Project Documentation

D4C Systems Inc.

Your partner in innovation

May 2021

Romy Peter | Logu R | Abdulrahman Syed | Krushang Sirikonda | Muzakkir Fazal

MIT License

Project Documentation
Authorization Memorandum

I have carefully assessed the Project Documentation for D4C Systems, Inc.'s WAD Project. This document has been completed in accordance with the requirements of System Development Methodology.

MANAGEMENT CERTIFICATION - Please check the appropriate statement.

_✓_ The document is accepted.

_____ The document is accepted pending the changes noted.

_____ The document is not accepted.

We fully accept the changes as needed improvements and authorize initiation of work to proceed. Based on our authority and judgment, the continued operation of this system is authorized.

Logu R	10th May 2021
Project Leader	DATE
Romy Peter	10th May 2021
Team Member	DATE
Abdulrahman Syed	10th May 2021
Team Member	DATE
Krushang Sirikonda	10th May 2021
Team Member	DATE
Muzakkir Fazal	10th May 2021
Team Member	DATE

TABLE OF CONTENTS

Page #

1.0 GENERAL INFORMATION

1.1 Project Overview

In this age of lockdowns and pandemics, there has been a surge in people working from home and thus, relying on online tools to keep the productivity bandwagon going. This has led to an increase in demand for file sharing apps to exchange information, virtual meetup applications to plan routines and services that can be used to serve video content. There are several free & paid websites that can be used by people to accomplish this. However, an app that integrates all these features into a single package and uses P2P for video conferencing is not available readily. Peer-to-Peer, also known as P2P, is an alternative to the client-server distribution model. Millions of consumers are routinely exchanging digital content files over P2P networks. People also tend to value their privacy these days. Therefore, we intend to address the aforementioned issue by creating a web app that can do primarily three things

- File Sharing
- Video Conferencing
- Video Server

1.2 Problem Statement

- During this period of COVID induced lockdowns to stop the spread of Coronavirus, a lot of people are now working from home. This led to a lot of work moving online and the demand for services that allowed to share, collaborate and work remotely to increase by a huge margin.

- Typically, a person who works remotely would need to manage a lot of files, connect with their colleagues and have virtual conferences and upload and download videos. However, it can be a daunting task to select a service considering good options for these services are becoming paid.

- Companies are looking to monetize the situation one way or another and the services that do not charge any fees for their services make up for the revenue by either harvesting user data or serving advertisements which can put the privacy of the end user at risk.

- To overcome the aforementioned issues, people might seek to get their own white-labelled version of these services or pay towards a company that values the privacy of customers or provide multiple functionalities in bulk. However, these might not be the optimal/ideal solutions.

1.3 Problem Solution

- Our solution to the problem is a three-fold approach: Provide all the required services in a single app, while providing the app for next to zero-cost and aim to protect the privacy of the user as much as possible within the scope available.

- Therefore, in this project we are designing a file sharing application which allows users to upload files they wish to collaborate and send the download link or email them, a video conferencing app that is able to provide identical features to other services provided on the market and a video server that allows quick uploading, downloading and viewing of important videos.

- Considering privacy & comfort in mind, we aim to store as less of the user's information as possible while providing the highest possible standards of safety and ease of use.

A Homepage

This project includes a homepage which is hosted at the link https://d4c-home.herokuapp.com/ and was created using ReactJS. The components that are used in the creation of the website include React Router and React Hooks. The website features a working navbar, some buttons and card components which lead to specific pages including the apps offered by our group.

A1. Installation

The website installation requires the following pre-requisites:

- NodeJS
- ReactJS
- Git
- Visual Studio Code or any other preferred text editor

Installation steps are as follows:

```
$ git clone https://github.com/scluzzy/d4c-home
$ cd d4c-home
$ npm install react-router-dom
$ npm start
```

Then, point your browser of choice to the address https://localhost:3000/ to access the homepage.

A2. Code Functionality:

There are various code snippets in this module that are essential to the functioning of the website and not all of them might be fully understood on first look. Therefore, some of the most essential code snippets and functions are presented here with their functions, so as to provide first time viewers of the project with some context. For fully commented source code, check the original file on Google Classroom.

- **app() in app.js:**
  In the app function, we can see the use of Router component from React-Router-DOM, which is a tool that helps in routing to inbuilt paths like the home page, services, and other modules.

- **navbar() in navbar.js:**
  This function is used to show the navbar, wherein we use hooks and usestate(). It is basically from React-Router-DOM.

The given below code snippet is a function that hides the buttons when viewing the website on a mobile device or when resizing windows.

```
const showButton = () => {
    if (window.innerWidth <= 960) {
      setButton(false);
    } else {
      setButton(true);
    }
  };
  window.addEventListener('resize', showButton);
```

The given below code snippet is a function that changes color of the navbar on scrolling.

```
const [navbar,setNavbar] = useState(false);
  const changeBackground = () => {
    if(window.scrollY >= 80) {
      setNavbar(true);
    } else {
      setNavbar(false);
    }
  };
  window.addEventListener('scroll', changeBackground);
```

The below given code snippet removes / fixes some errors which occur while revealing or hiding the buttons.

```
useEffect( () => {  showButton();} , []);
```

The below given code snippet which contains the ‘onClick()’ function closes the menu on a mobile device and ‘Link’ opens the given path, which is root here (/).

```
<Link to='/' className='navbar-logo' onClick={closeMobileMenu}>
```

The below given code snippet which contains ‘click()’ function changes the menu icon to the close icon when the condition is true.

```
<i className={click ? 'fas fa-times' : 'fas fa-bars'} />
```

The below given code snippet which contains ‘click()’ function changes the close icon to the menu icon when the condition is false.

```
<ul className={click ? 'nav-menu active' : 'nav-menu'}>
```

- **button() in button.js:**
  This function is basically used to load custom button components by varying different styles and sizes.

The below given code snippet basically checks and applies the style and size for the buttons present on the page.

```
  const checkButtonStyle = STYLES.includes(buttonStyle) ? buttonStyle : STYLES[0];

  const checkButtonSize = SIZES.includes(buttonSize) ? buttonSize : SIZES[0];
```

- **herosection() in herosection.js:**
  This function is used to load the section under the navbar in the home page.

- **cards() in cards.js:**
  This function is used to load the section under the hero section in the home page.

- **CardItem() in CardItem.js:**
  This function is used to take in props/parameters from cards.js and then make it, following which the card components are shown using those props in the cards section of the homepage.

- **footer() in footer.js**
  This function is used to show/call the footer component.

B Video Conferencing (JOYN)

This is a web-app which delivers a platform to the end-user in which one can easily communicate and interact with people like other conferencing applications on the market. Individuals can talk on video call, share content, hold meetings, classes or can simply chat. Using these basic features can bring about tremendous beneficial aspects in sectors like Healthcare, Education and Profession considering the COVID-19 induced lockdown these days. The app is serverless and also has a P2P based structure.

B1. Instructions to use the app:

Anyone who wishes to conduct a free meeting (with or without video) has to just click on the below given link:

https://joyn-g33.herokuapp.com/

A unique room ID will be generated each time the link is opened. A person who starts the meeting can invite others by clicking on the invite button, which provides the link in a shareable format. Webcam and Microphone permissions must be given when running for the first time. Users have the option to disable the audio and video using the buttons provided.

B2. Installation

The app installation requires the following pre-requisites:

- NodeJS
- Git
- npm
- Visual Studio Code (or any other text editor of your choice)

The following technologies were used in implementing the application:

- NodeJS
- ExpressJS
- Socket.IO
- PeerJS
- uuid
- express-sslify
- CORS

Installation steps are as follows:

```
$ git clone https://github.com/Logu-fosablanca/Joyn
$ cd Joyn
$ npm install
$ npm start
```

Once the server is up and running, point your browser of choice to the address https://localhost:5000/ to access the Joyn video conferencing application.

The steps involved for hosting the application online are as follows:

We used Heroku to host the app and it was free of any errors or issues. Therefore, it should be the preferred method to be used by anyone viewing this project for the first time.

- Firstly, before anything, clone https://github.com/Logu-fosablanca/Joyn into your Github
- Secondly, login into Heroku and connect your Github account.
- Thirdly, click ‘create an application’ on Heroku and provide the name of the app
- Finally, select the cloned Github repository and select the master branch to be deployed and enable automatic deployment and then click ‘deploy’

Congratulation! Joyn is now deployed on Heroku and is ready to be accessed worldwide.

B3. Code Functionality:

Typically, in every module, there is a Frontend and a Backend. But in the module for this app, there is no such separation because the it was easier to write it together as one combined version. Usually, for Frontend HTML is used widely but as we have combined the Frontend and Backend, it was easier to use Embedded JavaScript (EJS).

The project structure is:

```
root/Joyn/node_modules
root/Joyn/public
      /script.js
      /styles.css
root/Joyn/views
      /main.ejs
root/Joyn
      /gitignore
      /app.js
      /Readme
      /package.json
```

The general idea is that the callback is the last parameter. The callback gets called after the function is done with all of its operations. This part of the project can be a bit confusing for first timers as there are plenty of callback functions between files like script.js and app.js This module uses callbacks and events (i.e., event emitter object) as a main way to trigger functions

Target file: `root/Joyn/add.js`

This is the main file in the application which runs the main.ejs files and trigger the script.js in the public folder.

```
const express = require("express");
const app = express();
const server = require("http").Server(app);
const io = require("socket.io")(server);
//PeerServer helps establishing connections between PeerJS
clients.
const { ExpressPeerServer } = require("peer");//
const { v4: uuid } = require("uuid");
const cors = require("cors");
```

Here, we use express-sslify for turning the connection from an insecure HTTP one to a secure HTTPS one (line,app.js,19)

```
const enforce = require("express-sslify")
.
.
app.use(enforce.HTTPS({ trustProtoHeader: true }));
```

The below given code snippet allows the program to use EJS instead of HTML (line,app.js,16)

```
app.set("view engine", "ejs"); //Front End of the Application
```

The below given code snippet is a connect / express middleware that can be used to enable CORS (line,app.js,20)

```
app.use(cors({ origin: true }))
```

A cluster of callback functions and event listeners which are triggered using this single event listener function along with socket.io (line,app.js,39)

```
io.on("connection", (socket) =>
```

As soon as we enter the website, a unique room ID is generated and renders the main.ejs file for viewing while the main.ejs file uses the script.js file for its JS based functional scripting.

```
res.render("main", { roomID: id });//Rendering the main.ejs
through the view engine
```

Target file: `root/Joyn/script.js`

B4. Methods

addVideoStream(video, stream)

The srcObject property of the HTMLMediaElement interface sets or returns the object which serves as the source of the media associated with the HTMLMediaElement.

