

# BoilerBoard.

## **Team 21**

- Arunav Sen Choudhuri
- Arman Kumar
- Soham Sharma
- David Cox
- Maximus Beato

## **Purpose**

A prevalent issue among Purdue University students is the lack of a centralized platform for academic discussions, questions/answers, and resource sharing. This leads to fragmented information, making it challenging for students to find relevant academic help quickly. Our project, BoilerBoard, aims to address this by creating a Purdue-specific educational website that facilitates academic discussion and resource sharing for Purdue courses. Unlike existing services, BoilerBoard will be tailored to Purdue University's curriculum and community, providing unique features to students to enhance and personalize their student experience. Students will be able to create and join modules that correspond to Purdue courses and Instructors will be able to moderate these modules. These modules will consist of student-created learning materials, Q/A sessions, discussions, and voice-chat rooms.

### **Functional Requirements:**

- 1) As a user,
  - a) I would like to create an account with a Purdue email address.
  - b) I would like to log in to my account and manage my info.
  - c) I would like to be able to reset my password if I forget it.
  - d) I would like to be able to post learning materials for a course in the form of text and images.
  - e) I would like to make my own avatar for my profile.
  - f) I would like to be able to view my profile showing my account information and past posts that I have made.
  - g) I would like to be able to make practice quizzes that other users can take.
  - h) I would like to upvote certain posts and quizzes
  - i) I would like to have a scoring system based on the number of upvotes I get on my posts and practice quizzes and have other users view my score.
  - j) I would like to be able to sort discussions by the most recent or the most upvoted.
  - k) I would like to be able to edit and delete my own posts.
  - l) I would like to navigate from one page to another easily.
  - m) I would like to receive notifications when someone responds to my posts or discussions.
  - n) I would like to be able to follow specific courses and receive updates about new posts and discussions.
  - o) I would like to be able to report inappropriate content to the moderators.
  - p) I would like to be able to bookmark certain posts or discussions for easy access.

- q) I would like to be able to provide feedback on the platform's features and report any issues or bugs I encounter.
- 2) As a student,
- a) I would like to be able to easily join and leave courses.
  - b) I would like to be able to read and interact with other users' learning materials for courses I have joined.
  - c) I would like to take quizzes other users have made and immediately get the results after submitting them.
  - d) I would like to be able to view my quiz performance history to track my learning progress.
  - e) I would like to be able to make discussions in a specific course where other students in the course can respond to my question.
  - f) I would like to be able to mark a correct answer to my discussion question.
  - g) I would like to be able to create a voice chat that I can study and chat with fellow students.
  - h) I would like to be able to view voice chat rooms in session, and request to join.
  - i) I would like to be able to have access to a virtual whiteboard during voice chat sessions to enhance our collaboration experience.
  - j) I would like to view different modules within a course to better fit what I am learning.
  - k) I would like to be able to search course materials to find information that is relevant to me.
- 3) As a Session Leader,
- a) I would like to be able to change my session between private and public.
  - b) I would like to be able to invite students to my public or private session.
  - c) I would like to be able to remove students from my session.
  - d) I would like to be able to mute and unmute students in my session.
  - e) I would like to be able to have my sessions hidden from designated blacklisted users.
  - f) I would like to be able to clear the virtual whiteboard for all users in the session.
  - g) I would like to be able to remove whiteboard permissions from certain users in my session.
  - h) I would like to be able to schedule sessions in advance and send out invitations to participants.
- 4) As an Admin,
- a) I would like to create courses and course modules.
  - b) I would like to remove other users from a course in the event of repeated misbehavior.

- c) I would like to remove posts that are reported by other users or posts that can be considered as “unsafe”.
  - d) I would like to be able to manage user roles and permissions within a course.
  - e) I would like to be able to update course information and details.
- 5) As an Instructor,
- a) I would like to be able to endorse posts and practice quizzes.
  - b) I would like to view my course’s discussion to ensure Purdue’s honor code is upheld.
  - c) I would like to remove any posts that could be considered academic dishonesty.
  - d) I would like to be able to post announcements to all students in a course
  - e) I would like to be able to provide official answers to student questions in discussions.
  - f) I would like to be able to assign badges or rewards to students based on their participation or performance.

## **Non-Functional Requirements:**

### Architecture and Performance

BoilerBoard will be structured into two main components: the front end and the back end. The backend will be written in Django which is a Python web framework. Django features rapid development capabilities, security features, and scalability which will allow our web application to be robust. Django also features robust security features out of the box, helping us to avoid common security mistakes such as SQL injection and clickjacking. We will be using PostgreSQL for our database needs. PostgreSQL is a powerful relational database system that uses the SQL language combined with many features that can safely store and scale complicated data workloads.

The frontend will be developed using React which is a JavaScript framework for building user interfaces. React allows us to create reusable UI components, significantly speeding up our development process.

The combination of Django, PostgreSQL and React will provide a robust, scalable, and high performance solution for both the frontend and backend of BoilerBoard. This tech stack will ensure that we can effectively meet the needs of our users and provide them with a reliable experience when using our web application.

With this in mind, we have some performance goals. As a developer,

- a. I would like this application to run smoothly without crashing.
- b. I would like this application to be launched in 5 seconds.

- c. I would like this application to support 20,000 users.
- d. I would like requests to be handled within 500 ms.

### Security

Having good security measures is important for BoilerBoard, as there is sensitive information tied to users' accounts. The Django framework that we will be using comes with several security features to prevent common exploits such as cross-site request forgery (CSRF) protection and SQL injection protection. We will also implement different roles for users and a permissions system to ensure users can only access functions in which they should be able to. All requests to APIs will need to be authenticated to prevent abuse.

### Usability

The User Interface (UI) will be easy to use and navigate. The extent of our features requires the website to have an easy-to-understand interface that any user can use. Similar platforms have completely separate functionality and interfaces, but since this will be a central platform, we have to emphasize usability even further than our competitors for students, instructors, and administrators. Instructors and administrators will have a similar interface to the students but with additional capabilities. We want to ensure that our interface is easy to use on all resolutions, screens, monitors, and browsers for a universal experience.

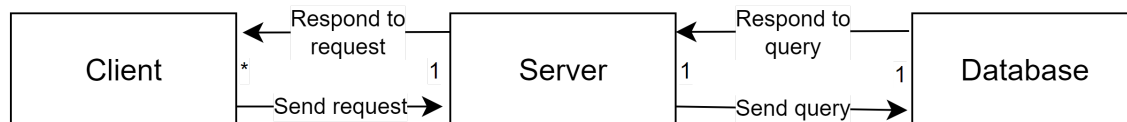
### Hosting/Deployment

Our front and back ends will be able to be deployed and updated separately. As we progress through the development process, we can deploy the back end to a DigitalOcean server to go through testing. The front end can be hosted for free on a service like Vercel.

## **Design Outline**

### **High Level Design Overview**

This system will be a web application that allows users to engage in discussions pertaining to Purdue courses and enhance their study experience. There will be forum posts and other social functions (e.g. private messages, study group creation, live voice chat sessions). This application will use the client-server model in which one server simultaneously handles access from a large number of clients using the Django framework in Python. The server will accept client requests, access / store data in the database, and provide feedback to client(s) accordingly.

