CPS 888 - Debatr.io Design Document and Test Plan

Authors: Taimoor Farooqi, Mohit Maguluri, Saima Munir, Tabish Rashidi (Group 7)

Full repository of the source code is available using the following link:
https://github.com/trashidi98/debatr-debate-application

Problem Statement

During an era of extreme division within society, it is crucial to have a space in which people can be exposed to opposing and challenging viewpoints. All of the mainstream social media and news platforms operate in a way that causes users to be stuck in a corner of the internet that only covers a specific set of topics and viewpoints. As a result, users of these platforms become out of touch with other perspectives, may not think critically, and lose empathy. These effects lead to extreme hostility, lack of empathy and ideological or dogmatic divides among ordinary people. Thus, Debatr.io was conceived to allow users to debate and be exposed to differing viewpoints, in order to alleviate some of this division.

Debatr.io is a full stack application of a debate platform with an intuitive UX/UI. Users can sign up to create profiles, and debate other users, or witness and rate debates between other people. The app hosts multiple rooms that are assigned a debate topic and a user may choose to join the room as a debater that is either for or against the issue, or as a viewer. Further development of the app also plans to implement a point system in which users accumulate points by rating debates, participating in debates and winning debates. With enough points, users can have the ability to create rooms and assign their own topics to it. Most importantly, this platform will try to cultivate an environment in which users will be able to find debate rooms of a broad range of topics. Hence, users will be pushed out of their comfort zones and have a better opportunity to gain a more holistic understanding of the world.

Specifications

Planned Functional Requirements

1.1 The system shall allow two users with opposing views to join a virtual debate room.

1.2 The system shall allow these two users to interact and debate via a text based chat.

1.3 The system shall let the two debaters exit the room (Mutual Quit, Surrender, Exit Session)

1.4 The system shall allow users other than the debaters to join the room and view the conversation.

1.5 The system shall allow users to gain points for participating in debates as listeners and more points for actively debating other users (Mutual Quit, Surrender, Exit Session, Rating the debate)

1.6 The system shall allow users to create profiles for each individual user and can edit, create, delete and change their profile.

In terms of completeness and validity; functions required by the user/client are met. Each listener will be able to join a room through a list of available rooms on the home page. After joining, the client will be able to actively examine and judge the debater chat. After examining the debater chat, clients will then be presented with a voting option, in which they will be able to make a decision on the winner of the debate based on their own judgment.

Based on current technology capabilities, this requirement is 100% realistic. Prior to joining a debating room as a listener, the client will be shown a list of available rooms (a simple list element with the name and topic of the debate room) and be shown a button for them to press and join the room. After they have joined, a list of the debate conversation will be displayed in which the client can scroll and examine each party's arguments. After the debate has concluded, the client will be shown 2 options (clickable buttons) in which they can vote for the best candidate as a winner. The application will use 3-tier architecture i.e. frontend, backend, database, and is not overly complex, thus the app is technically viable.

This particular system requirement can and will be tested and is verifiable. This will be done by assigning a metric to each of the requirements above. The standard way to test almost all of the requirements stated above is via unit tests and manual UI tests. For example, looking at Req 1.1 in order for two users to join a debate room, the room must allow two debaters to enter and be assigned the correct roles. The unit tests would check if users can enter, and the correct roles can be assigned. Req 1.2 would test that a mock user can send a message to another mock user. However 1.6 is a bit different as it needs both a unit test to check if fields can be edited but also a UI test to see if the correct popups are showing up in the case of bad input.

In terms of consistency, it does not seem like any of the requirements are in direct contradiction of one another. In fact they are all working in-tandem to create the desired product for the target audience.

Each individual requirement's origin is traceable to a fundamental need of a particular function on our platform. For ex: functional requirement 1.4 states that users/clients require the ability to join a room as a listener. This is a key functionality that then leads to viewers being able to rate each debate on a scale which then awards points to a debater that the audience perceived as the winner. Another example includes functional requirement 1.2 which states that 2 users should be allowed to join a room and begin a text-based discussion on a particular topic. This requirement is the main premise for our project and thus can also be used to trace the origin of the project.

Each individual requirement was designed and outlined with the intention of being concise and to-the-point. System functional requirements are meant to also have comprehensive structures which are outlined in the Use Cases section further in this document.

Implemented and In Development Functional Requirements

1.1 The system shall allow two users with opposing views to join a virtual debate room - **IMPLEMENTED**

1.2 The system shall allow these two users to interact and debate via a text based chat. - **IMPLEMENTED**

1.3 The system shall let the two debaters exit the room (Mutual Quit, Surrender, Exit Session) - **IN DEVELOPMENT**

Currently, if a debater closes their browser, they will exit the session and will need to re-login to the application in order to access the previous message communications. A mutual quit and surrender options are in development as they require further elements and logic to be implemented, i.e if one debater exits/surrenders the session, the opponent will have to be notified via chat or web notification that they have won the debate.

1.4 The system shall allow users other than the debaters to join the room and view the conversation. - **IMPLEMENTED**

1.5 The system shall allow users to gain points for participating in debates as listeners and more points for actively debating other users (Mutual Quit, Surrender, Exit Session, Rating the debate) - **IN DEVELOPMENT**

Users are able to join the debate session, but are unable to accumulate points as the point counter has not been integrated into the source code. A point scheme has been developed but not implemented, with participants gaining more points as they are actively hosting the discussion and participants are also able to gain points when they vote on the winner of the debate at the conclusion.

1.6 The system shall allow users to create profiles for each individual user and can edit, create, delete and change their profile. - **PARTIALLY IMPLEMENTED**

Users are able to create profiles with a custom username, profile picture and profile information. However, users can only change their profile picture after creating an account, but not their username. In order to change the profile username, they will need to create a new profile with the new username.

