Tech Stack

Front End : React, videojs, npm packages

Back End : Node.js, Express, api routes with micro service architecture

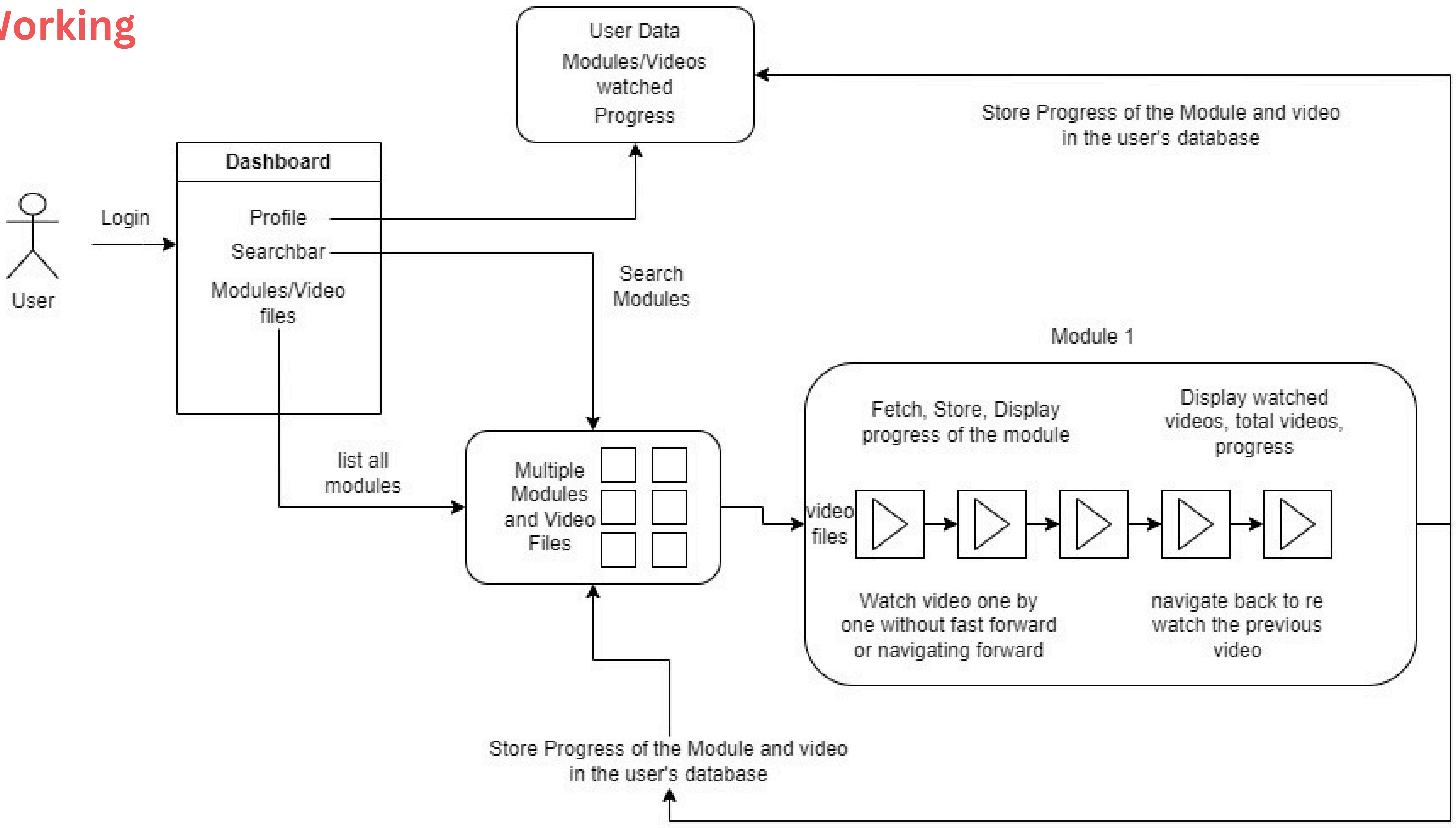
Database : Mongoose, Mongodb Atlas, Firebase for hosting videos

