



UNIVERSITY OF  
**LEICESTER**

**Department of Informatics**

**University of Leicester**

**CA7201 Individual Project**

**Online Video Platform**

Teja Bommineni

[tb302@student.le.ac.uk](mailto:tb302@student.le.ac.uk)

University ID: 199049133

**Final Report**

**Project Supervisor:** Prof. Elhag Anas Abdalla Osman

**Second Marker:** Dr. Paula Severi

Word Count: 10785

Date of Submission: 20<sup>th</sup> May 2022

## **Abstract**

Imagine sitting at home and opening a web application to share an important video to a person, who can only watch the video if he is registered and signed into the application. So, it was this intention that made me curious and select this project to create an application where users can login and upload, stream and share videos. In addition, users can comment on the video, reply to the comment, like, dislike and leave a rating to the video making it more interesting to use the application. All you need to require is a good internet connectivity to use the application.

## **Declaration**

All sentences or passages quoted in this report, or computer code of any form whatsoever used and/or submitted at any stages, which are taken from other people's work have been specifically acknowledged by the clear citation of the source, specifying author, work, date and page(s). Any part of my own written work, or software coding, which is substantially based on other people's work, is duly accompanied by clear citation of the source, specifying author, work, date and page(s). I understand that failure to do so amounts to plagiarism and will be considered grounds for failure in this module and the degree examination as a whole.

Name: Teja Bommineni

Date: May 20, 2022.

## Table of Contents

<b>1. Introduction</b>	5
1.1 Aim	5
1.2 Challenges faced	5
1.3 Risks	5
1.4 Background Research and Motive	6
<b>2. Prerequisites</b>	6
2.1 Prominent Requirements	6
2.1.1 Essential Features	6
2.1.2 Recommended Features	6
2.1.3 Optional Features	7
<b>3. Technical Specifications</b>	7
<b>4. Time Plan</b>	8
<b>5. Modus Operandi</b>	9
5.1 Project initial plan and Approach	9
<b>6. Design</b>	10
6.1 Use Case Diagram	10
6.2 Database	11
6.3 Architecture Diagram	12
<b>7. Project Final Output</b>	13
7.1 Frontend Modules	13
7.1.1 Registration Page	13
7.1.2 Login Page	14
7.1.3 Home Page	15
7.1.4 Home Button	16
7.1.5 Upload Video Component	16
7.1.6 Thumbnail Component	19
7.1.7 Video Page	20
7.1.7.1 Share video component	20
7.1.7.2 Like and dislike video features	21
7.1.7.3 Post Comment Component	24
7.1.7.4 Post reply component	25
7.1.8 Profile page	26
7.1.9 Sign out component	28
7.2 Frontend Libraries	29
7.3 Backend Modules	31

7.3.1 Models.....	31
7.3.2 Middlewares.....	32
7.3.3 Routing.....	32
7.3.3.1 Controllers.....	33
7.3.4 Utils.....	34
7.4 Backend Libraries .....	35
7.5 Build and Running the application.....	37
<b>8. Testing</b> .....	38
8.1 Functional testing:.....	38
8.1.1 API testing .....	38
8.2 Non-Functional testing.....	39
<b>9. Conclusion</b> .....	40
9.1 Challenges Faced .....	40
9.2 Feedback and observation.....	40
9.3 Further Improvement .....	41
<b>10. References</b> .....	41

# 1. Introduction

## 1.1 Aim

The aim of this project is to build a full stack web application using MERN Stack which comprises of (MongoDB, Express JS, React, Node JS) for uploading storing and viewing videos securely. There are many applications in the market currently for video storage but what separates this application from all of them is its security feature. User needs to be logged in in order to share the video through different social media platforms for other users and they need to be logged in to the application to watch the shared video. While uploading videos through account users are given the option to choose the category of the video so that when uploaded the video will be added to certain category. So, the videos are divided based on the category to make it easy for the users to search videos accordingly. Thumbnail filter also has been added so that users can choose an image which reflects the content in the video just by looking at it. To make the home page look beautiful animations are used and toastify is used to pop up the notifications according to the committed action. “The Chips” feature has also been added where specific word chips are provided on top of home page based on the video categories in the application and users can access all the videos of that category just by clicking on the chip. Users can comment, like, dislike and rate the videos accordingly. Overall, the application provides user friendly experience for the users and any of them can use it with ease just by following the steps.

## 1.2 Challenges faced

- Explore and learn new language and frameworks:
  - i) Express JS: Back-end framework of Node Js used for designing and building web applications and APIs.
  - ii) Node JS: A server-side code which helps to run JavaScript on server side and can act as a sever for content.
- Learn and acquire enough knowledge on designing a user-friendly web application.
- Following and implementing the best security measures to safeguard the sensitive information like user credentials.

## 1.3 Risks

**Risk 1:** The author has no prior experience in developing the backend part of the application.

**Action:** For building the full stack web application the author has to learn and implement the back-end framework (Express JS) and back-end language (Node JS) throughout the duration of the project.

**Risk 2:** The project involves storing of sensitive data which comprises of user details, their data, Videos information, etc., Mischievous users can try intrude in between to accumulate such sensitive data.

**Action:** As the project involves the sensitive data instead of opting for a normal hosting service, I opted for MongoDB high quality service so they provide complete security on my behalf and it is difficult to intrude into MongoDB. As an additional security measure, I used a library called bcrypt which encrypts user passwords and makes it difficult for the malicious users to go through them even though they intrude into MongoDB.

## 1.4 Motive and Background Research

I have always been interested to design and build websites. From the past year, I found myself in a space where I have been learning and building websites using java script and MERN Stack. As I am looking to build my career out of it, I have selected web development projects as my four preferences for my final project, and I have been allocated with 'Online Video Platform'. Once my project is finalized, I have started researching about other video streaming apps I noticed that they have different features for navigating throughout the application and watching videos. Based on these some speculations are made about the features which can be added to our application. One such speculation is that majority of them does not have this security feature which ask users to sign in to watch the video. So, in our application we made it mandatory to sign in order to watch the video to add additional security to the video. So even if an existing user shares a link for through any of the social media platforms to the end user, He (end user) can't watch the video without signing in to the application.

## 2. Prerequisites

### 2.1 Prominent Requirements

#### 2.1.1 Essential Features

- Authentication Feature: Users can register and login to the website
- Upload Video: Registered users can upload videos.
- Store uploaded videos securely using Video hosting provider.
- Display all videos in home screen along with the likes feature.
- Develop backend database to store users' information, video information and likes information, etc.,
- Advanced search feature with filter options like sort search results by using options like display videos using alphabetical order, video uploaded time, video length etc.,

#### 2.1.2 Recommended Features

- Share Video Option: Users can share their videos to different social media platforms.
- Profile page for registered users to access their personal information and upload history, etc.,
- Display videos category-wise.

- Animations in front end page to show the website more attractive.
- Display video uploading percentage using animations.
- Video download feature.
- Add security for shared video (User needs to login to watch the video).

### 2.1.3 Optional Features

- Show loading animation while home page displaying.
- Notification will popup when a user registers, uploads video successfully or likes a video.
- Delete video option.
- Display number of likes of a video.

## 3. Technical Specifications

Following my initial research, I have come across with certain technical specifications which are appropriate for developing this web application. You can find all the languages and frameworks used in the development of this web application below.

### Front end:

- i) React Js – Version 17.0.0: An open-source java script library used for building user interfaces.
- ii) React Hooks: React Hooks allow us to do a lot of things inside functional components like Use State, tapping to a life cycle method.
- iii) React Context: It provides clean and simple way to share state between components.
- iv) Yup: Yup is a java script schema validator used in this for forms validation.
- v) React Router DOM – version 6: Used for carrying out dynamic routing and navigation in web applications.

### Back end:

- i) Express JS: Back-end framework of Node Js used for designing and building web applications and APIs.
- ii) Node JS: Node JS is a server-side code which helps to run JavaScript on server side and can act as a sever for content.
- iii) NPM (Node Package Manager): A software registry that acts as a package manager for Node JavaScript.
- iv) MongoDB Atlas: Used for deploying, running and scaling MongoDB in the cloud platform of our choice.
- v) Json: Used for data transfer in the web applications that is from client to server and from server to client.
- vi) JWT: JWT (Json Web Token) is used for secure data transfer between client and server that is for authentication.

### Database:

- i) AWS s3: - A cloud storage on AWS used for safe and secure Image storage and video storage.

**Styles:**

- i) CSS3: - Latest version of CSS used for styling and describing the HTML content on the web page.
- ii) Bootstrap 4: - Latest version of Bootstrap an opensource CSS framework used for designing web applications.
- iii) Tailwind CSS: - An open-source CSS framework used for component-based design of web applications.
- iv) Font: - Presenting and for adding style to a web document.

## 4. Time Plan

A specific time plan is mandatory for the progress of the project to ensure that the project will be completed within the given time meeting all the requirements. So, a fortnightly plan is designed and certain milestones are set for every two weeks aiming to complete the whole project in 13 weeks. So, the initial phase of project started with initial setup of the project, designing and backend setup for login and registration forms. Moving forward MongoDB connection was given and AWS S3 bucket was set up and added logic to upload videos and all the features required to complete the application are added. All these features can be explored once the user logs in to the application. The entire plan for the project can be seen in the table below.

Table 1: Time plan for the project

Week	Module	Task
1-2	-	<ul style="list-style-type: none"><li>➤ Research, planning and Information gathering.</li><li>➤ Preliminary report writing.</li></ul>
3-4	Frontend	<ul style="list-style-type: none"><li>➤ Project initial setup.</li><li>➤ Adding required libraries.</li></ul>
5-6	Backend & Frontend	<ul style="list-style-type: none"><li>➤ Implement authentication features.</li><li>➤ Backend setup for registration and login.</li><li>➤ MongoDB connection.</li><li>➤ Login and registration forms.</li><li>➤ Interim report Submission.</li></ul>
7-8	Frontend	<ul style="list-style-type: none"><li>➤ Design forms for video upload.</li><li>➤ Setup Aws s3 bucket for video storage.</li><li>➤ Add routing using React Router- DOM.</li></ul>
9-10	Backend	<ul style="list-style-type: none"><li>➤ Add logic to upload videos to s3 bucket.</li></ul>



		<ul style="list-style-type: none"> <li>➤ Store them in MongoDB Atlas using unique ID.</li> <li>➤ Implement CRUD operations for videos in S3 bucket and MongoDB Atlas.</li> </ul>
11-12	Frontend	<ul style="list-style-type: none"> <li>➤ Display uploaded videos in homepage.</li> <li>➤ Add like, share options.</li> <li>➤ Download feature.</li> <li>➤ Profile update.</li> </ul>
13	Backend & Frontend	<ul style="list-style-type: none"> <li>➤ Tidy up UI.</li> <li>➤ Implement search.</li> <li>➤ Testing the application and Fix bugs.</li> <li>➤ Final report submission.</li> </ul>

## 5. Mode of Operation

### 5.1 Project initial plan and Approach

Initially the project started by researching the languages or frameworks I needed to learn to complete the project on time. So, accordingly a project plan was designed with a specific set of goals tasks to be completed for every two weeks as shown in table 1 of time plan. There were not many changes in the initial time plan and the project progressed smoothly according to it. The project initial setup was made and required libraries were added to start with. As the time went on login and registration forms have been added and MongoDB connection was setup. The next stage of project involved in developing all the required features of the application, create routing for them, add logic to upload videos to s3 bucket and testing these features all along. I was able to show the outcomes to the supervisor on fortnightly meetings by running the application on browser using “local host”. The project is built on Model-View-ViewModel (MVVM) architectural pattern which is used to differentiate logic of the program and UI controls.