Non-functional Requirements

2.1 Product Requirement - The system shall implement a seamless UI/UX to ensure an intuitive and user-friendly experience with the purpose of minimizing user mistakes or errors when navigating through the platform.

2.2 Organizational Requirement - The system shall be built with writing readable code, utilizing version control software (Git) to preserve the history of previous versions and using Agile project management tools such as Trello.

2.3 Ethical Requirement - The system shall allow a broad range of topics for a diverse audience, and is not restricted to any geographical locale or people (i.e. countries in the first world, people of a certain socio-economic class)

2.4 Security Requirement - The system shall allow all data to be secured and not allow third parties to access the data.

When it comes to non-functional requirements. they must be consistent, complete and realistic. For example, the essential premise of UX Requirements is consistency and completeness. In this environment, it is provided by a functional and user-friendly design. When a user needs to find a new solution to tackle a similar problem each time while working on a design, they will become both confused and frustrated. Because the consumer is comfortable with the offered experience, consistency helps to shorten the learning time for a product. Moreover, it's important to be realistic when it comes to Ethical requirements with users to maintain a strong experience, respecting your users' time by providing just the most important notifications when they're needed for example. It's also critical that you make it simple for people to change their notification options in this case.

All of the non functional requirements also meet the standard of being valid, verifiable, traceable, and comprehensible. In terms of validity, the ethical requirement meets this criterion as the purpose of this platform is to be a place where users can challenge themselves and step outside of their comfort zones. Most social media platforms recommend content to users that is similar to what they already view and thus, users only get exposed to certain topics and viewpoints. The ethical requirement is also verifiable as it can be tested easily by analyzing the database of debate topics to evaluate the broadness of the range of topics. For being traceable, the organizational requirement meets this need as organizational tools such as Git and Trello can allow the earlier versions of work to be tracked and traced. Finally, in terms of comprehensibility, the seamlessness of the UX/UI is further elaborated in the Use Cases section of the report.

It is also important to keep effort and implementation for requirements in mind. In terms of Ethical requirements, good policy conveys expectations and needs, gets buy-in from all levels of management, and develops benchmarks to assess if ethics are being kept and delivering the desired benefits. Help and documentation are available so users should be able to manage. Documentation can be required, depending on what type of solution.

Implemented and In Development Non-Functional Requirements

2.1 Product Requirement - The system shall implement a seamless UI/UX to ensure an intuitive and user-friendly experience with the purpose of minimizing user mistakes or errors when navigating through the platform. - **IMPLEMENTED**

2.2 Organizational Requirement - The system shall be built with writing readable code, utilizing version control software (Git) to preserve the history of previous versions and using Agile project management tools such as Trello. - **IMPLEMENTED**

2.3 Ethical Requirement - The system shall allow a broad range of topics for a diverse audience, and is not restricted to any geographical locale or people (i.e. countries in the first world, people of a certain socio-economic class). - **IMPLEMENTED**

2.4 Security Requirement - The system shall allow all data to be secured and not allow third parties to access the data. - **PARTIALLY IMPLEMENTED**

Most apps will have some security vulnerabilities. Most of the data in the app is secure, especially the data being pushed to and processed by ChatEngine.io. They have a strong commitment to security and will continually monitor their software for any vulnerabilities, more information can be found at https://chatengine.io/security. However, third party access to data was not specified in ChatEngine.io's Privacy Policy or Terms of Service, we cannot guarantee that they are not storing and analyzing user messages. For more information see the links below. https://chat-engine-assets.s3.amazonaws.com/legal/Chat-Engine-Privacy-Policy.pdf
https://chat-engine-assets.s3.amazonaws.com/legal/Chat-Engine-Terms-and-Conditions.pdf