Parameters:

Name	Type	Description
video	video	video – media device's video
stream	stream	stream - the set of video and audio streams that are associated with the function

Source: script.js, line 71

connectNewUser(userID, stream)

Allows a user into a meeting room and adds video stream of the newly admitted user by default and pops it out as soon as they leave and calls using PeerJS.

Parameters:

Name	Type	Description
userID	string	This is a unique id generated when the user enters the meeting room
stream	stream	This is the existing stream of other people in the meeting when the user joins the meeting.

Source: script.js, line 85

handleActive(buttonClass)

This function activates and deactivates buttons based on their usage. If they are active, the buttons are deactivated, if not active, they are activated to use further functions on them like handlescreen() and other above-mentioned functions.

Parameters:

Name	Type	Description
buttonClass	QuerySelectorButton	

Source: script.js, line 290

handleInvite()

Creates an alert box which provides the link to the current meeting in a shareable format.

Source: script.js, line 328

handleMicrophone()

This function provides the functionality of muting or enabling the microphone according to the user's preference. It is linked with the mute button on the Frontend.

Source: script.js, line 179

handleScreen()

This function handles the opening and closing of the chat screen and participant's list screen - hide and removing them in the activeScreen so that it would be easier to do the closing and opening of the other buttons and their functions.

Source: script.js, line 241

handleVideo()

This function provides the functionality of disabling or enabling the webcam according to the user's preference. It is linked with the camera button on the Frontend.

Source: script.js, line 207

isHidden(screen)

This function checks whether the chat box is hidden or not and acts accordingly.

Parameters:

Name	Type	Description
screen	QuerySelectorButton	

Source: script.js, line 233

Returns: Whether the respective QuerySelector (screen) object is hidden or not in boolean value.

sendMessage(message)

This sends a message that is typed in the chat box during an active meeting.

Parameters:

Name	Type	Description
message	String	This is a string data which the user entered in the chat box

Source: script.js, line 109

systemMessage(username, join)

This function gets the input parameters, username and join (boolean value to say whether the user is in the meeting or not). If the user has joined the meeting, this function indicates the same along with the username and if he/she leaves the meeting, it indicates that the respective user has left the meeting in the chat box.

Parameters:

Name	Type	Default	Description
username	string		The username entered by the user when he joins the meeting
join	boolean	false	The activity of the user - whether the user is in the meeting or not (join==false) means he is not there in the meeting

Source: script.js, line 306

C File Sharing (Sharelux)

In this module, which uses a client-server architecture, we have two different parts: the frontend and the backend. We will be talking about the backend first. It is implanted as an API which can be accessed by the frontend. The user uploads the file with the help of a package called Multer, after which a download link is generated. With the help of Nodemailer, the user can send to send emails via SMTP from the node server which contains the download link for the file. All files are deleted automatically after 24 hours of inactivity. Since the frontend doesn't have much here, it will be mentioned at the end with the snippets of code use in it.

C1. Instructions to use the app

This app is very simple to use: anyone can send files less than a size of 100 MB without logging in and the data gets deleted in the next 24 hours keeping the privacy of the user in mind. The download link can also be emailed to a recipient and any type of file can be transferred.

Given below is a sample download link:

https://innsharee.herokuapp.com/files/adea9d77-21ed-4a3d-9467-9431b6745ca8 (We have hosted the application under the domain of 'Innsharee' as the domain "Sharelux" name wasn't available).

C2. Technologies Used

- NodeJS
- ExpressJS
- MongoDB Atlas
- Config
- Dotenv
- EJS
- Mongoose
- Multer
- Nodemailer
- uuid
- CORS

C3. Installation

Pre-requisites for installation: (a) NodeJS, (b) npm

```
$ git clone https://github.com/Logu-fosablanca/File-Sharing-API
$ cd File-Sharing-API
$ npm install
$ npm start
```

Once the server is up and running, point your browser of choice to the address https://localhost:3000/ to upload file via APIs or otherwise.

The steps involved for hosting the application online are as follows:

We used Heroku to host the app and it was free of any errors or issues. Therefore, it should be the preferred method to be used by anyone viewing this project for the first time.

- Firstly, before anything, clone https://github.com/Logu-fosablanca/ File-Sharing-API into your Github

- Secondly, login into Heroku and connect your Github account.
- Thirdly, click ‘create an application’ on Heroku and provide the name of the app
- Finally, select the cloned Github repository and select the master branch to be deployed and enable automatic deployment and then click ‘deploy’

Congratulations! Sharelux Backend is now deployed on Heroku and is ready to be accessed as an API or otherwise worldwide.

C4. Code Functionality:

The structure of the code is as follows:

```
root/File-Sharing-API/~$
                        /.en
                        /Package.json
                        /script.js
                        /server.js

root/File-Sharing-API/config/~$
                        /db.js

root/File-Sharing-API/models/~$
                        /files.js

root/File-Sharing-API/public/~$
                        /css/Style.css
                        /img (static files)

root/File-Sharing-API/routes~$
                        /download.js
                        /files.js
                        /show.js

root/File-Sharing-API/services~$
                        /emailService.js
                        /emailTemplate.js

root/File-Sharing-API/Testing~$
                        /testSMTP.js

root/File-Sharing-API/uploads~$
                        /.gitkeep

root/File-Sharing-API/views~$
                        /download.ejs
```

Here too, we use EJS compared to HTML due to ease of use considering this is pure backend.

C5. Methods:

Target file: `root/File-Sharing-API/config/db.js`

connectDB(none)

This function gets the MongoDB Atlas password and connection URL from .env (environment) file on a deployment environment and connect to the MongoDB Atlas using Mongoose library and enables the program to do data transfer.

Parameters:

Name	Type	Description
none	*	

Source: db.js, line 10

Target file: `root/File-Sharing-API/models/file.js`

This function defines the way how the data should be stored in the database and in which format the data should be uploaded.

```
const fileSchema = new Schema({
. . . . .
}, { timestamps: true });
```

Target file: `root/File-Sharing-API/routes/files.js`

i) This function lets the data to be uploaded in the upload folder and from there it will be uploaded to the cloud and we use multer to do that function as well. To avoid conflict, we store the data as unique number with the combination of random numbers and time and date.

```
let storage = multer.diskStorage({
. . . . .
    },
});
```

ii) This function lets to the data to be uploaded into the cloud with a unique file name while catching errors and this helps us to overcome CORS error.

```
router.post('/', (req, res) => {

    upload(req, res, async (err) => {
```

iii) This function gets the data from the database and sends a mail to the receiver if needed. This is quite understandable as we use the sendMail Function developed in in services/emailService.js to send the generated download link to the respective user.

```
router.post('/send', async (req, res) => {
    const { uuid, emailTo, emailFrom, expiresIn } = req.body;
. . . .
…                 expires: '24 hours'
            })
        })
    return res.send({ success: true})
});
```

Target file: `root/File-Sharing-API/routes/show.js`

This is the file where we render the download.ejs in the views/download.ejs to present the download link to the receiver or the user who wants to download the file directly from the link. We render the download.ejs using the template-engine (line 5) and present the download option to the user.

```
router.get('/:uuid', async (req,res)=>{
    Try{
        //We Search in the MongoDB database for the unique id for the
file
        const file = await File.findOne({ uuid: req.params.uuid });
. . . .
  catch(err){
        return res.render('download',{error: "Something Went Wrong"});
 } })
```

Target file: `root/File-Sharing-API/routes/download.js`

The respective functions search using uuid and saves the data or enables the user to download the data from the rendered download.ejs and and show.js and enable and download the final data.

```
router.get('/:uuid', async (req, res) => {
    // Extract link and get file from storage send download stream
    const file = await File.findOne({ uuid: req.params.uuid });
. . . .
   res.download(filePath);
});
```

Target file: `root/File-Sharing-API/services/emailService.js`

This function is responsible for implementation of SMTP in the file sharing app, which allows the user to enter an email ID to which the link can be sent to as well as mentioning who it was sent from. Here, the SMTP server used is Gmail's built-in SMTP server. The function takes JSON as the input and sends the file to the respective user. It uses a library called Nodemailer to send the email via the address shareluxg33@gmail.com

```
module.exports = async  ({ from, to, subject, text, html }) => {
    let transporter = nodemailer.createTransport(
        {host : 'smtp.gmail.com',
.
.
.
            console.log('Email sent successfully');
        }

    });
}
```

Target file: `root/File-Sharing-API/services/emailTemplate.js`

We use the emailTemplate.js file, where the function returns a HTML code and the code is sent as a template to the E-mail in the which the user will access the download link and the file. This inbuilt template file has a composition of pure HTML, JS & CSS.

```
module.exports = ({ emailFrom, downloadLink, size, expiresIn }) => {
    return `
        <!doctype html>
            <html>
            <head>
                <meta name="viewport" content="width=device-width">
.
.
.
           </body>
        </html>
    `;
}
```

Target file: `root/File-Sharing-API/Testing/testSMTP.js`

(async) sendMAil(EmailDetails)

This is a unit test test for the SMTP setup in which a testing mail is sent from the email address shareluxg33@gmail.com to test whether the SMTP setup is working normally, and uses Nodemailer, Config.json to send the testing mail.