Hosting : Vercel and Render

Hosted Link : https://coursestream.vercel.app

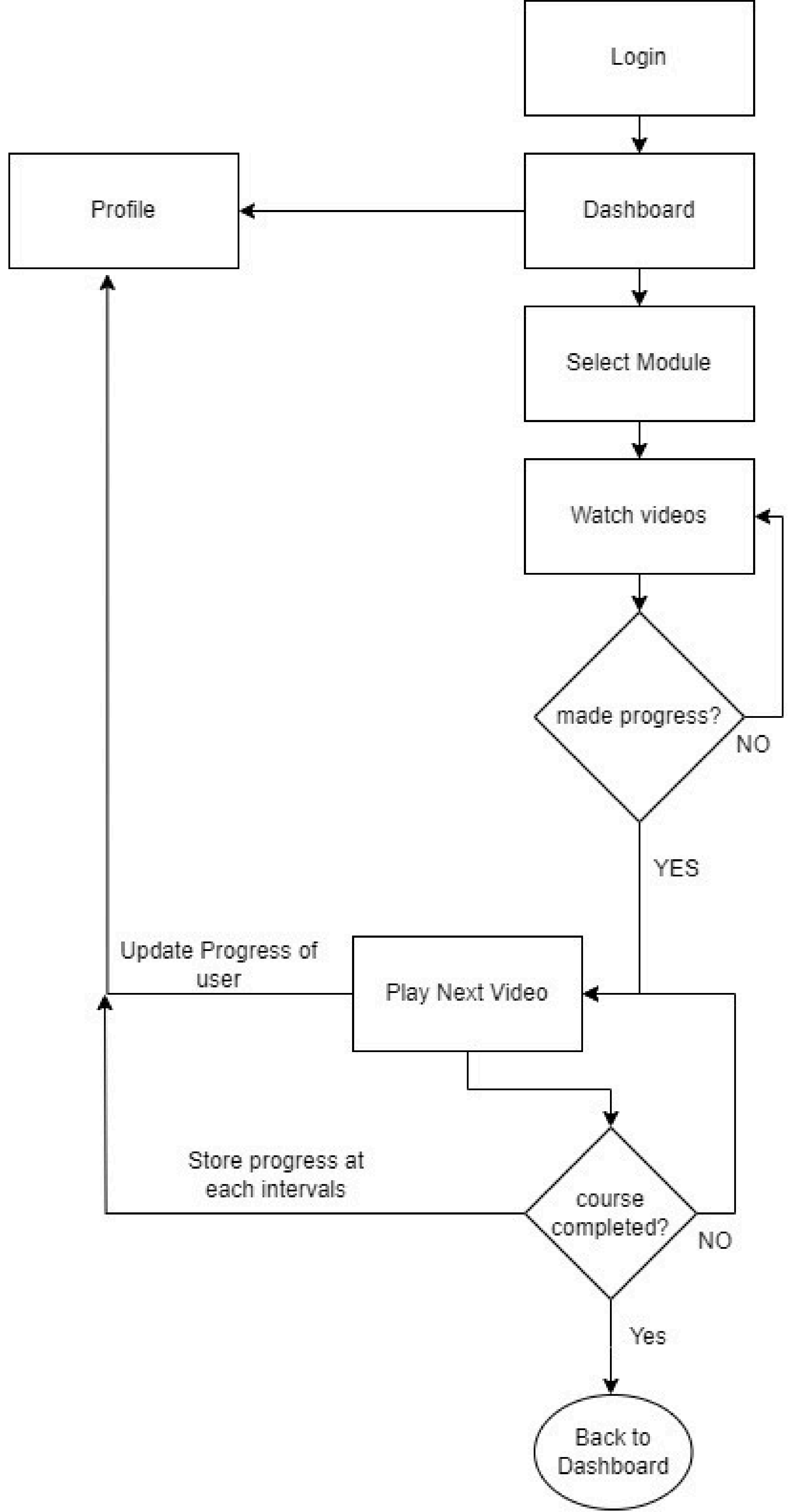
Github Link : https://github.com/Y-A-S-H-W-A-N-T/Internship_Assignment