Architecture diagram

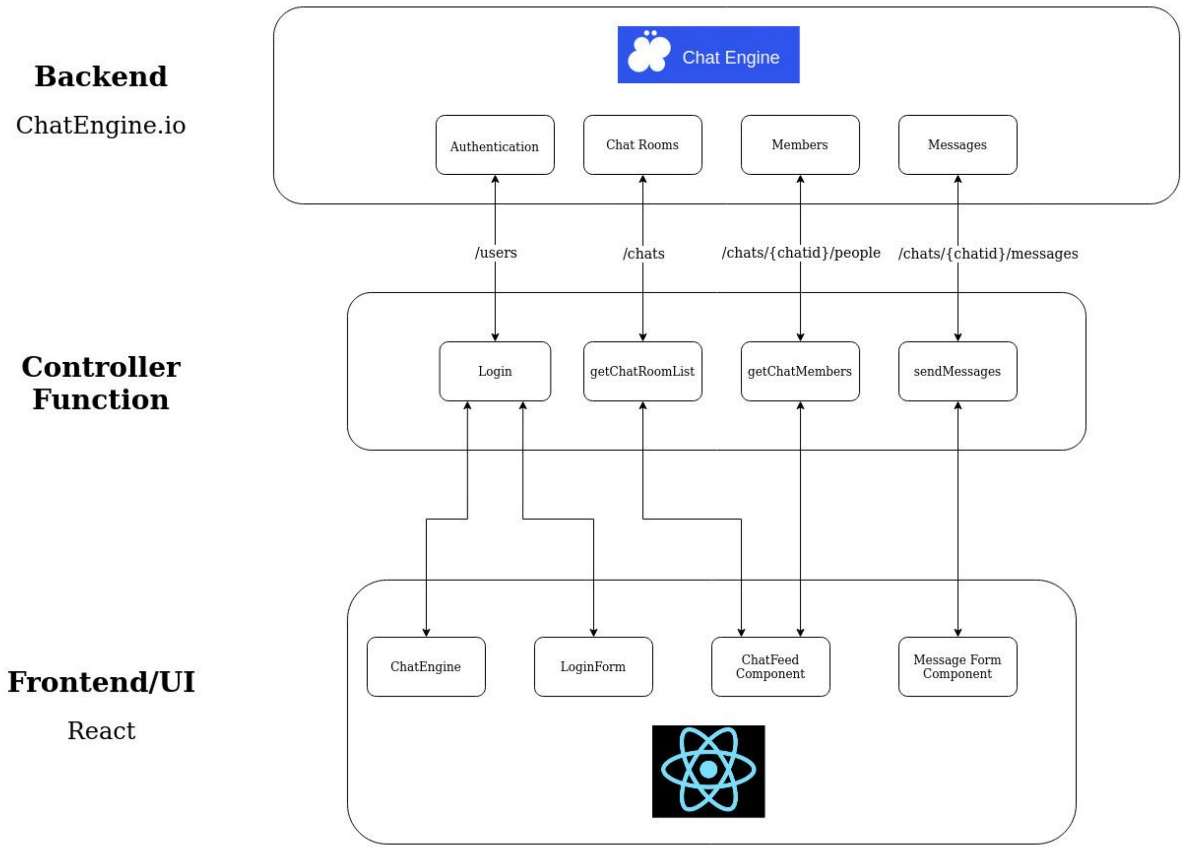


Figure 1: Architecture Diagram

The diagram above shows the overall architecture of the app. Although architecture diagrams can be rather informal, we have chosen this particular layout because it emphasizes the API functionality of the application, allowing readers to see the clear connection between frontend actions and backend functionality.

The **Backend** is handled by the third party chat engine called ChatEngine.io. It handles the authentication of users, and storage of messages and chat rooms. This level of abstraction allows for easier and faster development experience. The second part of the app is the set of **Controller Functions** which acts like the glue between the frontend and the backend. Specifically, it processes the input from the user, and forms API calls which are then sent to the backend. Finally, the **Frontend/UI** handles the rendering of the application, what the user sees, and the layout of the application. This is done by **ReactJS** which simplifies the process of using HTML, CSS and Javascript by providing a framework for frontend development.