## 6. Design

### 6.1 Use Case Diagram

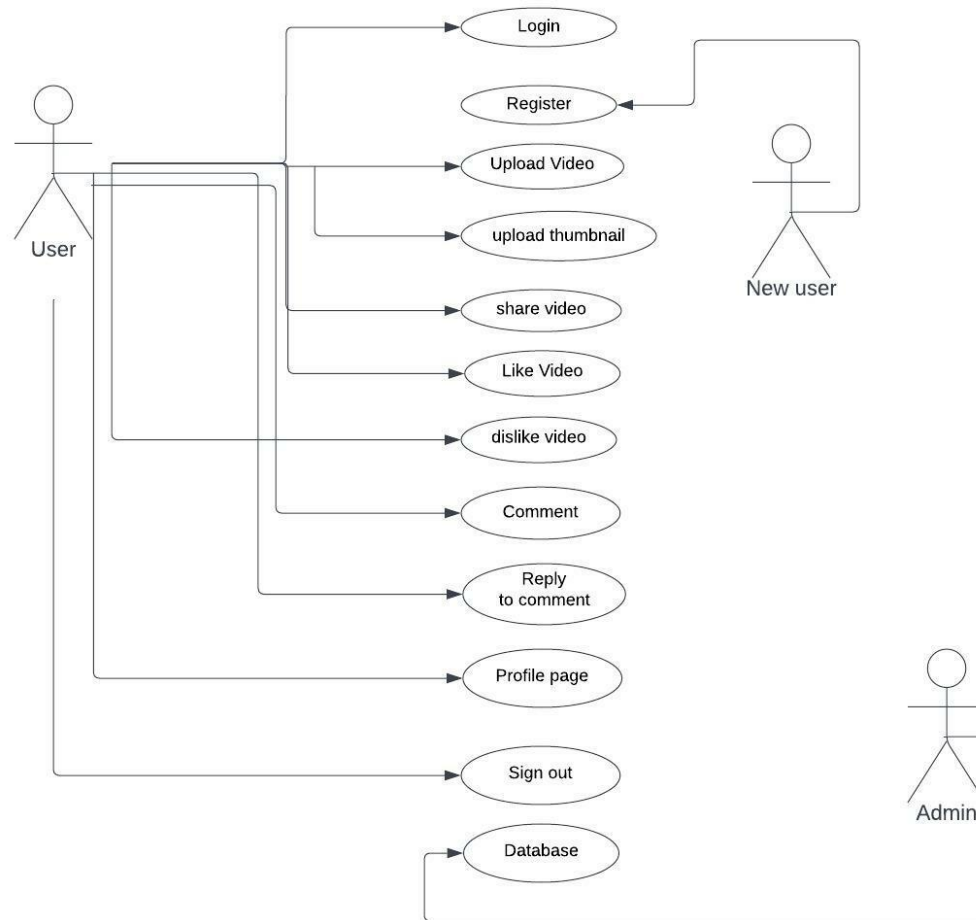


Figure 1: Use Case Diagram

Use case diagram in the above figure gives an overall understanding on functioning of the application. The diagram is also used to recognise the actions performed by the system and its actors. In the above use case diagram, we have three actors namely user, new user and admin. User can perform all the actions once he is signed into the system. Here is the list of services determined by the system.

**Login:** Verifies the log in credentials using email ID and password, it is mandatory for all the users.

**Register:** For a new user to login to the application it is mandatory to register to the application. Information required to register is full name, email ID and password. Only applicable to new users.

**Upload Video:** Existing users can upload any video of their choice once they sign in into the application.

**Upload Thumbnail:** Existing users are given an option to choose an image as thumbnail and is mandatory for the uploading video.

**Share Video:** Users are given an option to share the video to different social media platforms.

**Like & Dislike Video:** Allows users can like and dislike the video by clicking on like and dislike icons present at the right bottom of the video.

**Comment:** Users can comment on the video uploaded by any user in the comment section present below the video.

**Reply to a Comment:** Users can reply to the comment of other users in the video uploaded by a different user.

**Profile page:** Profile page section can be accessed by the user and it allows users to update their personal details.

**Sign out:** Users can sign out of the application at any given time using the sign out option.

**Database:** Data base can be accessed only by the admin if any of the details are to be manually entered into the database.

## 6.2 Database

The backend development of database is divided into two parts namely MongoDB and awss3 bucket. Out of these two MongoDB is completely used to store details of users, videos, likes, comments, etc., in this project whereas awss3 bucket is used solely for storing videos and images. MongoDB is a scalable, high-performance document database and processes data in semi structured format.

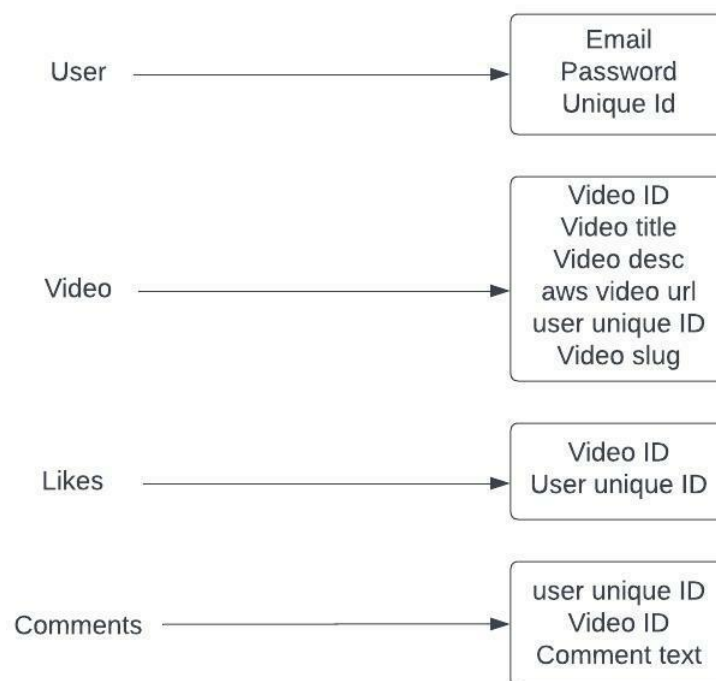


Figure 2: MongoDB Database Diagram

The above figure of MongoDB database gives an overall view of the database, in which format the data is stored in it and how it is stored. For every user or for every video uploaded unique ID is created and further actions can be committed with the help of that ID.

User: If user wants to register to the system, then the data will be sent to database using post method through the route “/create” where it verifies if whether the data is in the required schema. If everything is as per the validation rules it creates a Unique ID for the user and registers him to the application. These unique ID is checked every time using the route “/:id” if the user attempts to login and should be active if he has to sign in.

Video: Whenever the user uploads a video to the application from his account an unique video ID is created for that video. If the users click on the profile page the data will be sent to the database using post method through the route “/user/:id”, there it verifies the user details and get the videos uploaded by the user through “/:slug” which is associated with the user ID. Video slug is a simplified form of url which is created using video title to start with and ending with some unique text and is used to access the video.

Likes: Likes section contains video ID of that particular video and user ID. If any user likes the video to update the like count on the video page the data is passed on to the database through the route “/like/:videoID/:userID”, there it checks for the video and user IDs to update it.

Comments: Comment section contains user ID, video ID, commented text, video url and the time at which the comment was created and updated. It also has replies section where all the replies given by various users will be updated.

## 6.3 Architecture Diagram

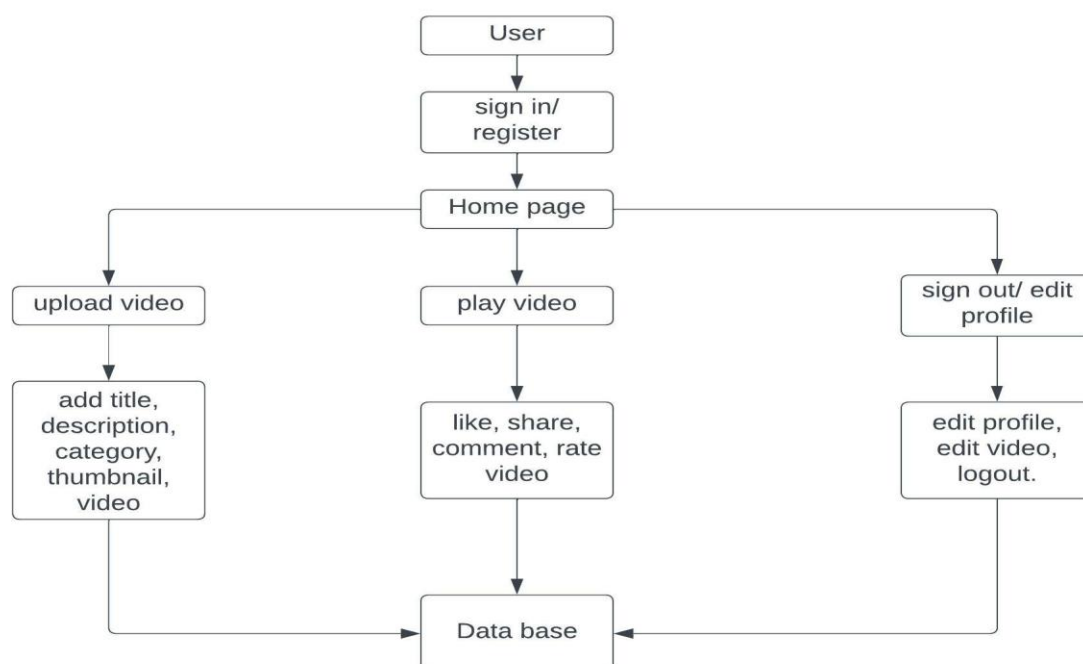


Figure 3: Architecture diagram.

The overall architecture diagram is represented in figure 3. The project is based on Model-View-ViewModel (MVVM) architectural pattern which is used to differentiate logic of the program and UI controls. The architecture diagram gives a complete overview of the application. For the user to sign in and go to the home page, play a video or to implement various features such as like, share, comment, rate video user has to communicate with the data base using backend API, same applies for the profile page and sign out options. Internet connectivity is mandatory for application to run on the browser.

## 7. Project Final Output

The project contains Register and Sign in components, to begin with. It contains various other components and functionalities which can be accessed by the user once he registers and logs in into the application. Overall, the pages in the application so far includes:

- Registration page
- Login page
- Home page
- Home button
- Upload video page
- Thumbnail Component
- Video page
  - i) Share video component
  - ii) Like video and dislike video components
  - iii) Comment Component
  - iv) Reply to comment component
- Profile Page
- Sign out component

### 7.1 Frontend Modules

#### 7.1.1 Registration Page

The registration page permits users to register for the application. It only allows the user to register only if he is a new user. If the user is already registered using the same email ID a notification will pop up saying that the user is already registered thereby denying the registration for the user. The information required for the registration are full name, email address, and password. Initial validation will be carried out by the application in order to verify the credentials given by the user. Firstly, it checks for the empty inputs, if any of the inputs are empty then an alert notification will be displayed notifying the user. If all inputs are filled out then it sends the request to registration API – “auth/register” where it checks if all the inputs are met with the required criteria and are within the server-side validation rules such as (same email ID cannot be used twice for registration). If everything is as per the validation rules then a Unique ID will be created for the user and the user will be registered to the application. The ID can also be used to check the user credentials whenever the user attempts to log in thereby giving access to the user to log in only if the ID is still active. After registration, once he clicks on the login button he will be redirected to the login page where the registered users can log in to the application.

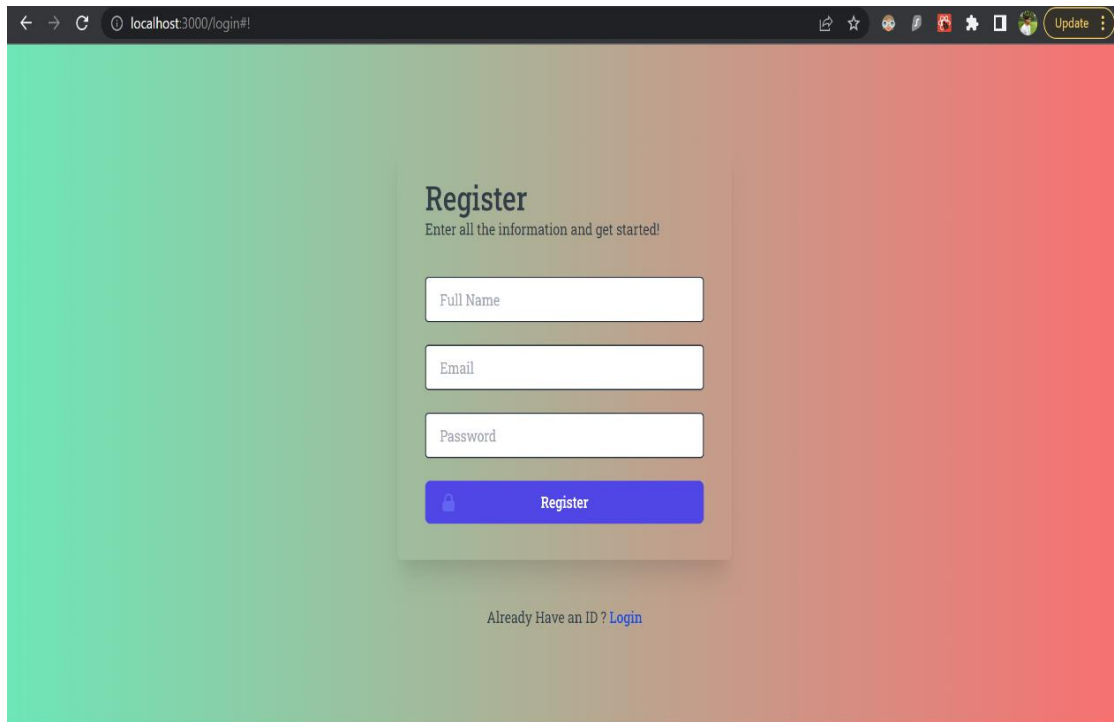


Figure 4 – Registration page for New Users

### 7.1.2 Login Page

Login page allows users to sign in to the application in order to use all the services of it. Full name and email address are required in order to login into the application. All the inputs should be filled out to sign in into the application. An error message will be displayed to alert the user if there is any empty input. Once all the inputs are filled out the credentials will be sent to login API – “auth/login” to check if they are present in the database. There it checks for the unique ID – which was created at the time of the registration. If the Object ID is still active then it authorizes sign in, else the user won’t be able to sign in.