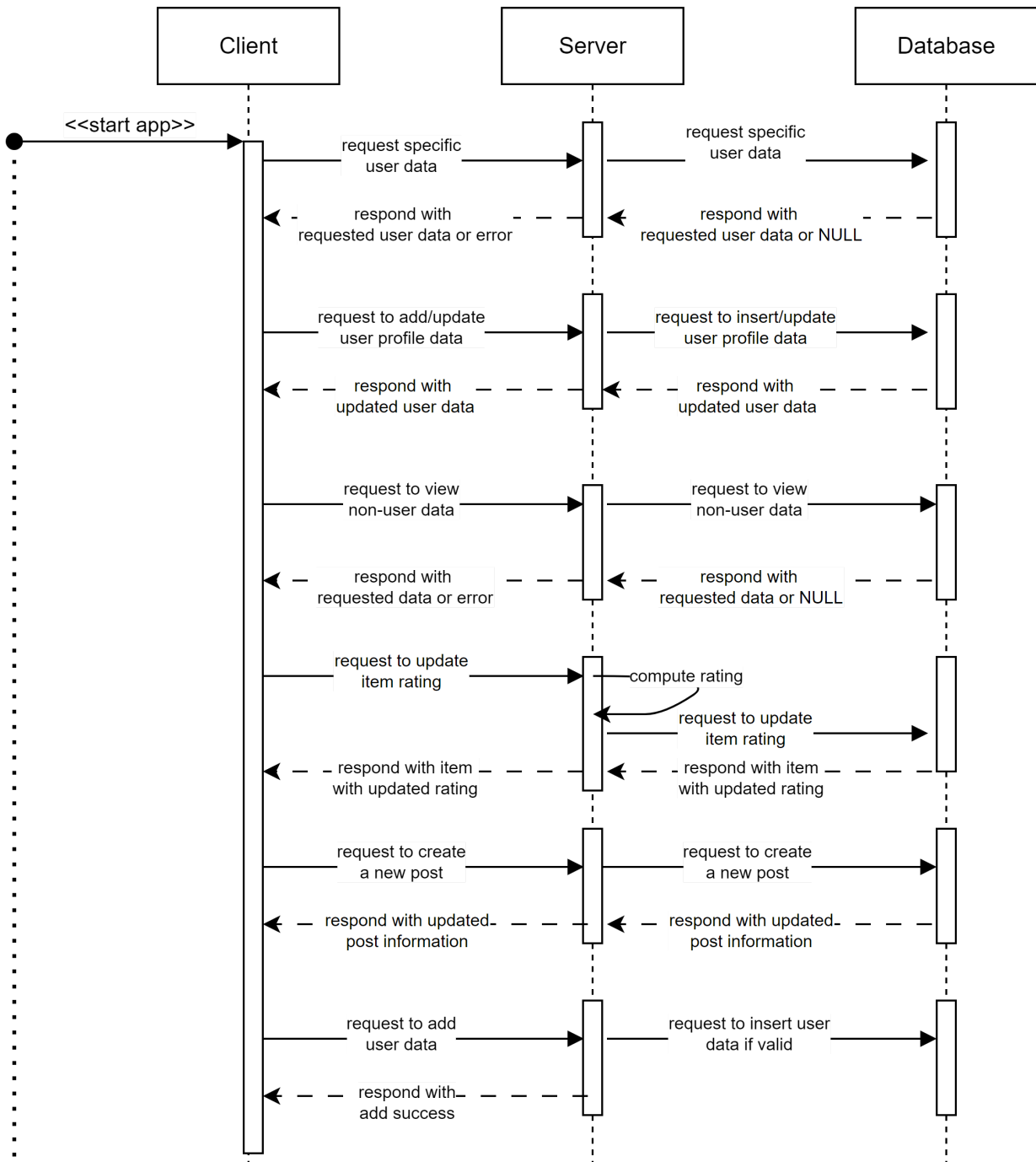


1. Client
  - a. Client provides the user an interface with our system.
  - b. Client sends ajax requests to the server.
  - c. Client receives and interprets ajax responses from the server and renders changes as necessary in the user interface
2. Server
  - a. Server receives and handles ajax requests that come from clients.
  - b. Server validates requests and sends queries to the database to add, modify, or retrieve data.
  - c. Server generates appropriate responses and sends them to targeted clients.
3. Database
  - a. A relational database stores all data utilized in the application (e.g. user information, course information, posts, replies, etc.)
  - b. Database responds to queries from the server, extracts data, and sends this data back to the server.

### **Sequence of Events Overview**

The sequence diagram below depicts the typical interactions that occur among clients, the server, and the database. The sequence is initiated by a user starting the web application. As the user logs into the system, the client sends a request to the server. The server handles the request, and then sends a query to the database and receives data from the database. After the user is logged in, the client further sends a request to the server for other actions,

such as managing the profile data, creating and updating discussions, responding to other discussions, and making quizzes. To respond to these requests, the server sends queries to the database to acquire data from the database. The database responds with updated data after receiving update requests. With the requested data from the database, the server returns the results back to the client.



## **Design Issues**

### **Functional Issues**

1. What information do we need to process in order to create an account?
  - Option 1: username and password only
  - Option 2: username, password and email address

Choice: Option 2

Justification: We wanted users to sign up using a Purdue email address to make the platform more secure. Signing up with a Purdue email means that users are easily identifiable and can not easily re-join the platform if they had to be removed for a violation. This also makes it possible for users to reset their password using their email if necessary.

2. Should we allow users to directly message other users?
  - Option 1: Allow users to directly message other users
  - Option 2: Don't allow users to directly message other users

Choice: Option 2

Justification: The primary purpose of BoilerBoard is to facilitate academic discussion and resource sharing in a community setting. Allowing direct messages could detract from this goal by encouraging private conversations which would lead to information being shared privately rather than the entire class. Not allowing private messages also eliminates the risk of users engaging in academic dishonesty.

3. Should we allow users to view each other's profiles?
  - Option 1: Allow users to view each other's profiles
  - Option 2: Allow users to view only their own profile

Choice: Option 1

Justification: We wanted to allow users to view each other's profiles as it can foster a sense of community. Users can see the other contributions of users whose posts they have liked. However we are not displaying other user's personal information such as mail to protect user privacy.

4. Should we allow all users to create quizzes?



- Option 1: Allow all users to create quizzes
- Option 2: Only allow instructors to create quizzes

Choice: Option 1

Justification: We wanted to allow all users to be able to create quizzes for a course they are enrolled in as it can lead to other students having more diverse quizzes to study with. Being able to create quizzes also encourages active learning among users as users have to engage with and learn course material to create their own quizzes

5. Should we allow users to edit their posts?
  - Option 1: Allow users to edit their posts
  - Option 2: Don't allow users to edit their posts

Choice: Option 1

Justification: Allowing users to edit their posts makes sure that their post is accurate. Users may realize that they have made a mistake or left out an important detail only after they have made a post.

## **Non-Functional Issues**

1. What database should we use?
  - Option 1: MySQL
  - Option 2: MongoDB
  - Option 3: PostgreSQL

Choice: Option 3

Justification: We chose to use a Relational database because of the nature of the data we are working with. User information, course details, posts have clear relationships that can be expressed as tables in a relational database. We then chose to use PostgreSQL over MySQL as it offers more data types and features than MySQL such as materialized views and stored procedures.

2. What backend language/framework should we use?
  - Option 1: PHP
  - Option 2: Node.js (JavaScript)
  - Option 3: Django (Python)

Choice: Option 3

Justification: The features associated with Django allow for rapid development of secure and maintainable websites. Django takes care of much of the hassle of web dev, so us as developers can focus on creating the meat and potatoes of our application without needing to reinvent the wheel. Also, Django can be used to build almost any type of website, which is optimal as we can utilize Django well to fit in our client-server-database model.

3. What frontend language/framework should we use?

- Option 1: raw HTML + JavaScript
- Option 2: React
- Option 3: Angular
- Option 4: Vue

Choice: Option 2

Justification: We chose to use React as it uses reusable UI components which improves the consistency of the application and speeds up development. React also uses a virtual DOM to optimize rendering which provides increased performance. Finally, React has a large community and a wealth of resources available which allows it to be easier to learn and use.

4. What web service should we use?

- Option 1: AWS
- Option 2: Azure
- Option 3: Google Cloud
- Option 4: Heroku

Choice: Option 1

Justification: We chose Amazon Web Services for hosting our project's backend components because the Amazon S3 can be utilized with a free tier. In addition, this package allows for secure, durable, and scalable object storage infrastructure. Amazon S3 facilitates low-latency data storage from the cloud, which will allow us to help achieve short load-times.

