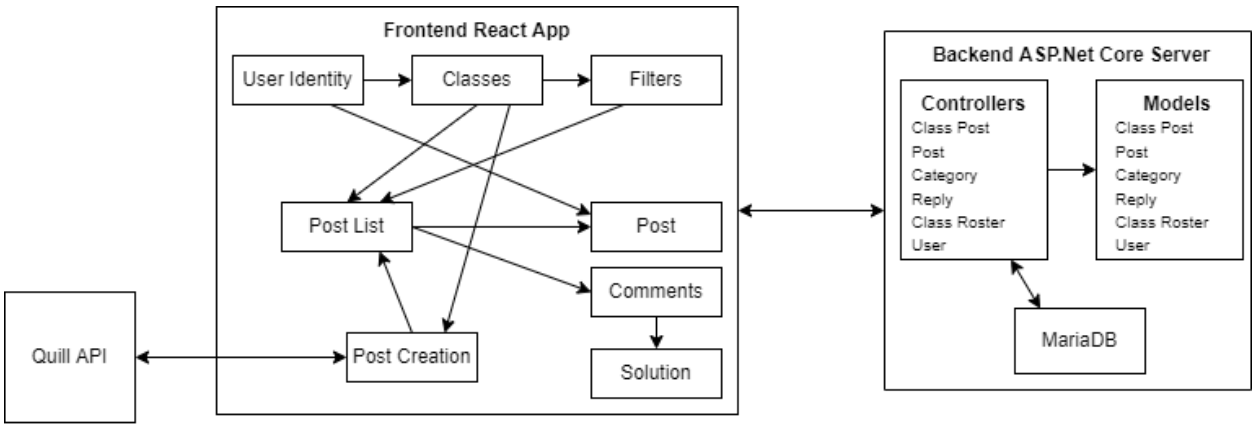
Currently, the only requirement for our web application is a computer that can run a web browser with JavaScript, which is nearly all of them. Despite this, we are using Bootstrap so that UDiscuss will look somewhat decent on phone and tablet browsers as well. We think it would be great if someone in the future turns UDiscuss into a mobile application, but we would like to focus on perfecting it as a website for the time being. There are no other requirements for our web application - the type of operating system,

memory, graphics card, browser, and other items are all flexible and up to the user to decide on.

3. Requirements Analysis

System architecture

From a high-level, UDiscuss consists of five major software components - the React app, Quill API, and the models, controllers, and database of the ASP.Net Core backend server. Within each of these, they can be further broken down into smaller components and interact as shown:



The React app consists of the post list, the selected post with a solution and comments, post creation via a rich text editor, various filters, classes, and user identity items (logins, roles, etc.). Each of these will be developed by our team, except for the identity items and rich text editor. The different user identity pages (login, signup, etc.) are going to use the auto-generated identity files provided by React and ASP.NET Core that can be customized to our liking. The rich text editor for post creation will use Quill, which is an API that allows developers to create customizable text editors with various functionalities and formats. The application will access it using a CDN, which quickly transfers the code needed for a component to the frontend upon request.

For each of the other React app components, data must be fetched from the API. Whenever the frontend requires data, it will send an HTTP GET or POST request to the backend server. The server reads this and calls the correct controller method based on the request's content. These methods handle the business logic of the application and send queries to our MariaDB database as needed. When queried, the database sends data back to the controller, which will send an appropriate HTML response back to the user when its method is complete. For each controller, there is a model of the same name that can be used as an object type. These models include classes, posts, post categories, replies, class rosters, and users.

The React components are all related in some sense. Upon logging in, the user's identity component will determine what classes they belong to and what posts they can see in each class's post list. Their identity also determines what role they belong to and what they are able to do as a result (i.e., instructors can edit a student's post and mark their own posts as announcements). The post list component retrieves data from the backend based on the current course selected in order to display previews of the course's posts. The posts on this list and the ordering of them can be changed through filters, whether it be a sorting filter or a class's categories. A class's categories are created by a professor and allows a student to find posts more easily by narrowing them down.

When a post preview in the list is selected, a query is sent to the API and retrieves the full post with its comments. If a post is a question, a solution is retrieved from the comments section if one exists. This solution will be a comment that is verified by the professor or, if none are verified, the comment with the highest number of votes tagged as a solution. The last component, post creation, can be used by any user in a given course and utilizes Quill API as mentioned above. After a user creates a post in a given class, it will show up in that class's post list.