Use Case Diagram

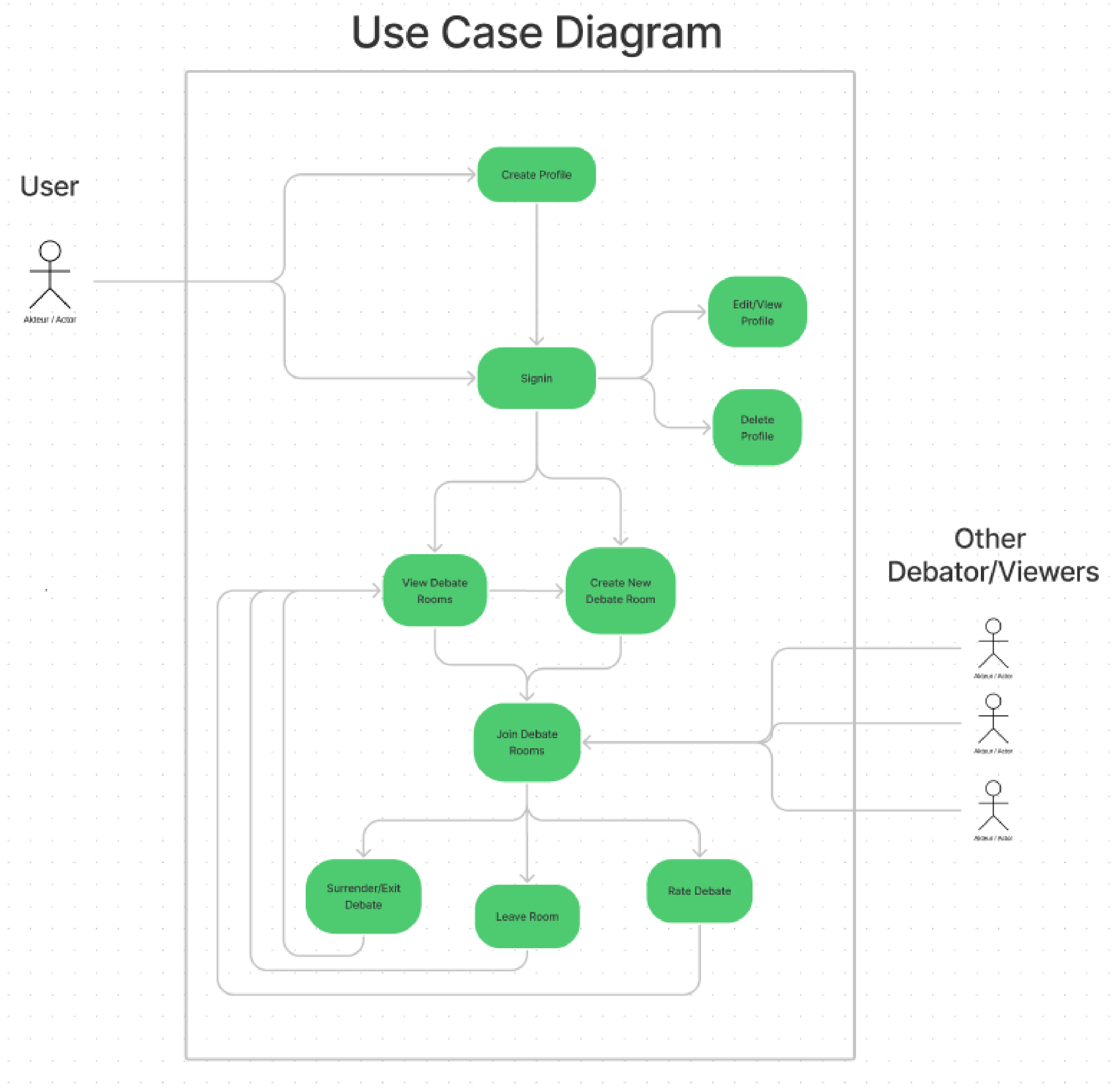


Figure 2: Use Case Diagram

In **Figure 2,** the use case diagram shows the final web application's functionalities. A user will be able to create a new profile or log in with an existing profile if they choose to do so. Next, if the user decides they would like to edit their profile, they will be able to navigate to the "edit profile" section and make their desired changes. In the case where the user wishes to begin debating, they will be able to view a full list of all existing and joinable debates and be able to join them either as participants or spectators.

If the participants of the debate choose to surrender the debate simultaneously, the debate will conclude and all participants and spectators will be returned to the home page where they can choose another debate room to join.

Sequence Diagrams

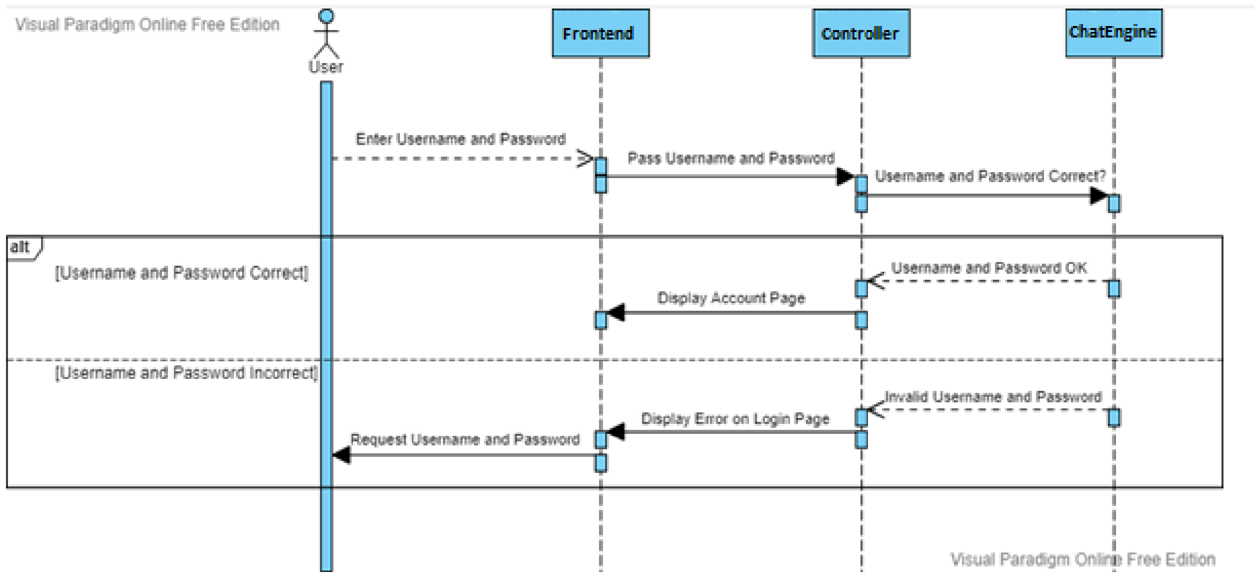

Visual Paradigm Online Free Edition
User
Frontend
Controller
ChatEngine
Enter Username and Password
Pass Username and Password
Username and Password Correct?
alt
[Username and Password Correct]
Username and Password OK
Display Account Page
[Username and Password Incorrect]
Invalid Username and Password
Display Error on Login Page
Request Username and Password
Visual Paradigm Online Free Edition

Figure 3: Login Sequence Diagram

In **Figure 3**, we can see the login sequence diagram which describes the authentication steps involved when a user attempts to log in to the Debatr.io application. This particular sequence diagram involves a single user and 3 unique sequences: Frontend, Controller and ChatEngine. The frontend is responsible for displaying relevant information to the user which includes feedback on the validity of the login credentials. The controller is responsible for all background processes that the user is not able to see. The controller will be able to cross-check the login credentials by searching the ChatEngine and sharing the feedback with the frontend.

