CS180 Design Document

Overview

1. Architecture
    a. Diagram
2. Data Model
3. Interfaces

Architecture

Stack

- Composed of Node.JS/Express for the backend.
- MongoDB for data persistence.
- React.JS for frontend.
- WebSockets for video synchronization.
- Youtube iFrame API for video manipulation.

The Node.JS server is used to manage rooms, interface the database, and provide a real-time WebSocket API for synchronizing videos. The user experience would roughly look as follows. The user arrives at the home page and creates a new room. On the backend, a room ID is created along with a WebSocket for that room. The room ID and the WebSocket are then persisted into the database. If the user is logged in, the user's ID is then associated with that room. If the user does not have an account they can proceed as a guest and a user ID will be provided for them.

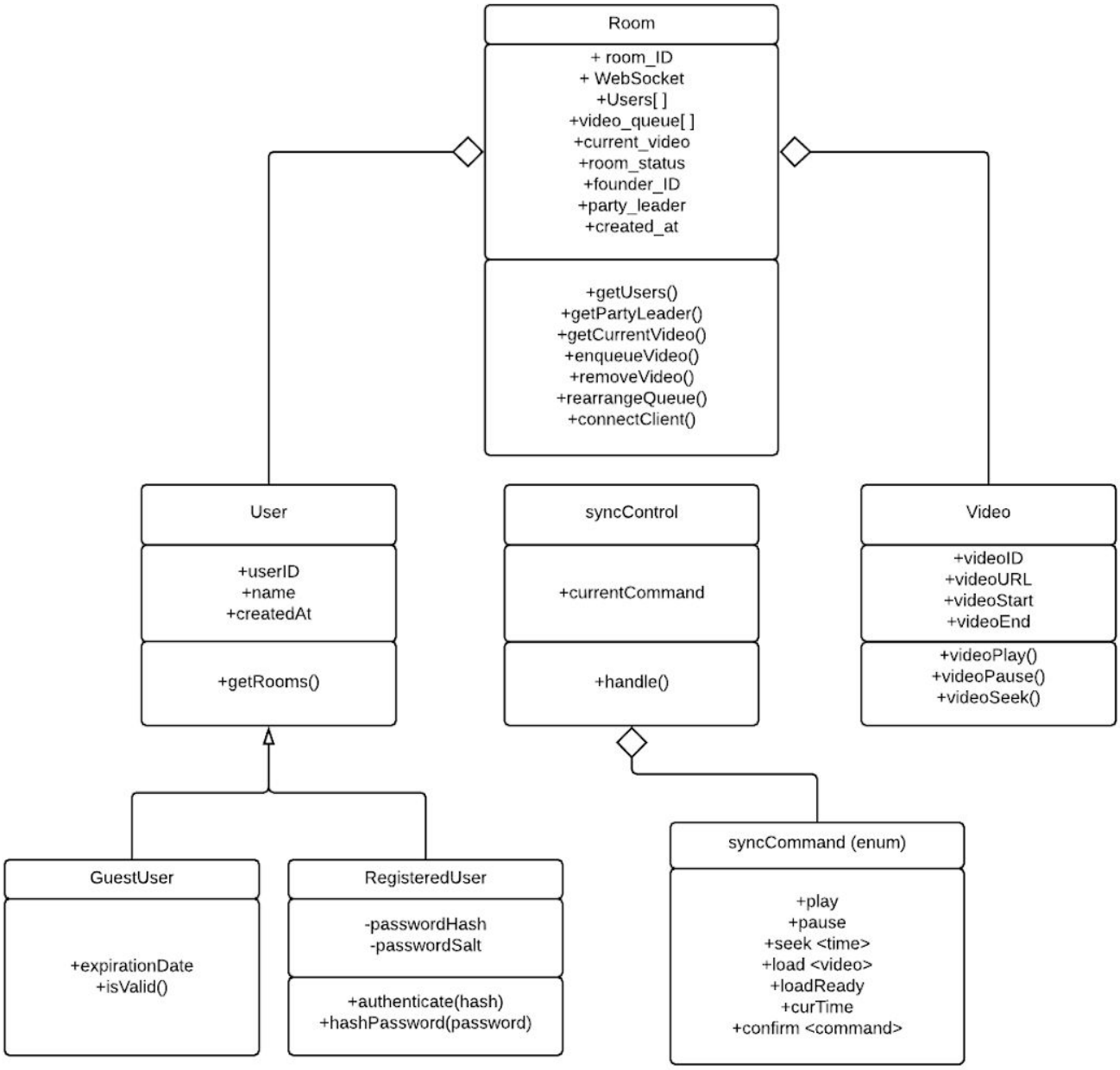
The user is now redirected to a new page in which he can view a blank player and an interface for adding videos. The room's founder is then selected as the party leader, the user that everyone else is synchronized to. The user can now share the room URL to other users so they can join the room. Each user upon joining the room will open a connection to the WebSocket for that specific room ID. Once the party leader selects a video a message is sent through the WebSocket to all users in the room, and the video is embedded on the page. Once all users have loaded the video, all users send a loadReady() command, indicating they're ready to play the video. All messages communicated over the WebSocket will be part of a standardized protocol called syncControl.

Every time the party leader interacts with the video, a message is sent using the syncControl protocol over the WebSocket and their interactions are replicated on the other user's machines. To ensure that the separate users are synchronized they will consistently send messages over the WebSocket, which the backend handles. Once a user is determined to no longer be in sync (a discrepancy of 0.5 seconds) they will be sent a seek command to match approximately where the party leader is in the video, accomodating for latency.

Data Model

In terms of data storage for our application, we do not have a lot of persistent data to store. We are using MongoDB to store our data. The database is simply going to contain registered users, which will be used to create persistent room URLs for those users. Passwords will be hashed, salted, and stored in the database in the same row as each user.

We will also be storing information about the room. We will store information in the room class. It will persist for up to 5 minutes after the last user leaves the room, after which the data is removed from the database.



Interfaces

API Routes

We have routes for interacting with the web application and they are specified as follows.

Request Type	**API Route**	**Description**
POST	api / login	Used to authenticate users.
POST	api / register	Creates a new user account.
GET	api / user /:id	Retrieves account information.
UPDATE	api / user /:id	Update account information.
GET	api / rooms	List all active rooms.
GET	api / room /:id	Gets information about a specific room.
POST	api / room	Creates a new room.
POST	api / enqueueVideo /:roomid	Adds avideo to room.

Sync Control Protocol

Sync Control is our custom standardized protocol for synchronizing videos among users connected to one of our rooms. It communicates over WebSockets by sending binary which represents commands. There will be a library for both the server and the client.

Protocol format looks as follows: <command><time><data>

The commands follow what is specified in the UML diagram on the second page.

Command	**Description**
play	Sends a play command to all clients.
pause	Sends a pause command to all clients.
seek <time>	Sends a seek command to all clients pointing to the specified time.
load	Dequeues the top video from the video queue and loads it for all clients.
loadReady	Used by clients to indicate that the video has been loaded.
curTime	Used by clients to send their current time in the video to the server.
confirm <command>	Used by clients to confirm that the command they were issued executed successfully.