CS180 Design Document

Table Of Contents

Architecture

Stack

- Composed of Node.JS/Express for the backend.
- MongoDB for data persistence.
- React.JS for frontend.
- WebSockets for video synchronization.
    - Utilizing both server and client socket.io libraries
- Youtube iFrame API for video manipulation.
- Youtube Data API for video information

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.

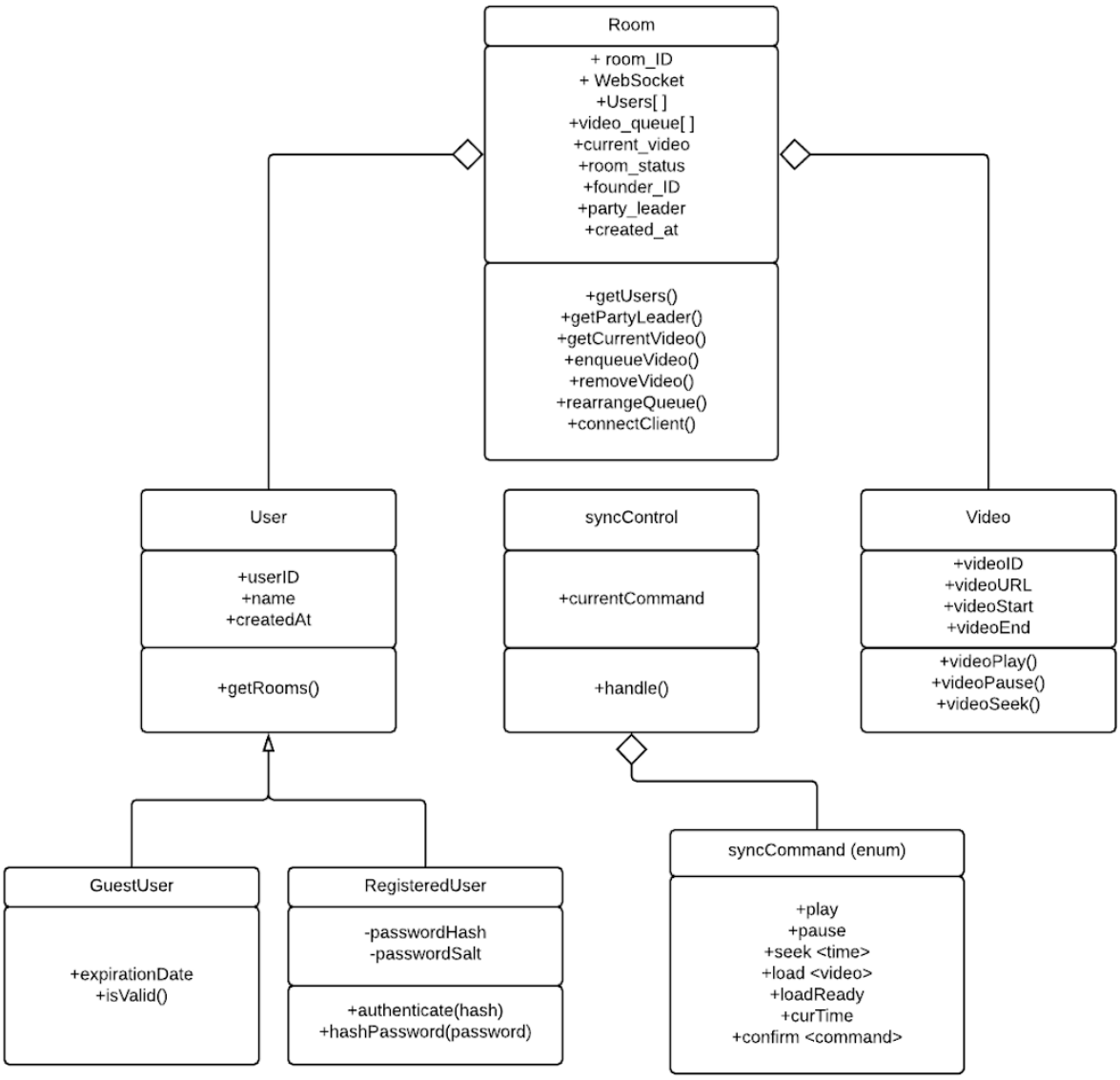
Every time the party leader interacts with the video, a message is sent using the syncControl protocol over the WebSocket and their interactions are replicated on the other user's machines. To ensure that the separate users are synchronized they will consistently send messages over the WebSocket, which 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

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 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.