Parameters:

Name	Type	Description
`EmailDetails`	JSON	The inputs are {from , to, subject in json format}

Source: testSMTP.js, line 15

Target file: `root/File-Sharing-API/scripts.js`

(async) fetchData(none)

This is a synchronous function that fetches data which are older than 24 hours and uses fs.unlink to delete all the old data from the database.

Parameters:

Name	Type	Description
none	*	

Source: script.js, line 13

Target file: `root/File-Sharing-API/server.js`

This is the application's main JS file. Here, we implement CORS so that the data can be sent between two different websites on a browser without any errors.

```
const express = require('express')
const app = express()
const path = require('path')
const cors = require('cors');
console.log(String(process.env.ALLOWED_CLIENTS))
// USING A cord to overcome the https http file transfer error
const corsOptions = {
.
.
.
}
app.use(cors(corsOptions))
```

Now, the Frontend modules will be listed. There is just a single JS file in the frontend and therefore, is easier to understand compared to the backend.

Target file: `index.js` (In this file we add the scripts that we use in the HTML file. Here, toast is used to display messages)

Functions:

dropZone.addEventListener("drop", __) is triggered when a file is dropped onto the dropbox element. In this we check if there is only one or multiple files and if the uploaded file is within the size limits or not. If it is not in the limit or if there are multiple files uploaded, we send a message with toast accordingly, else we upload the file.

dropZone.addEventListener("dragover"___) is triggered when a file is dragged over the dropbox zone.

dropZone.addEventListener("drag leave" ___) is triggered when the file is no longer dragged over the dropbox zone.

fileInput.addEventListener("change" ___) is a function that checks if file size in range, if not reset the input else just save it as upload file.

copyURLBtn.addEventListener("click", ____) is triggered when it is clicked, copies contents to clipboard and displays message “Copied to Clipboard “ by toast.

const uploadFile = () => is used to track the upload of the file and dynamically change the upload % bar . Also, we use XHR (xmlhtmalrequest), to actually upload the file we received.

emailForm.addEventListener("submit", ____) essentially checks for when the email address is submitted. When it is indeed submitted, it will save the values of mailed to and from address.

xhr.upload.onerror handles all errors that occur on the website.

D Video Server (Vidzilla)

This module includes login/signup along with token (jwt), video upload, thumbnail generation, video presentation, authentication middlewares, and much more functionalities.

D1. Installation

The app installation requires the following pre-requisites:

- NodeJS
- ReactJS
- Git
- MongoDB local version
- Visual Studio Code or any other preferred text editor

Installation steps are as follows:

```
$ git clone https://github.com/Logu-fosablanca/Vidzilla
$ cd videoServer
$ cd server
$ sudo service mongod start
$ npm install
$ npm start
$ cd ../client
$ npm install
$ npm start
```

Then, point your browser of choice to the address https://localhost:3000/ to access the homepage.

D2. Code Functionality

Target file: `videoThumbnail.js` (File responsible for creating video thumbnails using ffmpeg)

Functions – GenerateThumbnail (Line 13)

This function generates thumbnail and uploads them in the video_thumbnails folder. Then calls VideoDetails function to store in VideoDetails.

```
const generateThumbnail = (target, title, username) => {
```

Target file: `VideoDetailsSchema.js` (File responsible for storing video details in our database. This will be saving the entry after creating thumbnail)

Functions – uploadSchema (Line 3)

Function for video detail upload schema, saves uploader name, upload title, video path and thumbnail path

```
const uploadSchema = mongoose.Schema({
```

Target file: `checkauth.js` (File responsible for checking authorization and will receive request and it'll only call back next function if the token is valid. We check validity of token by jwt.verify)

```
    // Checking validity of token
    const decoded = jwt.verify(token, 'my_secret_key');
```

After which,

```
// storing decoded data in request
    req.userData = decoded;
```

Target file: `VideoList.js` (Route to create video thumbnail paths)

Line 7:

```
// Recieving data from VideoDetails module
router.get('/', (req, res, next) => {
```

`Dashboard`: It will contain all videos uploaded by everyone. Here, we will only render thumbnail which contains link to video an create a component VideoPlayer.

Line 14:

```
// If shouldredirect is true, we direct, else we dont
    let shouldRedirect = false;
    // If user has token we need to check if it's valid or not
    if (localStorage.getItem('userTokenTime')) {
```

Target file: `videoPlayer.js` – File responsible for playing videos from the URL

Line 19:

```
  componentDidMount() {
    // sending request
```

Line 49:

```
// before exiting, clear memory occupied by player
  componentWillUnmount() {
```

Line 55:

```
  // rendering the video
  render() {
```

Target file: `Navbar.js` – For implementing the navbar

Line 25:

```
// User will see Home, upload and signout option when logged in
              // And signin and signup when not logged in
                <React.Fragment>
```

Target file: `Upload.js` (This file is responsible for uploading and storing videos on the server)

Line 9:

```
// Storing the videos in media/uploads
const storage = multer.diskStorage({
```

Target file: `signin.js`
This handles everything related to checking user's legitimacy and generating a token for him. Here, we use JSWtoken to create a web token, and b.crypt in order to compare the encrypted password (hash) the user has entered, with the one in the database.

Functions:

- User.find: It takes in the credentials of the user on the signup page and searches the database to check if they exist. if the user is not found, we send back an auth failed message.
- bcrypt.compare: It converts the user entered password to hash and compares it to the user's password in database. If wrong password is provided, we send back an auth failed message.
- jwt.sign: Once we know the user is legitimate, we now move on to generating a token for him, and authorize the user, which is done by jwt.sign.

Target file: `signup.js`
This file handles everything related to signing up a user and storing his credentials in the database. Here we use mongoose for handling MongoDB and b.crypt for saving the user entered password as a hash.

Functions:

- User.find: It checks if the entered credentials already exists, if it does send a profile already exists message. Else, we save the new user credentials.
- bcrypt.hash: It used to convert the password in to hash and salting it before storing it db.

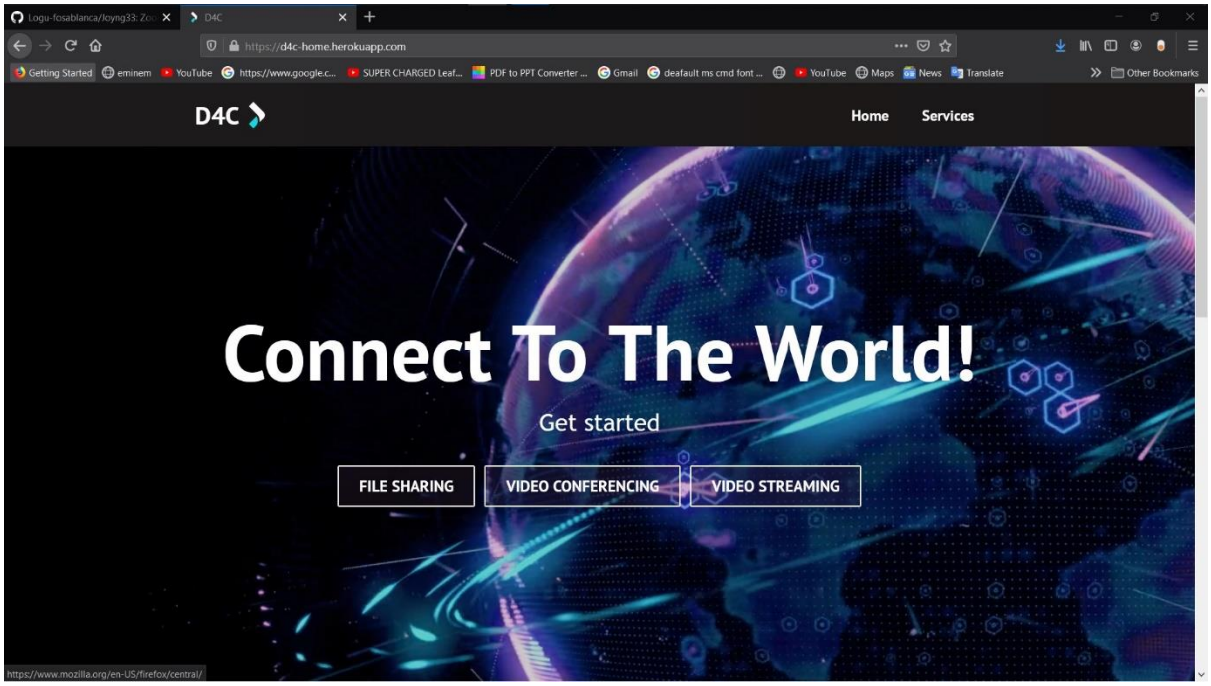
Target file: `Upload.js`
This file handles storing the uploaded file after generating the thumbnail. Here, we use multer to help with storing the file.