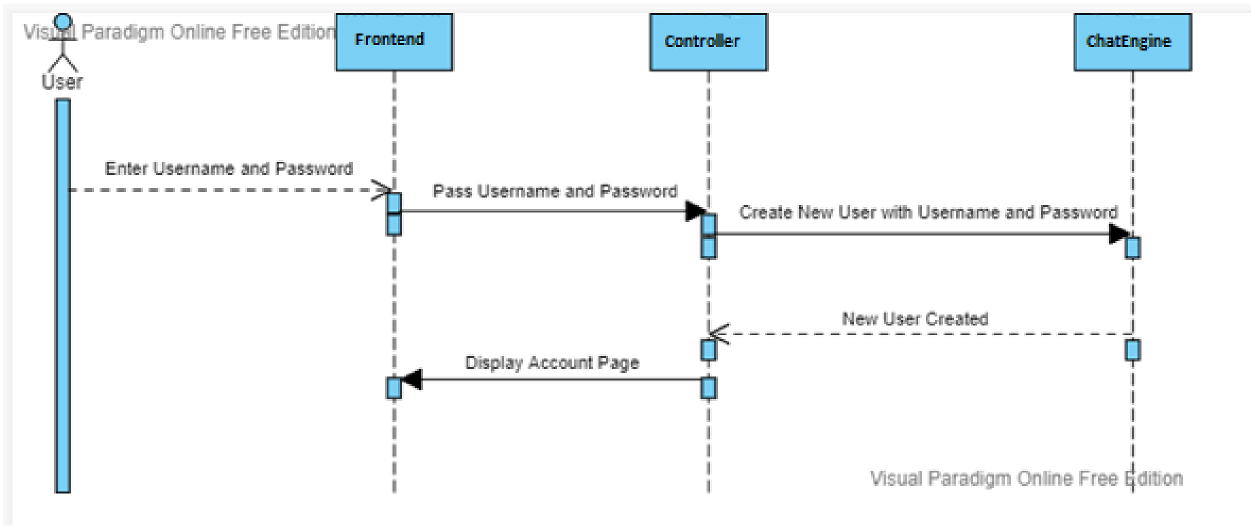

Visual Paradigm Online Free Edition
User
Frontend
Controller
ChatEngine
Enter Username and Password
Pass Username and Password
Create New User with Username and Password
New User Created
Display Account Page
Visual Paradigm Online Free Edition

Figure 4: Register Sequence Diagram

In **Figure 4**, we can see the User registration sequence diagram which describes the registration steps involved when a user attempts to register a new account in the Debatr.io application. This particular sequence diagram also involves a single user and 3 unique sequences: Frontend, Controller and ChatEngine. The frontend is responsible for displaying relevant information to the user which includes feedback about the status of the registration and if it was successful or not. The controller is responsible for all background processes that the user is not able to see. The controller will be able to create a new user entry in the ChatEngine and notify the frontend to display the account information after a successful registration.

Test Cases

Feature	Test Case	Description	Input	Expected	Result
Login	User Login	Displaying the user login fields	none	Render HTML	Render HTML
	Enter Credentials	User is able to enter their login credentials	Username, Password	Saved to page	Saved to page
	Successful Login	System successfully identifies both a correct and incorrect login	Button Click 'Login'	Response = 200 Status	Response = 200 Status
Room Creation	Users can create rooms	Create debate rooms, with specifying room name and the debate topic	Click '+' button	Response = 201 Status	Response = 201
	Users can delete rooms	Delete debate room after debate has concluded and/or when participants have surrendered	Click 'Delete' Icon	Response = 200 Status	Response = 200 Status
	List of available rooms	List of all joinable rooms is displayed after logging into the application	none	Render HTML	Render HTML
	Users can add other users	User will be able to add other other users to the current room	Search for username	Response = 201 Status	Response = 201 Status
Interaction during debates	Users are able to type and send text during the debate	Participants are able to communicate using text during a debate between one another.	none	Response = 201 Status	Response = 201 Status
	Users can send images in the debate chat	Participants are able to select and choose an image off of their local device and share it during a debate.	none	Response = 201 Status	Response = 201 Status
	Previously sent text/image messages are visible	During a debate, participants and listeners will be able to scroll up and previous communications will be visible.	none	Render image HTML in chat	Render image HTML in chat

Table 1: Test case breakdown

Unit Tests

The primary goal of unit testing is to guarantee that each individual component functions properly and as intended. Only if the different parts are in good operating order, can the system function properly.

To be clear, the table above is a proposed test plan for Debatr.io and the unit tests have not actually been implemented. That being said, the table above does outline the *core* functionality that is expected from our app. This core functionality was tested by the developers manually. The developers made sure that the correct elements were rendered on the pages, and the correct data was passed between components by adding in logging to the console. Furthermore, the API requests were verified by looking at the “body” and “headers” of each API call in the browser network analyzer.

An example of this manual testing is how the login functionality was tested. First the user details being entered into the form were saved to variables ‘username’ and ‘password’ in the backend. These variables were then logged to check that they were being populated correctly. Then the user details are passed to the Login controller, which then makes an POST API call to the /users endpoint to create the new user. We checked that the correct “body” and “headers” were attached to the request and also that a 200 HTTP response was returned.

Having these unit tests for application functionality (with good coverage) ensures ingress and egressive data objects are well formed and error free. This is useful during development and production since any errors introduced by new code are likely to be identified early by unit testing.

Design and Implement Challenges

During our requirements phase, we spent time thoroughly understanding the user requirements. Every stage of the project was described using all of the requirements, including prices, assumptions, risks, constraints, metrics, and completion timeframes. This was supported by a design stage, in which design specifications were generated and reviewed based on our set criteria for how the end project should look, as well as actionable activities that were required to get there.

After that, we went through the initial preparation phase, which included setting up our Git repository, establishing our environments, and breaking up the implementation tasks. We also consistently progressed through both frontend and backend by adding functionalities functionalities

In terms of developing specific functionality, the proposed strategy corresponds well with the order in which the functions are implemented. We began by implementing user login/registration, then moved on to chat room filtering and extra services, and finally to the subject's functionality. Finalize Phase 1 Deliverables -> Detailed Design -> GitHub and Environment -> Implementation Tasks -> Implement Login and Other Functionalities -> Testing -> Demo and Report is the project's roadmap. Our project also aligned with the proposal plan in this way because we used the waterfall approach during our execution. In terms of activity ordering and proportional time spent on each task, we stuck relatively close to the original planning phase model.