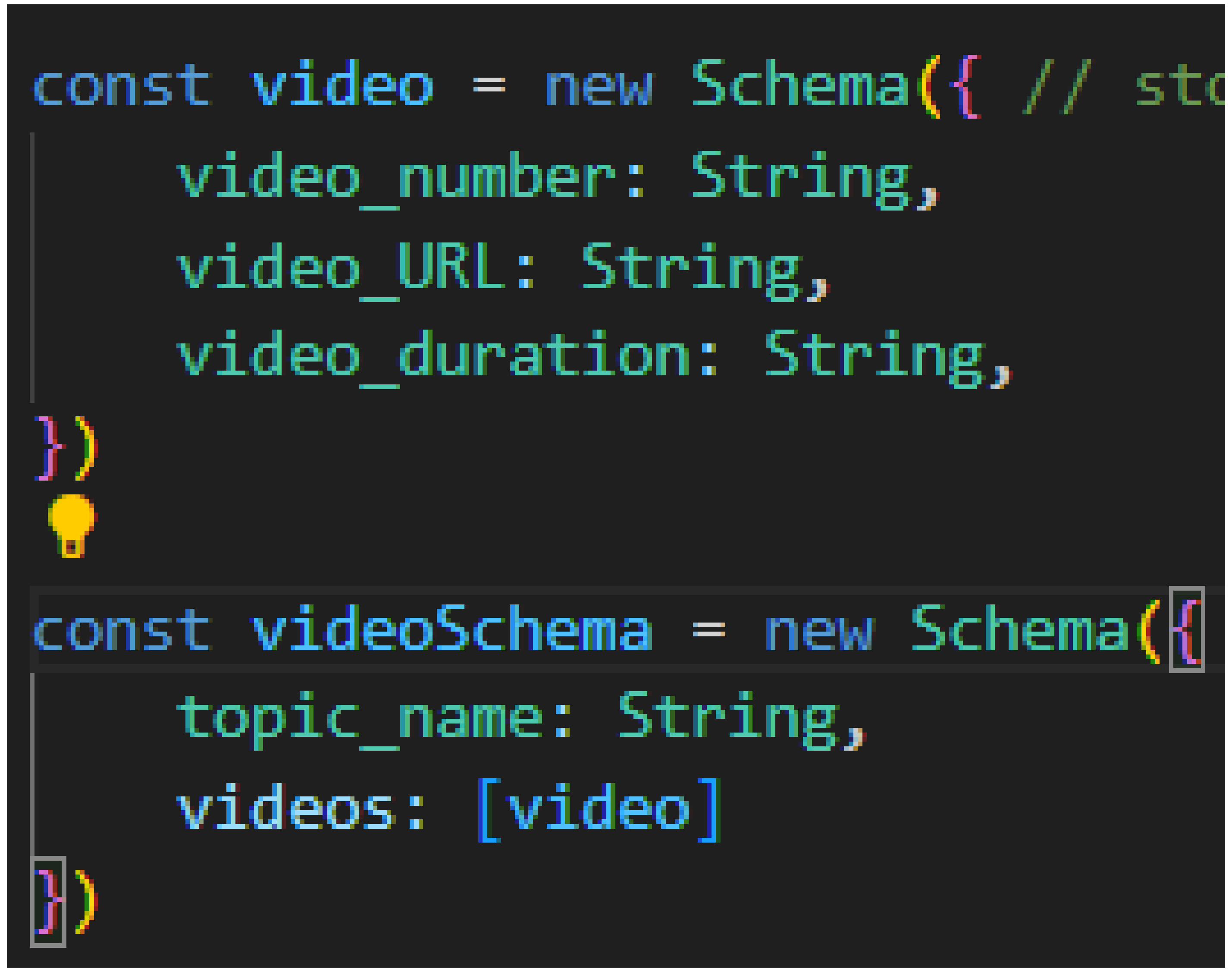
Working



User Data
Modules/Videos watched
Progress
Store Progress of the Module and video in the user's database
Dashboard
Profile
Searchbar
Modules/Video files
User
Login
Search Modules
Module 1
list all modules
Multiple Modules and Video Files
Fetch, Store, Display progress of the module
Display watched videos, total videos, progress
video files
Watch video one by one without fast forward or navigating forward
navigate back to re watch the previous video
Store Progress of the Module and video in the user's database