5. Should we use HTTP or HTTPS for our app?

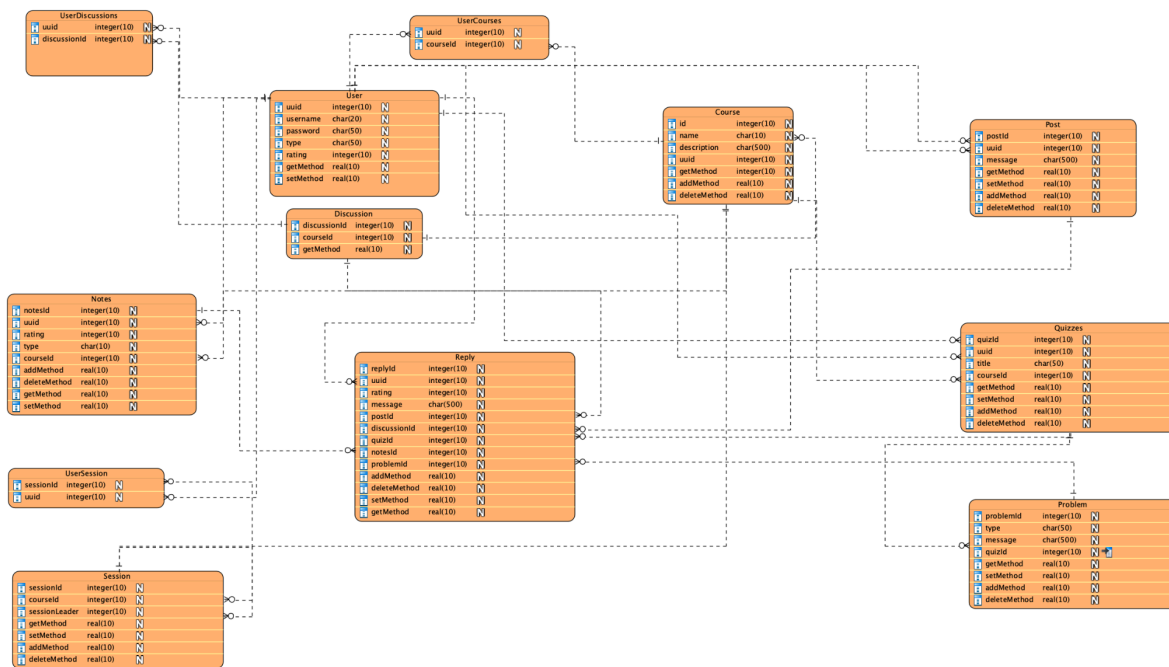
- Option 1: HTTP
- Option 2: HTTPS

Choice: Option 2

Justification: HTTP is 'HyperText Transport Protocol' and it is the way in which data gets passed back and forth between web servers and clients. HTTPS is the secure version of HTTP (hence the 'S' in HTTPS). This adds another layer of security that is ideal for our application as our system utilizes private information such as log-in information. Utilizing HTTPS will help minimize the likelihood of data theft.

## Design Details

### Class Design



#### Notes:

- The lines that end with a circle and three lines (like a triangle) denote the “many” relationship
- Zoom in for clearer resolution
- Due to the limitation of the software that we used, we could not add a separate method section, so the attributes that are denoted with the name “method” and a corresponding data type of “real(10)” are actually methods
  - Ex. getMethod real(10) is a get method for an the object

## Descriptions of Classes and Interaction between Classes

We have modeled our classes based on the objects in our projects. This is also similar to how our database will be configured, as Django utilizes an Object-Relational-Mapping (ORM) mechanism for storing data.

- **User**

- A User Object is created when someone signs up for our web application
- Each user will be assigned a unique user Id.
- Each user will have a username, password, and email for login purposes
- Each user will have a type. There are three different types: student, instructor, and admin. Each will have different functionalities and privileges given to them by their type, such as an instructor can block posts that could be considered academic dishonesty
- Each user will have a rating. The rating is part of a scoring system that encourages users to interact with courses, course discussions, and create new material on the website. It also encourages accuracy with what they post. Other users can “up-vote” and “down-vote” other user’s posts and creation.
- Each user can view their list of posts
- Each user can view their list of courses
- Each user can view their lists of practice quizzes
- Each user can view their list of replies and the corresponding post
- Each user can view their their list of discussions

- **Course**

- Course Object is created when an admin starts a course
- Each course will have a course Id and a name
- Each course will have a description
- Each course will have a list of all the users registered.
- Each course will have a list of quizzes
- Each course will have a discussion
- Each course will have a list of sessions

- **Post**

- A Post Object will be created when a user creates a post
- Each Post will have will have a postId
- Each Post will have an author. The author will be represented by the unique user Id.
- Each Post will have a message (the meat of the post)

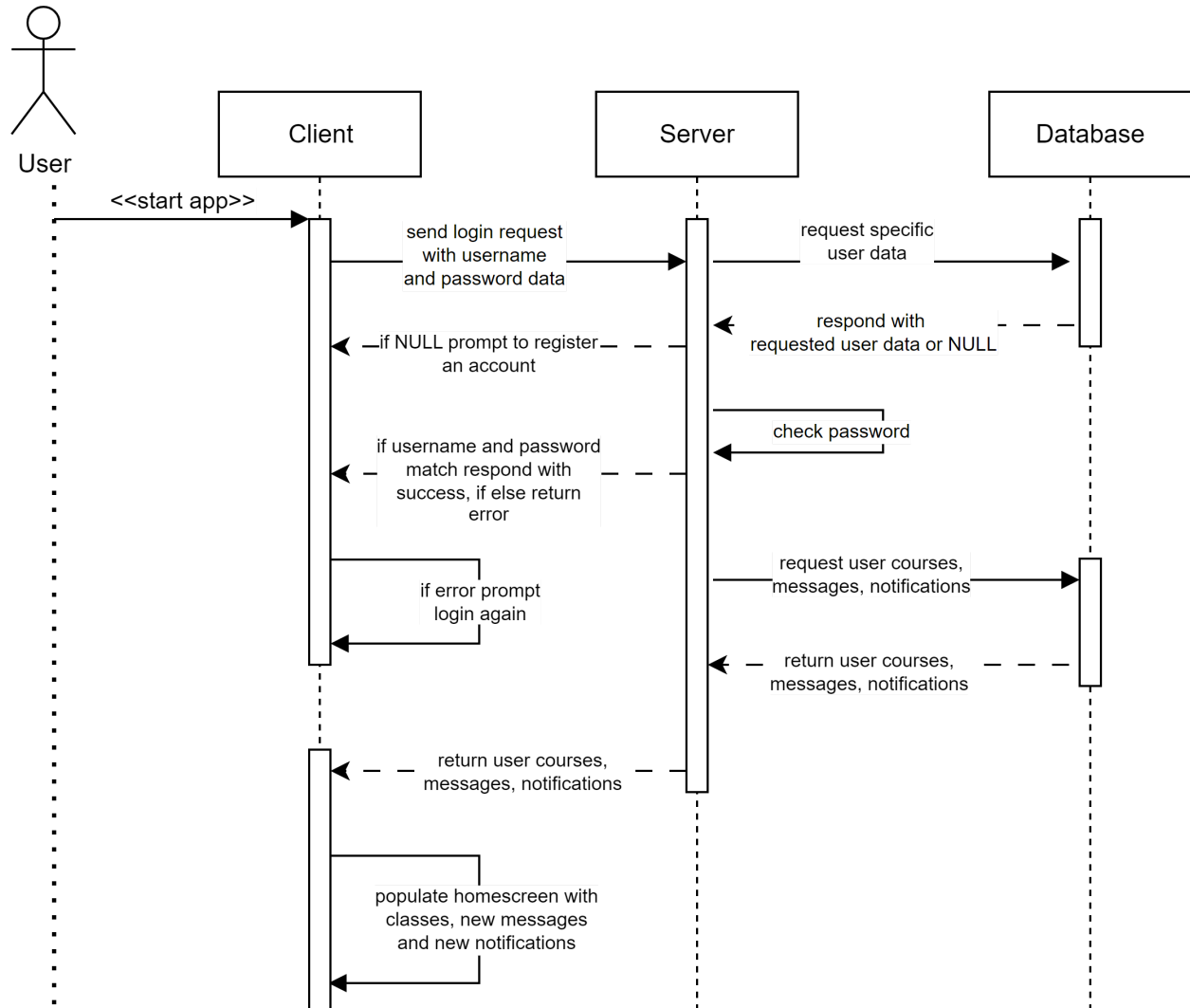
- Each Post will have a list of replies
- Each Post will have a discussion, linked together with a discussion Id
- Each Post has a course
- **Reply**
  - A Reply Object will be created when a user creates it
  - Each Reply will have an Id
  - Each Reply will have an author, linked with a unique user Id
  - Each Reply will have a message
  - Each Reply will have fields such as Id's for posts, quizzes, problems, and notes. Each of these fields are nullable, and only one of them will be filled. This is to identify what the reply will be replying to, since there are many different replies.
- **Quiz**
  - A Quiz Object will be created when a user creates a quiz
  - Each Quiz will have an Id
  - Each Quiz will have a title
  - Each Quiz will have an author linked with a unique user Id
  - Each Quiz will have a list of problems
  - Each Quiz will have a list of replies
  - Each Quiz has a course
- **Problem**
  - A Problem Object will be created when a user creates a problem
  - Each Problem will have an Id
  - Each Problem will have a type. The type refers to whether or not a problem is a free-response problem or a multiple choice problem
  - Each Problem will have an answer corresponding to the type, and put into the message category
  - Each Problem will have a message, which will include the problem statement and an answer
  - Each Problem will have an author linked with a unique user Id
  - Each Problem will have a quiz Id, which is nullable if the problem exists outside of a quiz (i.e in a discussion)
  - Each Problem has a Course
- **Session**
  - A Session Object is created when a user creates a session
  - A session refers to a voice chat call