UML Diagram


Room
+ room_ID
+ WebSocket
+Users[ ]
+video_queue[ ]
+current_video
+room_status
+founder_ID
+party_leader
+created_at
+getUsers()
+getPartyLeader()
+getCurrentVideo()
+enqueueVideo()
+removeVideo()
+rearrangeQueue()
+connectClient()
User
+userID
+name
+createdAt
+getRooms()
syncControl
+currentCommand
+handle()
Video
+videoID
+videoURL
+videoStart
+videoEnd
+videoPlay()
+videoPause()
+videoSeek()
GuestUser
+expirationDate
+isValid()
RegisteredUser
-passwordHash
-passwordSalt
+authenticate(hash)
+hashPassword(password)
syncCommand (enum)
+play
+pause
+seek <time>
+load <video>
+loadReady
+curTime
+confirm <command>

Video Model

A data model used to represent a video being played in the room

Video Model Attributes	
Attribute	**Description**
videoURL: <string>	The url of the video used to create the video
videoID: <number>	The primary key of the video
userID: <number>	The ID of the user who created the video
youtubeAPIKey: <string>	API key to use when calling the Youtube Data API

User Model

A data model used to represent a user account which can join many rooms

User Model Attributes	
Attribute	**Description**
userID: <number>	The primary key of the user in the database
userName: <string>	The public user name of the account
passwordHash: <string>	A hash of the uses password using sha-512
createdAt: <date>	A timestamp of when the user was created
Rooms: <array[]>	An array of roomID's that the user is currently in
isGuest: <boolean>	Indicates whether the account is a guest or not

Room Model

A data model used to represent a room. There can be many active rooms at a time so that groups of users can all watch videos together.

Room Model Attributes	
Attribute	**Description**
roomID: <number>	The primary key of the room in the database
founderID: <number>	The userID of the user who founded the room
partyLeaderID: <number>	The userID of the current party leader
syncRoom: <string>	The namespace the socket server is using for this room
users: <array[]>	An array of userID's that are currently in the room
videoQueue: <array[]>	A queue of youtube video IDs for the room
currentVideo: <string>	The youtube video ID of the currently playing video
roomStatus: <number>	Indicates the status of a room. See RoomStatus enum.
createdAt	A timestamp of when the room was created

Interfaces

The following describes the interfaces for the data models shown above.

Video Model Interface

Constructor(videoURL, userID)

- **Description:** Takes in videoURL and userID and automatically parses the videoURL to extract the youtubeID from the url.
- **VideoURL**: a string for the youtube URL
- **userID**: a number representing the userID of the user who is creating the video
- **Returns**: nothing

toJson()

- **Description:** Converts the model to a json object with all of its attributes
- **Returns: <**{}> Json object representing the video

save()

- **Description:** Saves the video to the database
- **Returns:** <promise> to a json record of the video returned form the database

retrieve()

- **Description:** Finds the video record in the database using the videoID attribute.
- **Returns:** <promise> to a json record of the video for the videoID set on the object.

parseVideoURL()

- **Description:** uses the videoURL attribute to parse out the youtube video id and saves this id on the object as videoID
- **Returns:** <string> the youtube video ID

getVideoID()

- **Description:** a getter for the videoID attribute
- **Returns:** <string> the youtube video ID for the video

setVideoID()

- **Description:** a setter for the video ID attribute
- **Returns:** nothing

getVideoDetails()

- **Description:** get the title and default thumbnail for the video from the Youtube Data API.
- **Returns:** <promise> to a json object with the thumbnail and title of the video

UserModel Interface

Constructor(isGuest)

- **Description:** initializes a new user either guest or regular user depending on isGuest
- **isGuest:** <boolean> whether or not the user is a guest
- **Returns:** nothing

generateUserID()

- **Description:** Generate a user ID for the user using a mongoDB auto increment work-around.
- **Returns:** <number> a userID for the record.

toJson()

- **Description:** Generate a json object with all attributes from the model.
- **Returns:** <{}> JSON object representing the user

save()

- **Description:** creates a new record of the user and saves it to the database.
- **Returns:** <promise> to a database record of the object

retrieve()

- **Description:** Finds the associated record for the data using userID attribute
- **Returns:** <promise> to a user object for the userID saved on the object

getRooms()

- **Description:** finds all the rooms a user is in and returns them
- **Returns:** <array[]> an array containing roomID's

