CS 407 Fall 2024 | Team 4

Course Clash

Design Document

Team Members

Thomas Carsello <tcarsell@purdue.edu>
Jason Bodzy <jbodzy@purdue.edu>
Zachary Heskett <zheskett@purdue.edu>
Sooha Park <park1343@purdue.edu>
Celia Patricio <cpatrici@purdue.edu>
Sergio Alvarez <alvar166@purdue.edu>

Table of Contents

Section I: Purpose

There are several tools available for studying course materials online, many of which contain features used to gamify the process of studying. However, most of these online study games are single-player games that fail to tap into the greatly motivating psychological effect of competition. For many, competition can be a motivator in and of itself, driving them to put effort into the necessary prerequisites in its pursuit, even if those prerequisites are not found to be intrinsically motivating. Of course platforms exist on which multiplayer study games can be played, but we have noticed that these games always require real-time, synchronous play. All of the parties playing the game must be present, on their devices, playing the game at the same moment as each other. This limits the setting in which, and duration for which, students can engage with these multiplayer study games.

Our goal is to create a platform on which teachers / professors / course coordinators (henceforth referred to as “coordinators”) can create course-specific study materials for their classes, with which students can engage through asynchronous, turn-based correspondence games. We believe that this style of game will enable students to engage with study materials at more convenient times, increasing their likelihood of participating over a long period of time. This will provide greater value to both the students and the coordinators as increased student engagement is expected to have a positive correlation with content absorption and academic performance.

Functional Requirements

User (Student of Coordinator):

- As a user, I would like to register for a Course Clash account with an email, display name, and password, and sign in/out via these credentials.
- As a user, I would like to manage my account details, including my information and password.
- As a user, I would like to upload a profile picture to my account.
- As a user, I would like to view all of the courses in which I am a member, and of which I am the coordinator.
- As a user, I would like to join, leave, create, and delete courses.
- As a user, I would like to view study materials (i.e. terms and definitions, quiz questions) within specific courses.
- As a user, I would like to receive email notifications on an opt-in basis.
- As a user, I would like to post, comment, and view others' posts on a course's discussion board.
- As a user, I would like to tag my posts on discussion boards as either Question, PSA, or General

Student

- As a student, I would like to challenge other students to head-to-head games, and accept/reject challenges received from other students.
- As a student, I would like to play head-to-head games against other students in a course.
- As a student, I would like to track my performance, and the performance of other students in the class, based on their games played (leaderboard).

- As a student, I would like to see all of the games I am currently participating in, and the games I have completed.
- As a student, I would like to see statistics about the games I have played within a course.
- As a student, I would like to play games when it is my turn.
- As a student, I would like to answer the quiz questions created by the Coordinator while playing a game.
- As a student, I would like to be notified if my selected choice during a game is correct or incorrect.
- As a student, I would like to surrender/resign from a game.
- As a student, I would like to view instructions on playing the game.
- As a student, I would like to focus my studying by filtering study materials by topic

Coordinator

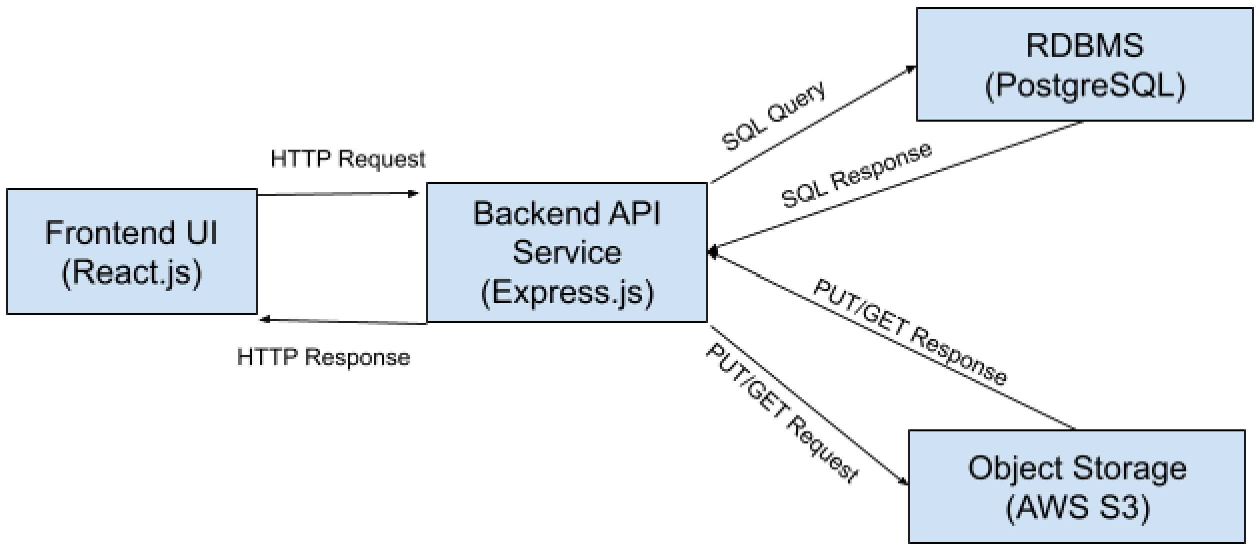
- As a coordinator, I would like to invite students via email to sign up for Course Clash and join courses.
- As a coordinator, I would like to remove students from my courses.
- As a coordinator, I would like to upload a photo for each of my courses.
- As a coordinator, I would like to upload study terms via CSV, or through direct entry.
- As a coordinator, I would like to upload quiz question sets via CSV, or through direct entry.
- As a coordinator, I would like to organize study terms and questions into categories (Topics).
- As a coordinator, I would like to manage (edit, delete) study terms and quiz questions.
- As a coordinator, I would like to view statistics on games played and student performance within the course, based on their games played.

- As a coordinator, I would like to moderate (delete) posts and comments on the course's discussion board.

Non-functional Requirements

- Our platform should function on all popular browsers.
- Our platform should securely manage sensitive information (emails, passwords, profile pictures, etc).
- Our application should be designed for scalable cloud deployment.
- Our application should have an intuitive, easy-to-use UI.
- Our application should authenticate all API requests pertaining to user data.

Section II: Design Outline



Overview

Our system will have four main components. The frontend UI will be the React.JS application our users will interact with through their web browser. This app will communicate with the Backend API Service through HTTP. The Backend API Service written in Express.JS handles all of the logic of receiving, processing, and responding to frontend requests. It also serves as the bridge between the two storage systems, which will not directly interact with each other. The RDBMS in SQL will hold tabular information while the Object Storage system (AWS S3) will be used for large items on a key/value basis. Rather than storing large objects itself, the RDBMS will maintain the keys that correspond to objects within the Object Store.

Design Decisions

We have chosen React JS for its efficiency, component-based architecture, and for their large ecosystem of libraries and tools as they provide a rich set of features and functionalities for building web applications such as ours. We chose Express JS for its simplicity, flexibility, and performance. Its middleware architecture will allow for modular and reusable code which makes it easier to build scalable and maintainable API's. SQL was chosen for its ability to efficiently store, retrieve, and manage structured data. SQL databases offer powerful query languages, ensuring efficient data access and manipulation. Using SQL will also provide features like ACID compliance and transaction support to maintain data integrity and consistency.