Personnel

Our team is made up of four people: Lucas, Kaden, Mark, and Ryan. Ryan will be handling all the backend work, such as managing the database and creating controllers for the API calls we need. Ryan is doing this because he has taken the databases course and is more comfortable with backend work than the rest of us. Mark will help with getting some things done on the backend, such as writing tests for the controllers, but will eventually switch over to the frontend during the second semester. Because he has taken the web development and databases courses, he can help wherever needed depending on the week. His main frontend task will be getting identity scaffolded into the project and making different authorizations for different user roles.

Kaden will be working solely on the frontend in React. Although Kaden is great with databases and has experience with backend technologies from his internships, he wanted to step out of his comfort zone and improve his skills in something he is less familiar with. Lucas will also be working on the frontend, but has more experience from the web architecture class. While Kaden will mainly focus on getting the collapsable comment section, post list, and filtering working, Lucas will be working on the navigation bar, post display, post solution box, settings, and creating posts. Overall, we have a well-balanced team when it comes to our skill sets. However, our group understands that there will be difficult parts to this project and that we will need to be flexible in helping each other as a result.

System Features

Rank 1 - Bare Essentials:

The first bare essential would be the requirements for a working website. This would include a working database with a frontend and backend that communicate with one another. As for UDiscuss specifically, allowing students to ask questions and professors to answer them would be essential. A detailed list of bare essentials that our application needs to function is:

- User identity - logins and personal information from the database tied to each user.
- User roles - admins, professors, instructors (TAs), and students.
- Admin page - allows an admin to change the role type of any user.
- Course creation page - a professor/admin page to create classes and add students/TAs to them.
- Allow users to view a list of posts for a specific course they are in.
- The ability to view the details of a specific post in the list (get the full body, poster's name, etc.)
- Create posts (questions or notes) by giving a title and description.
- Respond to posts using a basic comment section (not yet with threaded replies).
- The ability for a professor or TA to verify a solution from the comments.

Rank 2 - Planned Features:

We have some planned features already completed and others that we want to make sure are done by the end of the year. Our planned features are:

- the ability for a user to be able to sort by new, unread, unanswered, answered, and instructor posts.
- The ability to use a search bar based on the title of a post, body of a post, or the comments of a post
- Having flags in the post list for read/unread questions and also answered/unanswered questions.
- Putting posts into post categories created by the instructor. A user can then click a post category to filter by posts only in that category.
- Allowing students to post and comment anonymously.
- Tagging comments as solutions vs followup questions. To go along with this, we would like to pull the top-voted solution out of the comment thread and tag it as the solution until an instructor has verified one.
- Ability for users to easily tag/refer to other posts so that other users do not have to write out entirely new solutions.

- Upvotes for posts and comments.
- Using dark mode, where we simply swap the dark mode and light mode CSS files depending on the user.

Rank 3 - Bells and Whistles:

These are the features that we believe will get the university to use our project for years to come. They may be out of the scope for a one year project, but we hope to complete as many of them as we can. These are:

- A rich text editor that would have nice LaTex formatting and code blocks in questions and answers.
- Having a more abstract way of demonstrating searching for a post. This would involve speech bubbles that would grow as more words are being typed and would allow users to select posts that have similarities to what they are currently typing.
- Editing and deleting your own posts and comments.
- Saving posts as drafts to finish later.
- Comment threads that can be infinitely nested and collapsable.
- Using the University of Utah's SSO (if given permission).
- Setting email notifications. This would allow the user to select how often they get emailed and for what types of posts for each of their courses.

4. Software Engineering Tools and Techniques

Versioning

Our team is using Git for version control due to our comfort with it and its popularity in industry. One advantage to Git is the way that it handles separate branches. We will be able to have team members safely commit buggy code or unfinished features without it interfering with the rest of the team's work. Branches also allow us to review each other's code and give feedback before merging into the main branch. Other benefits of using Git include using the tag system, viewing commit history on GitHub, and ensuring each team member is contributing to the project.