Functions:

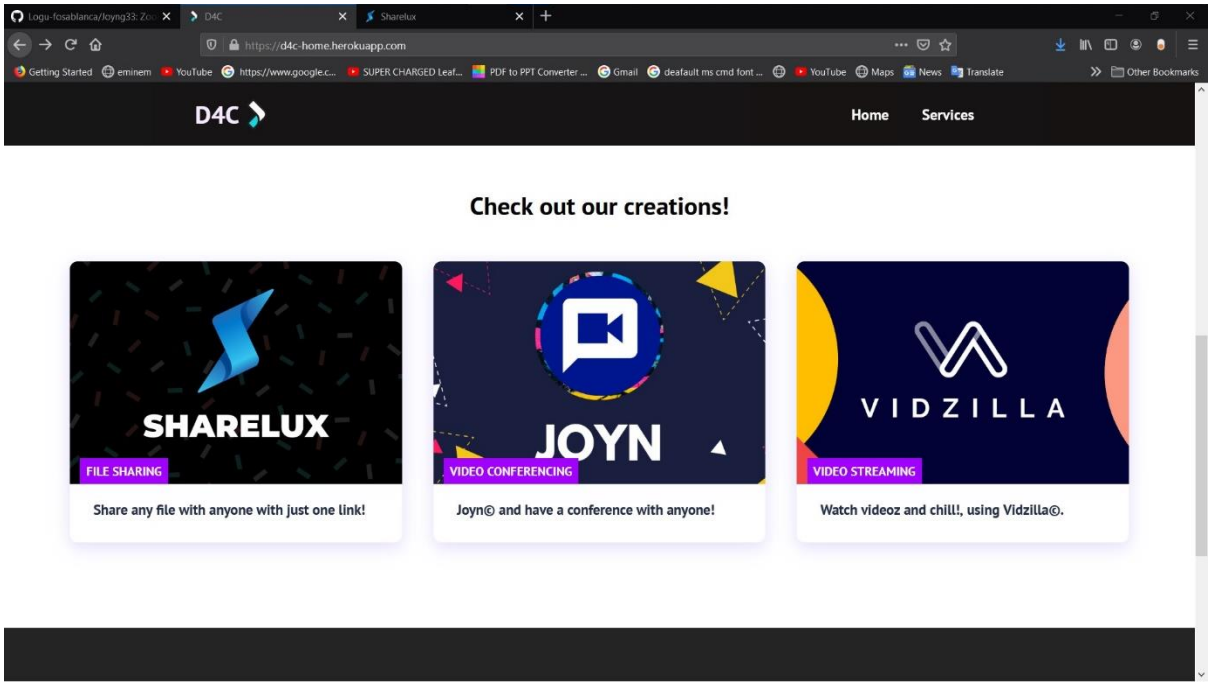
- multer.diskStorage :- Sets up multer for use.
- thumbnailGenerator.generateThumbnail :- Generates thumbnails from the uploaded video