System Components

1. <u>API Service</u>
    a. Interacts with the frontend via HTTP requests. Receives requests and sends responses with JSON body content
    b. Interacts with the PostgreSQL RDBMS system to read and manage data models in response to HTTP requests from the frontend
    c. Interacts with the S3 Object store to read and write large objects efficiently. The API service will also handle interdependencies between the object store and the RDBMS, namely in that the RDBMS may store the key used to reference a particular object
2. <u>Frontend UI</u>
    a. Interacts with the user directly through the browser interface
    b. Sends requests to the API service backend and receives responses
    c. Renders information received from the server, and collects information from the user
3. <u>RDBMS</u>
    a. Stores all of the application's core functional data
    b. Will use 3NF / BCNF for data integrity
    c. Stores keys to artifacts in the Object Store system
    d. Interacts only directly with the API Service
4. <u>Object Storage</u>

a. Stores large items such as images that would not be a good fit for RDBMS
b. Operates on a key/value basis
c. REST API through AWS managed service (S3)
d. Automatically scales

Deployment

For deployment, we will use AWS. Our RDBMS and Object Store will be hosted through AWS managed services, namely RDS and S3. To host the API service and React frontend, we will use Docker to containerize the application, and AWS ECS to dynamically scale.

Section III: Design Issues

Functional Issues

1. How will our users import and export question sets?

Option 1: CSV file

Option 2: Custom file format

Option 3: Text file

We want our users to be able to import questions they have made from a file. There are multiple ways we could have done this, but we decided to go with a CSV (comma separated values) file. We chose this because it is an existing file format that can be easily exported from programs like Excel. A custom file or text file would have no such support, and require more manual effort on the part of the user importing the file. A CSV file is also easy to read from our perspective, as the only parsing we have to do is on the commas. This also makes them easy to export. Exporting a CSV would then also allow our users to open the file in Excel or a similar program to look at questions.

2. Who is allowed to create courses and questions?

Option 1: Teacher/Professor accounts only

Option 2: All user accounts

Option 3: Anyone (no login required)

We want Course Clash to be a platform where teachers or professors can make courses with question sets, and then their students can study those sets via competition. However, if a student decides that they want to make a course and questions of their own to practice with their friends, we don't see why this should not be allowed. For this reason, we allow all accounts to create courses and questions, with no distinction between teachers and students at the account level. Only after a course is made will a distinction be made between course owner and course participant. We require a user to log-in to the website before creating a course because otherwise it would be too difficult to associate that course with an owner. We need our users to log in for this logistical reason.

3. How will students join courses?

Option 1: Public Join Codes

Option 2: Direct Invite via Email link

Option 3: Direct Invite on the platform

We will use a mixture of options 2 and 3. We will allow coordinators to invite students via their email addresses. If that email address corresponds to an existing user, they will receive an invite directly through the platform. If that user does not exist, they will receive an email prompting them to sign up. We want to avoid public join codes in order to avoid exploitation or leaking of the codes.

Non-Functional Issues

1. How will we tackle mobile functionality?

Option 1: Creating a separate mobile app alongside the website

Option 2: Developing the website to be mobile friendly at all times

Option 3: Developing the website to be more mobile friendly depending on viewport.

We decided to go with option 3, adjusting our website design to be friendly with mobile devices on typical mobile viewports. This approach requires less time investment on our end, as developing a whole new mobile app along the website could close to double our development time. A full fledged mobile app would likely be more desirable from an end user perspective, but it also wouldn't be able to be given the attention it deserves in the time we have. For this reason, we are focusing more heavily and ensuring our website design is easy to navigate in viewports typical of mobile devices. This includes making things like hamburger menus and large buttons. This gives mobile users a good experience without the overhead of an entire new app to develop. We chose this over option 2 because option 2 limits us in terms of good design for desktop users. Changing our design based on viewport size gives us the best of both PC and mobile use.

2. What kind of database will we use?

Option 1: SQL database

Option 2: NoSQL database

Option 3: From-scratch database

We chose to use a SQL database because our data is highly relational, which is what SQL excels at. Relationships like users who belong to a course, a game state that belongs to two users and a course, and a post that belongs to a user and a course are all relationships that SQL was designed to handle eloquently. We will not need to store super large JSON objects in our database, so the ability for databases like MongoDB to store these objects directly is not much of a benefit. Additionally, creating a performant database from scratch would be way too much effort in the scope of this project.

3. What database will we use?

Option 1: MySQL

Option 2: PostgreSQL

Option 3: SQL Server

We want our database to be able to run on the cloud in an AWS EC2 instance, since this gives us ease of access to our database at a relatively low cost without having to host it ourselves. SQL Server is very expensive and requires a license, so we will not use it. Between Postgres and MySQL, Postgres offers more features that are relevant to our use case, namely storing arrays natively which could be useful for storing our quiz lists. We also expect frequent updates to our data via the game states, and Postgres is better at handling this as well.

4. What backend will we use?

Option 1: ExpressJS

Option 2: Flask

Option 3: Custom Go backend

We don't expect extreme amounts of traffic to our backend, and for this reason, we do not think that a highly performant language like Go is necessary to create a satisfactory backend. Making a custom Go backend would take extra time to develop, which is not worth the benefit in our case. Between Flask and ExpressJS, we chose ExpressJS because of the language. ExpressJS can be written in TypeScript, the same language as our frontend. This makes it easier for our developers to switch between the front and backend of the project without having to be proficient in two languages.

5. How will we store large items (pictures)

Option 1: Amazon S3

Option 2: Host’s file system

Option 3: Encoded in SQL

We will use Amazon S3 for storing large files like images because of its RESTful API interface and simplicity in scaling and deployment. AWS S3 is a service that is managed by AWS and does most of the heavy lifting for us. We can take advantage of this to make our lives easier during the development process as well as the deployment process. This will also allow us to move our hosting environment easily, rather than having to also move all of the files from one host environment to the other.

6. How will we handle user authentication and authorization?

Option 1: Session IDs

Option 2: JWT

Option 3: Oauth

We will use JWT as our auth solution due to its simplicity and scalability. Session IDs require database entries for sessions, and a database access for validation, which can be slow and messy. JWT on the other hand does not require a centralized location for storing session IDs, and validation can happen quickly without database access. Oauth is more complicated and involved, which we will not need for this project.

Section IV: Design Details

Major Data Models

User Model - This model will hold information pertaining to a specific user such as their email, name, password hash, etc.

Course Model - This model will hold information pertaining to a specific course such as the course id, course coordinator, course name, description, etc.

Topic Model - This model holds information pertaining to a specific subtopic of a course, such as a unit or concept which contains study materials. This will be used for organizing study materials into groups based on these topics.

Term - This model holds study terms and definitions. These will be created by a course coordinator and associated with a topic. Students can view these terms as part of their studying functionality

Question - This model holds a specific question for the quiz game. These will be created by a course coordinator and associated with a topic. Students will see these questions when playing the head-to-head game.

Answer - This model holds potential multiple choice answers to questions. These will be created by the course coordinator alongside the questions, and these are the answer options available to students when playing the quiz game.

Post - This model holds a top-level post on the course's discussion board. These can be created by anyone in a specific course and will be visible to all others in the course.

Comment - This model represents a user's reply to a post on the discussion board

Invite - This model represents an invitation from a course coordinator to a user or potential user via their email address to join the course.