Communication

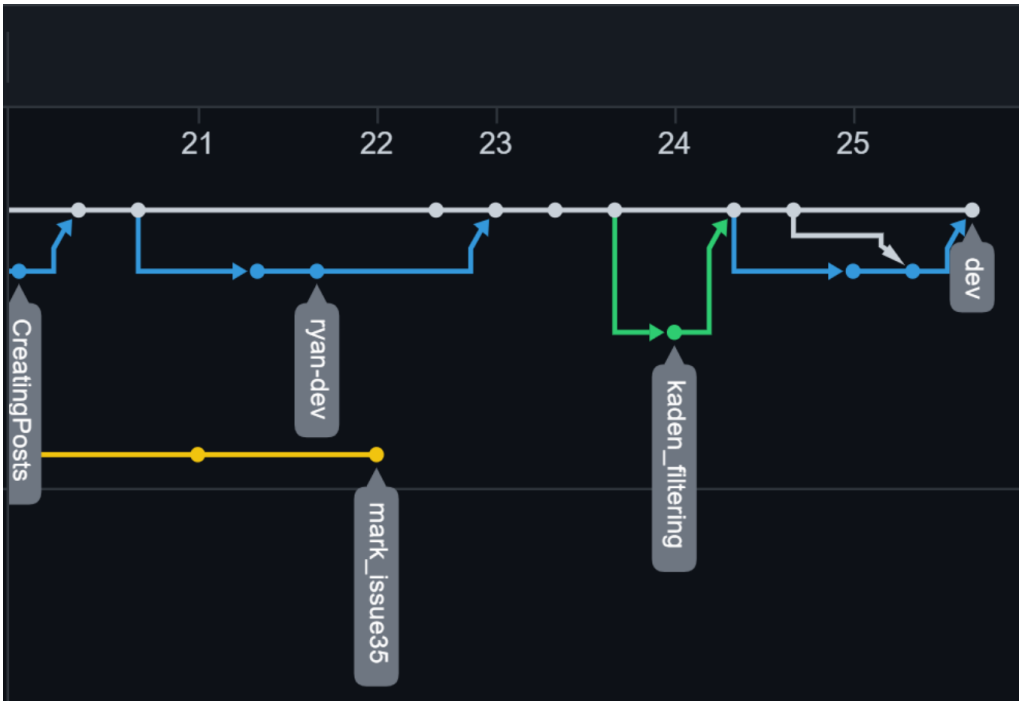
All of our team's messaging is done through Discord. We use our group server for most of this, but occasionally send personal messages to one another when debugging or doing other tasks that are irrelevant to the rest of the group. We are also using GitHub's Issues to assign weekly tasks and show each other whether or not they have been completed.

Team Meetings

Our team currently has stand-up meetings on Monday and Wednesday mornings on Discord. We each talk about what we did since our last meeting and what we need to accomplish moving forward. The first semester we were meeting for around 20 minutes each time, but have been talking a little longer now as we get deeper into our project and have more tasks that overlap with one another. We typically replace our stand-up meetings with longer, more formal meetings when extra tasks are assigned to us, such as presentations or papers. These generally are planned out and require us to do things like make important decisions or practice a presentation.

Development Process

A scrum development process is being used for our project. It is based on an iterative and incremental process, where work is broken down into sprints and is flexible over time. Although we haven't began it yet, we are planning on having Kaden be our scrum master moving forward. As a result, he will go over the issues we are trying to solve in a given week and assign them to group members if they aren't already. For each issue, a team member will create a branch for development off of our working branch. After an issue is tested and completed, the team member in charge of it will create a pull request, which will then be reviewed by either a teammate within a couple days or the whole team if a meeting occurs right after.



An example of the structure of how our branches are intermingling.

Bug-Tracking

We use GitHub Issues to track the majority of our issues and bugs. On the backend, we have created unit tests using XUnit to test various functionalities. We also have a Python program to show the basic use of our endpoints (before we moved over to auto-generated documentation). Overall, much of the tracking is done by creating an issue when a bug is discovered, creating a new branch based on our working branch, and then fixing the bug using that branch. Once we believe that the issue is fixed, we create a pull request, have a code review, and merge into the working branch.