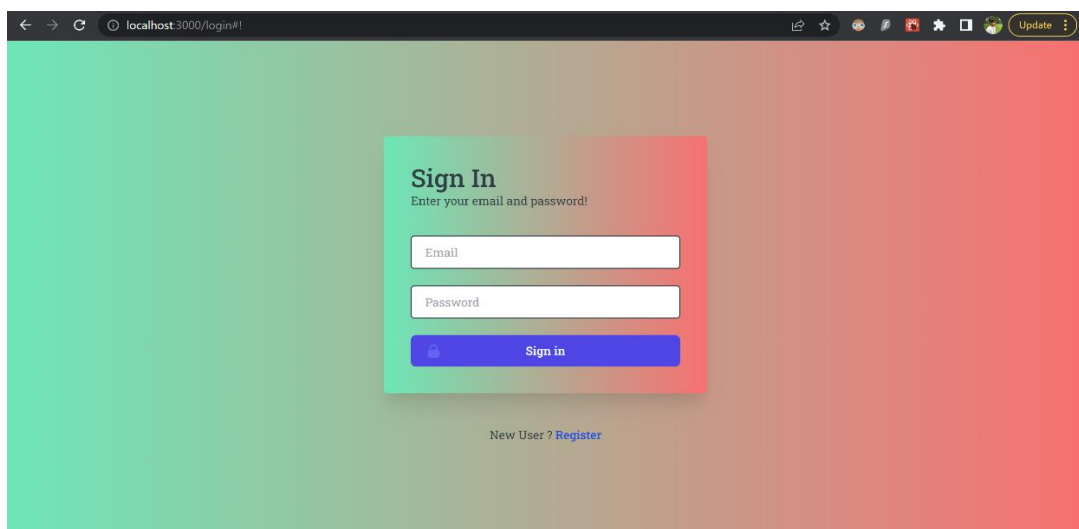


Figure 5 – Sign in page for existing users.

### 7.1.3 Home Page

Once the user gets access to sign in to the application, he will be redirected to the home page by default. All the videos which are uploaded by multiple users using this application are displayed in the home page. The home page is designed using a tool called Flexbox which is built in the latest version of Tailwind CSS. Flexbox comes into act when we are trying to display multiple elements in a same page at the same time. It is used as a part of responsive design of the application as it adjusts the size of the application according to the screen size and make it look attractive irrespective of the screen size. The home page functioning can be explained through the following code snippet.

Code snippet 1: Home page displaying all the videos in the application.

```
1  import React, { useContext, useEffect, useState } from "react";
2  import { Link, Redirect } from "react-router-dom";
3  import TopBar from "../../Components/Header/TopBar";
4  import ComponentLoader from "../../Components/Loader/ComponentLoader";
5  import { AuthContext } from "../../Providers/AuthProvider";
6  import { GetVideos } from "../../service/service";
7  import VideoCard from "../Components/VideoCard";
8
9  const HomePage = ({ history }) => {
10     const { user, authLoading } = useContext(AuthContext);
11     const [videos, setVideos] = useState([]);
12     const [loading, setLoading] = useState(true);
13
14     const getData = async () => {
15         const response = await GetVideos();
16         console.log(response);
17         if (response && response.isSuccess) {
18             setVideos(response.data.videos);
19         }
20         setLoading(false);
21     };
22
23     useEffect(() => {
24         getData();
25     }, []);
26
27     if (!authLoading && !user) return <Redirect to="/login" />;
28 }
```

Home page acts as the parent class for all the existing pages, that is the functionalities of all other pages have been imported and used in the home page using import statement. React Hooks like useState, useEffect and useContext are imported from react and used in the home page. In simple terms useState is used to keep track on changes made to the initial state as it is tend to alter any time. useContext acts as the manager for useState by passing data to the entire hierarchy and useEffect as the name suggests effects the state change. Similarly, TopBar, ComponentLoader, AuthContext, GetVideos, VideoCard are imported from other modules. Using useState() initial value is set as “true”, loading is the current value and setLoading is the function used for updating its value. Similarly using useState initial value is set as null array “useState([])”, videos is the current value and setVideos is the function used for modifying its value. getData() function is called and used inside useEffect(), async() is used infront of the function as it returns a promise and waits for the response i.e., get videos. If the response is success the initial state of useState([]) is updated to “videos” which is the current state using the function “setVideos(response.data.videos)” and initial state of useState(true) is updated to current

state “loading” using the function `setLoading(false)` and user will be redirected to the home page and all the videos will be displayed in the home page.

If not authloading and not a user “if (!authloading && !user)” then the user will be redirected to the login page using the route “/login”.

When the user clicks on a video in order to play the video, the video page of that particular video will open. This process of redirecting from home page to video page is defined as navigation or ‘routing’. The code for the link which is responsible for all of this is mentioned below.

```
<Link to={`video/${video.slug}`}>
  <VideoCard video={video} />
</Link>
```

While giving routing to the video slug is used. Slug is a unique string which starts with video title followed by some unique characters. The slug is used as a part of url so that the users can get a good idea of the type of video just by looking at the url when it is shared across different social media platforms.

#### 7.1.4 Home Button

The home button is present in the header section of the application. It helps in routing between different pages to the home page. After the user is done with uploading videos, watching videos, updating user details or performing any other type of task. Once you click on the home button it helps in redirecting the user directly to the home page as it is linked to “/” route and exact path of the “/” component is home page. It is clearly shown in the code sample provided below.

```
<Route exact path="/" component={HomePage} />
```

#### 7.1.5 Upload Video Component

Upload video component helps users to upload videos to the application. Once the users click on the upload video button present in the header section of the application they will be redirected to upload video page which has multiple options to choose for uploading a video namely Title, description, video and thumbnail. Video title, description, video file which is ready to be uploaded and a thumbnail image is to be chosen in each of the sections respectively which helps in identifying the type of video uploaded. All the fields need to be filled up with the required information in order to upload a video else an error message will be displayed below each section suggesting to fill that particular section as



shown in figure 6 below.

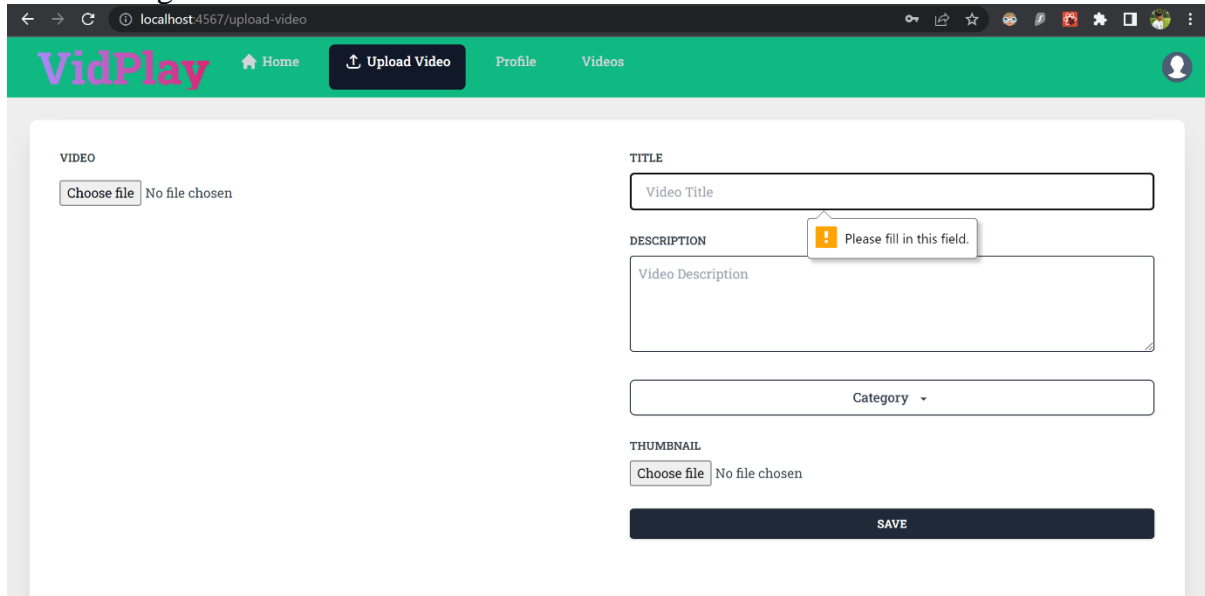
The screenshot shows a web browser at localhost:4567/upload-video. The page has a green header with the 'VidPlay' logo and navigation links: Home, Upload Video, Profile, and Videos. A user profile icon is in the top right. The main form is titled 'VIDEO' and contains several sections: 1. A file selection area with a 'Choose file' button and the text 'No file chosen'. 2. A 'TITLE' section with a text input field labeled 'Video Title'. 3. A 'DESCRIPTION' section with a text area labeled 'Video Description'. A tooltip with an exclamation mark icon and the text 'Please fill in this field.' is pointing to the description area. 4. A 'Category' dropdown menu. 5. A 'THUMBNAIL' section with another 'Choose file' button and the text 'No file chosen'. At the bottom of the form is a dark blue 'SAVE' button.

Figure 6 : Upload video form

A java script package called toastify has been installed and used in the development of the application as it produces notifications in toaster style. Technical side of uploading video and thumbnail functionality can be explained with the code snippet below.

Code Snippet 2: Code explaining functionality of upload video

```
const UploadVideoPage = ({ history }) => {  
  const { user, authLoading } = useContext(AuthContext);  
  const [video, setVideo] = useState(null);  
  const [thumbnail, setThumbnail] = useState(null);  
  const [saving, setSaving] = useState(false);  
  const [uploadingVideo, setUploadingVideo] = useState(false);  
  const [uploadingThumb, setUploadingThumb] = useState(false);  
  const [uploadPercentage, setUploadPercentage] = useState(0);  
  
  if (!authLoading && !user) return <Redirect to="/login" />;  
  
  const handleUploadVideo = async (e) => {  
    setUploadingVideo(true);  
    const files = e.target.files;  
    try {  
      const response = await UploadVideo(files[0], setUploadPercentage);  
      console.log(response);  
      if (response && response.isSuccess) {  
        setVideo(response.data);  
      }  
    } catch (err) {  
      if (err.response.status === 500) {  
        console.log("There was a problem with the server");  
      } else {  
        console.log(err.response.data.msg);  
      }  
    }  
    setUploadingVideo(false);  
    setUploadPercentage(0);  
  }  
}
```

For uploading video using `useState()` initial value is set as false and `uploadingVideo` is the current value of the state and `setUploadingVideo` is the function used for updating the value of the state. Similarly, `setVideo`, `setSaving`, `setThumbnail`, `setUploadingThumb` and `setUploadingPercentage` are the functions used for updating the current values of their states which are video, saving, thumbnail, uploadingThumb and uploadingPercentage respectively. Any errors in uploading and the person is not a user then the person will be redirected to login page using the route “/login”. AsyncI operation used for the function `handleUploadVideo` as it returns a promise on click that is it opens the files in the system using the operation “`files = e.target.files;`”, `setUploadingVideo` as true and awaits for the response which shows uploading video file and uploading percentage. If the response is success, then chosen video file will be added to the page using `setVideo(response.data)`, if there is any problem with the server or it catches any error `setUploadingVideo` will be corrected to false and upload percentage as zero.

Code Snippet 3: Code showing creation of video data and usage of toastify.

```
const data = {
  title: e.target.title.value,
  description: e.target.description.value,
  video: video,
  thumbnail: thumbnail,
  user: user?._id,
};

console.log(data);
const response = await CreateVideo(data);
if (response && response.isSuccess) {
  toast.success('Video Uploaded!', {
    position: toast.POSITION.BOTTOM_RIGHT,
    autoClose: 3000,
  });
  history.push("/");
} else {
  toast.error(`${response.data.toString()}!`, {
    position: toast.POSITION.BOTTOM_RIGHT,
    autoClose: 3000,
  });
}
console.log(response);
setSaving(false);
} else {
  toast.error('Please Upload Video and Thumbnail!', {
    position: toast.POSITION.BOTTOM_RIGHT,
    autoClose: 3000,
  });
}
```

If the video and thumbnail are added then using `useState()` initial value of false is updated to current value saving which is true using `setSaving` function. Once data is logged it awaits for the response which in this case is the creation of video data. If the response is success, then a notification will pop up at the bottom of the application as we have used toastify, stating that “Video uploaded” and it closes automatically in 3 seconds and the video is pushed to home page using `history.push("/")`. If the video file is not uploaded or there is any error while uploading then error notification will pop up and no action will take place.