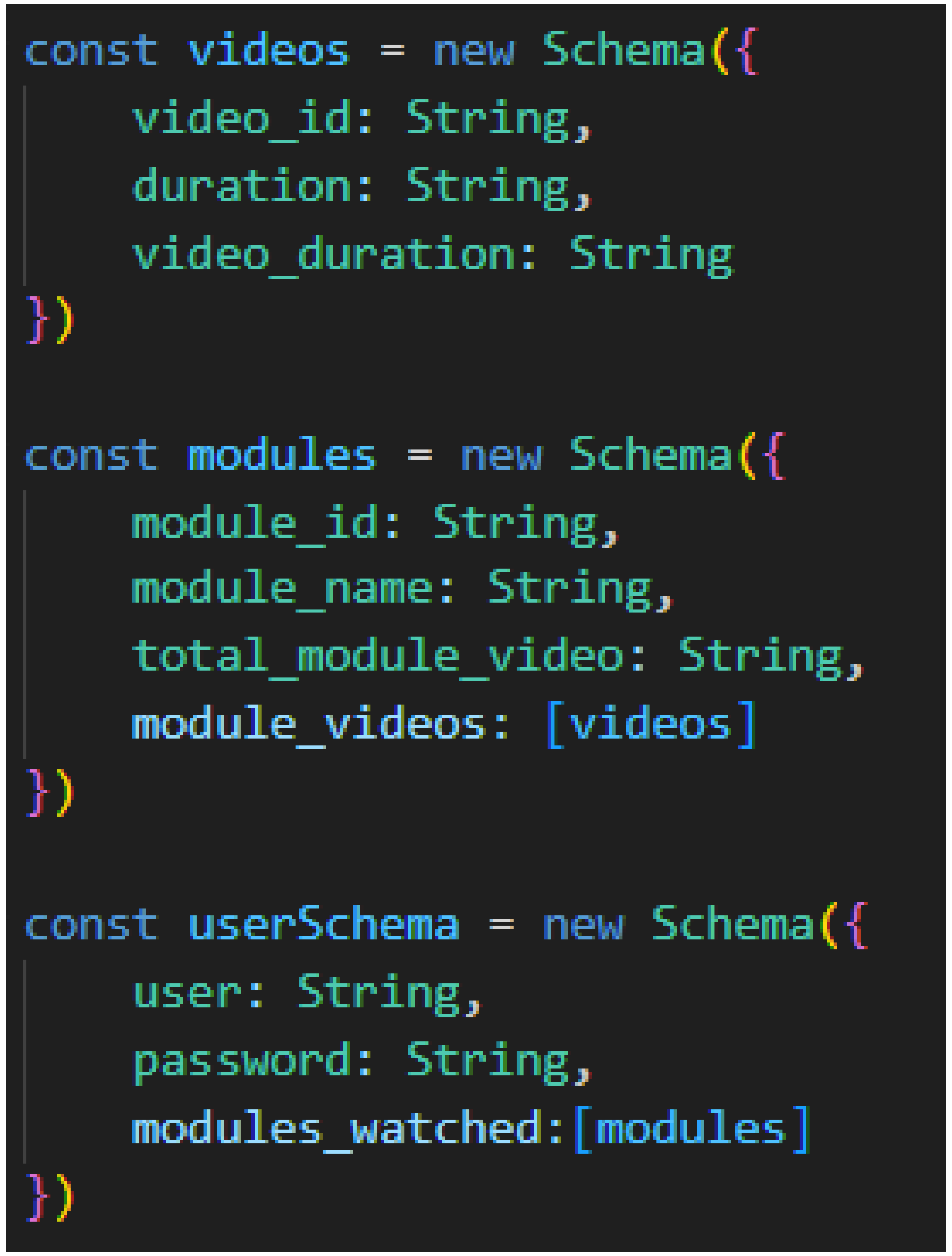
Flowchart


Login
Dashboard
Profile
Select Module
Watch videos
made progress?
NO
YES
Update Progress of user
Play Next Video
Store progress at each intervals
course completed?
NO
Yes
Back to Dashboard



```
const video = new Schema({ // sto
    video_number: String,
    video_URL: String,
    video_duration: String,
})

const videoSchema = new Schema({
    topic_name: String,
    videos: [video]
})
```