E Screenshots

E1. Home Page

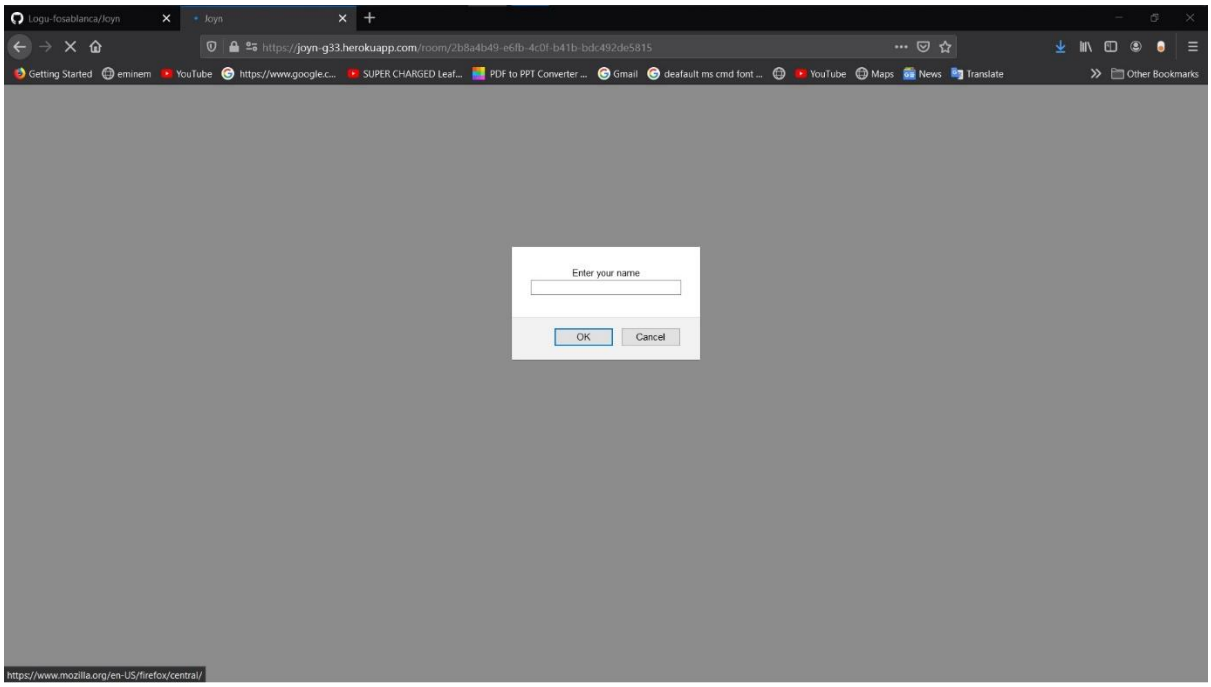


Screenshot of the landing page

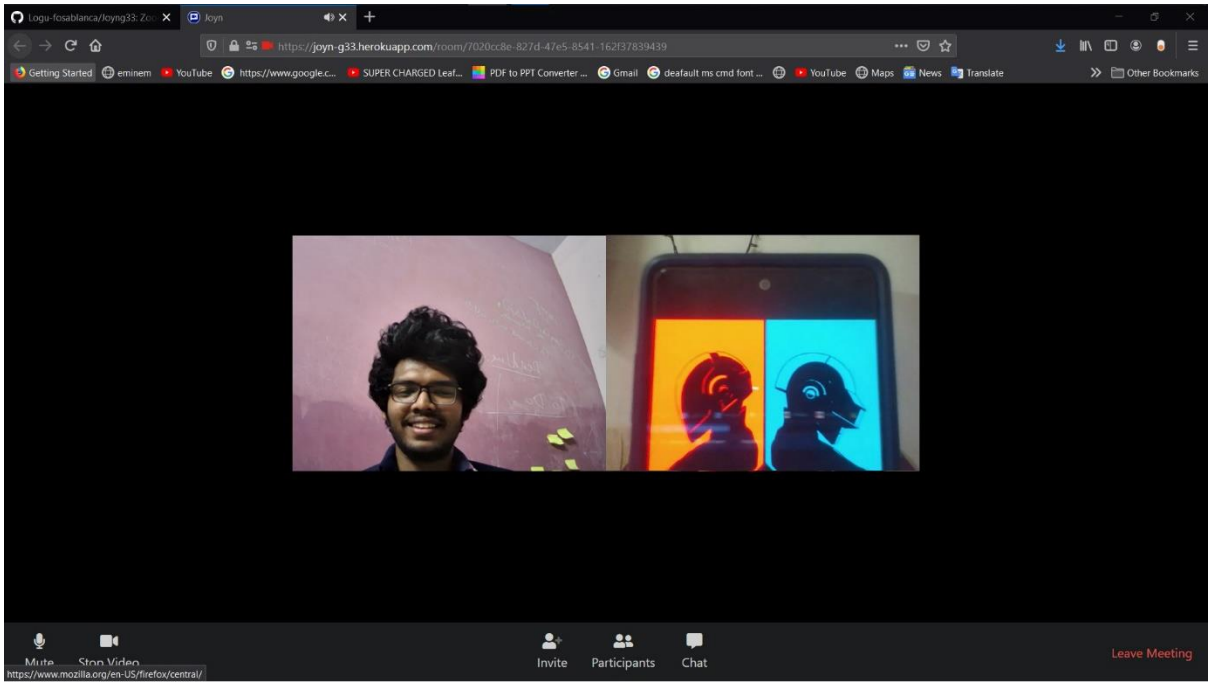


Screenshots which shows links to the apps developed for the project

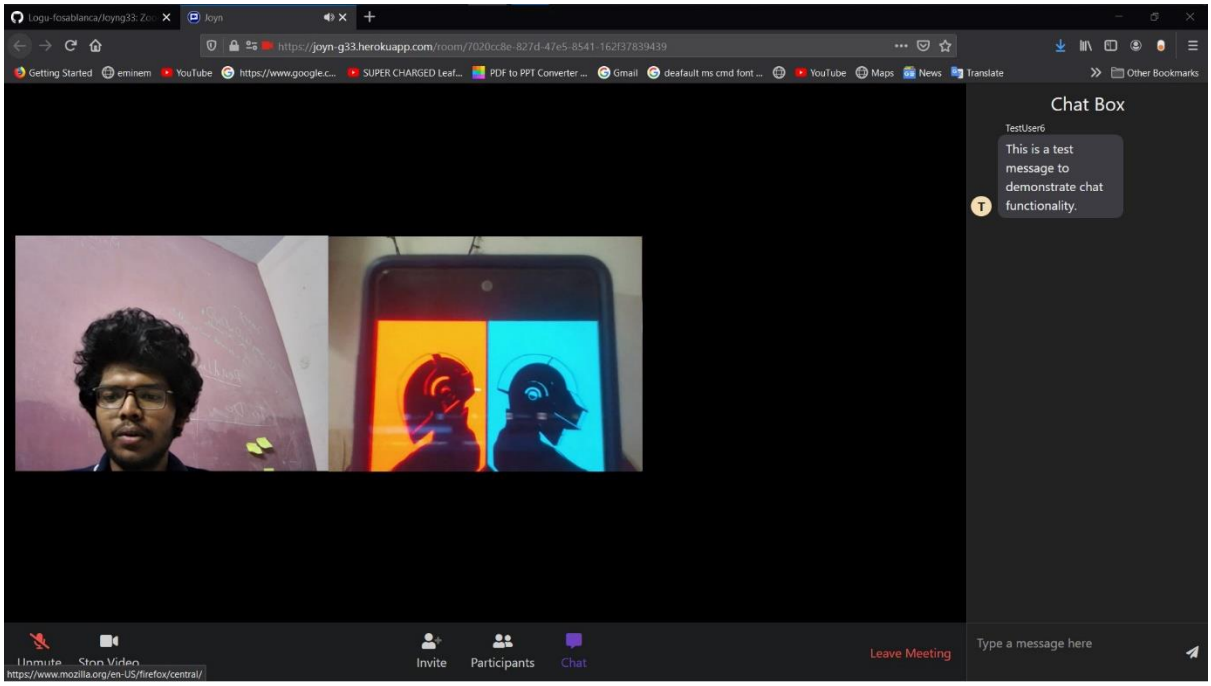
E2. Joyn



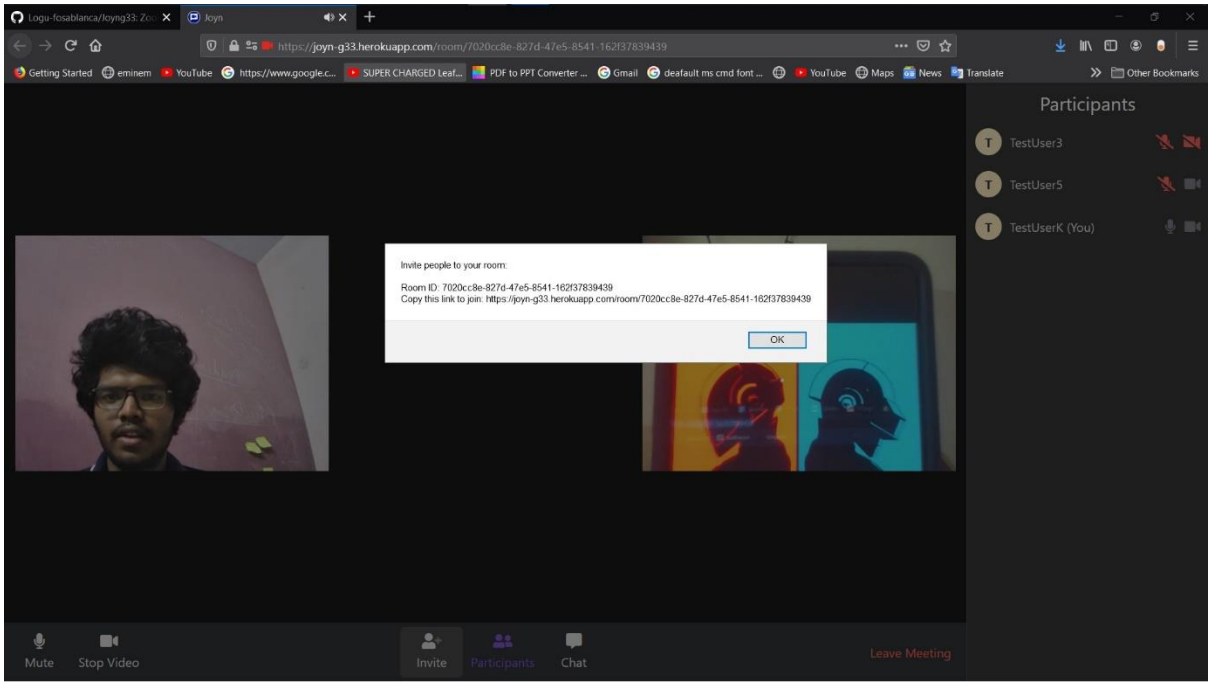
Demonstration of entering username before joining meeting room