- A Session as an Id
- A Session has a session leader, the user who created the session, which is linked with unique user Id
- A Session has a list of users that are invited, and if null, has the entire user list of the course invited (a Public voice chat)
- Each Session has a Course
- **Discussion**
  - There will be a main discussion board, and multiple smaller discussions that other users create. All discussions will be public
  - Each Discussion will have an Id
  - Each Discussions will have a course
- **Note**
  - Each Note will be created when a User creates a note
  - Each Note will have an author, linked with an unique user Id
  - Each Note will have a list of replies
  - Each Note will have a pdf or png or jpg attachment
  - Each Note will have a course, linked with a Course Id

## Sequence Diagram

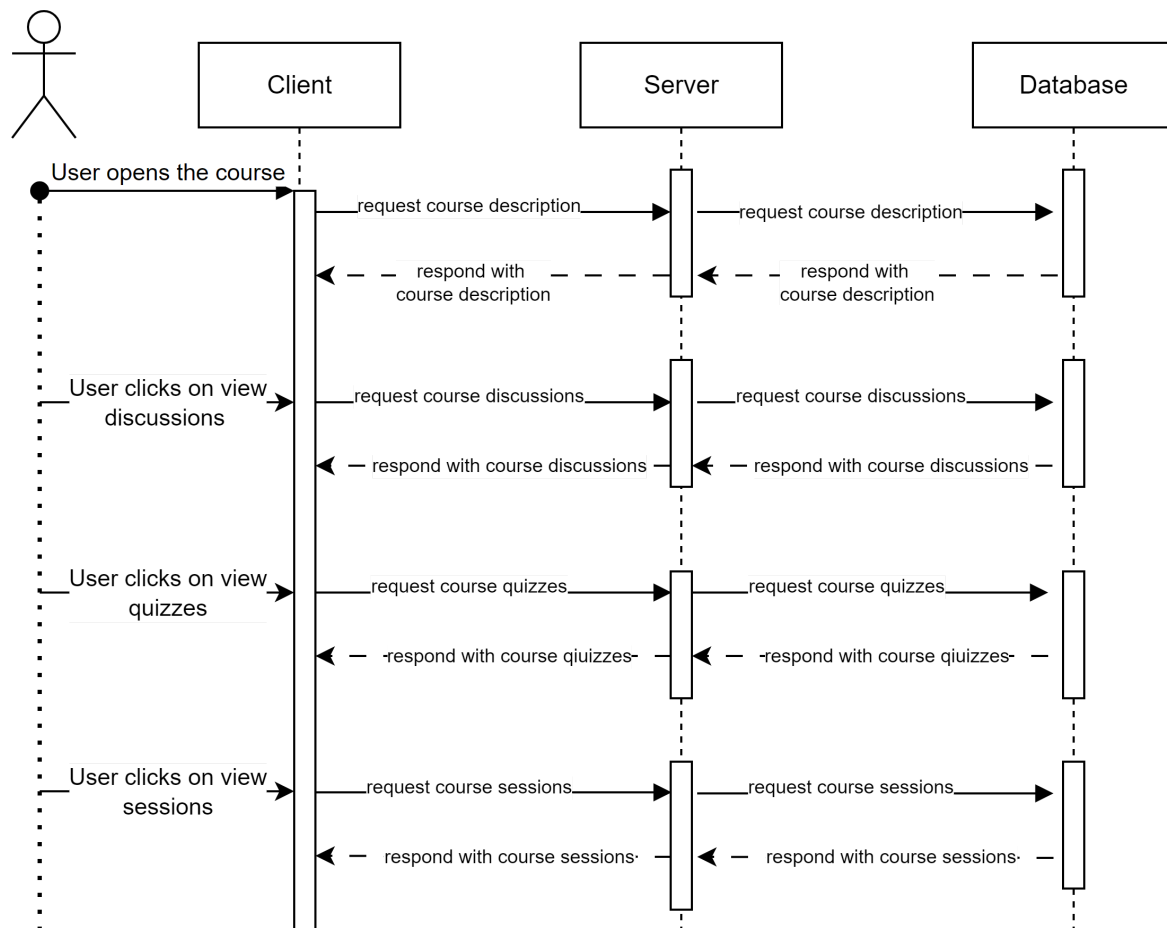
### 1. Sequence of events when users start the app and login

When the user opens the application, they are prompted with a login screen. The user enters their username and password into the appropriate fields on the login screen. The application then sends the entered credentials to the database for verification. The database checks the entered credentials against existing user data and returns the result of the verification; success if the entered credentials match a user otherwise it fails. If the verification fails, the application displays an error message and prompts the user to retry. If it is successful, the home screen is populated with the users data such as their courses and profile.