5. Timeline

TASK NAME	DUE DATE	TEAM MEMBER
Research Scaffolding in React	2/21	Lucas, Kaden
Research ASP .NET Core Scaffolding	2/21	Mark, Ryan
Database Schema Modeling	2/21	Ryan
Polish UI Sketches	3/21	Lucas
Start coding front-end	3/28	Lucas, Kaden
Design Document first draft	3/30	Lucas, Mark
Add Scaffolding into C#	3/30	Ryan
Create basic post list item	3/30	Kaden
Create basic controller methods	3/30	Mark, Ryan
Create database	4/1	Ryan
Make the selected post on UI	4/1	Lucas
Match Post Componenets to database	4/1	Kaden
Create controller test suite	4/7	Mark
Use Swagger to create API docs	4/10	Ryan
Add Identity to React and C#	4/22	Mark
Create a reply field	4/22	Kaden
Add Setting Page, Home Page	4/22	Lucas
Finish API endpoint testing python	4/22	Ryan
Prepare for Pre-Prototype	4/22	Everyone
Create voting components	4/25	Lucas
Connect replies to DB	4/25	Kaden
Finish Design Doc	5/3	Mark

First Semester

TASK NAME	END DATE	TEAM MEMBER
Create team website	8/26	Mark
Split project into 2 projects (frontend and backend)	8/26	Ryan (API), Kaden and Lucas (UI)
Create loading spinner for fetching data	8/26	Lucas
Set up mock API and add data to it	9/2	Kaden
Dockerize the backend	9/2	Ryan
Implement light/dark modes with toggle	9/2	Lucas
Create basic rich text editor on UI	9/2	Lucas
Fix comments in API to nest correctly	9/9	Ryan
Design Doc Update 1	9/9	Team
Style comments to be like mockups	9/9	Lucas
Add authentication and login pages	9/12	Mark
Polish Post list styling from last semester	9/12	Kaden
Polish UI sketches for admin pages	9/12	Kaden
Connect authorization on frontend and backend	9/13	Mark
Post list indicators on API for questions (answered/unanswered) vs notes, read vs undread, and flagged vs unflagged.	9/16	Ryan (API)
Post list item indicators UI (read/unread, flagged/unflagged, post types)	9/16	Kaden (UI)
Creating replies to comments on UI	9/16	Lucas
Get voting functional on UI	9/16	Lucas
Admin page for changing users' roles	9/16	Mark
Create basic class creation/edit page	9/16	Kaden
Class creation/editing on API	9/23	Mark
Allow professors to create classes	9/23	Kaden
Queries for user roles	9/23	Ryan
Comments to be collapsed and nested	9/23	Lucas
Protect API from user's accessing or posting data they don't have access to.	9/30	Mark
Preparation for Alpha Demo	9/30	Team

Second Semester part (a)

Professor assign TAs to a class	9/30	Mark
Show correct pages and data based on user and their role on UI	9/30	Kaden, Lucas, Mark
Put discussions page into css grid system	9/30	Lucas
Dropdown to change class	9/30	Lucas
Fix Solution box to work from comments	9/30	Kaden (UI), Ryan (API)
Professor insert/remove students	10/7	Kaden
Anonymous posting for comments	10/7	Mark (UI), Ryan (API)
Search bar functionality	10/7	Lucas
User Guide Draft	10/21	Team
Instructor create categories, students use them	10/21	Kaden (UI), Ryan (API)
Refer to other posts/comments w/ link	10/21	Lucas
Add advanced filtering for front end	10/21	Kaden
Saving drafts of posts for later	10/21	Mark, Ryan
Design Doc Update 2	10/21	Team
Hook settings page up to database	10/28	Mark
Setup API for sending email notifications	10/28	Kaden, Ryan
Preparation for Beta Demo	10/28	Team
Ensure/fix UI to look good on all devices	11/4	Mark, Lucas
Allow backend to store images and files	11/11	Ryan
Upload/display images and files on UI	11/11	Lucas
Creating similar post feature w/ bubbles	11/18	Mark, Kaden
Integration with Canvas (if permission given)	11/18	Mark, Ryan
Replace login w/ U of U SSO (if permission given)	11/18	Lucas, Kaden
Format date and times throughout the app	11/25	Kaden
Loading symbols center in containers.	11/25	Lucas
Consistent fonts throughout application.	11/25	Mark
Final Design Document	12/2	Team
Final User Guide	12/2	Team
Demo Preparation	12/2	Team

Second Semester part (b)