Implementation Details

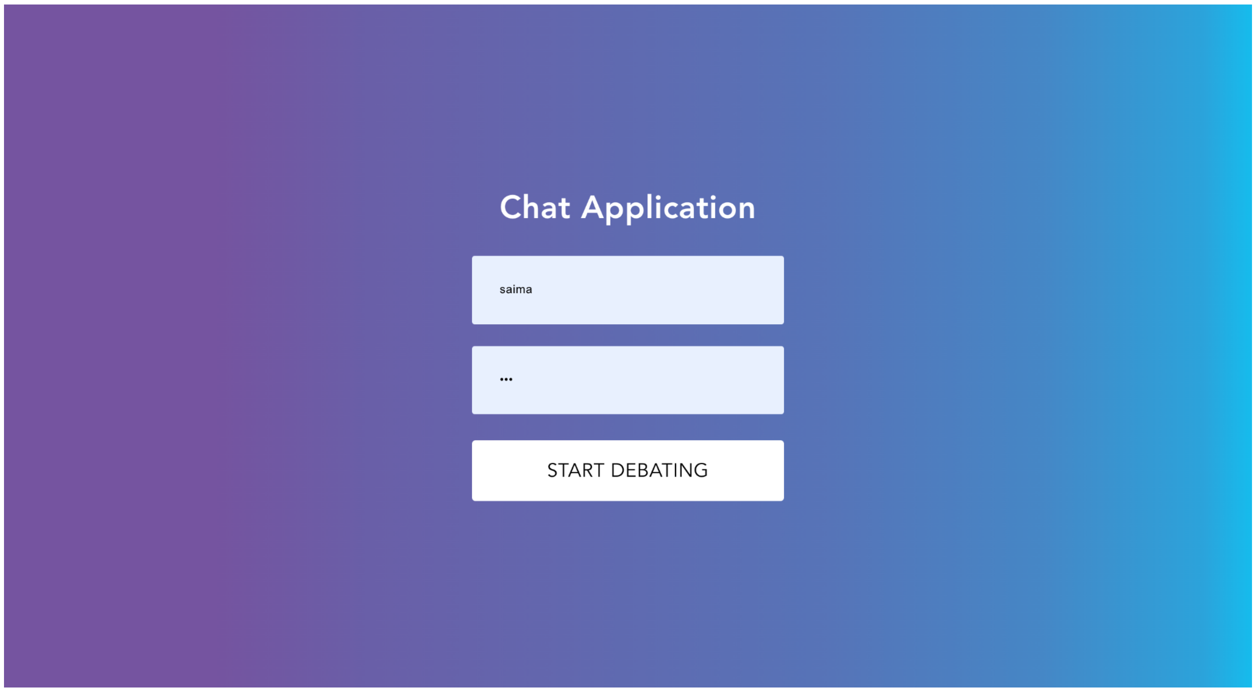
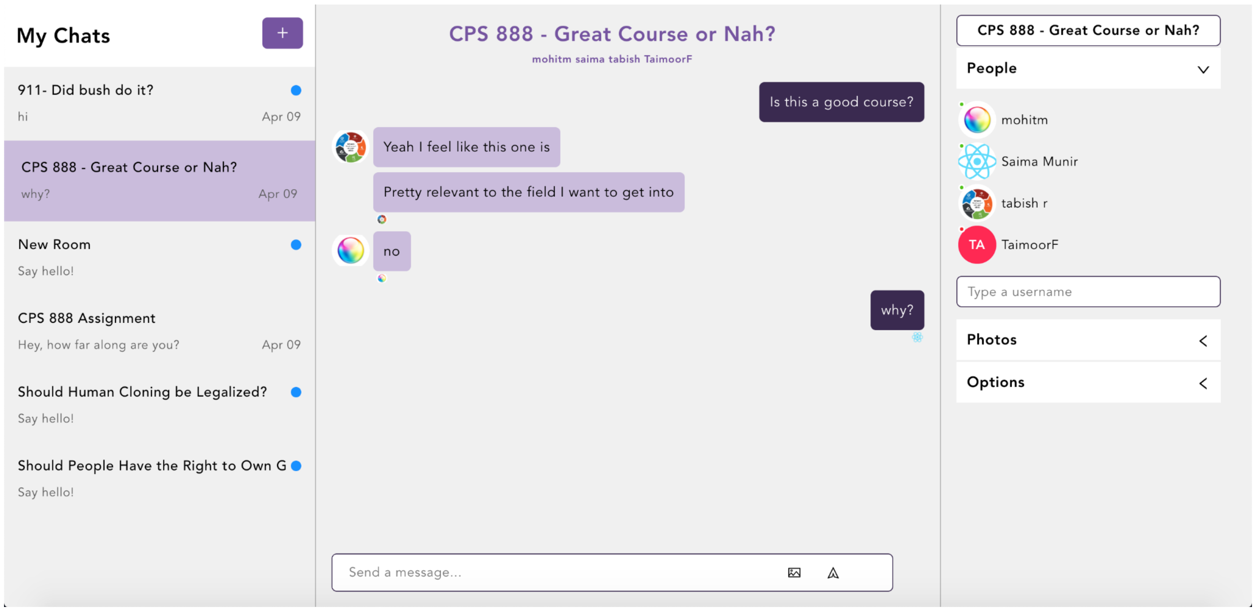


Figure 5: Login Page

Going through the flow of the application, we can start at the **LoginForm** component, this is the form shown in **Figure 5** that allows users to sign up to the app. The user details are passed to the Login controller which then makes an POST API call to the /users endpoint (seen in **Figure 1** arrow going towards Authentication in Backend) to create the new user. Now the user enters the app, but the components have not rendered yet, at least for a few milliseconds.

Next, the **ChatEngine** (**Figure 1** in the Frontend) component provides context to the application by essentially creating a connection between ChatEngine.io and the app. It pulls essential data like the projectId, current user information and other metadata needed to render the application. This metadata is then passed to the **ChatFeed component** (also in Frontend)**.**


My Chats

911- Did bush do it?
hi Apr 09

CPS 888 - Great Course or Nah?
why? Apr 09

New Room
Say hello!