## 2. Sequence of events when users open a course

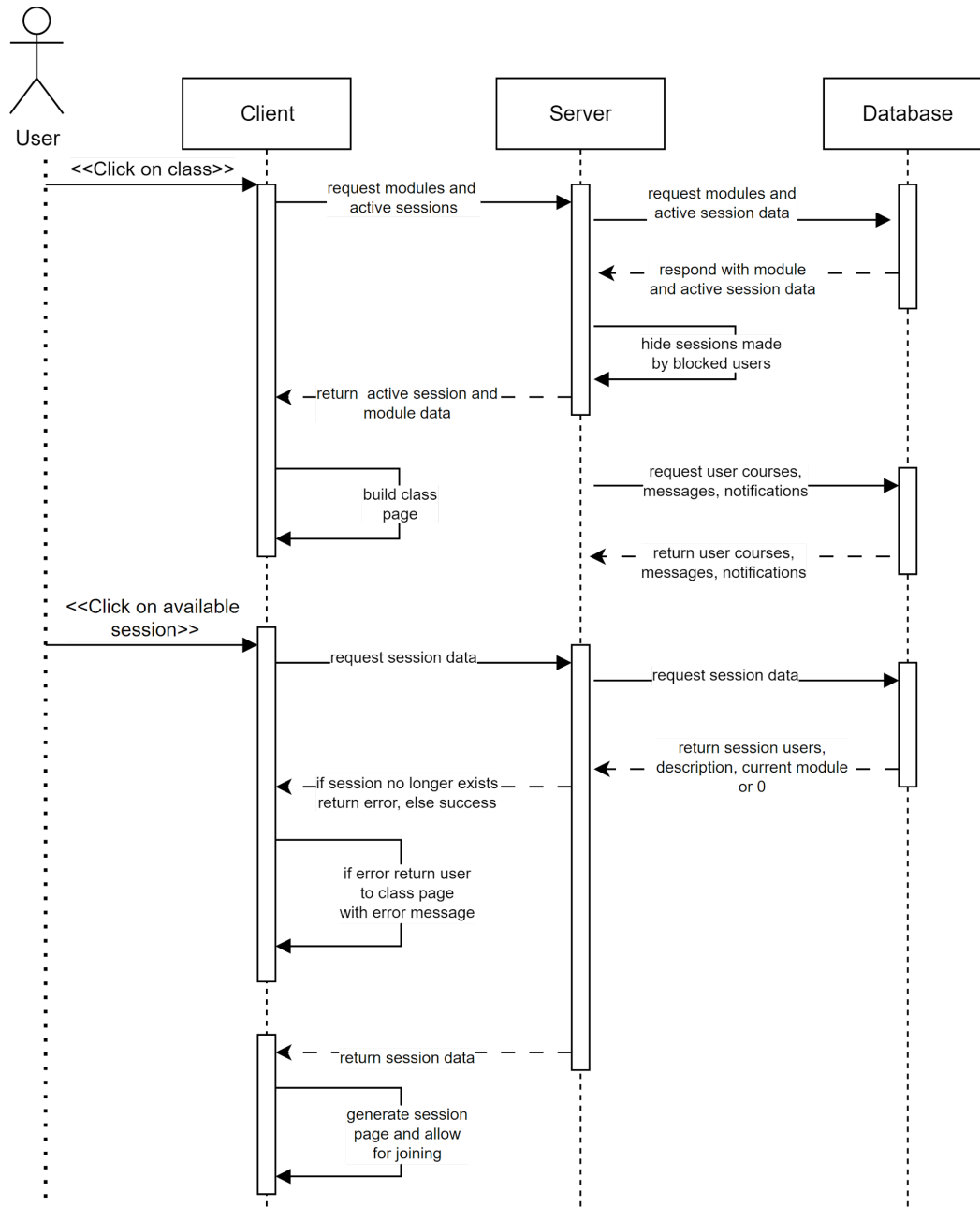
When the user opens a course, they are given the course description and prompted with the available options. These options allow the users to view discussions, quizzes or active sessions currently active in the course. When the user clicks on one of these options, the requested data is fetched from the server and displayed to the user.





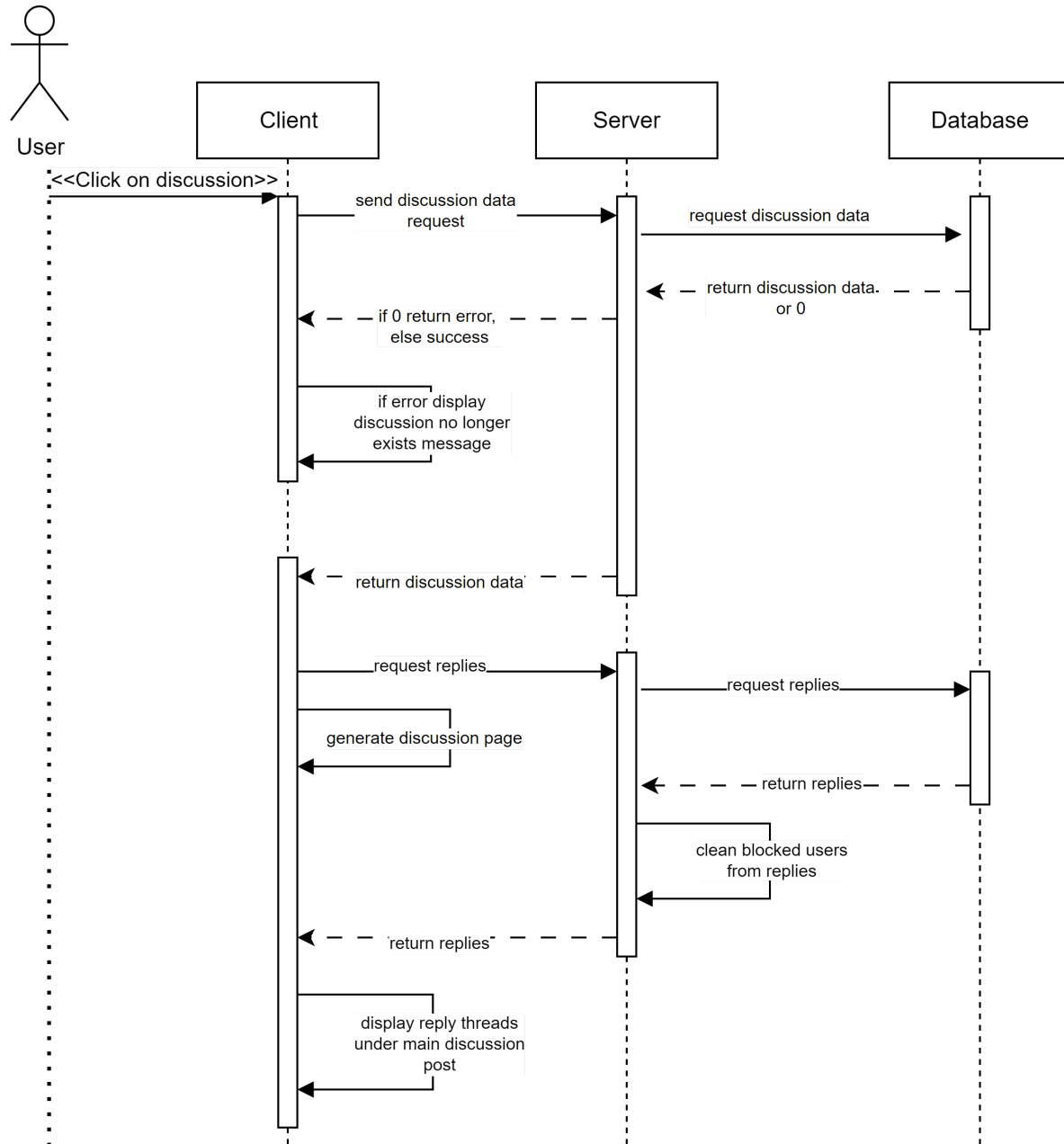
### 3. Sequence of events when users view a session

When the user clicks on the button to view sessions within a course, a new page is opened which displays the available sections in the course. This includes the ability to see the name of the session and the number of people within the sessions. When the user clicks on a session, they are added to the session unless it is not available to join anymore.