Challenge - This model represents a challenge sent from one student to another in the same course to play in the head-to-head game

Game - This model represents an instance of the head-to-head game between two students within a course

Round - This model represents a round of a Game, and holds information pertaining to each player's score during that round.

Data Model Interactions and UML

User-Course Interaction

Users can be members of many courses, and courses can have many members. This many to many relationship is achieved through the UserCourse junction table

A course only has one coordinator, but a user can be a coordinator of many courses. This One to many relationship is achieved by the coordinator foreign key on the Course table, which references the user_id primary key on the User table.

Course-Invite-User Interaction

A course coordinator may create many invites. This one to many relationship is through the foreign key reference to course_id on the invite table. The invite is done via email address, since a user may or may not exist with that email address. This is okay, since email addresses per user will be unique, natural keys. The pair of (course_id, email) will be unique, and serve as the primary key for the Invite table.

Course-Post-UserCourse Interaction

A course can have many posts. These posts can only be made by users belonging to the specific course. This is ensured by using the UserCourse junction table from earlier as the foreign key reference on the Post table, facilitating the many-to-many relationships between users and posts, as well as between courses and posts, while ensuring posts are only made by users on courses to which they have access to.

Post-Comment-UserCourse Interaction

Posts can have many comments, and this one to many relationship is achieved through the foreign key reference to the post_id on the Comment table. The comment must be made by a user belonging to the course on which the post is made, so there is a foreign key reference to the UserCourse table. This facilitates the many-to-many relationship between users and comments as well as courses and comments.

Course-Topic Interaction

A course can have many topics, but a given topic can only belong to one course. This one to many relationship is achieved through a foreign key reference to the course_id located on the Topic table.

Topic-Term and Topic-Question Interaction

Terms and Questions are both study materials that coordinators can create. They belong to specific topics within courses, so this one to many relationship is achieved through a foreign key reference of the topic_id on the Term and Question tables respectively.

Question-Answer Interaction

Questions can have several Answer choices, but only one of those answer is correct. This relationship is achieved through the QuestionAnswer table which pairs question_id and and answer_id together to form a primary key. This junction table also has a field to determine if that question-answer pair is correct or incorrect.

<u>User-Course-Challenge Interaction</u>

Challenges can be created by students within courses, but must be between two students in the same course. We can use the UserCourse junction table from earlier to ensure this is the case as we did with the discussion board's posts and comments earlier. The challenge model will reference two records on the course-challenge table to denote a challenge request between those two students.

<u>Challenge-Game Interaction</u>

If and only if the challenged player accepts a challenge, will a game be created. This one-to-zero-or-one relationship is achieved through a foreign key reference on the Game table to the challenge that was used to create the game.

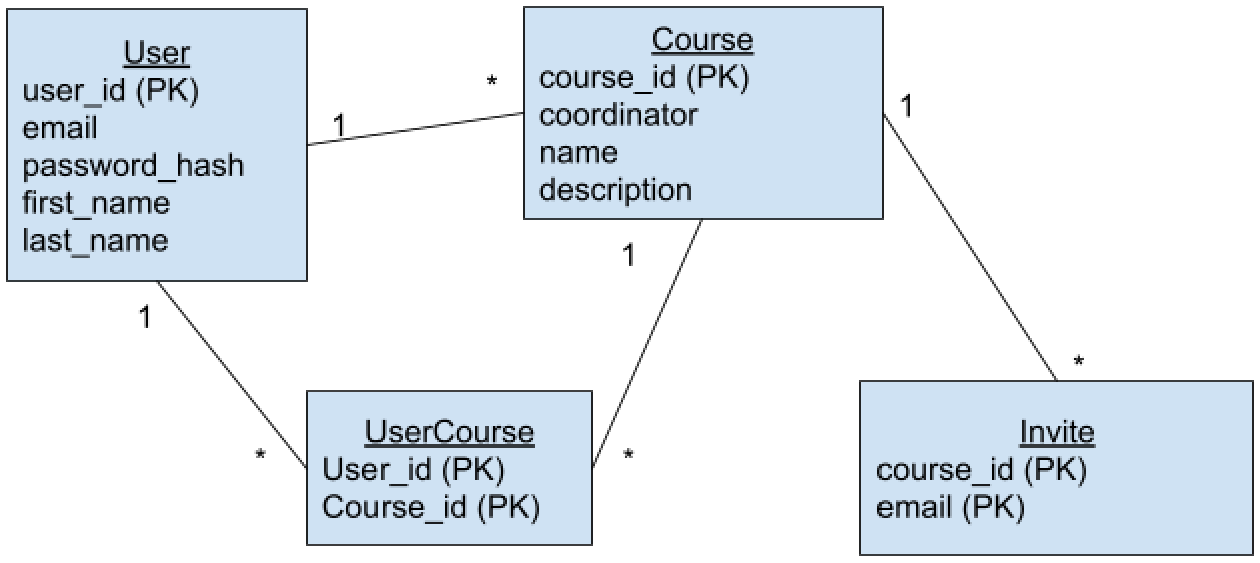
<u>Game-Round Interaction</u>

Games can have several rounds, but a round only belongs to one game. This one to many relationship is achieved through a foreign key reference on the Round table to the game_id of which that round belongs.

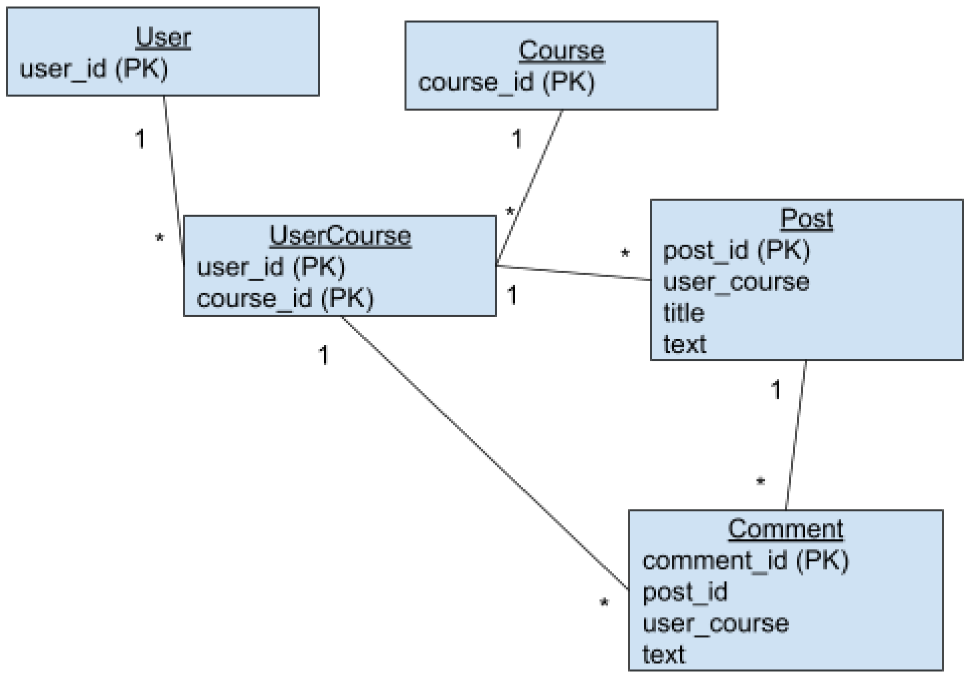
** UML Diagrams: For simplicity, we will break the UML diagram into separate subsystems, and only display fields on each table relevant to that subsystem.*