CPS 888 Assignment
Hey, how far along are you? Apr 09

Should Human Cloning be Legalized?
Say hello!

Should People Have the Right to Own G
Say hello!

CPS 888 - Great Course or Nah?
mohitm saima tabish TaimoorF

Is this a good course?

Yeah I feel like this one is

Pretty relevant to the field I want to get into

no

why?

CPS 888 - Great Course or Nah?

Figure 6: Main Page (Includes chat feed, messages that show read receipts and shows which users are online/offline)

The ChatFeed component then gets the most important data to the user, namely the rooms that the user is in and the associated messages in those rooms by calling getChatRoomList and getChatMembers respectively. In the getChatMembers call in **Figure 1** (arrow pointing to Members) we can see that there is an extra parameter {chatid} that needs to be included in the body of the API call so that the correct members are returned from the room. Once those API calls return the relevant information, the app is then rendered to the user. They are shown the ChatFeed component, shown in **Figure 6**, where they can see different rooms on the left, current chat in the center, and chat details like media and chat members on the right.

Finally the **MessageForm** component (**Figure 1** in the Frontend) allows the user to type in their message, which passes this to the controller, and makes a POST request to /chats/{chatid}/messages/ for each message that the user types. This stores the messages and allows other users to pull down those messages and render them in their own UI.

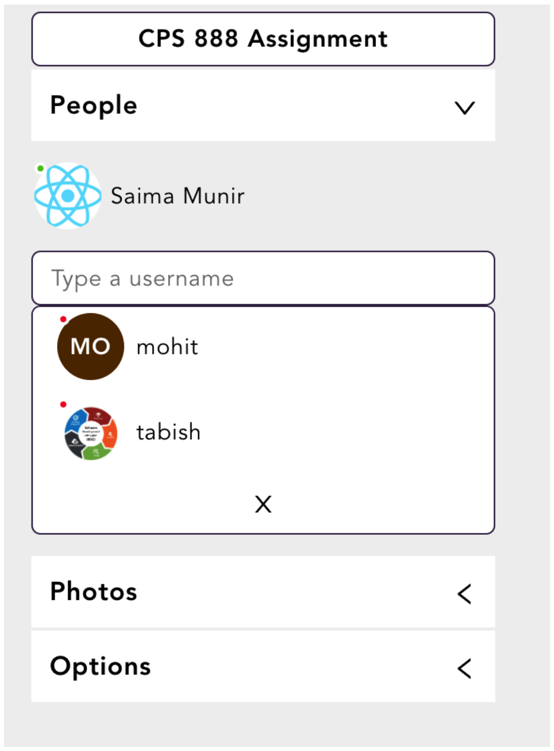


Figure 7: Add a User to a Chat

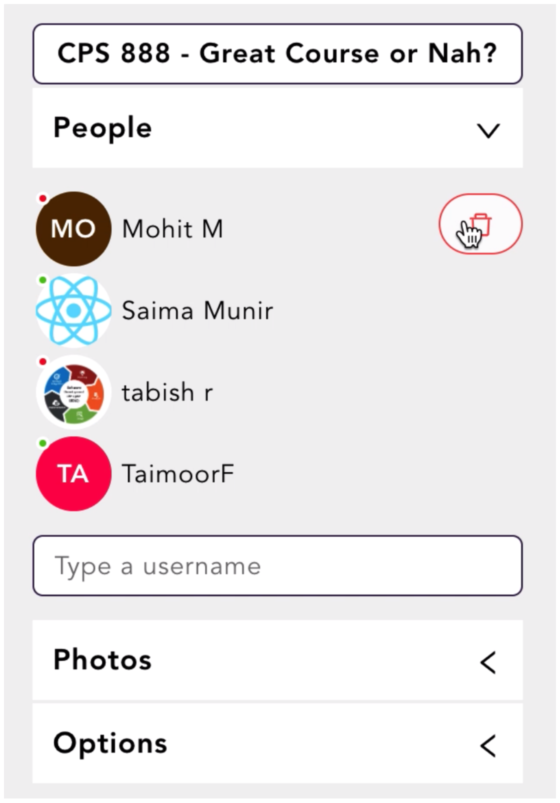
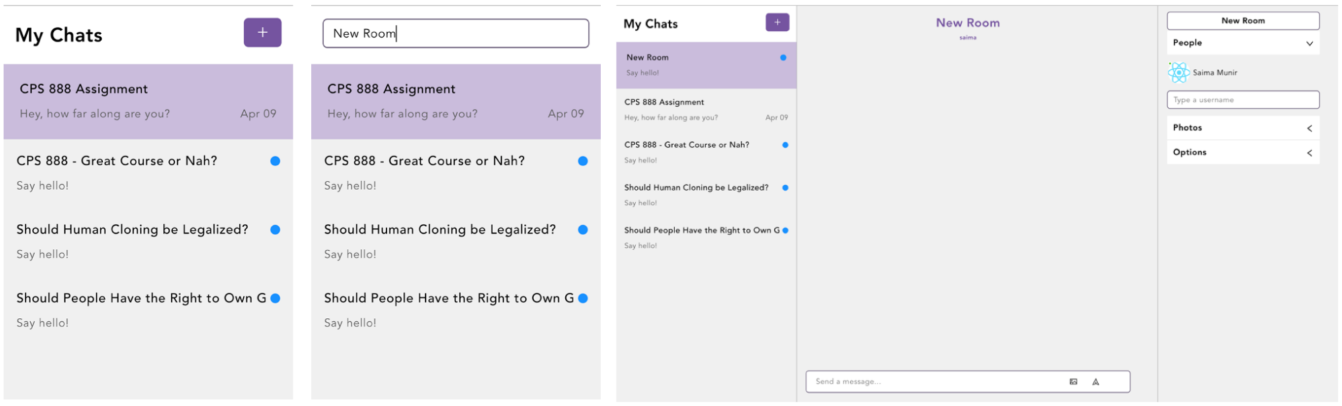


Figure 8: Delete a User from a Chat



My Chats

New Room

CPS 888 Assignment	
Hey, how far along are you?	Apr 09
CPS 888 - Great Course or Nah?	●
Say hello!	
Should Human Cloning be Legalized?	●
Say hello!	
Should People Have the Right to Own G	●
Say hello!	