User & Videos Schema

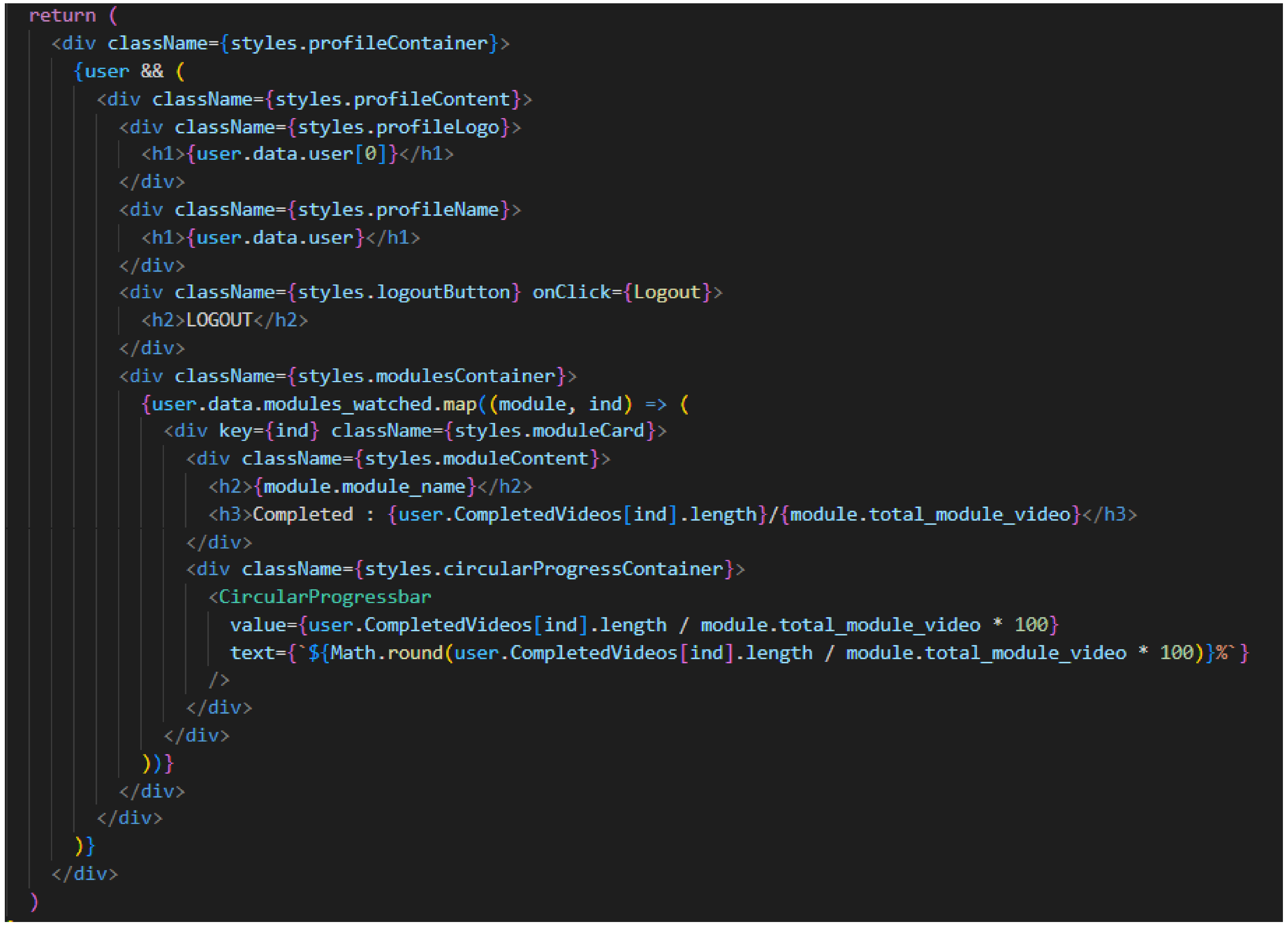


```
const videos = new Schema({
    video_id: String,
    duration: String,
    video_duration: String
})

const modules = new Schema({
    module_id: String,
    module_name: String,
    total_module_video: String,
    module_videos: [videos]
})

const userSchema = new Schema({
    user: String,
    password: String,
    modules_watched:[modules]
})
```