** Many-to-Many relationships will be implemented using Junction Tables to maintain 3NF / BCNF compliance.*

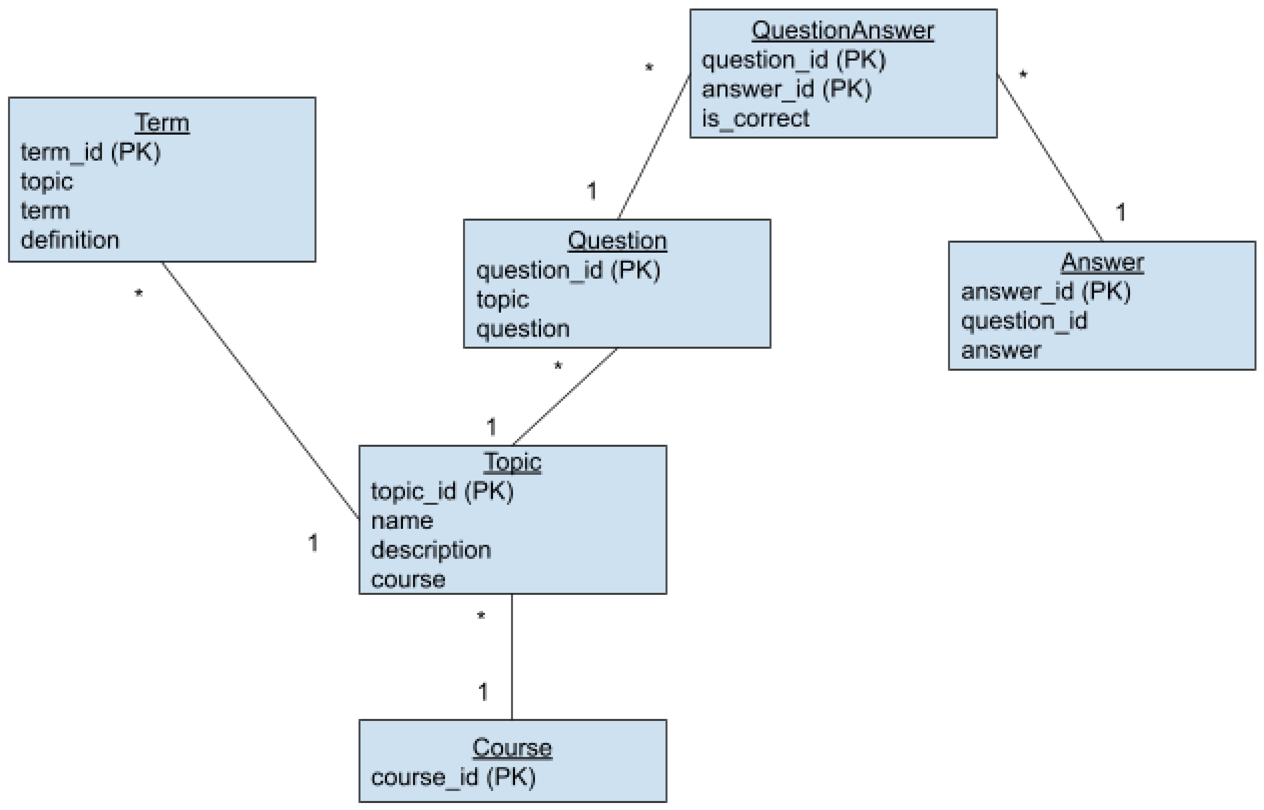
<u>Users and Courses Subsystem</u>



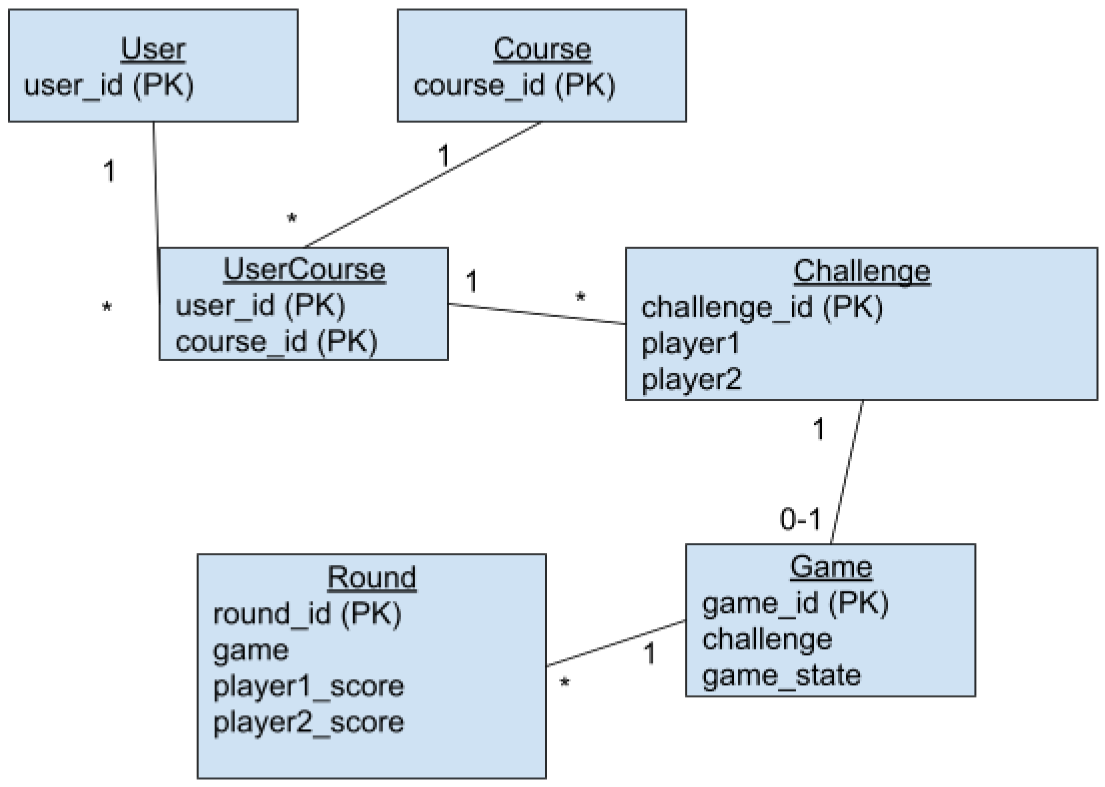
Course Discussion Forum Subsystem



Course Study Materials Subsystem



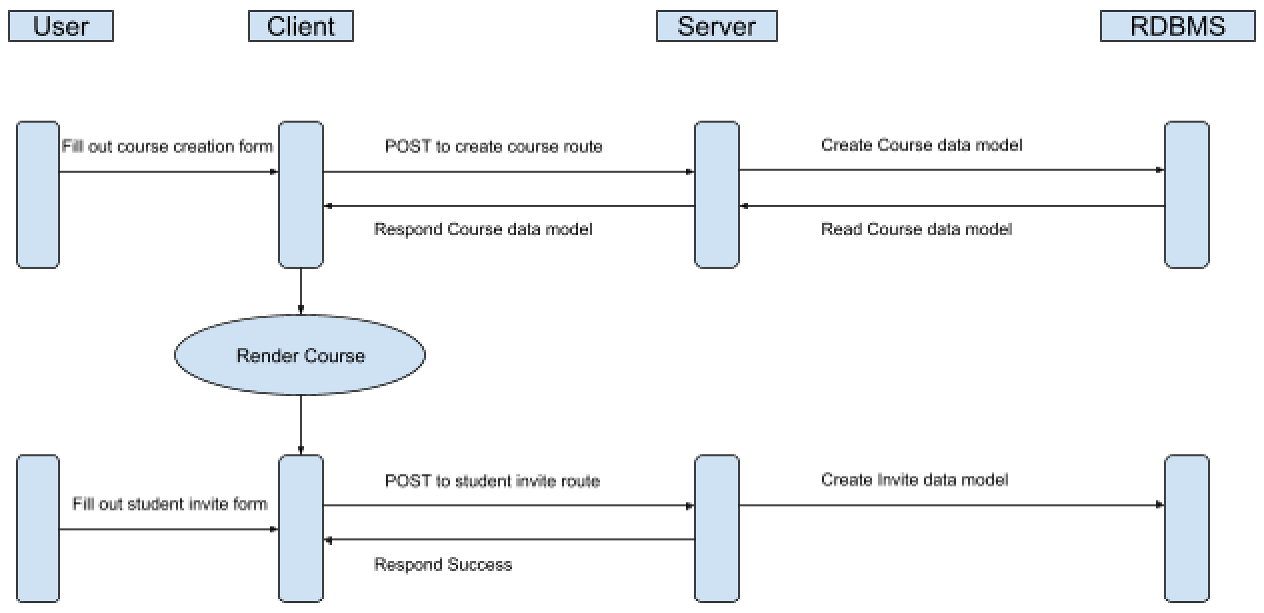
Course and Game Subsystem



Event Sequence Diagrams

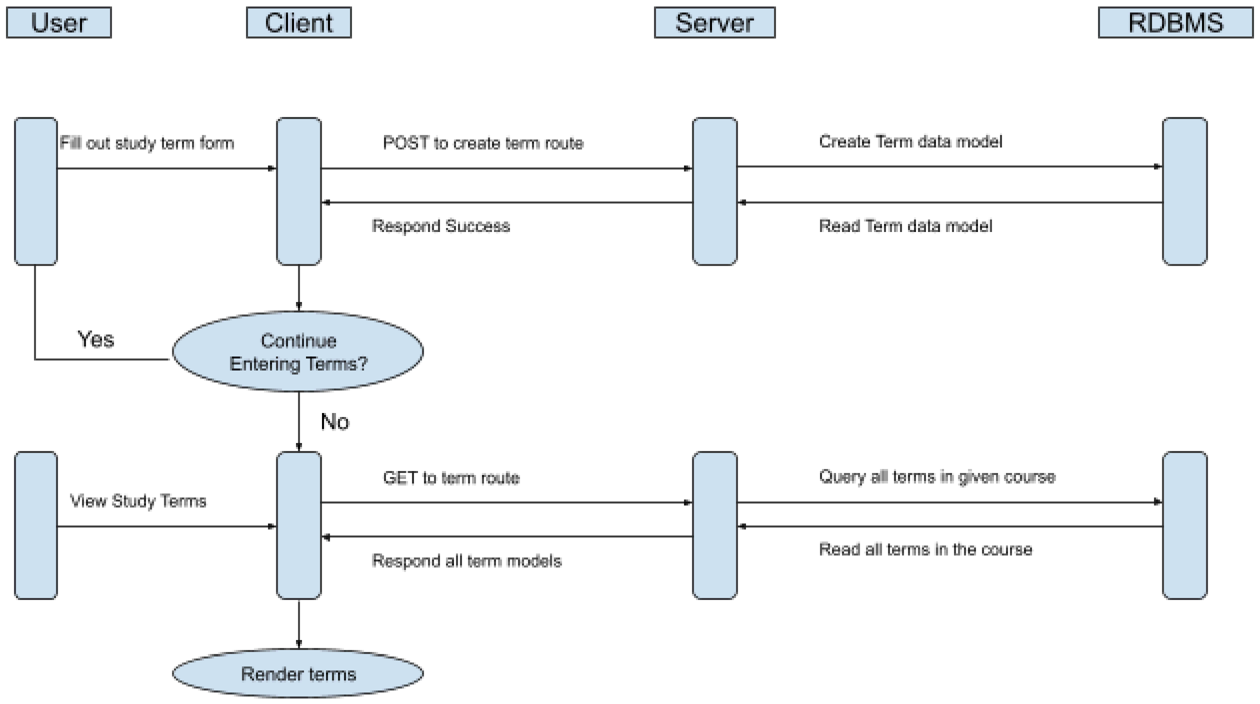
Create a Course and Invite Students

A user will create a course, and then inside that course, select “invite students” and enter the emails of students to invite.


User
Client
Server
RDBMS
Fill out course creation form