Figure 9: Create a New Room

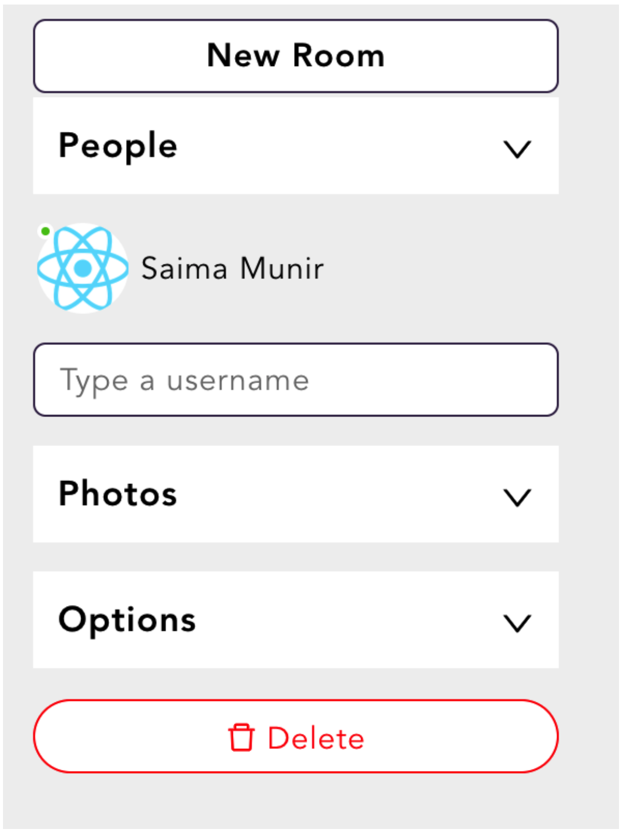


Figure 10: Delete a Room

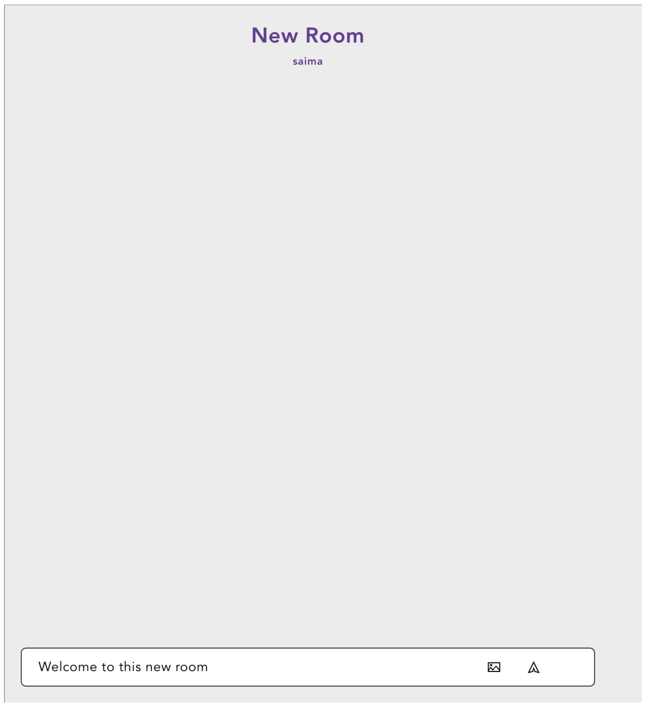


Figure 11: Sending a Message

Conclusion and Challenges

To conclude, Debatr.io is a successful full stack application as envisioned with an intuitive UX/UI. Users can register to make profiles, debate other users, or watch and rate other people's debates. The software has many rooms with a discussion topic allocated to them, and users can join the room as either a debater for or against the issue, or as a watcher. The app's future development goals include implementing a point system, in which users earn points for rating debates, participating in debates, and winning debates. Users who have earned enough points can build rooms and attach their own subjects to them. Most significantly, this platform will strive to create an environment where users may locate debate rooms on a variety of issues. We believe this will push our users out of their comfort zones and have an opportunity to participate on a platform to expand their knowledge also while being empathetic to a variety of topics.

Some challenges that occurred while working on the project were through waterfall development methodology. When a particular stage of development has a delay, all the other stages are delayed as well. This issue can be fixed in the future through an agile approach where it follows an iterative process. The waterfall process also limits the potential to get user feedback through a midway stage, leading to changes until all the way till end.

Another challenge we faced was making axios API requests to the backend. In order to automatically allow users to have access to certain chat rooms once they sign up, a POST request needs to be made to the specific chat room API with information about the user's credentials and the application credentials. However, the API calls were unsuccessful with 403 and 404 errors, meaning that the authentication credentials were not provided or the request was not found. Hence, more research into the ChatEngine API and troubleshooting is required to solve this roadblock.

Some future challenges to consider are regarding onboarding. We believe engagement and retention are heavily reliant on onboarding. It has an impact on metrics such as productivity, longevity, finances, and culture. It's critical to focus on a new user workflow and establish a solid relationship in order to acquire loyalty and devotion to the platform.