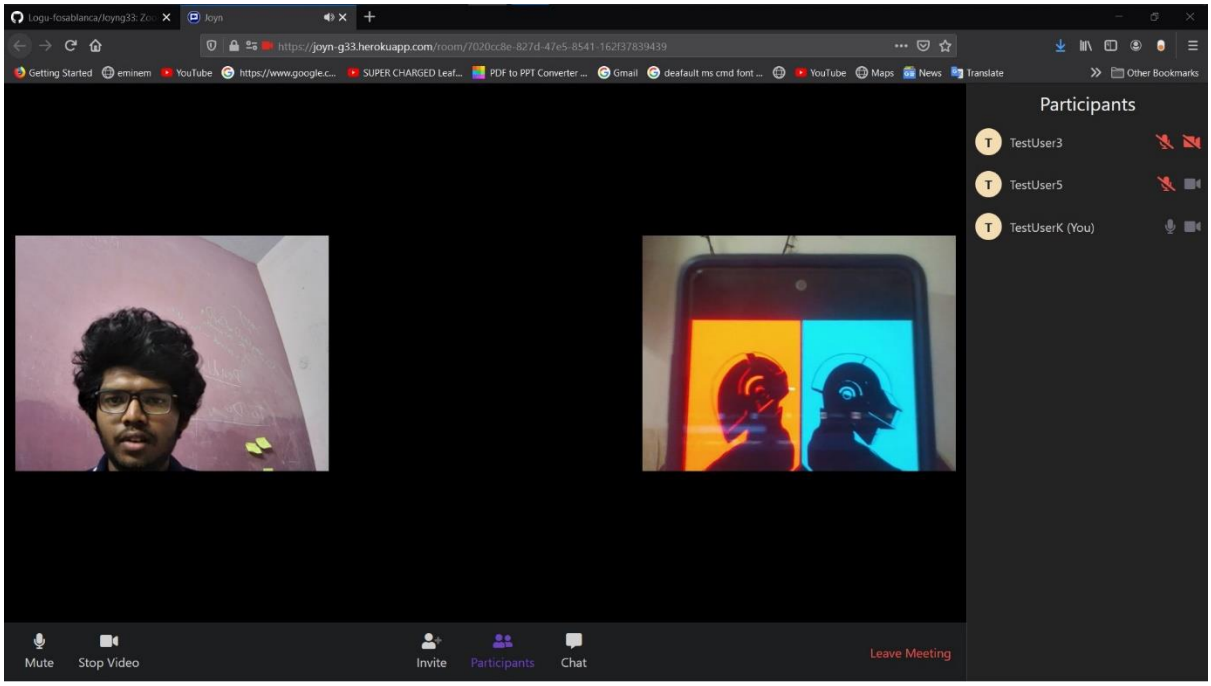
Screenshot of Meeting Room



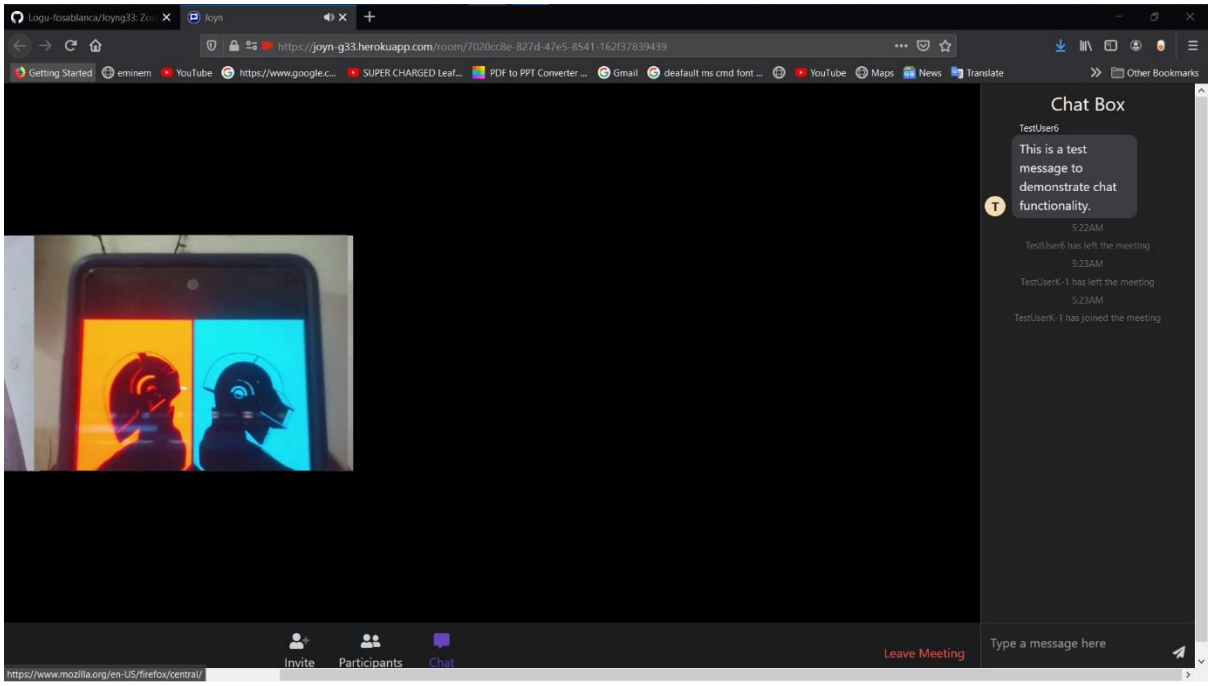
Demonstration of Chat Functionality



Demonstration of Invite Functionality

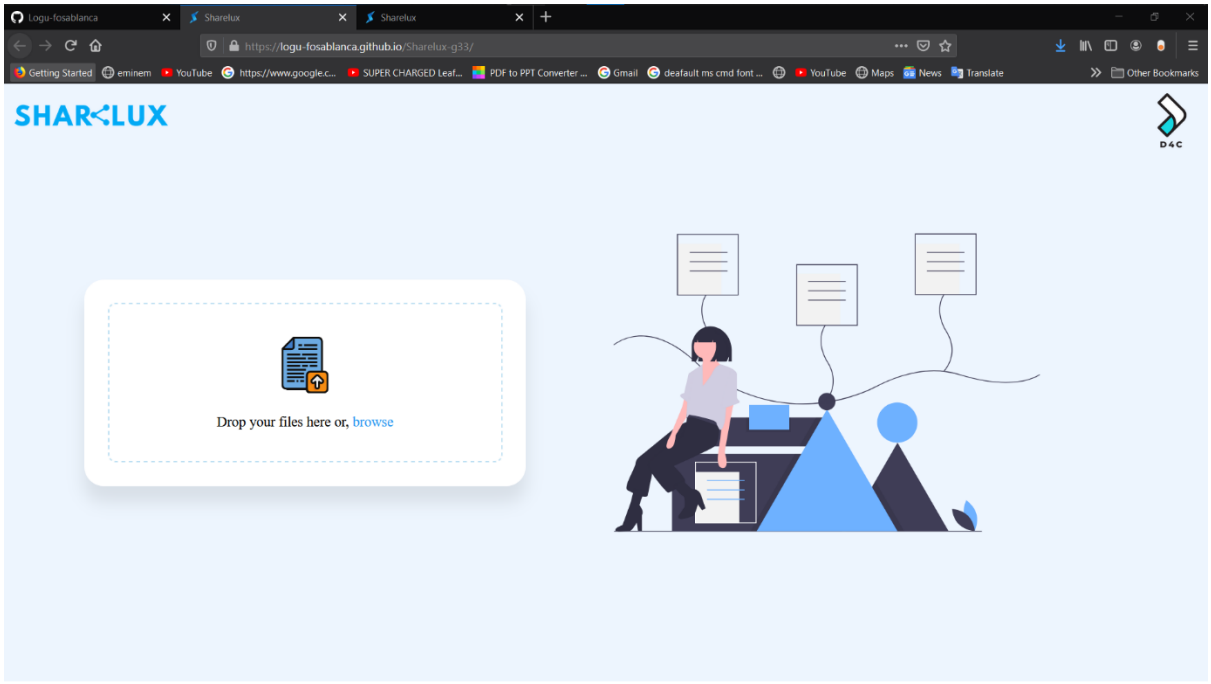


Demonstration of Participant's List

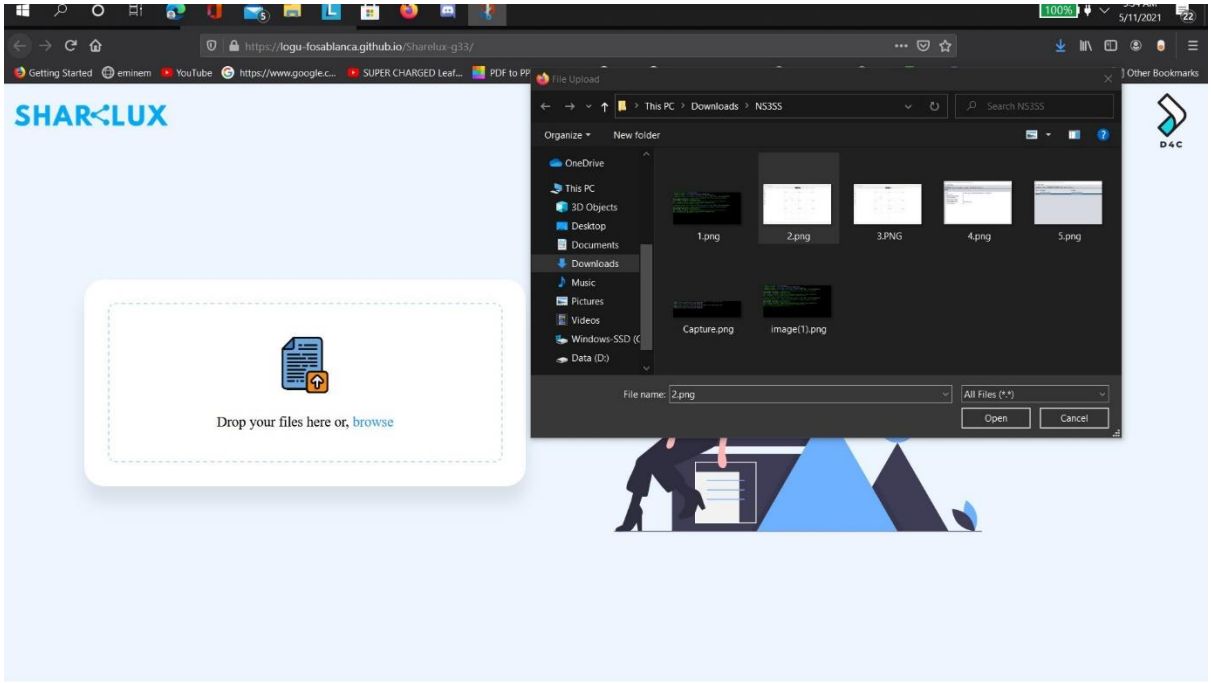


Demonstration of notifications stating participants joining and leaving.

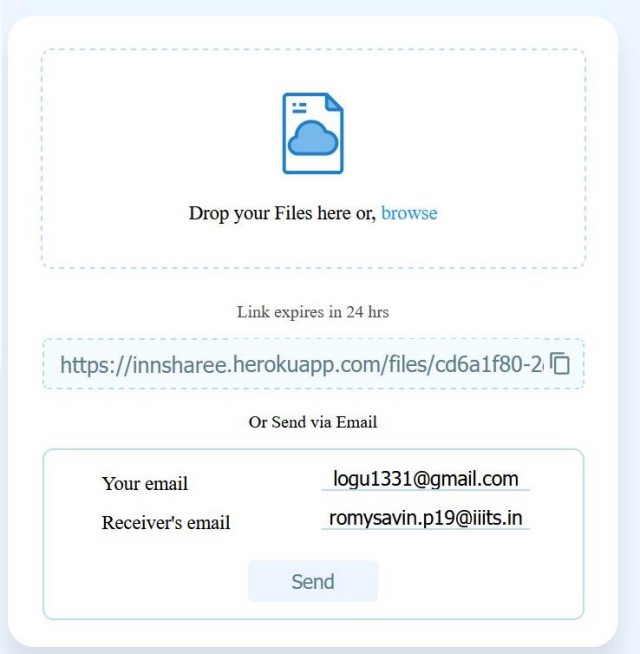
E3. Sharelux



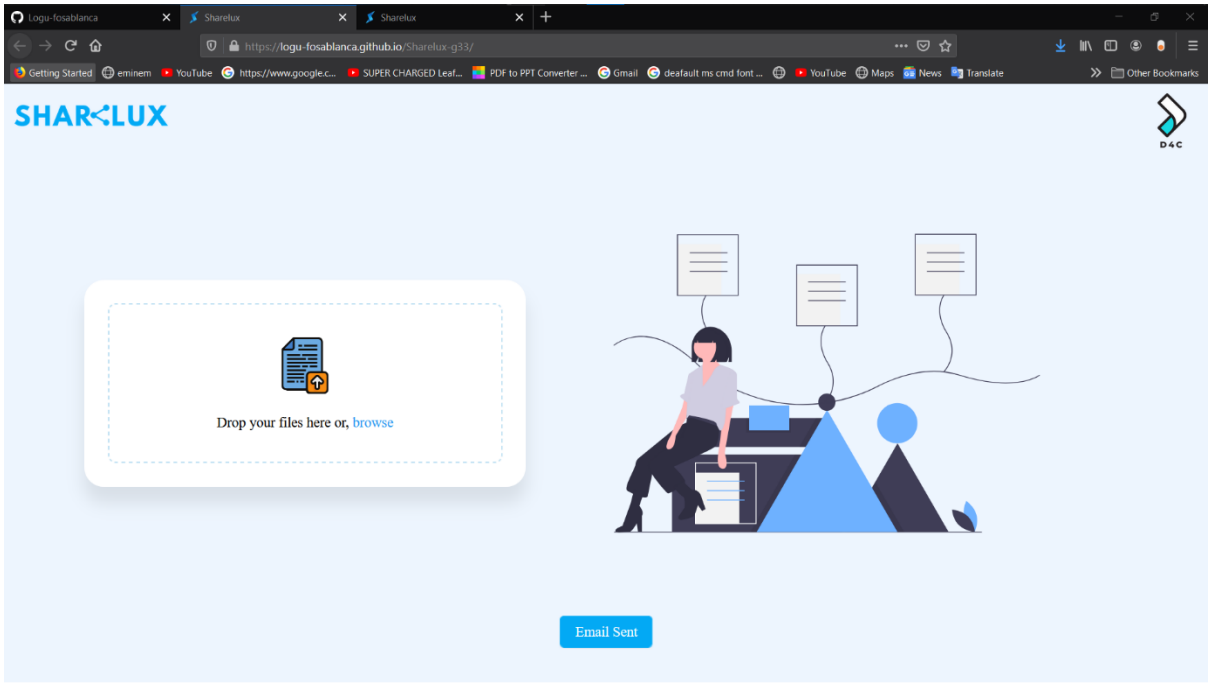
Screenshot of file upload page



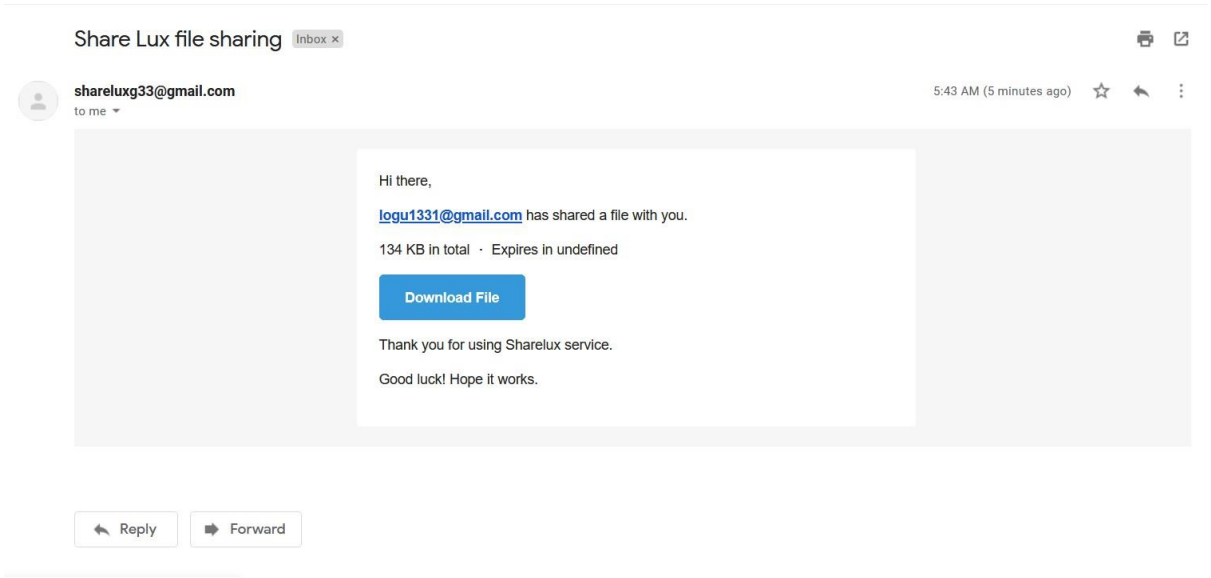
Demonstration of selecting file to be uploaded