#### 4. Sequence of events when a user views a discussion

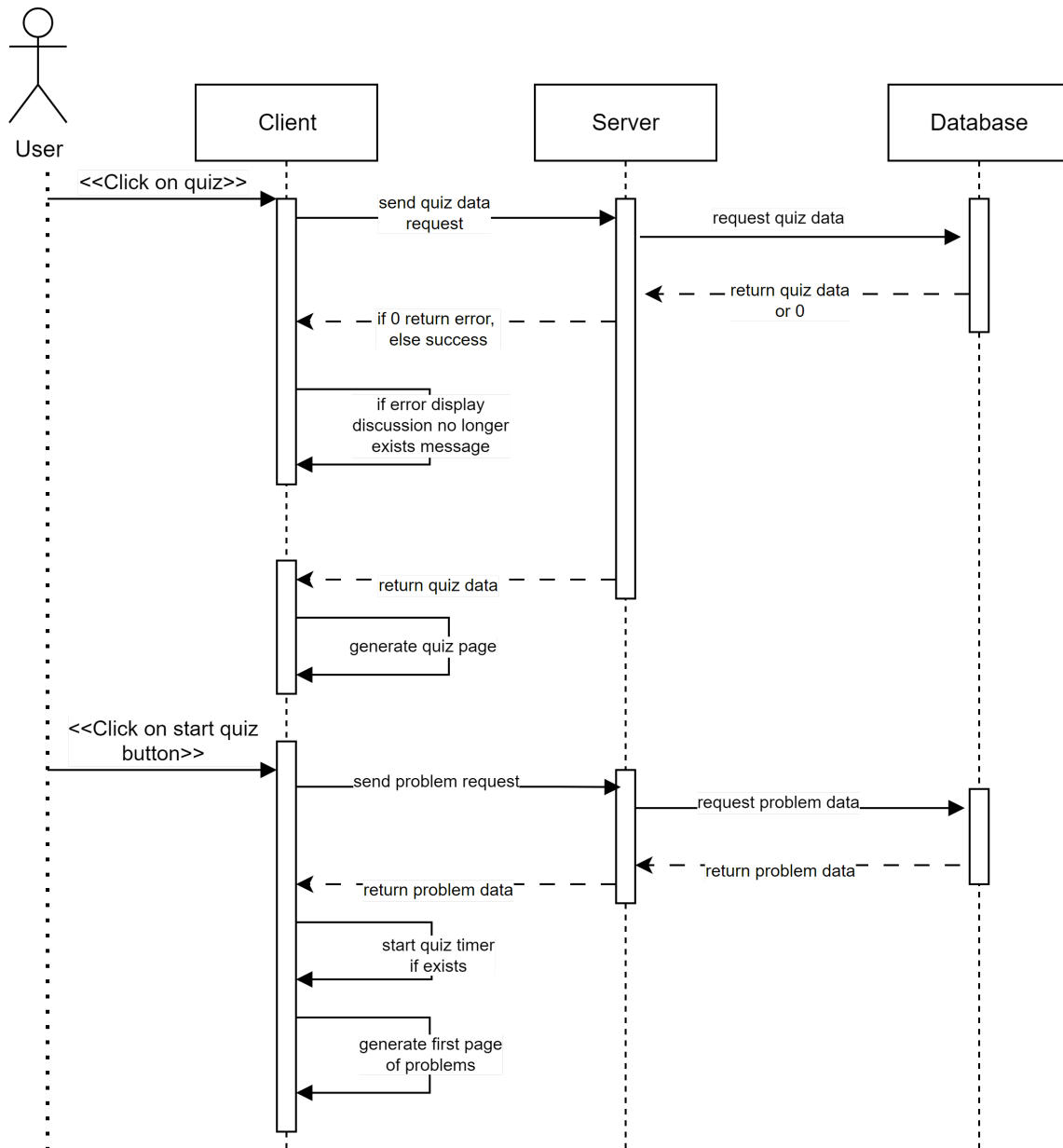
When the user selects a specific discussion to view, the application sends a request to the database to fetch the discussion data. The application displays this fetched data to the user. The application then sends a request to the server to get the replies of the discussion. The application then displays the updated discussion with replies after the server cleans the information from blocked users.



##### 5. Sequence of events when a user starts a quiz

When the user requests a specific quiz to view, the application sends a request to the database to fetch the quiz data. If the database was able to fetch the quiz, the quiz is displayed to the user alongside a start quiz button, otherwise the application returns an error message saying the quiz no longer exists. The user can then click on the start quiz. At this point, the application sends a

request to the server for the quiz problems, the timer is started if the quiz specified and the first page of problems are displayed to the user



## Navigational Flow Map

The design of our navigation emphasizes simplicity and ease of access. To access our web application, we encourage users to log in to their account using their Purdue email and password. If a user does not yet have an account they may create a new account using their Purdue credentials. The homepage contains the user's currently enrolled classes and a search bar for looking up and joining additional classes. The function banner on every page contains pages in the web application including going back to the homepage, links to currently enrolled class pages, and view user profile. On each class homepage, it will display the associated forum, buttons to go to the study materials, and it will display current open rooms with an option to create a new room. On the profile page, it will display user information and options to change certain aspects.

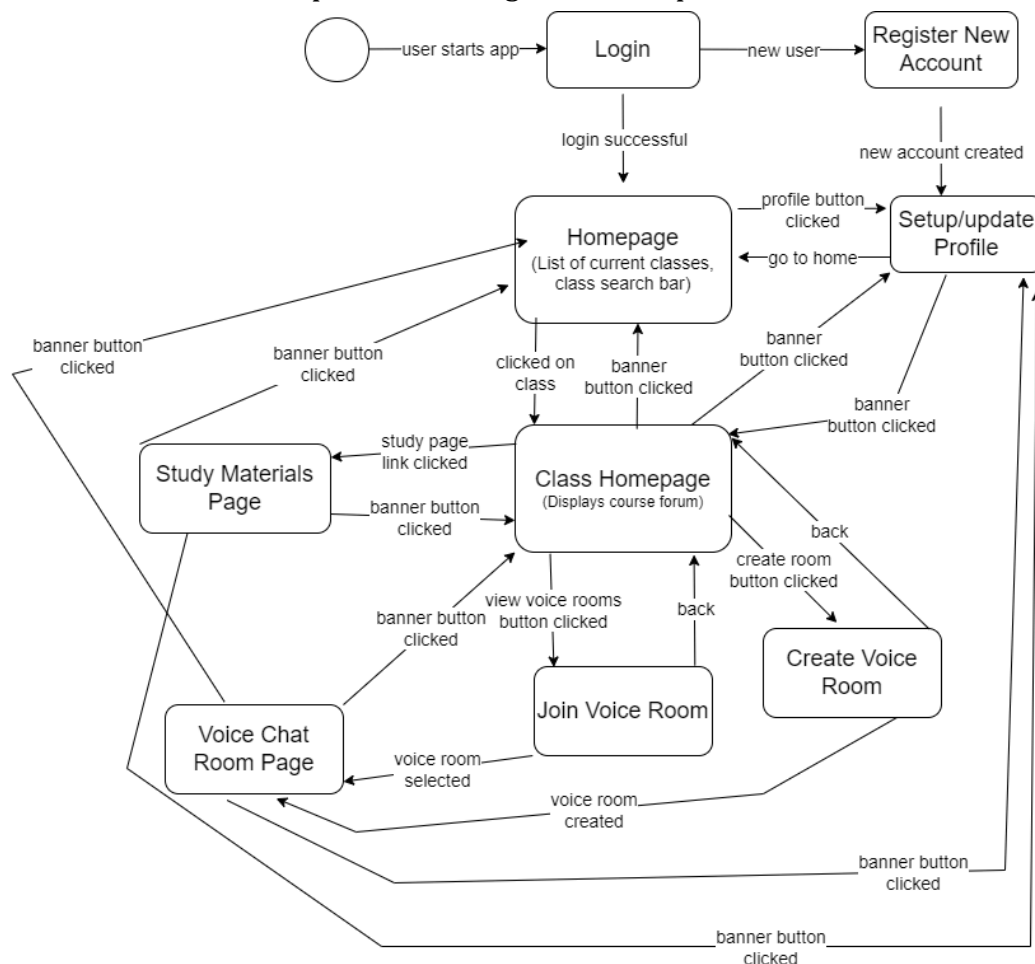
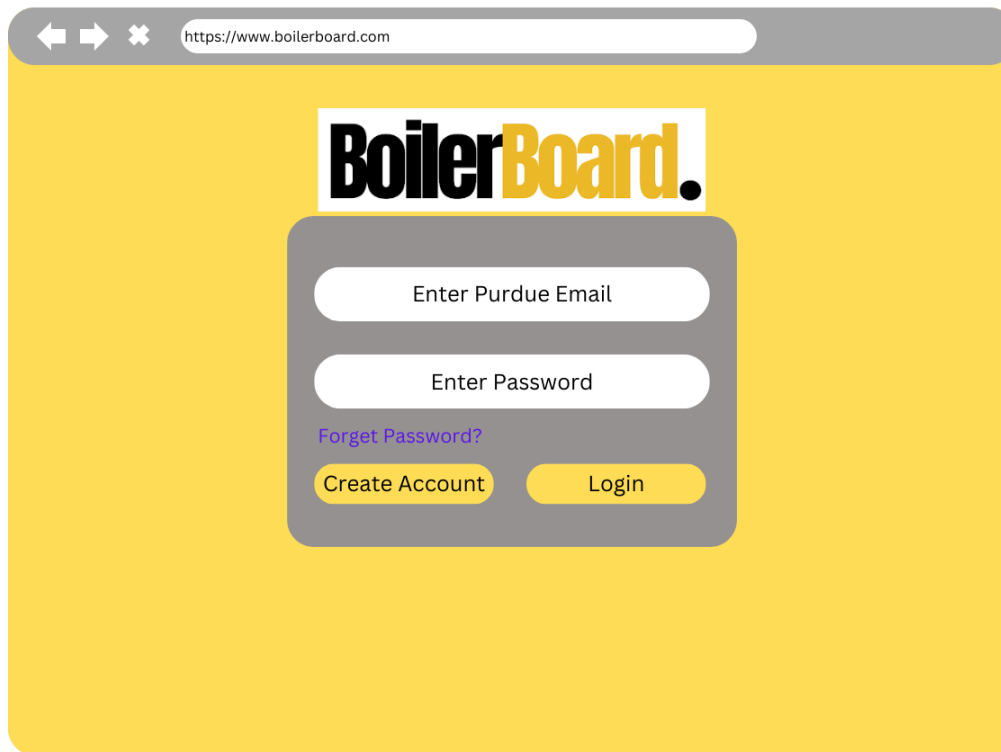