Keeping the schema minimal for optimal response and lightweight application

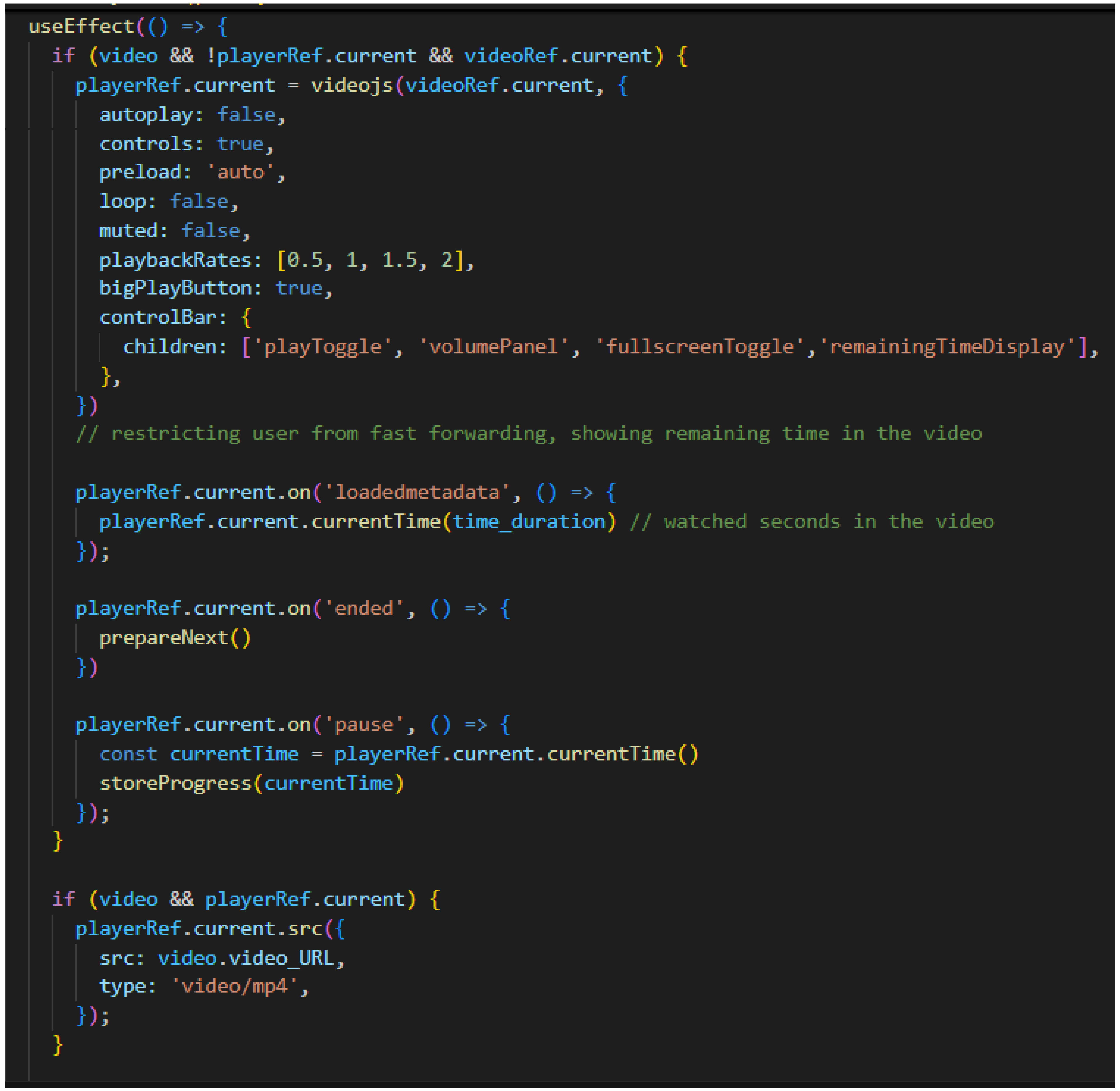
Code Snippets

Displaying user data and user progress in each modules he watched



```
return (
  <div className={styles.profileContainer}>
    {user && (
      <div className={styles.profileContent}>
        <div className={styles.profileLogo}>
          <h1>{user.data.user[0]}</h1>
        </div>
        <div className={styles.profileName}>
          <h1>{user.data.user}</h1>
        </div>
        <div className={styles.logoutButton} onClick={Logout}>
          <h2>LOGOUT</h2>
        </div>
        <div className={styles.modulesContainer}>
          {user.data.modules_watched.map((module, ind) => (
            <div key={ind} className={styles.moduleCard}>
              <div className={styles.moduleContent}>
                <h2>{module.module_name}</h2>
                <h3>Completed : {user.CompletedVideos[ind].length}/{module.total_module_video}</h3>
              </div>
              <div className={styles.circularProgressContainer}>
                <CircularProgressbar
                  value={user.CompletedVideos[ind].length / module.total_module_video * 100}
                  text={`${Math.round(user.CompletedVideos[ind].length / module.total_module_video * 100)}%`}
                />
              </div>
            </div>
          ))}
        </div>
      </div>
    )}
  </div>
)
```