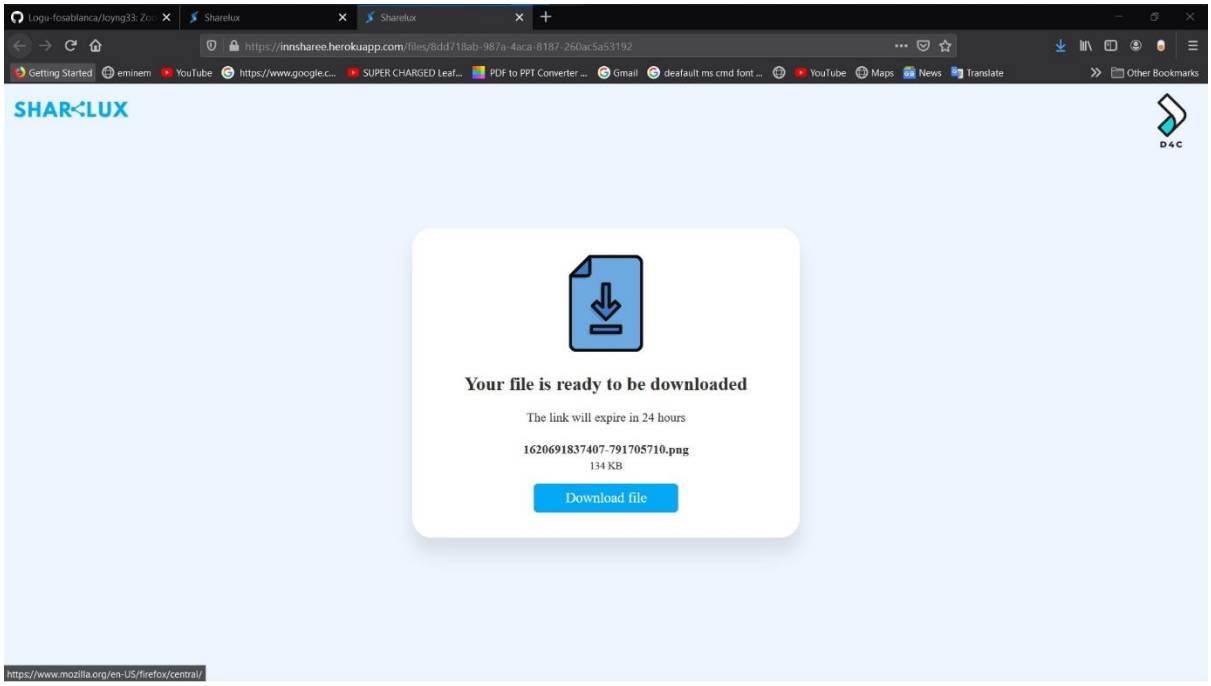
Screenshot of download link, along with option to send via email