POST to create course route
Create Course data model
Respond Course data model
Read Course data model
Render Course
Fill out student invite form
POST to student invite route
Create Invite data model
Respond Success

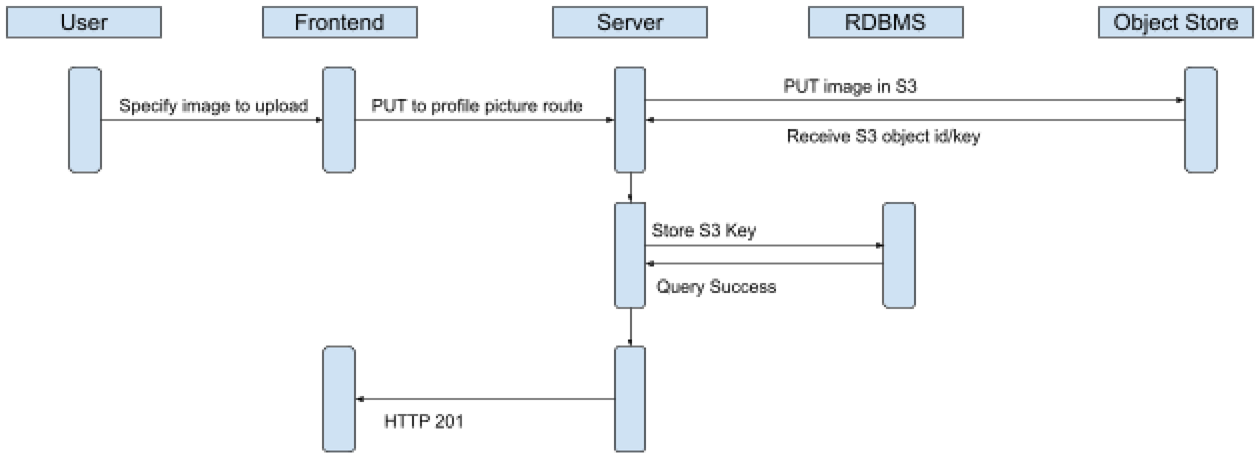
Create and View New Study Terms

A course coordinator will enter terms and their corresponding definitions into the front end form, and send those values to the server which will be stored in the database. This process can continue for as many terms as the user would like to upload, until they decide they are done. Then, they can view the terms in the course via a GET route which returns all terms form the database.