### 7.1.6 Thumbnail Component

Thumbnail component is displayed at the bottom of the upload video page. While uploading a video user is given an option to choose a thumbnail image for the video, which is mandatory. Thumbnail component helps the users to choose a background image for the video, the image acts as a face of the video and will be displayed on the back of the video when the video is uploaded. This component has been added in order to make it relatively easier for the users to identify the type of video on the basis of chosen image just by looking at the video. The following code snippet explains the functionality of the thumbnail component.

Code Snippet 4: Code explaining functionality of thumbnail component

```
const handleUpload = async (e) => {
  setUploadingThumb(true);
  const files = e.target.files;
  try {
    const response = await UploadFile(files[0]);
    console.log(response);
    if (response && response.isSuccess) {
      setThumbnail(response.data);
    }
  } catch (err) {
    if (err.response.status === 500) {
      console.log("There was a problem with the server");
    } else {
      console.log(err.response.data.msg);
    }
  }
  setUploadingThumb(false);
};

const handleSave = async (e) => {
  e.preventDefault();

  if (video && thumbnail) {
    setSaving(true);

    const data = {
      title: e.target.title.value,
      description: e.target.description.value,
      video: video,
      thumbnail: thumbnail
    };
  }
};
```

The functionality of thumbnail component is similar to upload video component. It uses asyncI operation used for the function handleUpload as it returns a promise on click and opens the files in the system using the operation “files = e.target.files;”, setUploadingThumb as true and waits for the response which shows uploading thumbnail file and uploading percentage. If the response is a success then chosen video file will be added to the page using setThumbnail(response.data), if there is any problem with the server or it catches any error setUploadingThumb will be corrected to false and use will be redirected to the video page.

### 7.1.7 Video Page

When the user clicks on the video in order to play the video then the video page opens up along with the video displaying in the centre of the page. Video page contains several functionalities such as like and dislike videos, comment under the video, reply to the comment and share video. The figure 6 below shows the entire video page along with all the features in it.

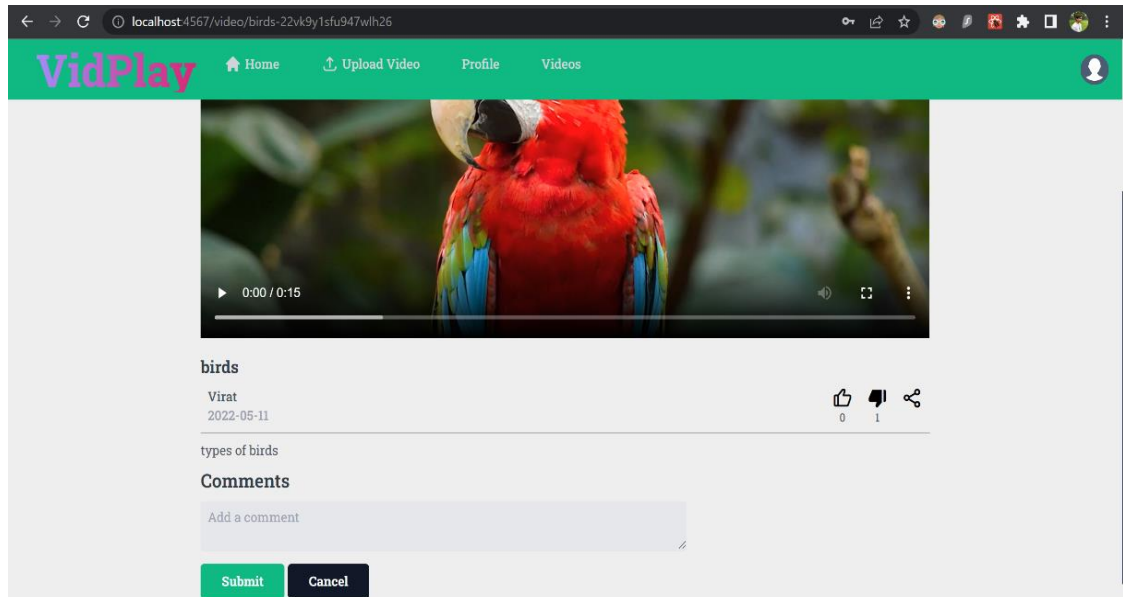


Figure 7: Video page.

#### 7.1.7.1 Share video component

Once the users open the video page to play the video, they can share the video to other users through different social media platforms by clicking on the share icon present in the bottom right corner.

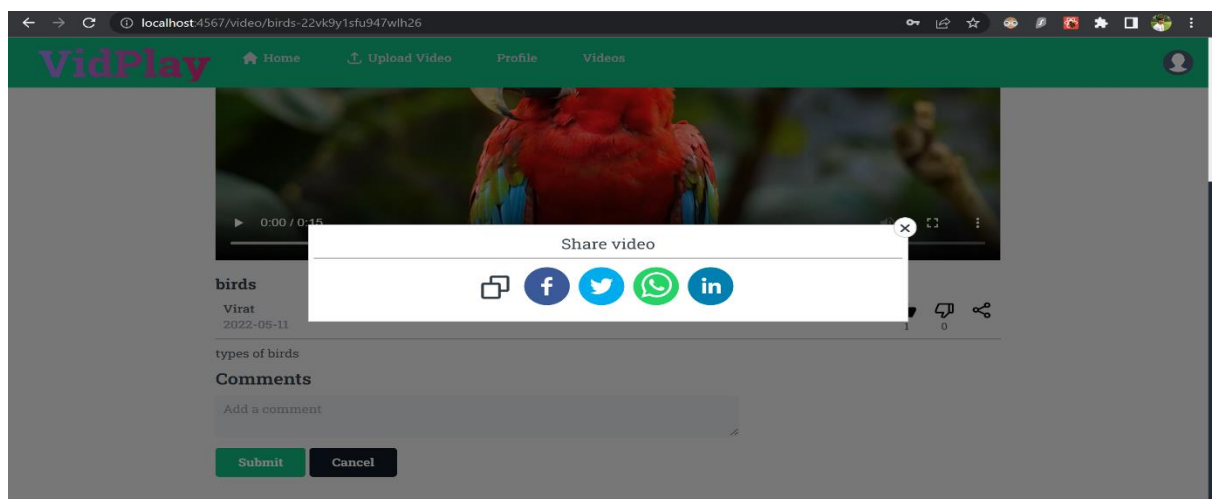


Figure 8: Share Video Feature

Once they click on the share icon it opens up a panel which contains icons of different social media platforms. By clicking on the icon of your choice the video url (which is required to identify the video, is generated using video slug), is copied and the user is redirected to the selected social media platform and can share the link to selected user. In the panel copy video icon is present just before all the social media icons which on click copies the url to clipboard and users can paste the url anywhere, not just limiting to the given social media platforms. By copying the url users have to go to the specific platform and paste it manually whereas by clicking on the specific social media icon all of this will be done automatically for the user. Video sharing option for different social media platforms is clearly presented in figure 7.

Code Snippet 5: Code explaining the functionality of share video option through different social media platforms

```
<div className="mr-2">
  <FacebookShareButton
    url={window.location.href}
    quote={videoData?.title}
  >
    <FacebookIcon size={50} round />
  </FacebookShareButton>
</div>
<div className="mr-2">
  <TwitterShareButton
    url={window.location.href}
    title={videoData?.title}
  >
    <TwitterIcon size={50} round />
  </TwitterShareButton>
</div>
<div className="mr-2">
  <WhatsappShareButton
    url={window.location.href}
    title={videoData?.title}
  >
    <WhatsappIcon size={50} round />
  </WhatsappShareButton>
</div>
<div className="mr-2">
  <LinkedInShareButton
    url={window.location.href}
    title={videoData?.title}
    summary={videoData?.description}
  >
    <LinkedInIcon size={50} round />
  </LinkedInShareButton>
</div>
```

className is given as “mr-2” for every element. FacebookShareButton tag is opened and inside that tag the url required for accessing Facebook is specified along with the title. Facebook icon is also specified and the size of it is set as {50px} and the shape as round. FacebookShareButton tag is closed meaning that the design of the Facebook icons and logic for accessing the Facebook is within the tags. The design part is same for the remaining platforms i.e., size is 50px and is in round shape as shown in the code snippet 5 above but the url required for accessing those particular platforms changes accordingly.

#### 7.1.7.2 Like and dislike video features

Like video option allows users to like the video by clicking on the like icon present at the bottom right corner as shown in figure 4. Dislike video option allows users to like the

video by clicking on the dislike icon present at the bottom right corner next to like icon. The code snippets below explain the functionality of the like and dislike features.

Code snippet 6: Code explaining data fetching for like and dislike features

```
const [likedVideo, setLikedVideo] = useState(false);
const [shouldLike, setShouldLike] = useState(false);
const [disLikedVideo, setDisLikedVideo] = useState(false);
const [shouldDisLike, setShouldDisLike] = useState(false);

const { slug } = match.params;

// Get video data
useEffect(() => {
  (async () => {
    if (user && user._id) {
      const response = await GetVideoDetails(slug);

      if (response && response.isSuccess) {
        setVideoData(response.data.video);

        if (response.data?.video?.likes?.includes(user._id)) {
          setLikedVideo(true);
        }
        if (response.data?.video?.dislikes?.includes(user._id)) {
          setDisLikedVideo(true);
        }
      }
      setLoading(false);
    }
  })();
}, [slug, user]);

export const LikeVideo = (videoID, userID) =>
  postApi(`video/like/${videoID}/${userID}`);
export const UnlikeVideo = (videoID, userID) =>
  postApi(`video/unlike/${videoID}/${userID}`);
export const DislikeVideo = (videoID, userID) =>
  postApi(`video/dislike/${videoID}/${userID}`);
export const RemoveDislike = (videoID, userID) =>
  postApi(`video/removedislike/${videoID}/${userID}`);
```

The state hook “useState” is used to keep track on the changes made to the current state. It is used in four instances here where useState is the initial state and is set as false in all four instances. It returns a set of values in each instance; one is the current state and the other one is the function which updates it. In the first instance it returns the current state which is likedVideo and the function to update is setLikedVideo. Slug is also used here to fetch the video details. Slug and user dependencies used at the bottom suggest that only they will change when the state change occurs using useEffect (), await keyword is used inside the async function as it blocks the execution of the code and waits for the response or promise to be resolved. Once you click on the video based on the user and userID, using GetVideoDetails() the video details are posted to the backend server through getApi(`video/\${slug}`) and waits for the response. If the response is success, then initial state of useState() is updated to videoData from null using the function setVideoData(). If data, video and likes includes (user.\_id) setLikedVideo is set as true, or if data, video and dislikes includes(user.\_id) setDisLikedVideo is set as true according to the response. Once any of these actions take place using setLoading() loading is set as false meaning it is done and not loading at that particular time.



## Code snippet 7: Code explaining the functionality of like and dislike features

```
// Like or Unlike the video
useEffect(() => {
  (async () => {
    if (shouldLike && user && videoData) {
      setDisLikedVideo(false);
      if (!likedVideo) {
        const response = await LikeVideo(videoData._id, user._id);

        setVideoData(response.data.result);

        if (response && response.isSuccess) {
          setShouldLike(false);
          setLikedVideo(true);
        }
      } else {
        const response = await UnlikeVideo(videoData._id, user._id);

        setVideoData(response.data.result);

        if (response && response.isSuccess) {
          setShouldLike(false);
          setLikedVideo(false);
        }
      }
    }
  })();
}, [likedVideo, shouldLike, videoData, user]);

// Dislike the video
useEffect(() => {
  (async () => {
    if (shouldDisLike && user && videoData) {
      setLikedVideo(false);
      if (!disLikedVideo) {
        const response = await DislikeVideo(videoData._id, user._id);

        setVideoData(response.data.result);

        if (response && response.isSuccess) {
          setShouldDisLike(false);
          setDisLikedVideo(true);
        }
      } else {
        const response = await RemoveDislike(videoData._id, user._id);
        setVideoData(response.data.result);
        if (response && response.isSuccess) {
          setShouldDisLike(false);
          setDisLikedVideo(false);
        }
      }
    }
  })();
}, [disLikedVideo, shouldDisLike, videoData, user]);
```

For the like and unlike video feature four dependencies namely likedVideo, shouldLike, videoData, user are declared using useEffect() suggesting that only they will change when the state change occurs. Await is used inside the async function as it makes the code wait till the promise is resolved. Disliked video is set as false using setDisLikedVideo() function. If the video is not liked yet then using LikeVideo() the data required for it (videoData.\_id, user.\_id) is posted to backend through postApi(`video/like/\${videoID}/\${userID}`) and it waits for the response from the server. Initial state of useState() is updated to videoData from null using the function setVideoData() and if the response is success, then setShouldLike() is set as false and setLikedVideo() is set as true meaning video is liked, else the data is posted to backend using UnlikeVideo() through postApi(`video/unlike/\${videoID}/\${userID}`) and initial state of useState() is updated to videoData from null using setVideoData() and if the response is success, then setShouldLike() is set as false and setLikedVideo() is also set as false meaning the video is neither liked nor about to be liked.