Room Model Interface

constructor()

- **Description:** initializes a new room
- **Returns:** nothing

generateRoomID()

- **Description:** generates a new roomID using a mongoDB work-around
- **Returns:** <number> roomID for the room

toJson()

- **Description:** serializes the room into a json object using object attributes
- **Returns:** <{}> a JSON object representing the room

fromJson()

- **Description:** Deserializes a json object by loading data into object attributes
- **Returns:** nothing

getFounderID()

- **Description:** Returns a founderID for creating a new room. If founderID is -1 a temporary user account is created and that founderID is returned. Otherwise the founderID saved on the object is returned.
- **Returns:** <number> founderID

create()

- **Description:** creates a new room object by saving it to the database
- **Returns:** <promise> to a room object from the database

update()

- **Description:** Updates a room object as specified by the roomID attribute
- **Returns:** <promise> to a room object from the database

retrieve()

- **Description:** Fetches a room from the database as specified by the roomID attribute.
- **Returns:** <promise> to a room object from the database

getPartyLeaderID()

- **Description:** returns the partyLeaderID for the room
- **Returns:** <number> partyLeaderID

getCurrentVideo()

- **Description:** returns the youtube videoID of the currently playing video
- **Returns:** <string> youtube video ID

enqueueVideo(videoID)

- **Description:** adds a youtube video the room's video queue.
- **videoID:** <string> youtube video ID
- **Returns:** nothing

dequeueVideo()

- **Description:** Dequeues the top youtube video ID from the room's video queue.
- **Returns:** <string> the top youtube video ID of the video queue

getVideoQueue()

- **Description:** returns an array of video details. For every video in the video queue an object with a thumbnail and title of the video is returned.
- **Returns:** <promise> to <array[{}]> an array of video detail objects.

swapVideosInQueue(index1, index2)

- **Description:** swaps two video positions in the queue
- **Index1:** <number> index of video to be swapped
- **Index2:** <number> index of video to be swapped
- **Returns:** nothing

getRoomStatus()

- **Description:** returns the room status (playing, creating, deleting, etc)
- **Returns:** <number>

getCreationDate()

- **Description:** returns a timestamp of when the room was created
- **Returns:** <string> createdAt timestamp

RoomModelFactory Interface

getRoom(id)

- **Description:** finds a room object in the database as specified by the id given.
- **Id:** <number> the id of the number to ‘get’
- **Returns:** <promise> to a Video Model object

getAllRooms(limit)

- **Description:** finds all rooms in the database. Up to limit rooms are returned.
- **Limit:** <number> the maximum number of rooms to ‘get’
- **Returns:** <promise> to <Array[Video Model]>

updateRoom(id, doc)

- **Description:** Updates the room specified by ID with the given json doc
- **Id:** <number> roomID
- **Doc:** <{}> JSON object representing the room
- **Returns:** <promise> to an instantiated Room Model

deleteRoom(id)

- **Description:** deletes a room record in the database specified by the id passed in.
- **Id:** <number> roomID
- **Returns:** <promise> to a JSON object which indicates number of records changed

API Routes

We have routes for interacting with the backend API. 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 Event Library

Sync Control is our custom event library for communicating video player manipulations over a websocket. The follow events are sent to and from the server to the client.

Server Events:

There is a server library at: ***/src/api/lib/sync.lib.js***
These are the events emitted by the server to connected clients.

Server Events		
Name	**Data**	**Description**
loadVideo	videoID	Indicates a client should load the videoID
changeSpeed	speed	Alerts client to change playback speed
playVideo	-	Party leader has pressed play
pauseVideo	-	Party leader has paused the video
seekVideo	time	Party leader has seeked the video to time
updateQueue	-	Client should update queue by making API call
resVideo	videoID	The response event of reqVideo

Client Events

There is a client library at ***/src/frontend/src/lib/sync-lib.js***
These are the events emitted by the client to the server.

Client Events		
Name	**Data**	**Description**
join	{roomID, userID}	userID requests to join roomID
pauseVideo	{roomID, userID}	userID in roomID has paused video
playVideo	{roomID, userID}	userID in roomID has played video
reqVideo	{roomID}	Client requests current video in roomID
sync	{roomID, userID, curTime}	Sync info for a client's playback
seekVideo	{roomID, userID, time}	userID in roomID has seeked to time
doneVideo	{roomID, userID}	userID in roomID video playback has ended
updateQueue	{roomID, userID}	userID in roomID changed queue, emit to all