6. Appendix

UI Sketches

Figure 1: Overall Homepage Design

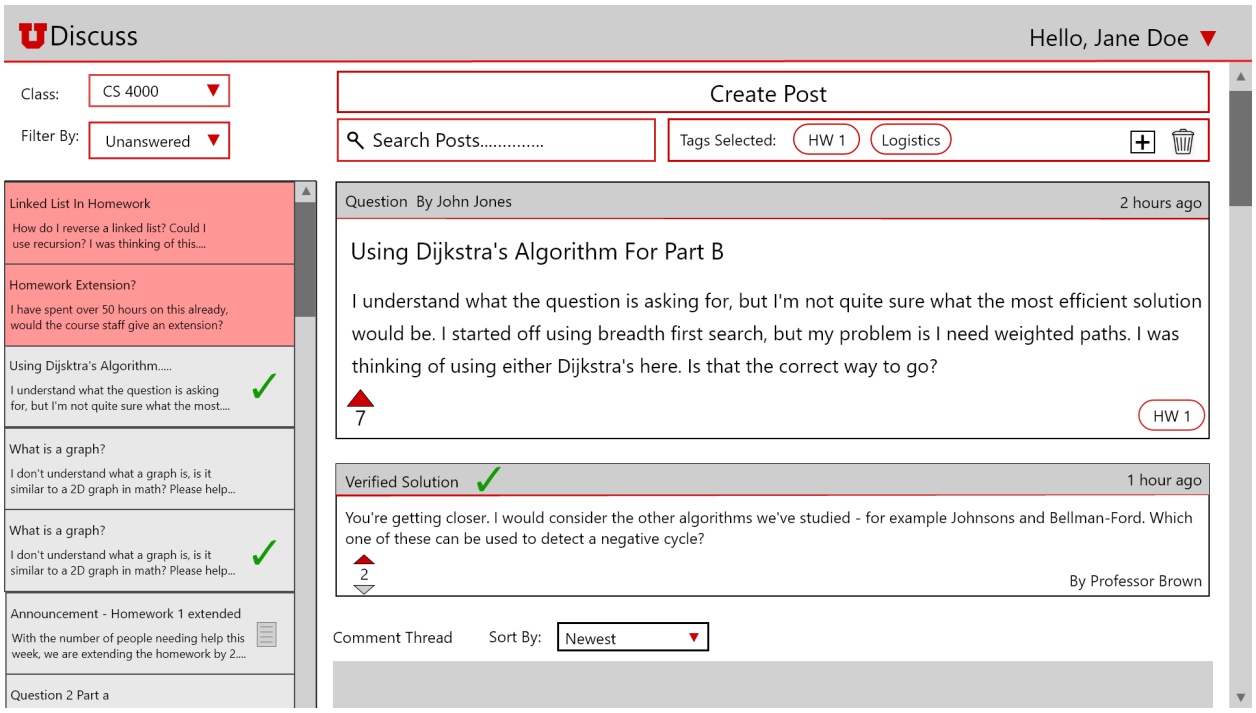


Figure 2: Settings Page