For the dislike video feature four dependencies namely disLikedVideo, shouldDisLike, videoData, user are declared using useEffect() suggesting that only they will change when the state change occurs. Await keyword is used inside the async function as it makes the rest of the code wait till the promise is resolved and liked video is set as false using setLikedVideo(). If !disLikedVideo then using DislikeVideo() the data is posted to server through postApi(`video/dislike/\${videoID}/\${userID}`) and waits for the response from the server. Initial state of useState() is updated to videoData from null using the function setVideoData() and If the response is success then setShouldDislike() is set as False and setDisLikedVideo is set as true meaning the video is disliked, else the same process repeats for removing dislike but using RemoveDislike() function. If the response is success then setSholudDisLike() and setDisLikedVideo is set as false meaning the video is neither disliked nor about to be disliked.

### 7.1.7.3 Post Comment Component

Signed in users can comment on the particular video in the textbox present below the video. Once you type your comment and click on the submit button your comment will be posted to the video page. All the users who watch the video will be able to see the comment along with the name of the person who posted the comment.

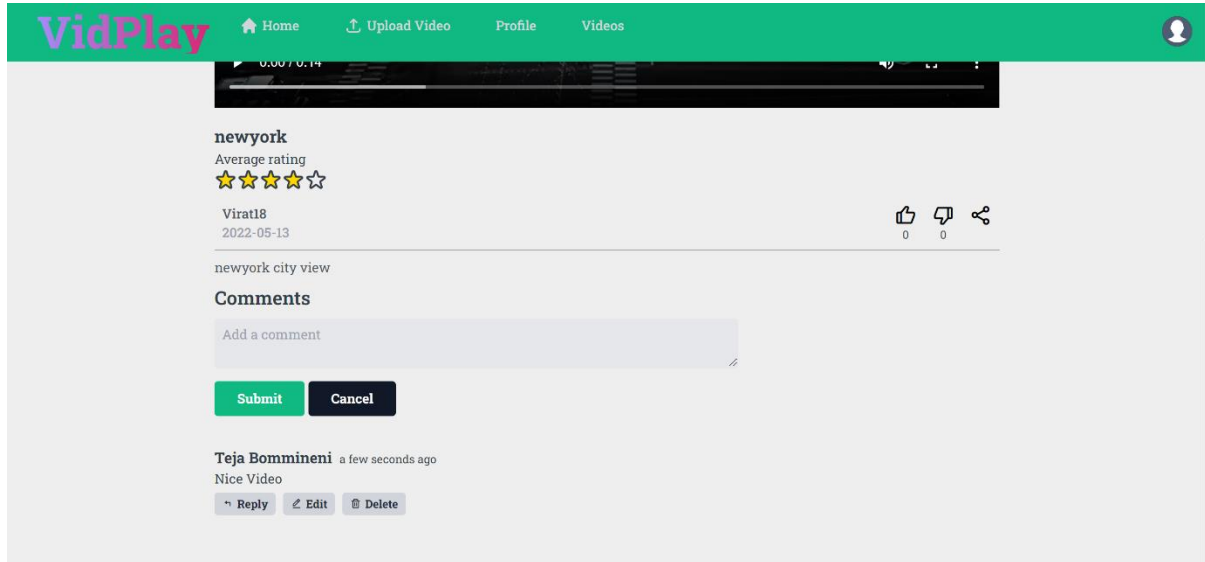


Figure 9: Post comment component

As evident from the above figure 6, only the user who posted the comment has the access to delete and edit to the comment. Any user can reply to the comment using the reply button. The functionality of posting a comment is explained in the code snippet 8.

Code snippet 8: Code explaining the functionality of post comment

```
const PostComment = ({ videoData, dispatchComments }) => {
  const [textValue, setTextValue] = useState("");
  const [disableSubmit, setDisableSubmit] = useState(true);
  const [shouldComment, setShouldComment] = useState(false);
  const { user } = useContext(AuthContext);

  useEffect(() => {
    if (textValue.trim().length > 0) {
      setDisableSubmit(false);
    } else {
      setDisableSubmit(true);
    }
  });

  return () => {
    setDisableSubmit(true);
  };
}, [textValue];

useEffect(() => {
  (async () => {
    if (shouldComment) {
      const response = await CommentVideo(videoData._id, {
        text: textValue,
        userID: user._id,
      });
    }
  })();
}, [shouldComment, videoData, textValue, user]);

if (response && response.isSuccess) {
  const commentData = { ...response.data.result };
  const commentUser = { ...response.data.result.user, ...user };
  commentData.user = commentUser;

  console.log(commentData);

  setTextValue("");
  dispatchComments({
    type: "create",
    payload: commentData,
  });
} else {
  toast.error(`${response.data.toString()}`, {
    position: toast.POSITION.BOTTOM_RIGHT,
    autoClose: 3000,
  });
}
})();

return () => {
  setShouldComment(false);
};
// eslint-disable-next-line react-hooks/exhaustive-deps
}, [shouldComment, videoData, textValue, user]);
```



The state hook “useState” is used to keep track on the changes made to the current state, “useContext” for the state management. `textValue` is declared inside `useEffect` stating that only they should change when the state change occurs. If the length of the comment is greater than 0, then using `setDisableSubmit(false)` submit button will be enabled or else it will be disabled. Four dependencies namely `shouldComment`, `videoData`, `textValue`, `user` are declared inside the next `useEffect`. When the user submits the comment, then by using `CommentVideo()` function the data is posted to backend through post request. The video id, user id will be created and stored in MongoDB along with the text value. If the response is success, by using http-status-code “200” the comment will be posted and the comment box will return to null state which is its original state. If there is any error the data stays same in the comment box and an error message will be displayed at the bottom using `toastify`.

#### 7.1.7.4 Post reply component

Users can reply to a comment which is posted by another user by clicking on the reply button which is displayed in figure 9. The number of times user can reply to a video is not limited. The functionality of reply to a comment is explained using code snippet 9.

Code snippet 9: Code explaining the functionality of post reply comment.

```
const PostReply = ({ comment, dispatchComments, setShowReplyBox }) => {
  const [textValue, setTextValue] = useState("");
  const [disableSubmit, setDisableSubmit] = useState(true);
  const [shouldReply, setShouldReply] = useState(false);

  const { user } = useContext(AuthContext);

  const inputRef = useRef();

  useEffect(() => {
    inputRef.current.focus();
  }, []);

  useEffect(() => {
    if (textValue.trim().length > 0) {
      setDisableSubmit(false);
    } else {
      setDisableSubmit(true);
    }
  });

  return () => {
    setDisableSubmit(true);
  };

  useEffect(() => {
    (async () => {
      if (shouldReply) {
        console.log("should reply");
        const response = await ReplyComment(comment._id, user._id, {
          if (response && response.isSuccess) {
            const payload = response.data.result;
            setTextValue("");
            dispatchComments({
              type: "update",
              payload,
            });
          } else {
            toast.error(`${response.data.toString()}!`, {
              position: toast.POSITION.BOTTOM_RIGHT,
              autoClose: 3000,
            });
          }
          setShouldReply(false);
        })();
      }

      return () => {
        setShouldReply(false);
      };
    }, [shouldReply, comment, textValue, user]);
  });
};
```

The state hook `useState` and `useContext` are used. In `useEffect` single dependency `textValue` is defined and if the length of the comment is greater than 0, then using `setDisableSubmit(false)` submit button will be enabled or else it will be disabled. When the user submits the reply, then by using `ReplyComment()` function we are creating comment `._id`, `user._id` and the data is posted to backend through post request and stored in MongoDB along with `textValue`. If it is stored in the MongoDB and the response is success then it the reply will be posted and reply box in which we typed the comment becomes empty and returns to its original state. If there is any error the data stays same in the reply box and an error message will be displayed at the bottom using `toastify`.

### 7.1.8 Profile page

Profile page is where users can update their credentials related to the account. Once the users click on the profile page option present in the top bar of the application they will be redirected to a page where they can update their details. All the videos uploaded from that particular account are present in the profile page.

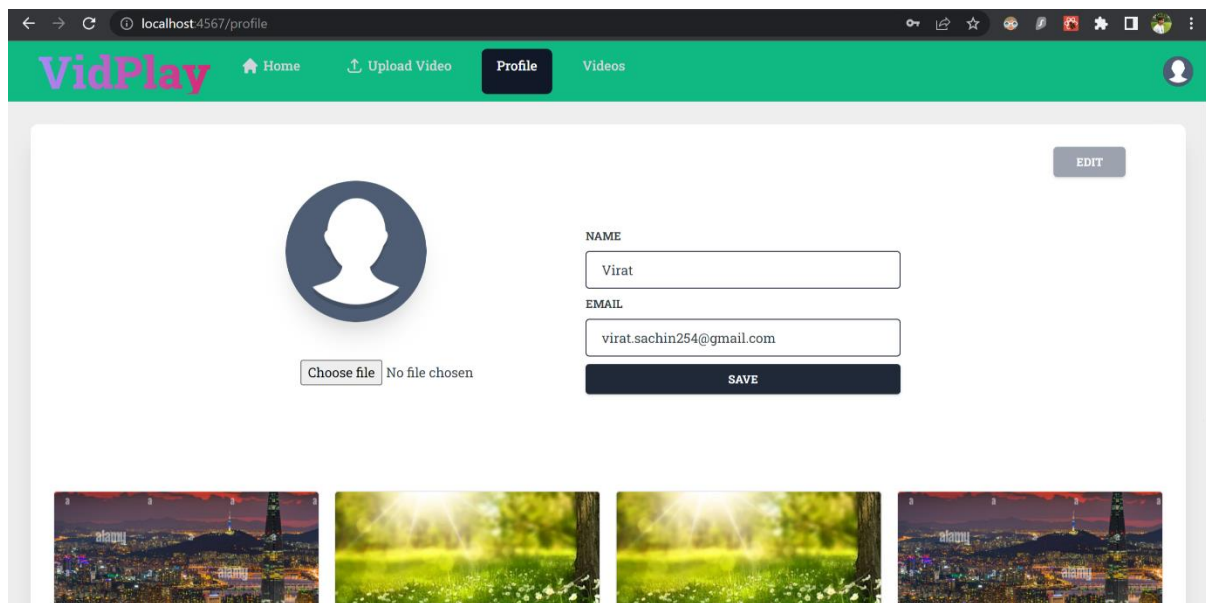


Figure 10: Updating user details in the profile page

As you can see in the figure 10 after going into the profile once you click on the edit button present in the top right corner you will be redirected for editing your details, where you can update your name, email address and can also choose an image for your profile. Once everything is done and there are no more changes to make, click on the save button so that everything will be updated according to the changes made displaying a notification stating “profile update”.

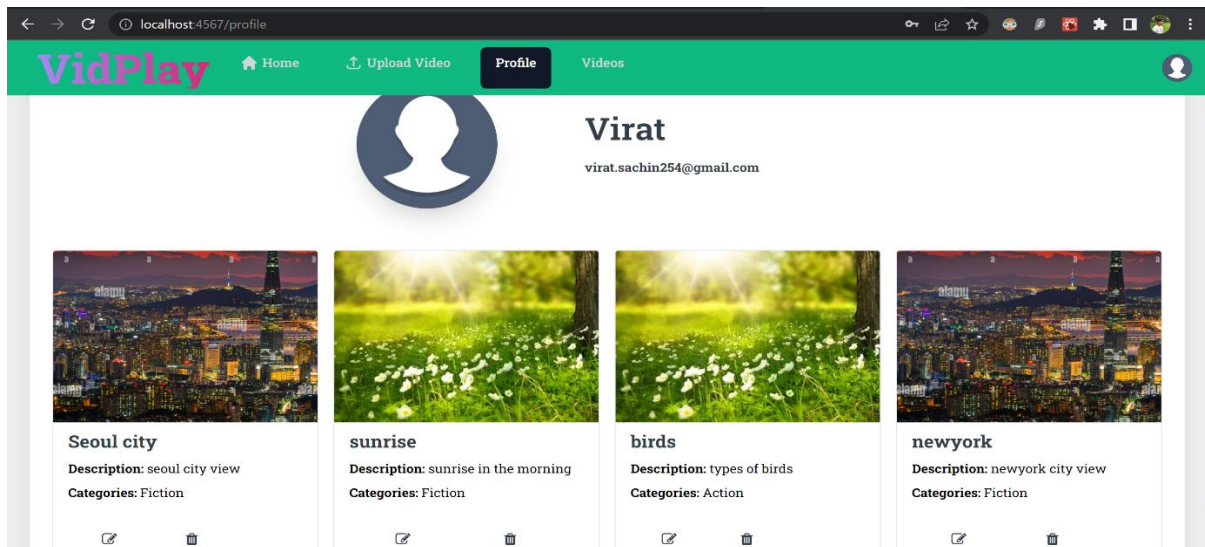


Figure 11: Displaying all the videos updated from on particular account.