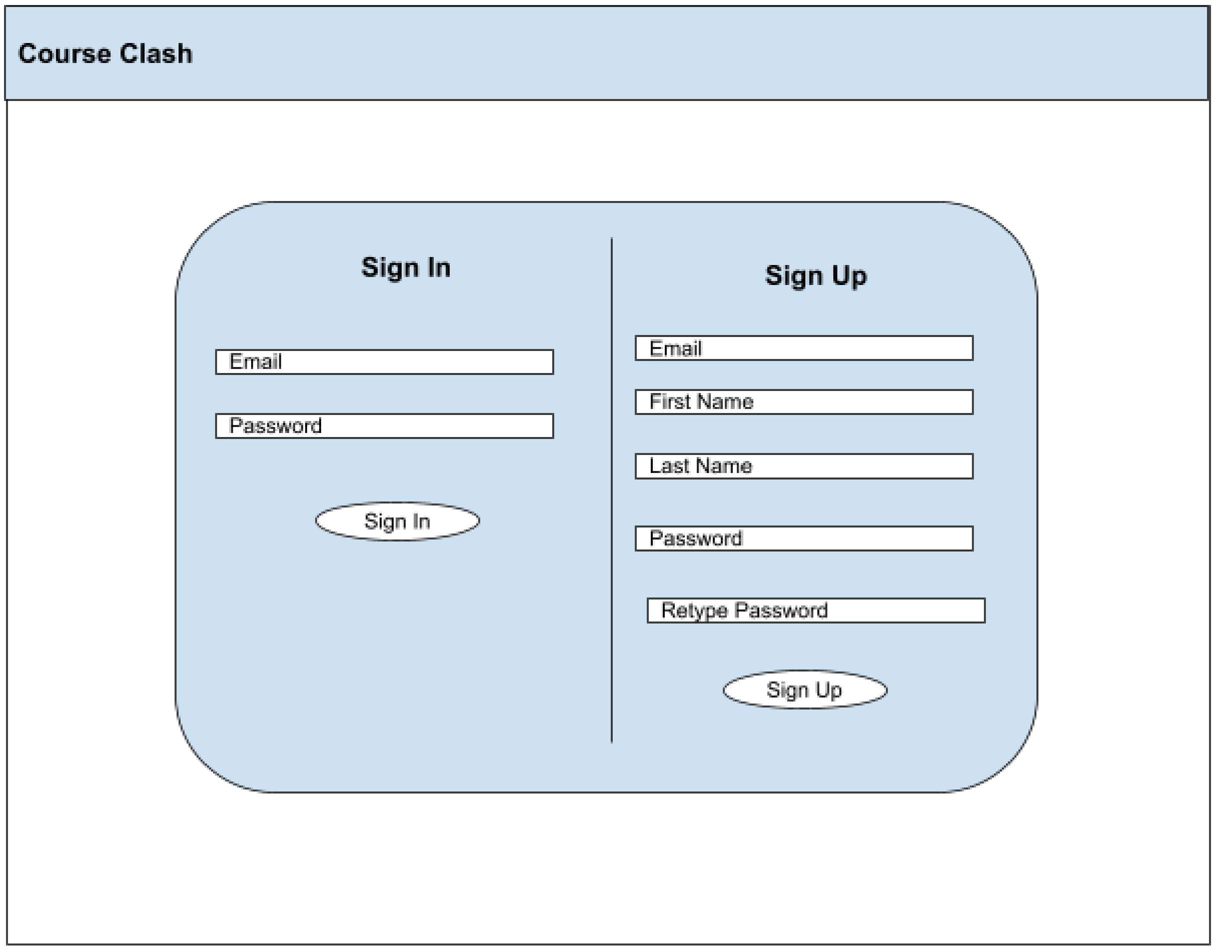
Upload Image (Profile image / Course image)

The user will specify an image to upload in the frontend, which will send an HTTP request to the API server. The API server will use the AWS S3 REST interface to upload the image to the object storage system. The Object storage system will respond with the unique key it generated to access that object later. This key then gets stored in the database for lookup at a later point.