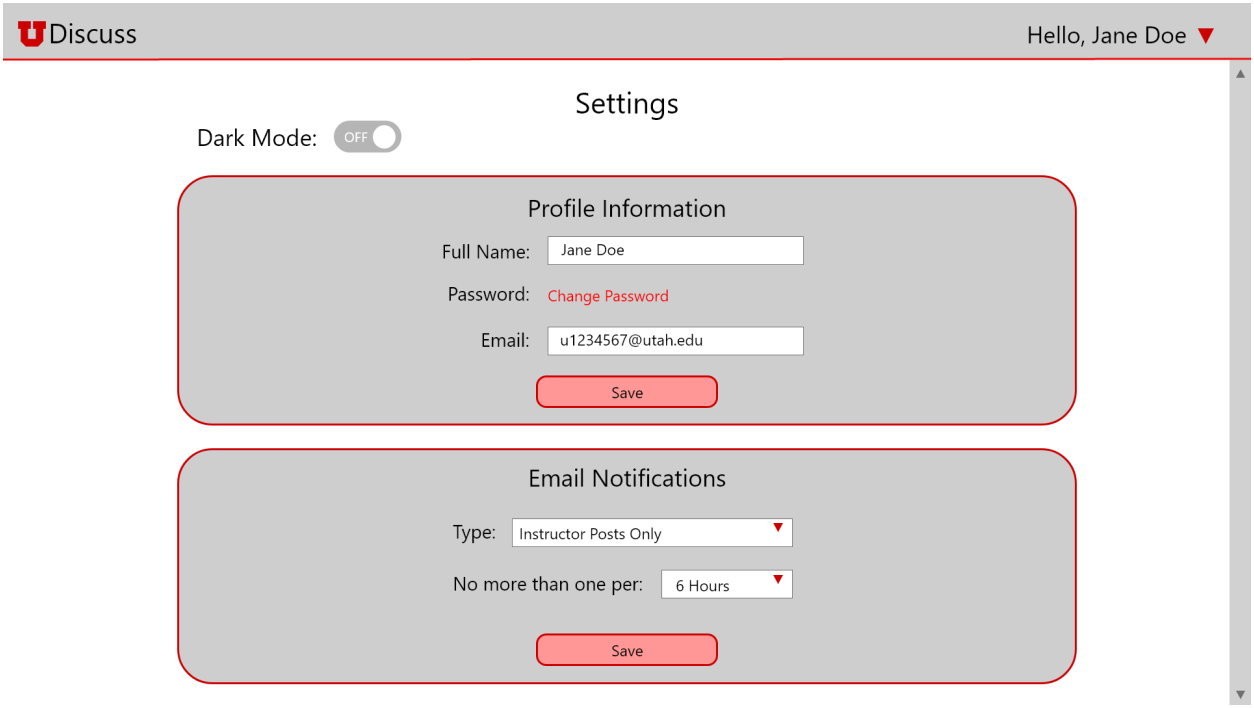


Figure 3: Creating a Post

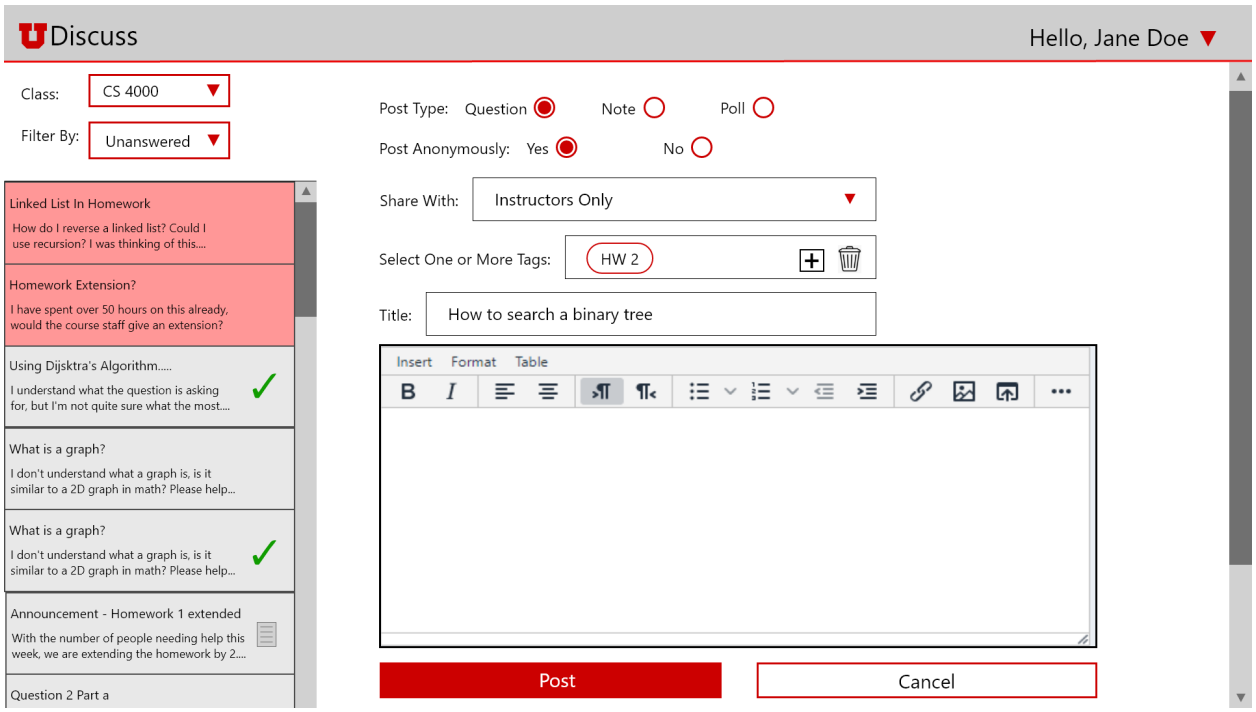