As mentioned earlier all the videos uploaded from that particular account are present in the profile page. As evident from the above figure four videos are uploaded from the account. As the user has uploaded these videos, user has given the option of editing and deleting the videos using the edit and delete icon present at the bottom of each video. Once the users click on the edit icon, they will be redirected to the upload video page where they can update all the details of the video namely title, description, category, thumbnail of the video and also video file. Users have to click on the save button present at the bottom right corner in order for the video to update. On clicking the save button a notification will be popped up stating that “video uploaded”, there by updating the video details and the user will be redirected to the home page. The functionality of all the components in the profile page is explained using code snippet 10.

Code snippet 10: Code which explains all the components in profile page.

```
const ProfilePage = () => {
  const { user, authLoading } = useContext(AuthContext);
  const [userInfo, setUserInfo] = useState(null);
  const [photo, setPhoto] = useState(null);
  const [showEdit, setShowEdit] = useState(false);
  const [saving, setSaving] = useState(false);
  const [uploading, setUploading] = useState(false);
  const [loading, setLoading] = useState(true);
  const [videos, setVideos] = useState([]);

  const handleUpload = async (e) => {
    setUploading(true);
    const files = e.target.files;
    try {
      const response = await UploadFile(files[0]);
      if (response && response.isSuccess) {
        setPhoto(response.data);
      }
    } catch (err) {
      if (err.response.status === 500) {
      } else {
      }
    }
    setUploading(false);
  };

  const handleSave = async (e) => {
    e.preventDefault();
    setSaving(true);

    const data = {
      name: e.target.name.value,
      email: e.target.email.value,
      photo: photo,
    };

    const response = await UpdateUserProfile(user._id, data);
    if (response && response.isSuccess) {
      setUserInfo(response.data.user);
      setPhoto(response.data.user.photo);
      toast.success('Profile Update!', {
        position: toast.POSITION.BOTTOM_RIGHT,
        autoClose: 3000,
      });
    } else {
      toast.error(`${response.data.toString()}!`, {
        position: toast.POSITION.BOTTOM_RIGHT,
        autoClose: 3000,
      });
    }
    setSaving(false);
    setShowEdit(false);
  };
};
```

UseContext is used for the state management and the state hook “useState” is used to keep track on the changes made to the current state. To handle the uploading of a photo handleUpload() function is used and setUploading() is set as true meaning you can upload a file. Once you click on choose file button all the files or images in the system will open up for the user to choose from using “e.target.files” and waits for the user to choose a file. If the user clicks on a file to choose then using setPhoto method the image is selected and using UploadFile method the image will be uploaded to the profile. If it catches any error in between then the error response code will be displayed and setUploading is set as false meaning you can’t upload the file.

The data which needs to be saved when you click on save button are declared using handleSave() function. Once you click on save it posts the data to backend api and waits for its response and if the response is success, it updates user details based on user\_id and data using UpdateUserProfile method. A notification will pop up at the bottom right corner stating ‘profile update’ and will auto closes in 3 seconds as designed using toastify else an error message will pop up at the same area. setSaving() and setShowEdit() are set as false at the end meaning it returns to the profile page from edit page on you click save.

Code snippet 11: Code for fetching user data

```
useEffect(() => {
  (async () => {
    const response = await GetUserProfile(user?._id);

    if (response && response.isSuccess) {
      setUserInfo(response.data.user);
      setPhoto(response.data.user?.photo);
    }
    setLoading(false);
  })();
  if (user) {
    getUserVideos({ userId: user._id })
      .then((res) => {
        if (res && res.isSuccess) {
          setVideos(res.data.videos);
        }
      })
      .catch((err) => {});
  }
}, [user]);
```

User details will be fetched based on user.\_id using GetUserProfile method, if the response is success, then using setUserInfo and setPhoto methods user details and user photo are fetched respectively from the database. Through getUserVideos method the user.\_id of the logged in user is matched with user.\_id of the user on the uploaded videos and by using setVideos method the matched videos are fetched from the database and displayed on the profile page.

#### 7.1.9 Sign out component

Sign out Component is used by the users when they want to sign out of the application. It is present in the top right corner in the menu bar of the application. A sign out button pops up on click on the icon which on click allows users to log out of the application.

Code snippet 12: Code for the sign out button

```
const handleSignOut = async () => {
  localStorage.removeItem("auth_token");
  setUser(null);
};

<button
  onClick={() => handleSignOut()}
  className="block px-4 py-2 text-sm text-gray-700"
  role="menuitem"
  type="button"
>
  Sign out
</button>
```

As evident from the code snippet 10 handleSignOut() function is used in the sign out button to handle the sign out operation for the user. The sign out button on click removes the login details of the user from the local storage using removeItem("auth\_token") method which is declared inside handleSignOut() function and logs out user from the application.

## 7.2 Frontend Libraries

Some pre-defined libraries are associated with the development of this project. These libraries are used in order to build the operationality and the behaviour of the application within the given time frame as time is limited. All the libraries are available in the Node package manager(npm). For using most of these libraries in the development of your application you have to install them in your system using syntax "npm install" followed by library name. Once they are installed in your system, then they are ready to be used in development of any component just by importing it to that component. Following are the libraries used for the frontend development of application.

Table 2: Frontend Libraries

Name	Version	Usage	Installation
<a href="#">autoprefixer</a>	9.8.6	Autoprefixer is a postCSS extension (plugin) for determining vendor prefixes to CSS rules utilizing values present in caniuse.com. autoprefixer is used in this part of development because it utilizes the data depending on present browser popularity and property support to put in prefixes for you.	Npm I autoprefixer
<a href="#">Postcss</a>	7.0.36	It is used in the development as it is a powerful tool and has been helpful in altering and modifying the styles using JS plugins. One such plugin for postcss is autoprefixer.	Npm I postcss
<a href="#">Tailwindcss</a>	2.2.4	Tailwind CSS is a CSS framework used for quickly	Npm I tailwindcss



		developing traditional user interfaces. In the development it helps in designing the application rapidly in the limited amount of time.	
<a href="#">@craco/craco</a>	6.1.2	Used to obtain all the advantages of create-react-app and customization. It is also necessary for using tailwind CSS properly in the application	<code>npm install @craco/craco -save</code>
<a href="#">React – toastify</a>	7.0.4	Used to add various types of notifications to the application handily without worrying about anything.	Npm install – save react-toastify
<a href="#">Axios</a>	0.21.1	Used as a HTTP client support for browser and node js	npm install axios
<a href="#">jwt-decode</a>	3.1.2	It is a minor browser library used in this application development to assist in decoding JWT tokens.	Npm install jwt-decode
<a href="#">moment</a>	2.29.3	used in the development for parsing, validating, manipulating and formatting dates.	Npm I moment
<a href="#">react-copy-to-clipboard</a>	5.1.0	React component used to copy text to the clipboard automatically.	Npm I react-copy-to-clipboard
<a href="#">react-router-dom</a>	5.2.0	React-router-dom consists all the requirements for utilizing react-router in the application. React-router is a library used for routing in react as it permits to alter the browser url and connects it to UI.	Npm I react-router-dom
<a href="#">reactjs-popup</a>	2.0.5	A modest react popup component used to design easy and complicated models, menus for the web application.	Npm I reactjs-popup
<a href="#">uuid</a>	8.3.2	Used to generate unique IDs in the database for users, videos, likes, etc.,	npm I uuid

## 7.3 Backend Modules

Backend is the part of web application which is not visible to the user and is mandatory for the application to function properly and smoothly. Backend code commonly known as server-side code is developed to run on server and not on users' system. When the user enters some data to the application through the frontend it communicates to the backend through the HTTP requests to validate the data and transfers the data to users portal.

### 7.3.1 Models

Models is the backend part of the code and is the base of backend part of development which is defined on the node.js server and dependent on schema. This schema demonstrates the properties of the model. All the data files in this project are stored in the MongoDB database, models define the structure or format of the data in which it should be stored in MongoDB. Models are the path amongst MongoDB database and the logic written for the application. So, when the user is signing in, commenting on a video, uploading a video or photo it has to meet the requirements specified in the model or else user won't be able to commit the action which was meant to do. The backend code of models and how they are stored in MongoDB are clearly presented in the figure 12.

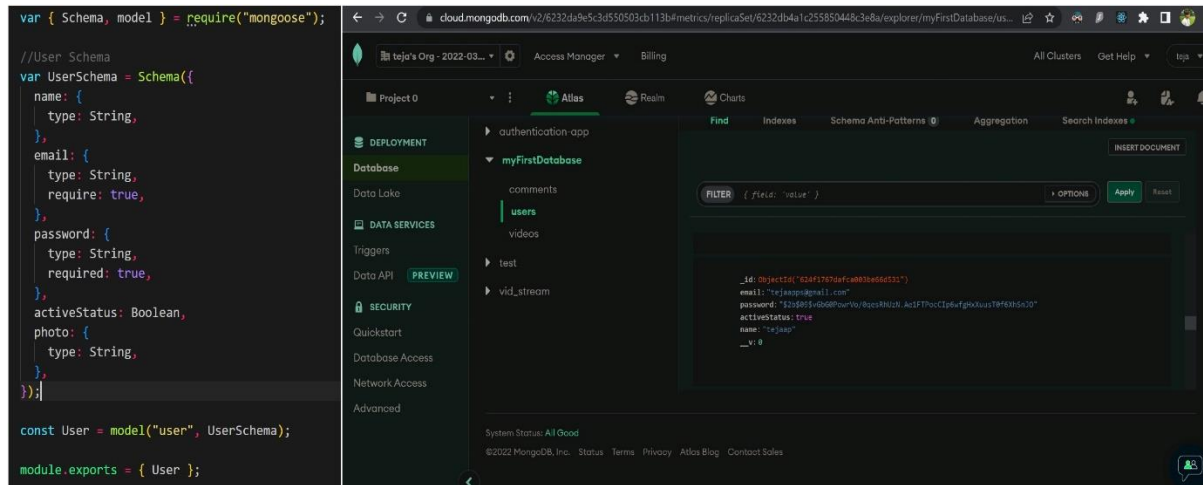


Figure 12: Code specifying the user schema and user details stored in MongoDB.

As evident from the code user details are stored in the specific format. Name is stored in the string format, so is email and password but email and password are necessary for a user to login to the application. So, they are made mandatory using “require: true” and the active status is also shown in the string format. They are stored in the exact format in the MongoDB except the objectID which is automatically generated by the database drivers.

### 7.3.2 Middlewares

The functions which will have permissions to requested object, response object and the next middleware function in the request and response cycle of the application, which is the means of communication between the frontend and backend are known as middlewares. As the name itself states the middleware sits in between the client requests which are made on front end and the response from the backend. The main responsibility of the middleware is to allow and simplify access to the backend resources. Middlewares are used in this project development to check whether the user ID and the video IDs are valid MongoDB ID's or not i.e., it is used in validation for all HTTP requests. While logging in the middleware checks for the user ID and email ID provided by the user are valid or not with MongoDB. If the user is authenticated it accepts the request and permits the user to proceed further into the application or else the user will be redirected to the login page.

Code snippet 13: Backend code to verify user id and video id using middleware

```
const { response } = require("../utils/response");
const isValidMongoID = require("../utils/isValidMongoID");

const verifyUserID = (req, res, next) => {
  try {
    // check if the user id is a valid mongodb id or not
    if (!isValidMongoID(req.params.userID)) {
      return response(res, 400, false, {}, "Invalid user id");
    }

    next();
  } catch (error) {
    return response(
      res,
      StatusCodes.INTERNAL_SERVER_ERROR,
      false,
      {},
      error.message
    );
  }
};

module.exports = verifyUserID;
```

```
const { response } = require("../utils/response");
const isValidMongoID = require("../utils/isValidMongoID");

const verifyVideoID = (req, res, next) => {
  try {
    // check if the video id is a valid mongodb id or not
    if (!isValidMongoID(req.params.videoID)) {
      return response(res, 400, false, {}, "Invalid video id");
    }

    next();
  } catch (error) {
    return response(
      res,
      StatusCodes.INTERNAL_SERVER_ERROR,
      false,
      {},
      error.message
    );
  }
};

module.exports = verifyVideoID;
```

In the code snippet 10 `isValidMongoID` and `response` functions which are declared in `utils` folder are used and are required for validating the user details with MongoDB. Request object(`req`), response object(`res`) and next middleware function (`next`) are declared inside `verifyUserID` function. After validation on the basis of input parameters, if it is not a valid MongoDB then it catches error and returns error message as response object using HTTP status codes in this case 400.