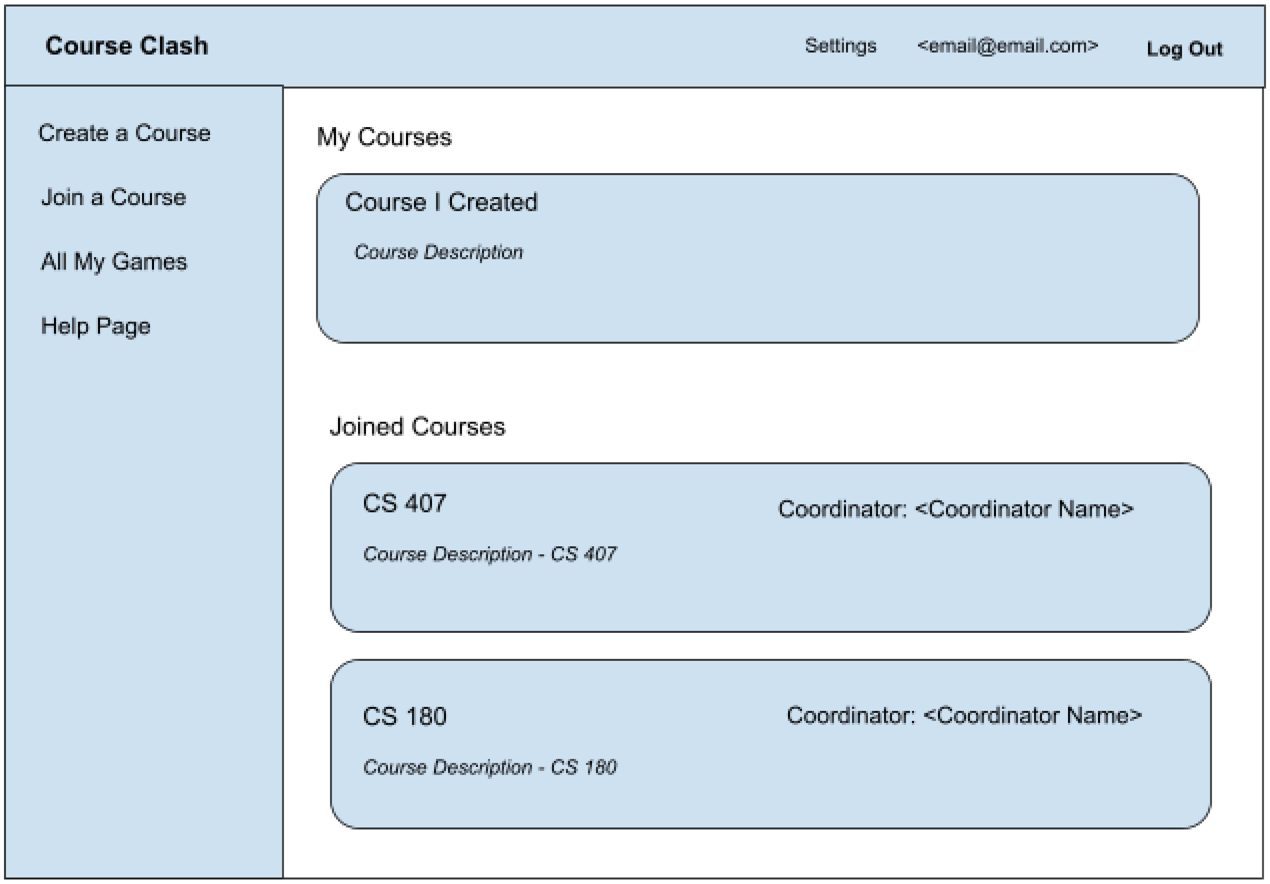


UI Mockups

<u>Login / Signup Page</u>



Main Homepage, Course List Mockup



My Courses

Course I Created

Course Description

Joined Courses

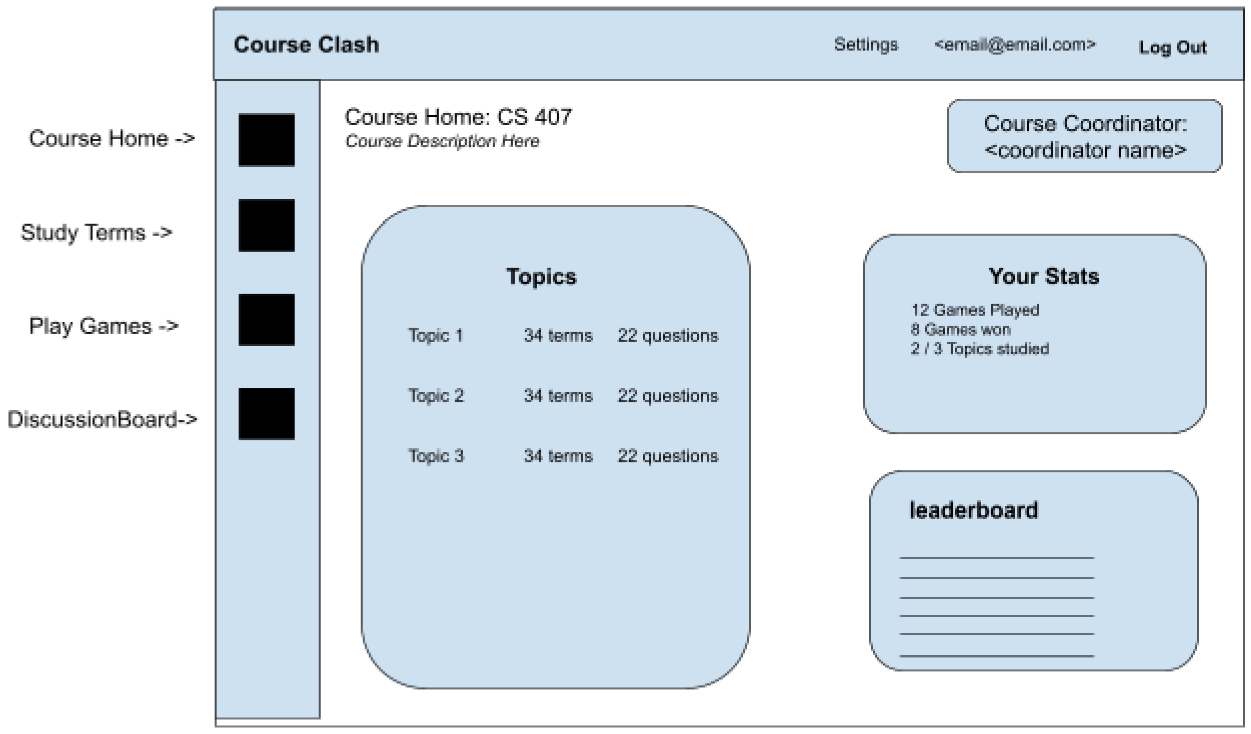
CS 407 Coordinator: <Coordinator Name>

Course Description - CS 407

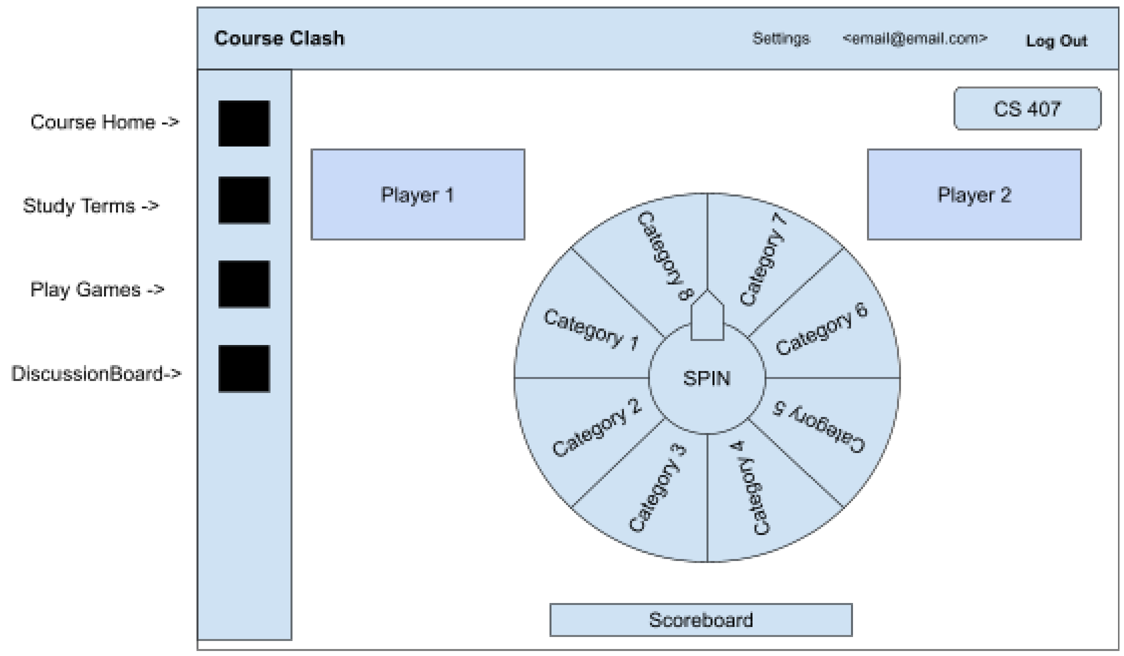
CS 180 Coordinator: <Coordinator Name>

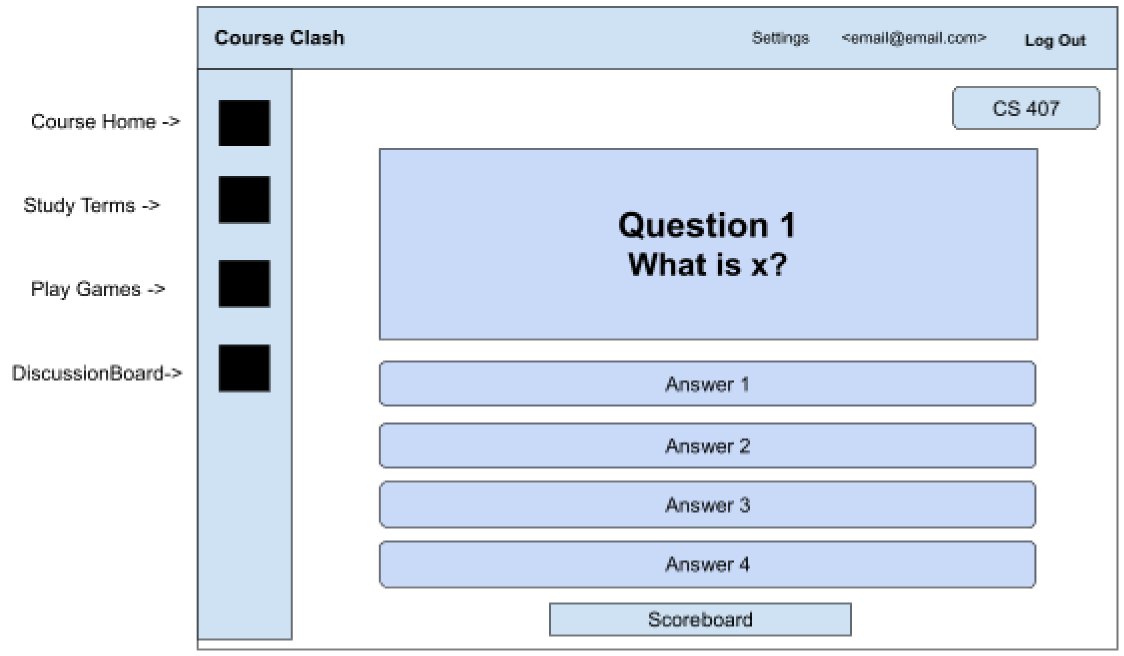
Course Description - CS 180

Course Homepage

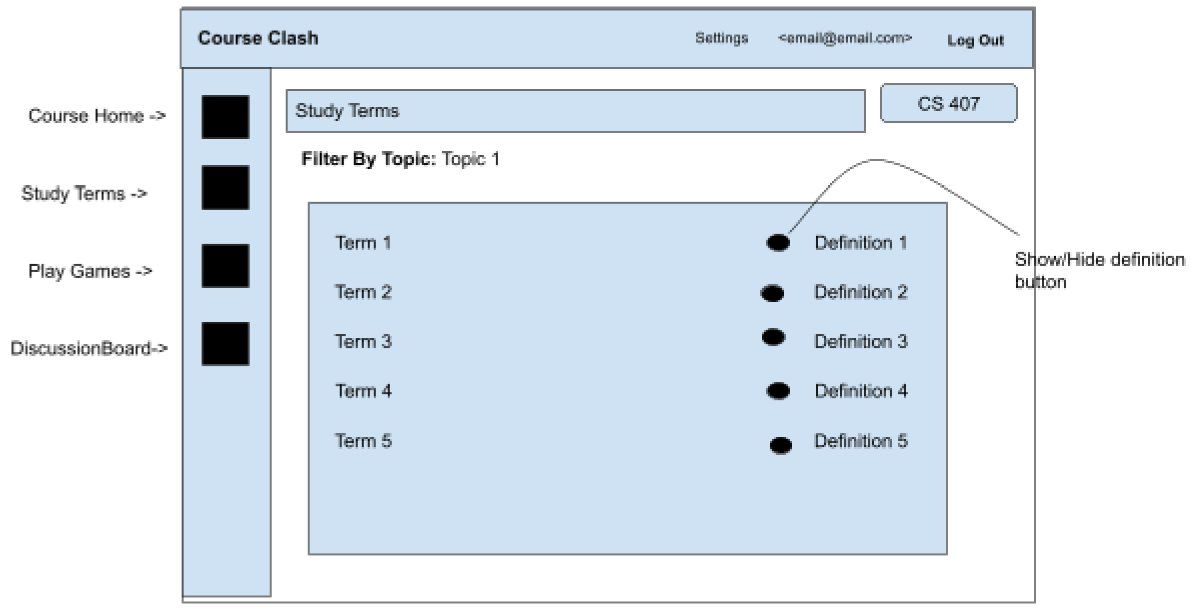


<u>Game Pages</u>

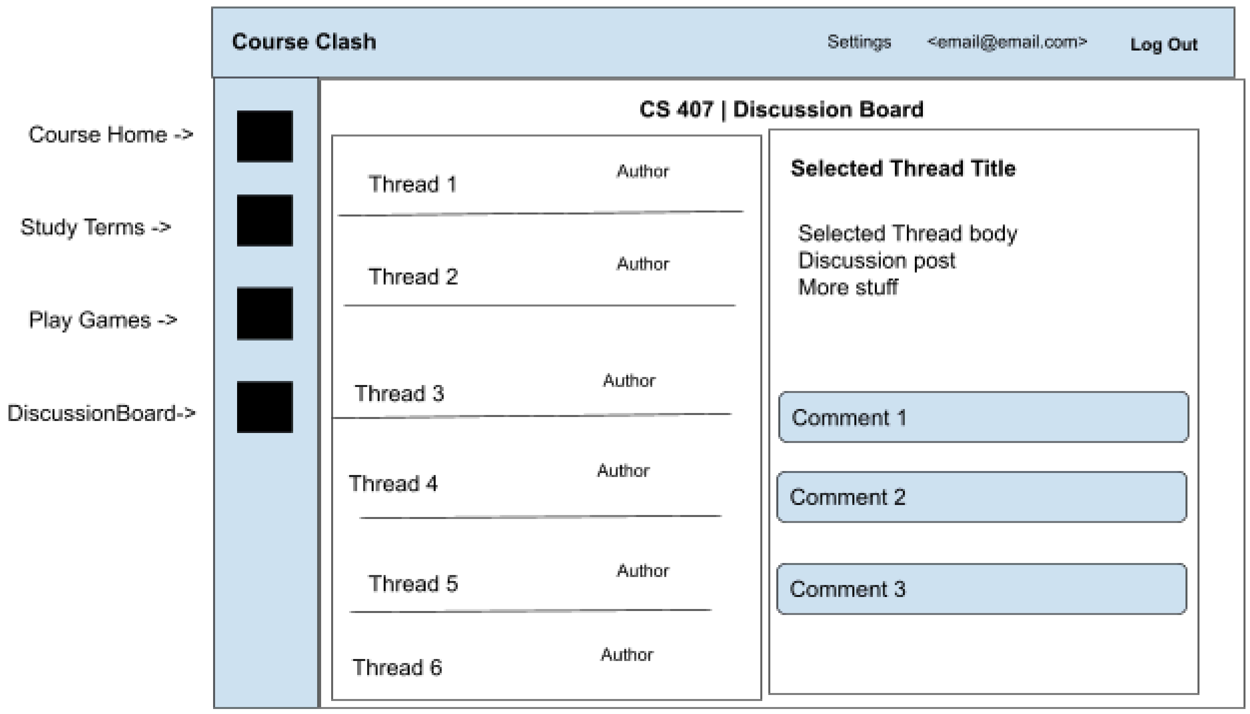

Course Clash
Settings
<email@email.com>
Log Out
Course Home ->
Study Terms ->
Play Games ->
DiscussionBoard->
CS 407
Player 1
Player 2
Category 8
Category 7
Category 1
Category 6
SPIN
Category 2
Category 5
Category 3
Category 4
Scoreboard


Course Clash
Settings
<email@email.com>
Log Out
Course Home ->
Study Terms ->
Play Games ->
DiscussionBoard->
CS 407
Question 1
What is x?
Answer 1
Answer 2
Answer 3
Answer 4
Scoreboard

<u>Study Term Page</u>



Course Discussion Board Page


Course Home ->
Study Terms ->
Play Games ->
DiscussionBoard->
Course Clash
Settings
<email@email.com>
Log Out
CS 407 | Discussion Board
Thread 1
Author
Thread 2
Author
Thread 3
Author
Thread 4
Author
Thread 5
Author
Thread 6
Author
Selected Thread Title
Selected Thread body
Discussion post
More stuff
Comment 1
Comment 2
Comment 3