Figure 4: Comment Threads

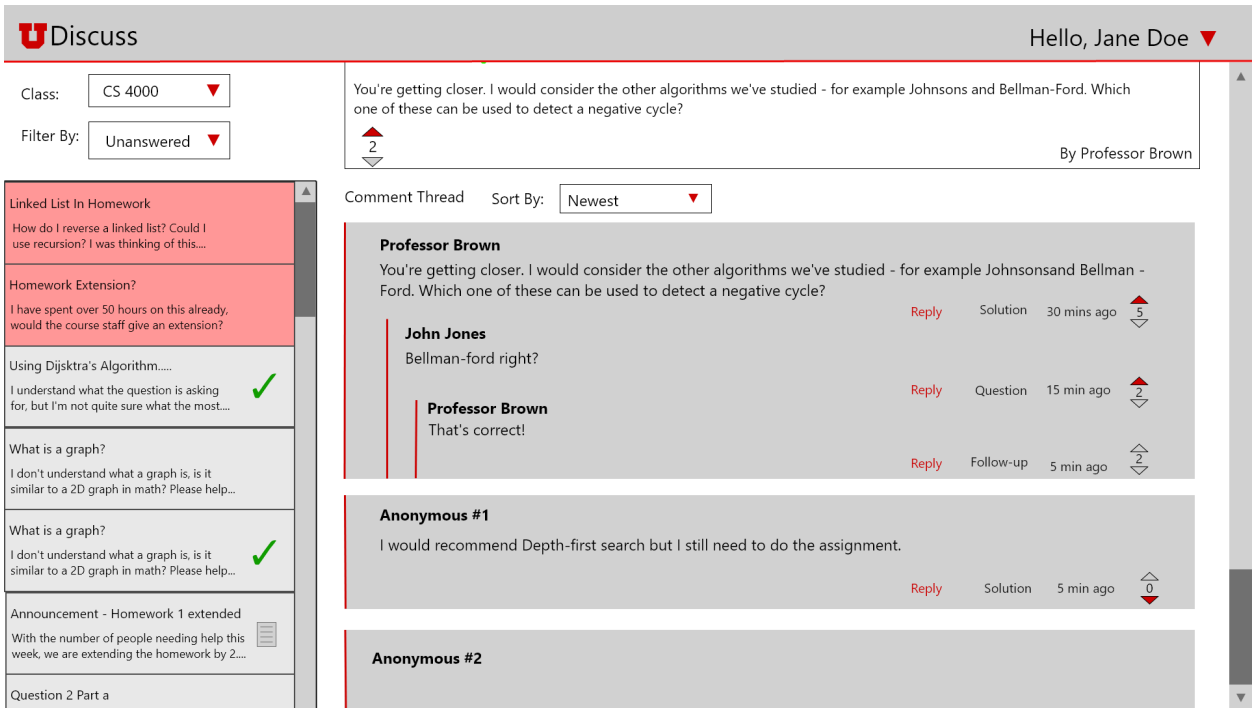
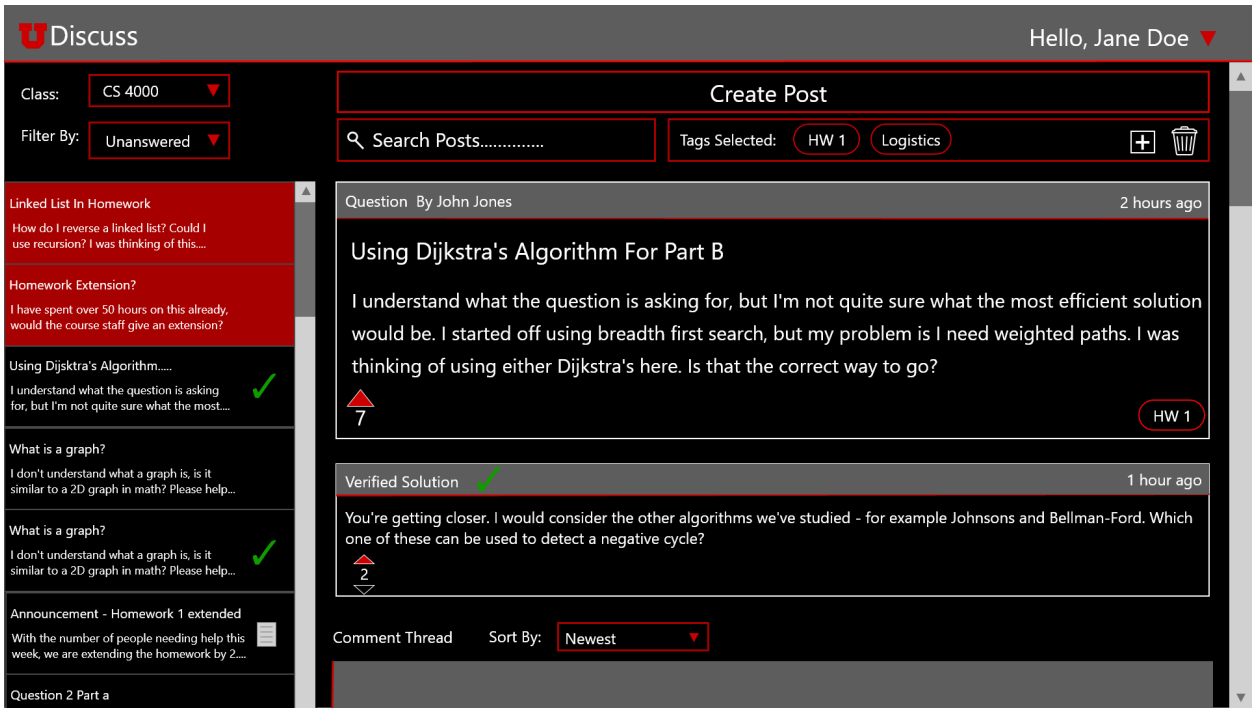