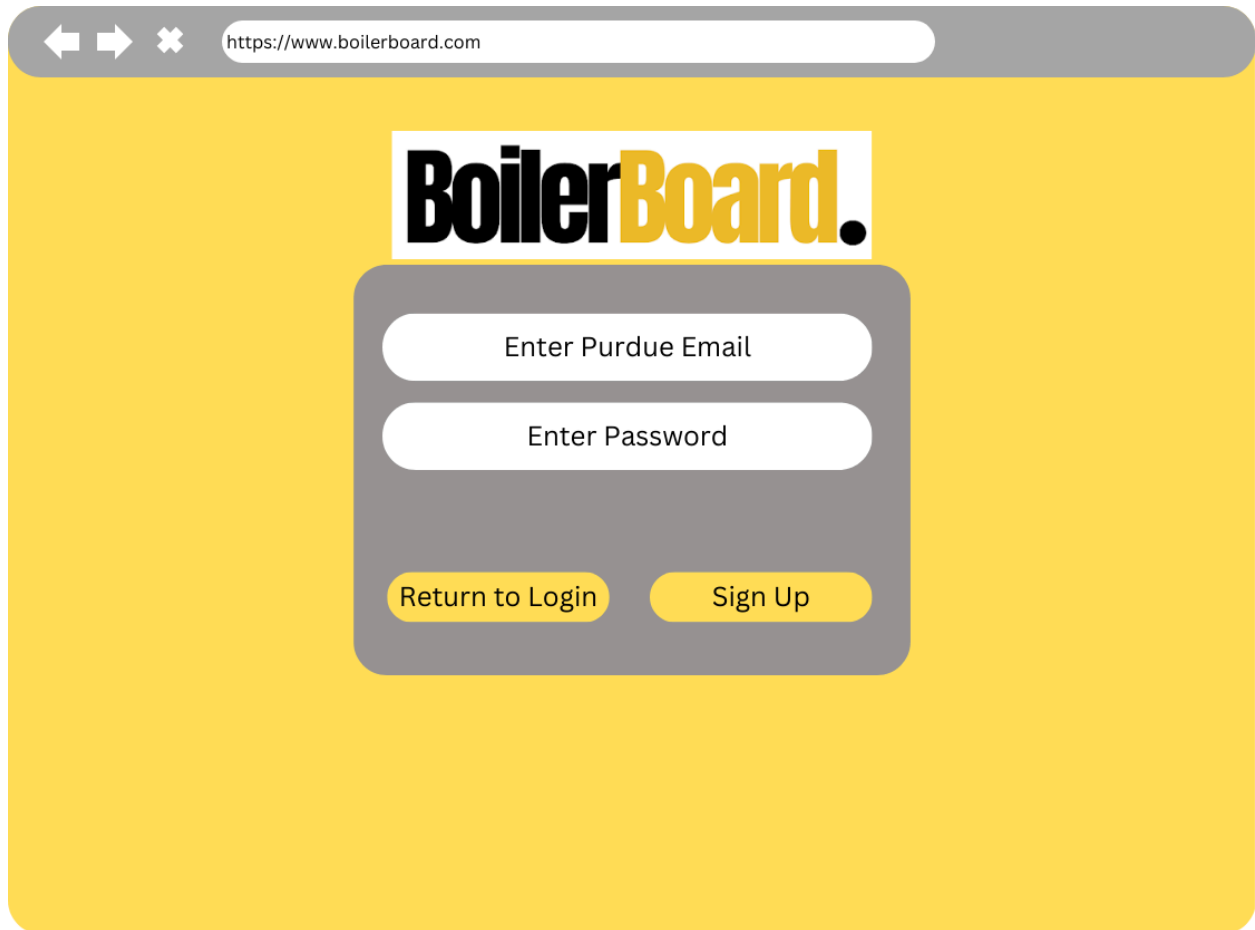
Diagram shows the ease of access. All modules after login are accessible from the function bar at the top. All submodules and modules are easy to access through the function bar.