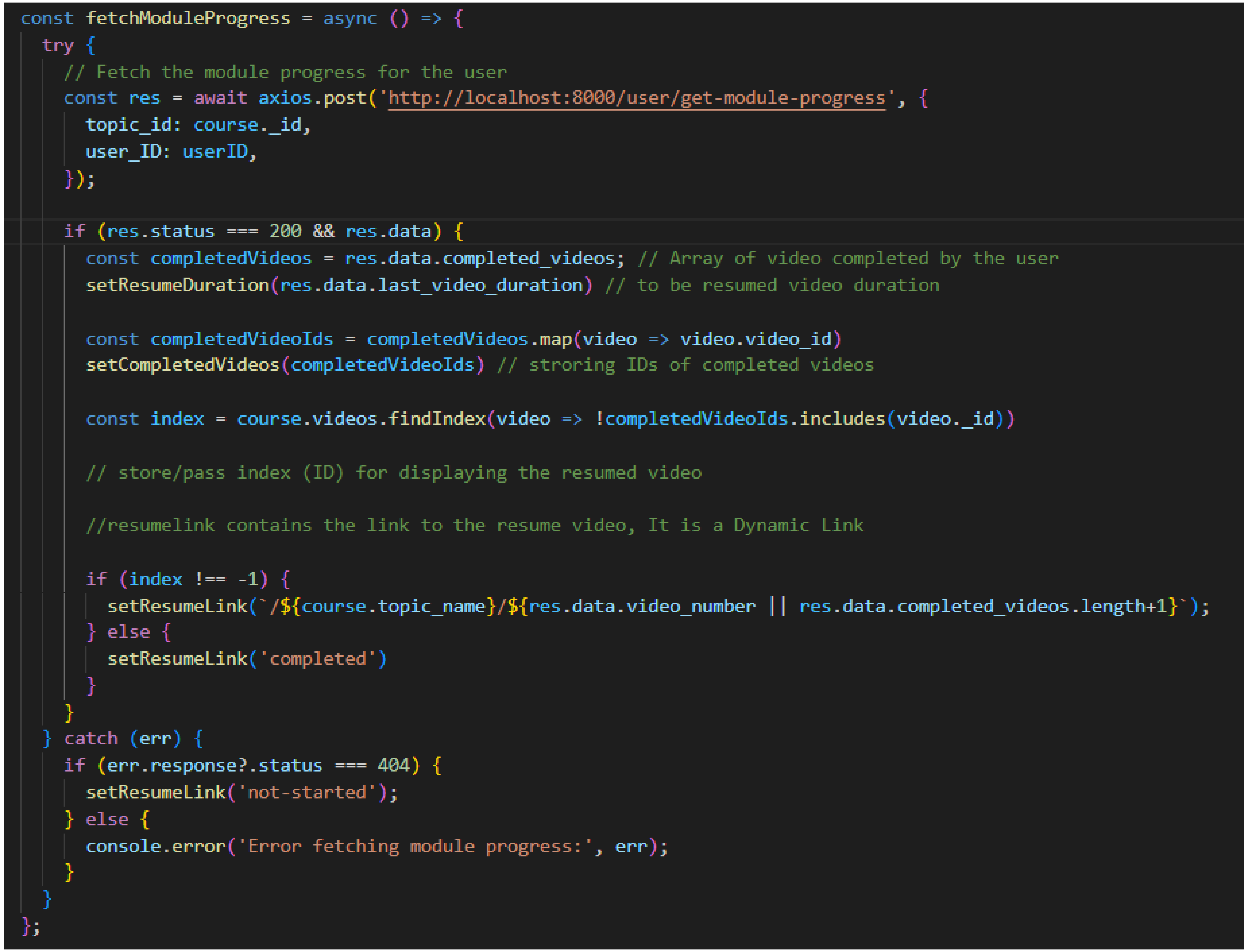
Playing video using VideoJS, restricting user from fast forwards and navigations in the video