Figure 5: Dark Mode



Use Cases

Use Case #1 - Creating a Post

To create a post, a user must be logged in and on the home screen (see figure 1). First, they must select the correct class using the class dropdown on the top left. Then, they must click the “Create Post” button. This will swap out the selected post information on the right to now display the post creation information (see figure 3). The user can then select what type of post they are making, if they want to post anonymously or not, and who to share the post with. Lastly, they fill out the title and body, then click the “Post” button.

Use Case #2 - Find a Post to Help you With Homework

As in use case #1, first navigate to the home page with the correct class selected. Then, you can try to find a question similar to yours in the post list on the left of the screen. You can narrow down these posts by selecting tags (see the box right below the “Create Post” button), using the search box (the box to the left of the tags box), or by using the “Filter By” dropdown right above the post list. As mentioned in this document, you can filter/sort by new, unread, unanswered, answered, and instructor posts. You can then select the post you would like to view. As in figure 1, you’ll see the post with either a verified or unverified solution below it (in the case of an unverified solution, the top-voted solution comment is used). You can also scroll down to read the comments of the post (see figure 4).

Use Case #3 - Help Classmates by Answering Questions

To answer your classmates' questions, you again need to be on the home page (navigate there using the dropdown on the top right if you aren’t there already). If you would like to, you can use the filter on the top left to sort by unanswered questions. Otherwise, you can leave the filter on new or whatever it was on before. Once you find a post you would like to answer, you can scroll down to the comments and click “reply” on any comments to reply to a followup question or you can scroll to the very bottom and click the reply box (not pictured in sketches). When writing a comment, you simply fill out the textbox and mark the post as a solution rather than a followup question. Once you click post, your suggested solution is complete!

Use Case #4 - Editing Notification Settings

Let’s say that a user would like to make sure that they are receiving emails for only their own posts and receive no more than one per day. In this case, the user will navigate to the settings page by using the top right dropdown menu (see any of the figures above) and navigate to the “Settings” tab (now on figure 2). Once there, the user can change their display name, password, and email for notifications as desired and click the “Save”

button. They can then change their email notification settings by clicking the dropdown for type and selecting “My Posts Only” and then for “No More Than One Per” selecting the “Day” option. Clicking save in this notifications box completes the process.

Use Case #5 - Changing to Dark Mode

Similar to as in use case #2, the user must navigate to the settings page. They can then simply click the dark mode toggle to switch back and forth between light mode and dark mode (see figure 5 for our projected dark mode look).