### 7.3.3 Routing

Routing is one of the top most feature of any web application, it is robust and gives the flexibility of navigating throughout the entire application. Routing is just a part of Express code which connects an HTTP request (post, get, put, etc.), an URL path and lastly the function which is called in order to handle and control that particular pattern. Some of the route methods used in the development of this application are: `post()`: used to post the data to the server, `put()`: used to modify the data, `get()`: used to obtain the data from url. Routing is clearly demonstrated in the following code snippet 11.



Code snippet 14: authentication routes and comment routes

```
const { Router } = require("express");
const { login, reAuth, register } = require("../controllers/auth.controller");

const router = Router();

//api: url/auth/_

//login
router.post("/login", login);

//register
router.post("/register", register);

//re-auth
router.post("/reauth", reAuth);

module.exports = router;
```

```
const { Router } = require("express");
const verifyVideoID = require("../middlewares/verifyVideoID");
const verifyUserID = require("../middlewares/verifyUserID");

const {
  postComment,
  getVideoComments,
  deleteComment,
  postReply,
  deleteReply,
  updateComment,
  updateReply,
} = require("../controllers/comment");

const router = Router();

// Post a comment
router.get("/:videoID", verifyVideoID, getVideoComments);
router.post("/:videoID", verifyVideoID, postComment);
router.post("/delete/:commentID/:userID", verifyUserID, deleteComment);
router.post("/update/:commentID/:userID", verifyUserID, updateComment);

router.post("/reply/:commentID/:userID", verifyUserID, postReply);
router.post("/updateReply/:userID", verifyUserID, updateReply);
router.post("/deleteReply/:userID", verifyUserID, deleteReply);

module.exports = router;
```

As evident from the above code snippet login, reAuth and register functions are declared in a separate folder named auth.controller.js and routed into this part of code to use them here. As the routing structure itself defines router.post("/login", login) defines the data entered in the login form is posted to backend using the route "/login" and the login function specified handles the pattern. Same functioning is followed in the register form as well but using different route "/register" and different function to handle the pattern which is register. In the routing section of comments which are specified on the right side middlewares are imported from middlewares section and used in here as it is mandatory to verify the user id and video id when the user posts a comment. Post() method is used to post the data to server using route "/:videoID" and middleware is used to validate video credentials and the postComment function handles all these patterns. Get() method is used to obtain the data related to the video using the route "/:videoID" and middleware is used to validate video credentials and using getVideoComments is the function specified to handle the patterns.

#### 7.3.3.1 Controllers

Controllers are the part of express code written for the backend and as the name itself suggests they are used to control the flow of requests to the server and send appropriate response to the frontend for the user. If the user wants to delete a comment, then the user clicks on the delete icon. As soon as user clicks on the delete icon the request is posted to backend then the deleteComment controller checks whether the video id is a valid MongoDB id or not using middleware and pass the appropriate result to the user, basically controlling the entire request flow. The sample code for functioning of controllers used in this project is shown in code snippet 12.

Code snippet 15: Backend code to get user based on video details

```
const { StatusCodes } = require("http-status-codes");
const { Video } = require("../models/Video.model");
const { response } = require("../utils/response");

const getVideoByUser = async (req, res) => {
  const { userId } = req.body;
  Video.find({ user: userId })
    .then((videos) => {
      if (videos) {
        return response(res, StatusCodes.OK, true, { videos: videos }, null);
      } else {
        return response(
          res,
          StatusCodes.NOT_FOUND,
          false,
          {},
          "No video Found!"
        );
      }
    })
    .catch((err) => {
      return response(
        res,
        StatusCodes.INTERNAL_SERVER_ERROR,
        false,
        {},
        err.message
      );
    });
});

const createVideo = require("./createVideo");
const getVideos = require("./getVideos");
const getVideoDetails = require("./getVideoDetails");
const updateVideo = require("./updateVideo");
const getVideoByUser = require("./getVideoByUser");
const likeVideo = require("./likeVideo");
const unlikeVideo = require("./unlikeVideo");
const dislikeVideo = require("./dislikeVideo");
const removeDislike = require("./removeDislike");
const deleteVideo = require("./deleteVideo");
const searchVideoFromKeyword = require("./searchVideoFromKeyword");
const submitRating = require("./submitRating");

module.exports = {
  createVideo,
  getVideos,
  getVideoDetails,
  updateVideo,
  getVideoByUser,
  likeVideo,
  unlikeVideo,
  dislikeVideo,
  removeDislike,
  deleteVideo,
  searchVideoFromKeyword,
  submitRating,
};
```

In order to get video based on user details “http-status-codes”, video.model.js folder and response.js file in utils folder has to be imported and used in the code. userID is required and based on the user id, by using find () method videos are identified in the database. After the videos are identified in the database a response is sent using http-status-code “200” that videos are there and videos are fetched and displayed in the application. If there are no videos in the database using http-status-code “400” a notification is displayed stating that “No videos Found”. An error will also be displayed if the server catches any internal error.

In the index.js file present in the videos folder in controllers the code for all the features and methods applicable to the video like createVideo, updateVideo, getVideoDetails, likeVideo, dislikeVideo, deleteVideo, submitRating, etc., are imported and used here through routing. As evident from the code snippet on right side all of these are exported using module.exports{ } making the code flexible. Just by importing the index.js all these features can be used anywhere in the development.

### 7.3.4 Utils

When creating a full stack web application any specific keys and urls related to the backend part of project are stored in utils folder. In the development of this project jsonwebtoken which is used for login and registration, aws s3 bucket keys, video and image keys, etc., are stored in the utils folder and it checks the validation of the keys. When the user tries to sign in to the application jwt token is generated by MongoDB and the token is stored in the browser cookies. Each time user tries to login into the application the token stored in the cookies is cross verified with the token in MongoDB and the user is signed into application automatically until the browser cookies are cleared.

## Code snippet 16: Backend code which explains the uploading of a file to aws s3 bucket

```
const ID = "AKIAZM7TFLDPQ354Q7HC";
const SECRET = "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";

const BUCKET_NAME = "vidstream-files";

const uploadFile = async (req, res) => {
  if (req.files === undefined || !req.files.image) {
    let msg = "No file found!";
    return response(res, StatusCodes.BAD_REQUEST, false, null, msg);
  }

  try {
    const s3 = new AWS.S3({
      accessKeyId: ID,
      secretAccessKey: SECRET,
    });

    const file = req.files.image;
    const fileFormat = file.name.split(".").pop();
    const fileName = uuidv4() + "." + fileFormat;

    const params = {
      Bucket: BUCKET_NAME,
      Key: fileName,
      Body: file.data,
    };

    s3.upload(params, function (err, data) {
      if (err) {
        throw err;
      }
      return response(res, StatusCodes.ACCEPTED, true, fileName, null);
    });
  } catch (error) {
    return response(
      res,
      StatusCodes.INTERNAL_SERVER_ERROR,
      false,
      {},
      error.message
    );
  }
};

module.exports = {
  uploadFile,
};
```

The accessKeyId and secretAccessKey are declared at the beginning and Bucket Name is given as “vidstream-files”. In the next step using uploadFile function the files are uploaded to the s3 bucket. If the required files or images are not selected then it is considered as a bad request and using http-status-codes an error notification “No File Found!” will be popped up. Then try with aws s3 bucket using the ID and access key, the file is required and the file name should be uuidv4.fileformat where uuidv4 is the unique ID generated for the file using uuid. The file type of how it should be while uploading to the s3 bucket are declared using params{ }. While uploading to the s3 bucket using s3.upload() function if there is any error it throws error, else it returns the response as success using http-status-codes. If it catches any error after uploading to the s3 bucket then the error message will be displayed as internal server error.

## 7.4 Backend Libraries

Some predefined backend libraries are used in the development process to finish the development within the given time frame. All these libraries used in the backend part of the development are available in the node package manager(npm). Once you install them in your system using the syntax “npm install” followed by library name you can use them in your development just by importing them into your code. All the libraries used in the backend part of the development are listed in table 3 below.

Table 3: Backend Libraries

Name	Version	Usage	Installation
<a href="#">Nodemon</a>	2.0.9	Nodemon assists in building and developing the applications that are based on node.js by restarting them if any changes are made to the file.	Npm I nodemon

<a href="#">@azure/storage-blob</a>	12.6.0	Storage blob is a cloud storage solution which is improvised for reserving large amounts of unstructured data. The data which does not follow particular order or any specific data model.	Npm install @azure/storage-blob
<a href="#">aws-sdk</a>	2.958.0	Aws-sdk for java script gives access to the web services in node.js. aws-sdk allows to use AWS services such as Amazon s3 which is used in the development of this project for storing videos.	Npm I aws-sdk
<a href="#">bcrypt</a>	5.0.1	bcrypt is a library for node.js which is used as the part of development to encrypt passwords and make them in unreadable format.	Npm I bcrypt
<a href="#">body-parser</a>	1.19.0	body-parser is a node.js middleware used for parsing incoming request bodies even before you can handle it.	Npm I body-parser
<a href="#">cors</a>	2.8.5	Cors is basically a node.js package for providing Express middleware which is utilized to authorize cors with different options.	Npm I cors
<a href="#">express</a>	4.17.1	Express is a quick, assertive and minimalistic web framework for node.	Npm I express
<a href="#">Express-fileupload</a>	1.2.1	It is an express middleware helpful for uploading files.	Npm I express-fileupload
<a href="#">http-status-codes</a>	2.1.4	HTTP status codes are the constants which are used to check whether a HTTP request is completed.	Npm I http-status-codes
<a href="#">jsonwebtoken</a>	8.5.1	An open standard which is utilized to share key information or data among two parties.	Npm I jsonwebtoken
<a href="#">Mongoose</a>	5.12.15	It is a MongoDB object modelling tool which balances relations among data and provides schema validation.	Npm I mongoose

<a href="#">uuid</a>	8.3.2	Used to generate unique IDs in the database for users, videos, likes, etc.,	npm I uuid
<a href="#">Validator</a>	13.6.0	A library which contains string validators and sanitizers.	Npm I validator

## 7.5 Build and Running the application

It is important to run the application during development to check on the progress of various features. I have used Visual Studio code for the development of the project, to run the app in browser using visual studio code involves these steps:

- Open the terminal and split the terminals for running both frontend code and backend code, one on each terminal.
- In the terminal go to that particular folder where the front end and backend codes are using the command 'cd filename'.
- Install the node package manager(npm) in both terminals in front end and backend code using 'npm install'.
- Establish database connection where the backend code is in the terminal using 'npm start', the front-end part of the application can be run on the browser using local host with the same command 'npm start'.

If you want to terminate the database connection or stop running of the application that can be done by going to that particular terminal and by simply clicking on 'Ctrl+c' and typing 'y' you can terminate the connections. Running of the application from the terminal in vs code is clearly shown in the figure 13 below.

```

1 import React, { useContext, useEffect, useState } from "react";
2 import { Link, Redirect } from "react-router-dom";
3 import TopBar from "../Components/Menu/MenuBar";
4 import ComponentLoader from "../Components/Progress/ComponentLoader";
5 import { AuthContext } from "../Providers/AuthProvider";
6 import { GetVideos } from "../Service/service";
7 import VideoCard from "../Components/VideoCard";
8
9 const HomePage = ({ history }) => {
10   const { user, authLoading } = useContext(AuthContext);
11   const [videos, setVideos] = useState([]);
12   const [activeVideos, setActiveVideos] = useState([]);
13   const [loading, setLoading] = useState(true);
14   const [currentCategory, setCurrentCategory] = useState("All");
15
16   const getData = async () => {
17     const response = await GetVideos();
18     if (response && response.isSuccess) {
19
20     }
21   }
22 }
23
24 export default HomePage;

```

Database Connected  
 Terminate batch job (Y/N)? y  
 PS C:\Users\tejab\Desktop\New folder\video-bac kend> npm start  
 > express-rest-api@1.0.0 start  
 > node server.js  
 Server is Running on 5000  
 Database Connected

Figure 13: Running the server and application on local host from vs code terminal

## 8. Testing

Testing plays a major role in finding out the bugs in the application. Web application testing identifies the bugs in the code and allows developer to sort out the issues in the code and ensures in delivering the application as per the specifications within the given amount of time when developing the project on a large scale. So, it is important to test the web application through different phases before deployment. Manual testing is done throughout the course of the project using chrome developer tools to ensure that web application functions properly as per the specifications. The manual testing is done in two phases of testing namely functional testing and non-functional testing.