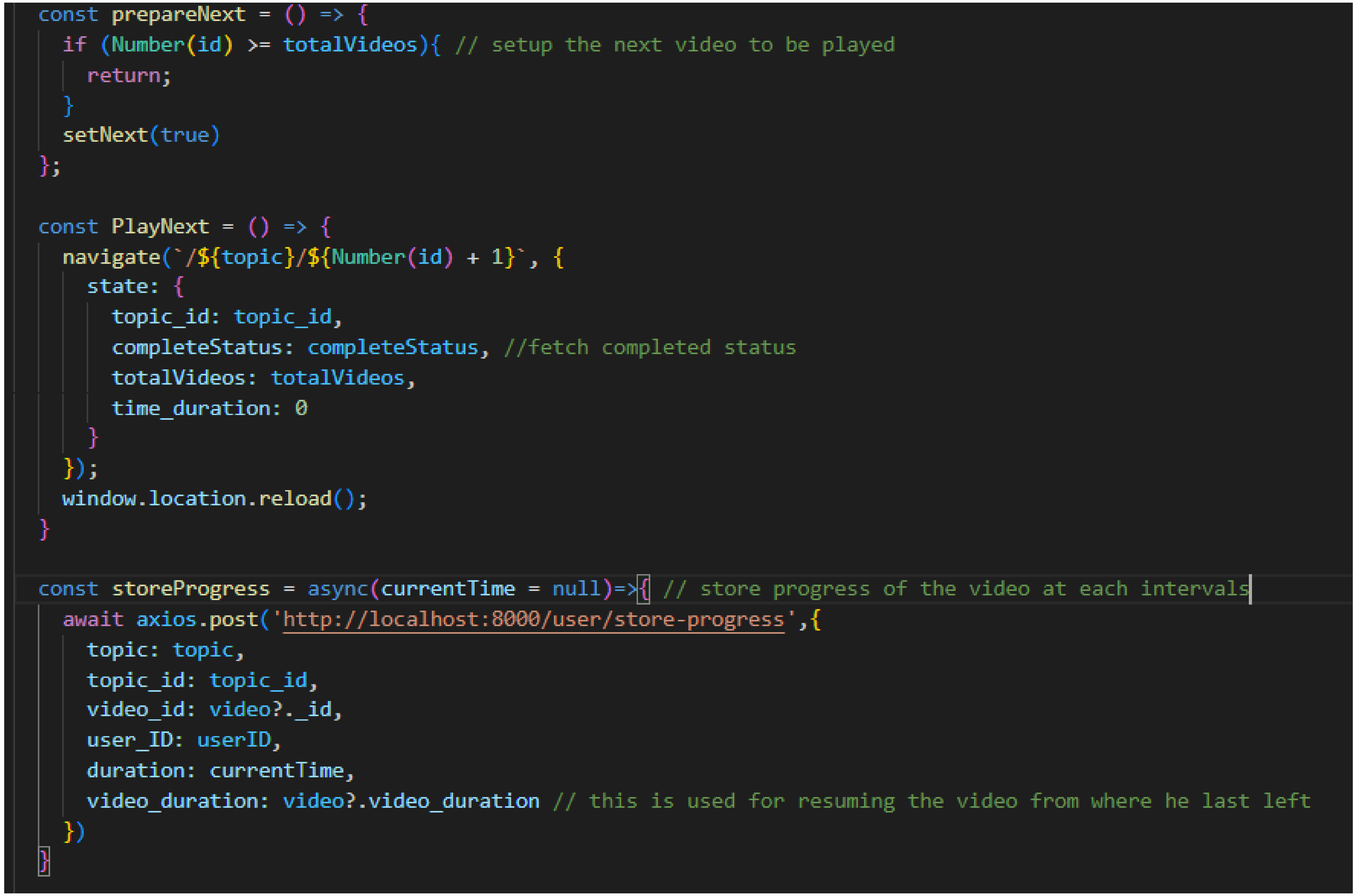
```
useEffect(() => {
  if (video && !playerRef.current && videoRef.current) {
    playerRef.current = videojs(videoRef.current, {
      autoplay: false,
      controls: true,
      preload: 'auto',
      loop: false,
      muted: false,
      playbackRates: [0.5, 1, 1.5, 2],
      bigPlayButton: true,
      controlBar: {
        children: ['playToggle', 'volumePanel', 'fullscreenToggle','remainingTimeDisplay'],
      },
    })
    // restricting user from fast forwarding, showing remaining time in the video

    playerRef.current.on('loadedmetadata', () => {
      playerRef.current.currentTime(time_duration) // watched seconds in the video
    });

    playerRef.current.on('ended', () => {
      prepareNext()
    })

    playerRef.current.on('pause', () => {
      const currentTime = playerRef.current.currentTime()
      storeProgress(currentTime)
    });
  }

  if (video && playerRef.current) {
    playerRef.current.src({
      src: video.video_URL,
      type: 'video/mp4',
    });
  }
```

Fetching last played video and its duration

Resuming video from where the user left



```
const fetchModuleProgress = async () => {
  try {
    // Fetch the module progress for the user
    const res = await axios.post('http://localhost:8000/user/get-module-progress', {
      topic_id: course._id,
      user_ID: userID,
    });

    if (res.status === 200 && res.data) {
      const completedVideos = res.data.completed_videos; // Array of video completed by the user
      setResumeDuration(res.data.last_video_duration) // to be resumed video duration

      const completedVideoIds = completedVideos.map(video => video.video_id)
      setCompletedVideos(completedVideoIds) // stroring IDs of completed videos

      const index = course.videos.findIndex(video => !completedVideoIds.includes(video._id))

      // store/pass index (ID) for displaying the resumed video

      //resumelink contains the link to the resume video, It is a Dynamic Link

      if (index !== -1) {
        setResumeLink(`/${course.topic_name}/${res.data.video_number || res.data.completed_videos.length+1}`);
      } else {
        setResumeLink('completed')
      }
    }
  } catch (err) {
    if (err.response?.status === 404) {
      setResumeLink('not-started');
    } else {
      console.error('Error fetching module progress:', err);
    }
  }
};
```



```
const prepareNext = () => {
  if (Number(id) >= totalVideos){ // setup the next video to be played
    return;
  }
  setNext(true)
};

const PlayNext = () => {
  navigate(`/${topic}/${Number(id) + 1}`, {
    state: {
      topic_id: topic_id,
      completeStatus: completeStatus, //fetch completed status
      totalVideos: totalVideos,
      time_duration: 0
    }
  });
  window.location.reload();
}

const storeProgress = async(currentTime = null)=>{ // store progress of the video at each intervals
  await axios.post('http://localhost:8000/user/store-progress',{
    topic: topic,
    topic_id: topic_id,
    video_id: video?._id,
    user_ID: userID,
    duration: currentTime,
    video_duration: video?.video_duration // this is used for resuming the video from where he last left
  })
}
```

Playing next video of the module in a sequence

Storing Progress of the video in user's database