## UI Mockups

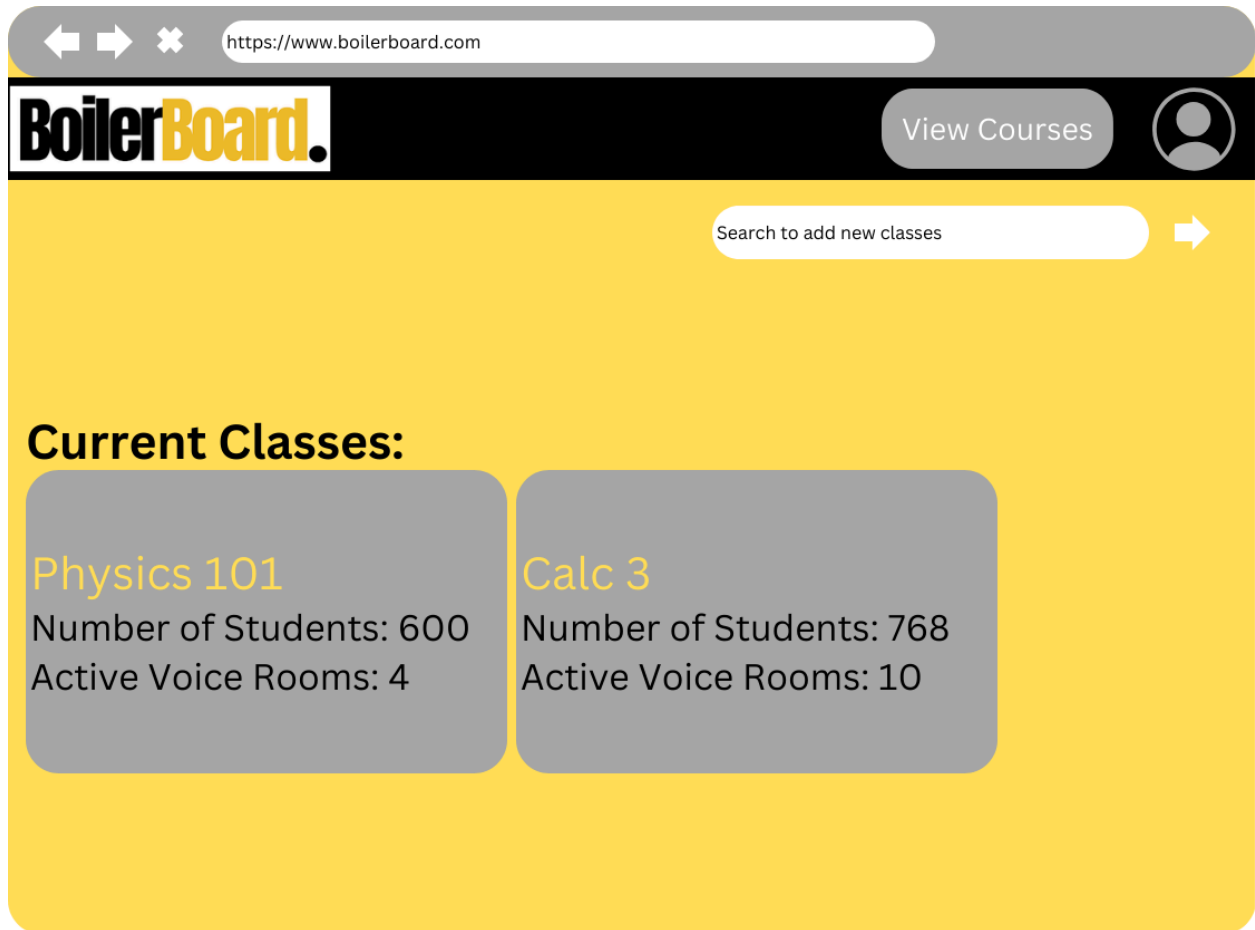
We designed the function bar on top of each page after the user logs in. The function bar contains quick access to all other pages such as the logo which when clicked returns to the homepage, a button to see a dropdown of all current classes, and a button to go to the profile page. The UI is clean and well organized. All related functions are on the same page.



This is the login page. This is where users start the application. It allows users to sign in using their Purdue email if they have an account or they can click the create an account button which will redirect them to the setup account page.

A screenshot of a web browser displaying the BoilerBoard login and sign-up page. The browser's address bar shows the URL "https://www.boilerboard.com". The page has a bright yellow background. At the top center is the "BoilerBoard." logo, with "Boiler" in black and "Board." in yellow. Below the logo is a grey rounded rectangle containing two white input fields: "Enter Purdue Email" and "Enter Password". At the bottom of this grey rectangle are two yellow buttons: "Return to Login" and "Sign Up".

This is the page where users can create a new account. It also allows for users to click the return to login page in case they accidentally entered this page. When a user clicks "Sign Up" it will send that information to the server in order to process the user's information. The server will then send a response whether the information is correct and an account does not already exist for the given Purdue email.



This is BoilerBoard's homepage. It contains the currently enrolled in classes with the number of other students enrolled in the class and how many active voice rooms there are currently. There is additionally a search bar to add new classes. The function bar at the top contains the logo to return to the homepage, a button to view a dropdown of the currently enrolled classes and a button to go to the profile page.



← → × https://www.boilerboard.com

**BoilerBoard.** View Courses

## Profile View/Setup

Email: cox407@purdue.edu

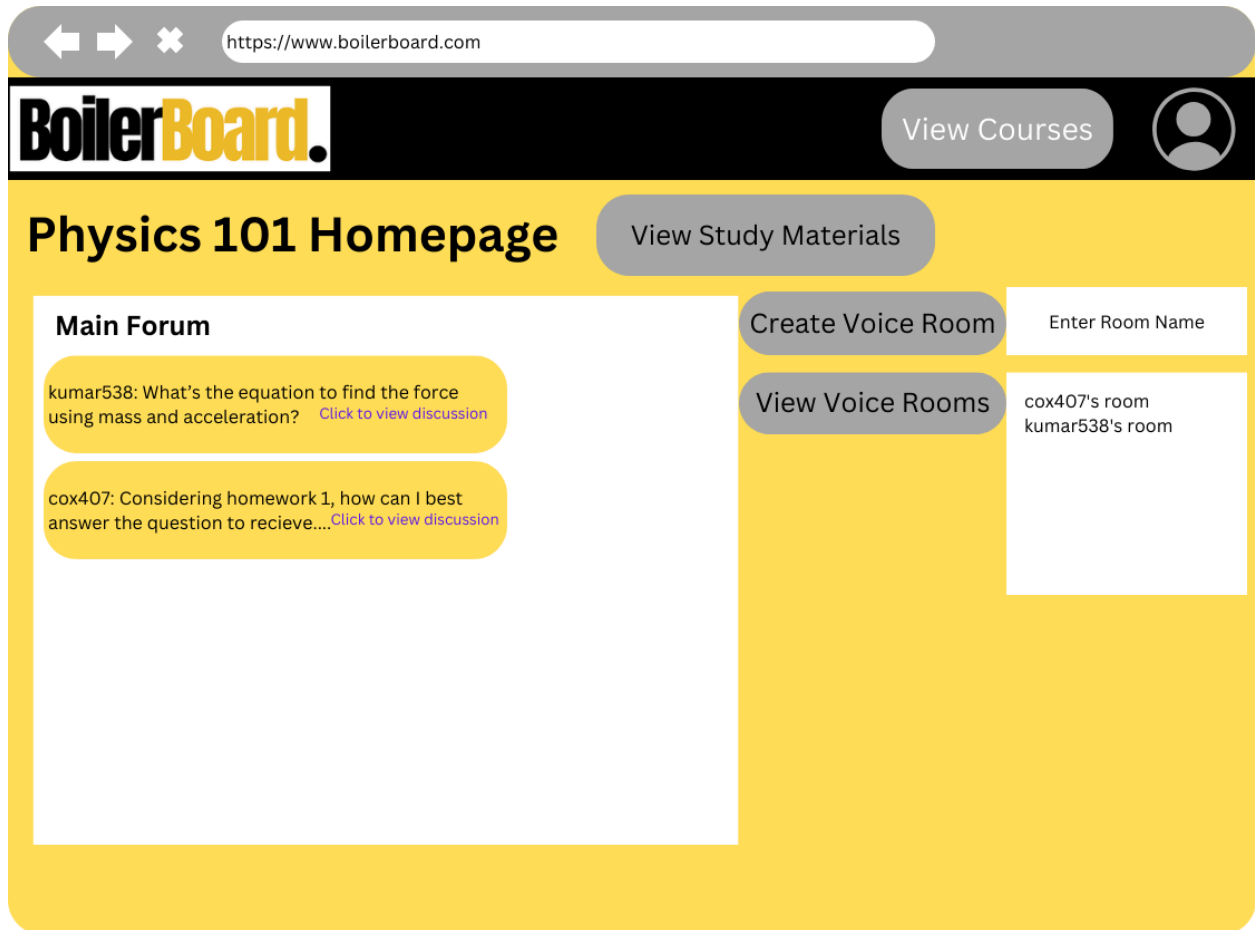
Password: 123456

Submit Changes

### User Statistics

Number of questions answered: 76  
User rating score: : 8.5/10  
Number of questions answered: 76  
Upvotes: 108 Downvotes: 6

This is the profile view and setup page. This is where users can view their statistics for BoilerBoard and they can change their password.



This is the homepage for a course registered to the user. On this page, users are able to view discussions related to the course. Users can create and join voice rooms. When the view voice rooms button is clicked a drop down will appear showing the available rooms which a user can click on to join. Users can also click on the view study materials button which will take them to the related study materials page.