### 8.1 Functional testing

Functional testing is done to the application to make sure that the application is functioning in line with the specifications and that it meets the user requirements. There are many types of functional testing but depending on the type of this application API testing and network testing are preferred over the others.

#### 8.1.1 API testing

All the failure scenarios are tested using the API testing. This API testing can be done in two ways, one is sending request by changing the IP address of the local host and the second one is testing by closing the backend server. The figure 14 below clearly shows the error which is occurred by sending the request from wrong IP address during API testing.

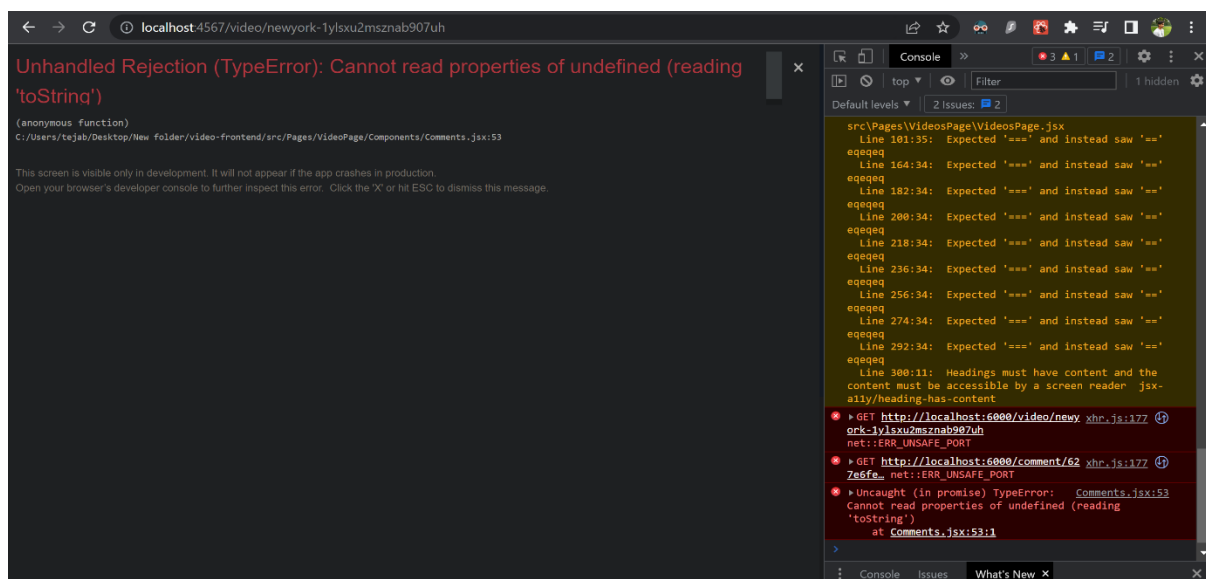


Figure 14: Error occurred due to changing the IP address and sending request from it.

The applications runs on localhost:5000 but in order to check the server functioning I changed the IP address to localhost:6000. As you can see in the above figure the get()



method which is used to obtain data from the url tries to get videos from localhost:6000, so it throws an error as the port doesn't exist. In the second scenario I have terminated the database connection and tested the application by trying to login into the application, but I have been refused the connection because the data which has to be posted to the backend using the post method for verification has not been posted as the database connection is terminated.

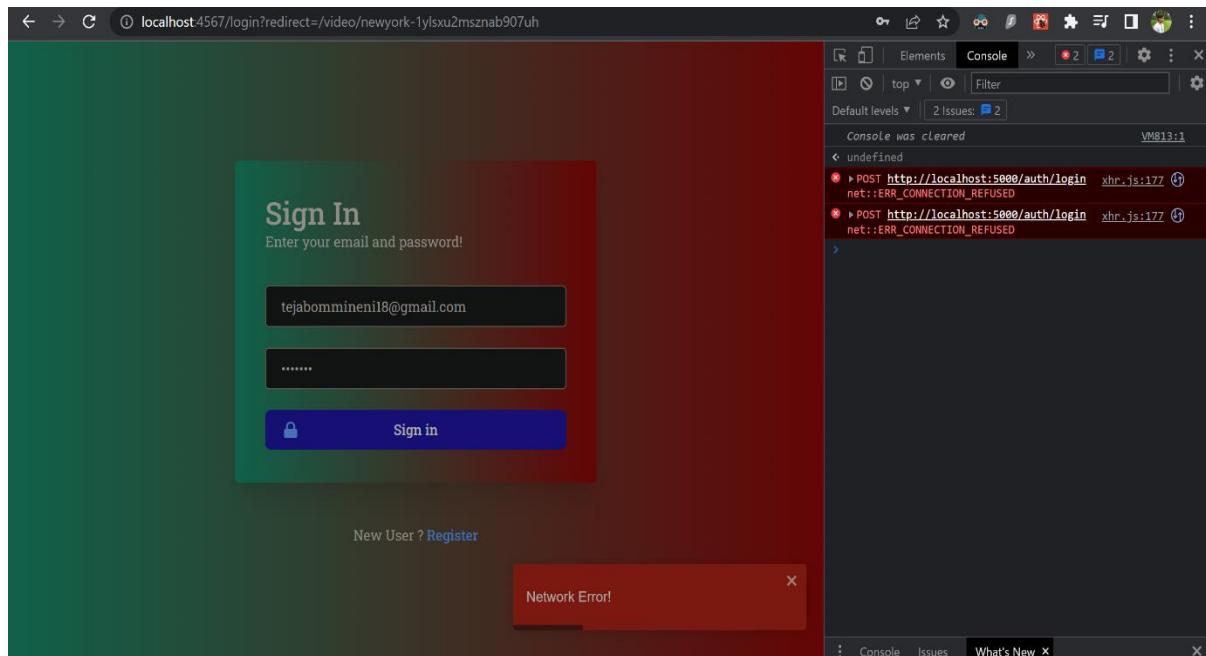


Figure 15: Error occurred during API testing due to the database termination.

## 8.2 Non-Functional testing

Nonfunctional testing involves the testing of non-functional features or aspects of the application such as performance. It is used for testing the preparedness of the application using nonfunctional parameters. It is as important to test the application using nonfunctional testing because it can affect the user satisfaction.

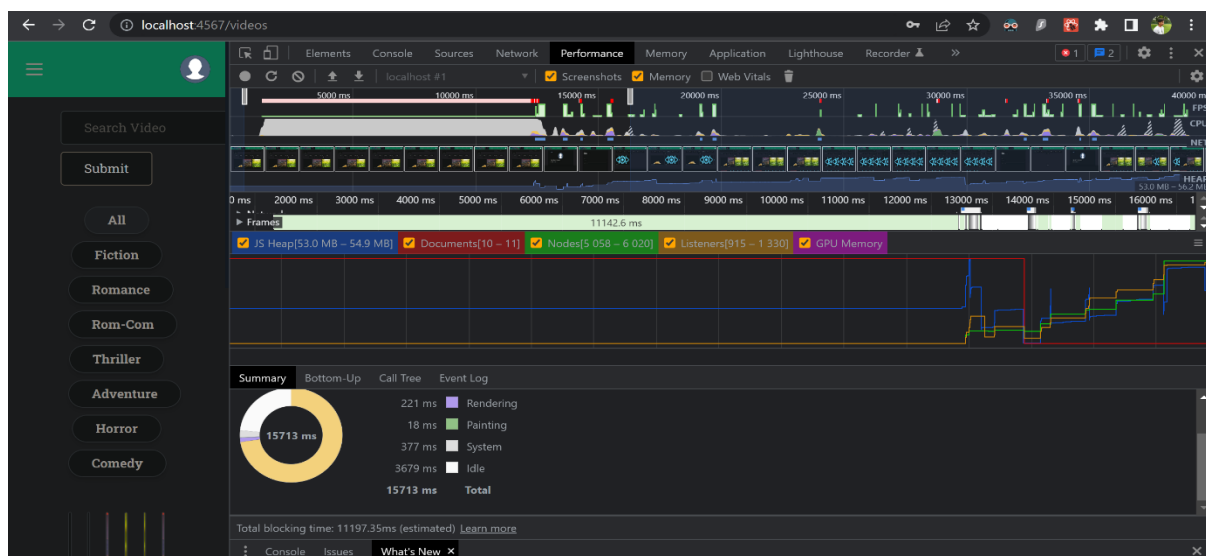


Figure 16: Performance testing of the application

The application is tested using the performance testing to see how it is performing and responding when we navigate quickly throughout the application. As we can see from the figure 11 the performance of the application is quite good under the workload and various factors such as GPU memory consumption, number of documents accessed, nodes and listeners are taken into consideration. The time consumed for performing every single action is also shown in the testing.

## 9. Conclusion

All the essential features are completed. Rest of the recommended and optional features has also been fully completed except the download feature in recommended features which has been left out due to sheer learning schedule, and this has been discussed with supervisor in fortnightly meetings. Additionally, the ratings feature which allows the users to leave a rating to the video after watching it has been added as suggested by the supervisor and second marker. All the features have been tested manually using google developer tools throughout the course of development to check the functionality and performance of the application.

### 9.1 Challenges Faced

One of the main challenges faced during the development of the project is while developing “search with chips” feature on the home page as it involved a bit of learning because I haven’t developed or used the chips feature before. In this feature specific word chips are provided on top of home page based on the video categories in the application and users can access all the videos of that category just by clicking on the chip instead of going through all the videos.

Uploading a video by integrating it with aws s3 bucket and connecting that data to MongoDB. The database is divided into two parts as the MongoDB stores only details of all the features and users in text format, so aws s3 bucket is used for storing videos and images.

### 9.2 Feedback and observation

In order to get the feedback for the application I contacted a friend of mine to join with me in testing. After going through the functionality of the application and navigating through all the features he gave a very good and positive feedback. The most he liked about the project is ‘the chips’ feature on top of the home page which allows the users to go through a particular category of videos just by a click on it instead of going through all the videos and provides a user-friendly experience.

Another suggestion he gave is about blocking the copyrighted content, which can be used in further development of the application in future.



## 9.3 Further Improvement

This project can be improved further and taken to the advanced level by using automated tools. For example, copyrighted content can be blocked from uploading into the application by integrating machine learning tools. These tools allow to search through database for any copyrighted content and only allows uploading only if the video is unique without any copyright violation. One more area where the project can be improved further is where video quality automatically changes without our interference according to the internet speed i.e., if the internet speed increases video quality also increases along with it.

## 10. References

- i) Developing cloud applications with Nodejs and React, Edx Course  
<https://www.edx.org/course/developing-cloud-applications-with-nodejs-and-react?index=product&queryID=59589cd2d613f57593ee7419fd4968fe&position=2>
- ii) GeeksforGeeks, for java script  
<https://www.geeksforgeeks.org/javascript/?ref=lbp>
- iii) Node package Manager(npm)  
<https://www.npmjs.com/>
- iv) MongoDB Atlas for MERN Stack  
<https://www.mongodb.com/languages/mern-stack-tutorial>
- v) Querying MongoDB in the browser with React  
<https://www.mongodb.com/developer/how-to/querying-mongodb-browser-realm-react/>
- vi) Building a REST API with Express, Node and MongoDB  
<https://www.mongodb.com/languages/express-mongodb-rest-api-tutorial>
- vii) NoSql vs Sql <https://www.mongodb.com/nosql-explained/nosql-vs-sql>  
and <https://www.ibm.com/cloud/blog/sql-vs-nosql>
- viii) Introduction to NoSql  
<https://www.learnhowtoprogram.com/react/react-with-nosql/introduction-to-nosql>

- ix) React js vs Angular js: a comparison of both the frameworks  
<https://www.imaginarycloud.com/blog/angular-vs-react/>
- x) How to use Axios with React js for API calls  
<https://www.freecodecamp.org/news/how-to-use-axios-with-react/>
- xi) Securely store passwords using bcrypt  
<https://auth0.com/blog/hashing-in-action-understanding-bcrypt/>
- xii) Simple login with JWT Tokens  
<https://auth0.com/learn/json-web-tokens/>
- xiii) Store data securely in AWS s3 bucket  
<https://www.paloaltonetworks.com/blog/prisma-cloud/guide-protect-aws-s3/>
- xiv) Testing of the web application  
<https://tms-outsource.com/blog/posts/web-application-testing/>