Demonstration of sending link via email, shows 'email sent' at bottom

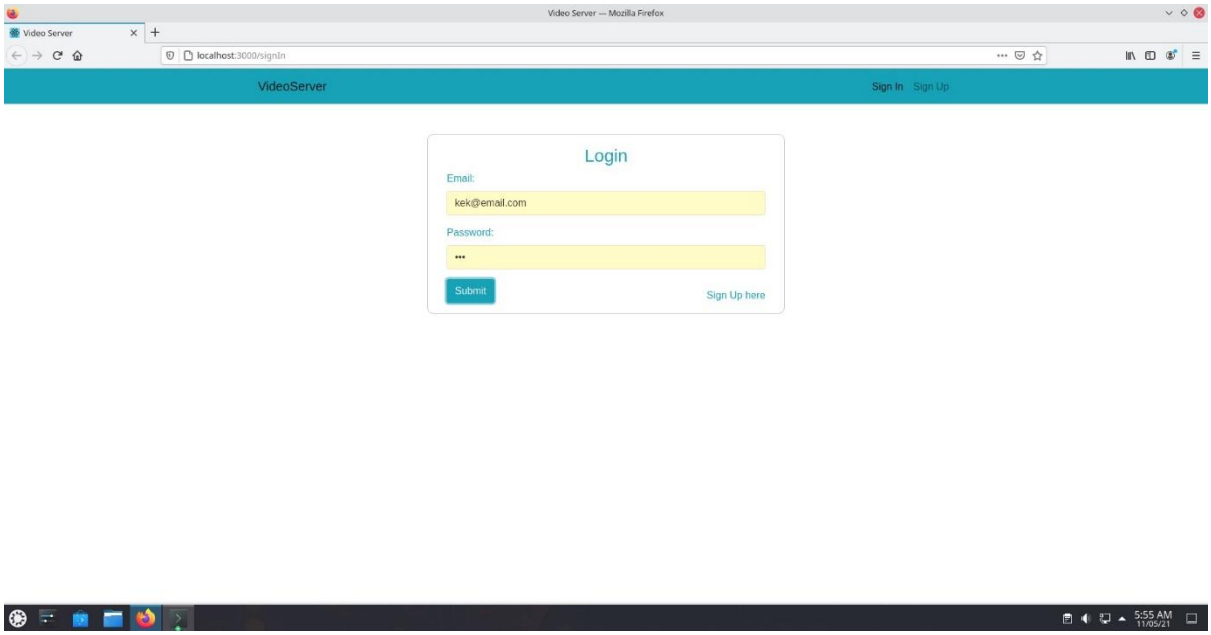


Screenshot of the email containing the download link

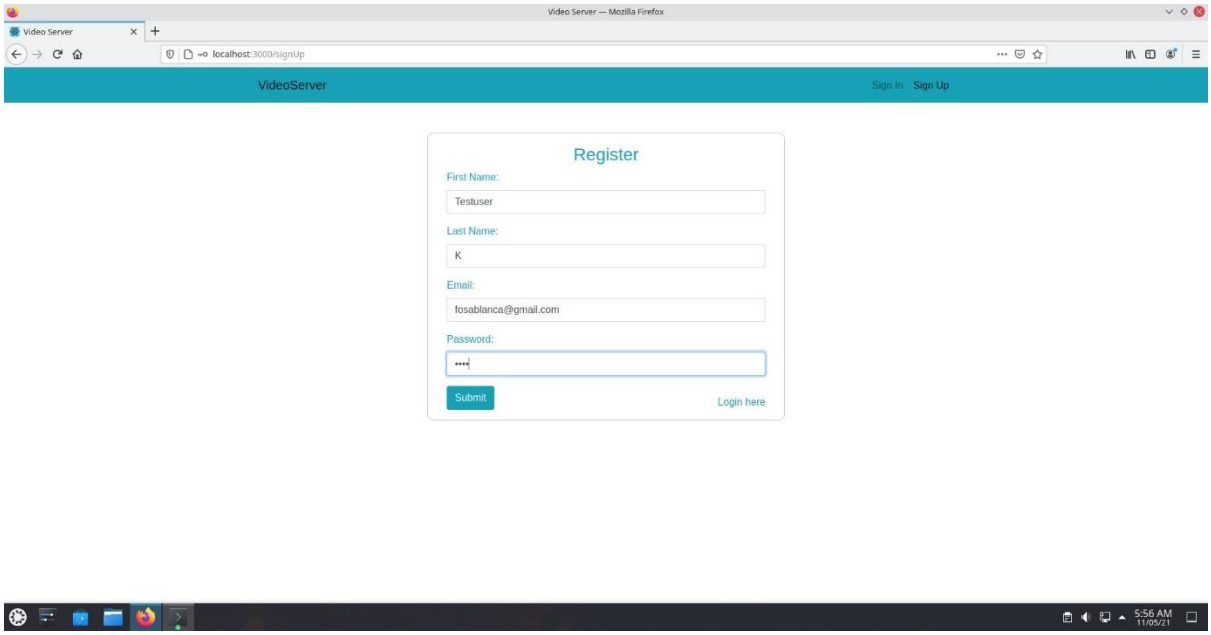


Screenshot of the file download page

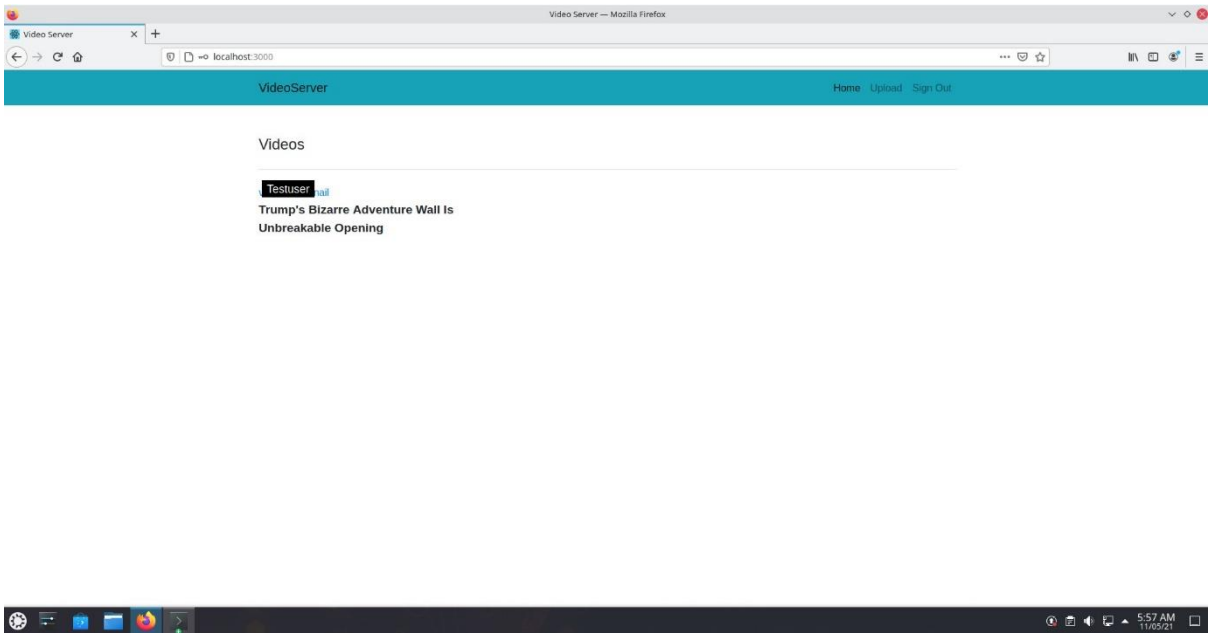
E4. Vidzilla



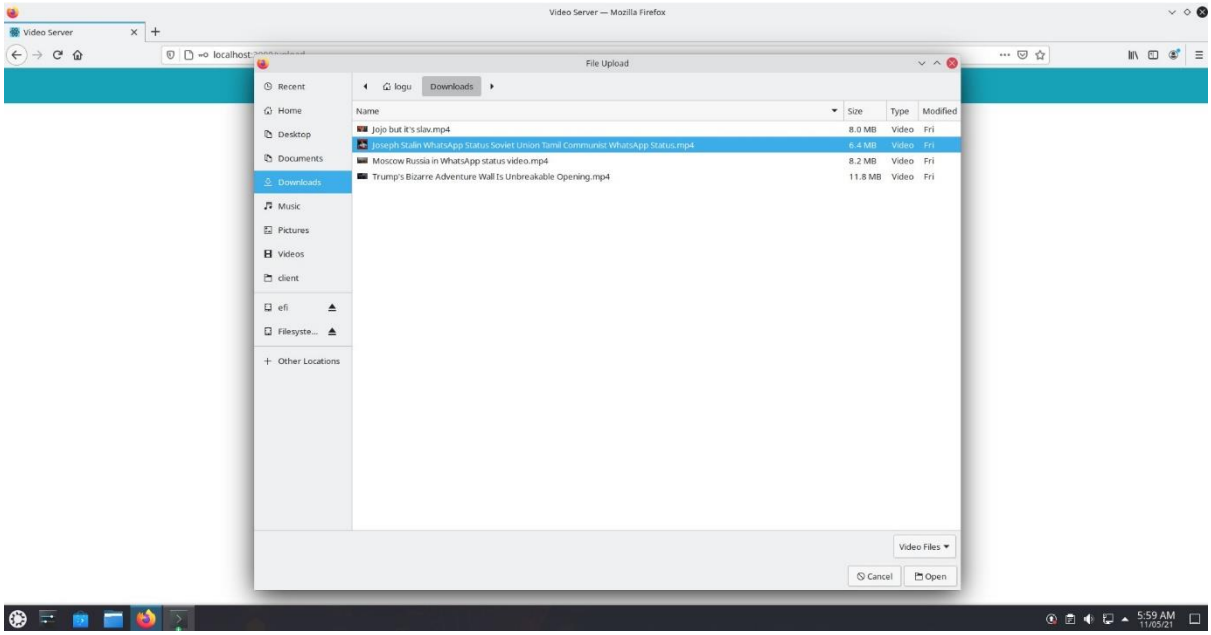
Screenshot of the sign-in page



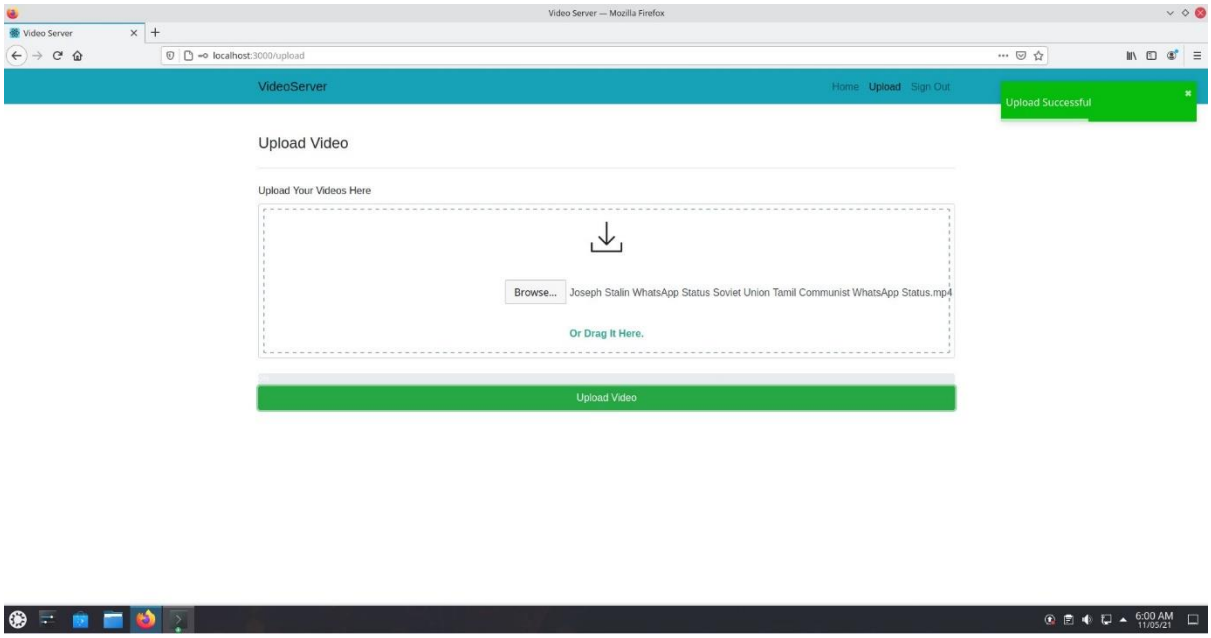
Screenshot of the sign-up page



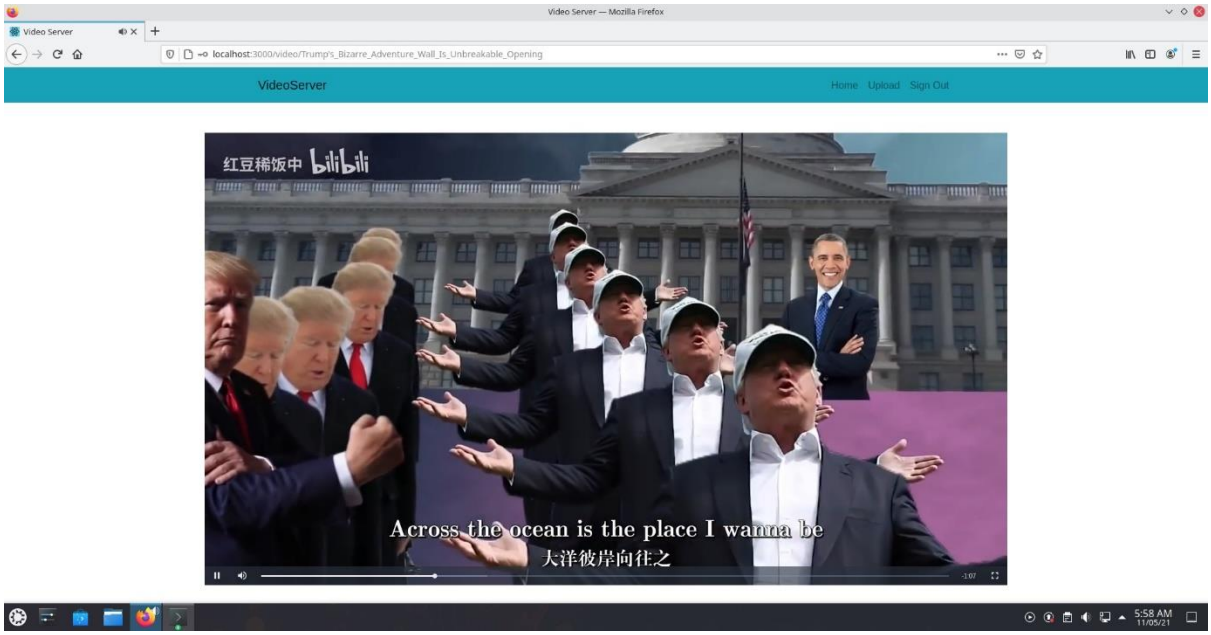
Screenshot of the main landing page after signing in



Demonstration of uploading a video file



Screenshot showing that uploading of a video file is successful



Screenshot of video player, which plays the uploaded video files

END
OF
PROJECT
DOCUMENTATION