Media Streaming with IBM Cloud Video Streaming

Phase 1: Problem Definition and Design Thinking

In this part you will need to understand the problem statement and create a document on what have you understood and how will you proceed ahead with solving the problem. Please think on a design and present in form of a document.

Problem Definition:

The project involves creating a virtual cinema platform using IBM Cloud Video Streaming. The objective is to build a platform where users can upload and stream movies and videos on-demand. This project encompasses defining the virtual cinema platform, designing the user interface, integrating IBM Cloud Video Streaming services, enabling on-demand video playback, and ensuring a seamless and immersive cinematic experience.

Design Thinking:

1. **Project Overview:**

   The "Media Streaming with IBM Cloud Video Streaming" project aims to design, develop, and implement a media streaming solution utilizing IBM Cloud Video Streaming services. This project will enable organizations to efficiently deliver live and on-demand video content to a global audience, ensuring high-quality streaming experiences while leveraging the robust infrastructure and features provided by IBM Cloud.

2. **Project Objectives:**
   - **Streamlined Content Delivery:** Implement a seamless media streaming platform that allows for the delivery of live and on-demand video content to a wide range of devices and locations.
   - **High-Quality Streaming:** Ensure the highest quality of video streaming with adaptive bitrates, low latency, and minimal buffering to provide an exceptional user experience.

- **Global Scalability:** Build a scalable architecture that can handle varying levels of audience demand and geographic distribution.
- **Security and Access Control:** Implement security measures to protect intellectual property and restrict access to authorized viewers.
- **Analytics and Monitoring:** Integrate monitoring and analytics tools to gather insights into user engagement, content performance, and network efficiency.
- **User Engagement:** Enhance user engagement through interactive features like chat, polls, and social media integration.
- **Content Management:** Develop a user-friendly content management system to organize and manage media assets efficiently.

3. **Project Scope:**
   - **Platform Selection:** Evaluate and select the appropriate IBM Cloud Video Streaming services and solutions based on project requirements.
   - **Streaming Infrastructure:** Design and deploy the necessary infrastructure to support media streaming, including content delivery networks (CDNs), edge servers, and data storage.
   - **Integration:** Integrate the streaming platform with content management systems, user authentication, and analytics tools.
   - **User Interface:** Develop user interfaces for content creators, administrators, and viewers to interact with the streaming platform.
   - **Security Measures:** Implement access controls, encryption, and digital rights management (DRM) to protect content and user data.
   - **Analytics and Reporting:** Set up tools for real-time monitoring, analytics, and reporting of streaming performance and user engagement.
   - **Documentation and Training:** Create documentation and provide training for administrators and content creators to effectively use and manage the streaming platform.
   - **Testing and Quality Assurance:** Conduct rigorous testing, including load testing, to ensure the platform performs reliably under varying conditions.
4. **Project Deliverables:**
   - Media streaming platform with IBM Cloud Video Streaming integration
   - User interfaces for content management and viewing
   - Security protocols and access control mechanisms
   - Analytics and monitoring dashboard
   - Documentation for administrators and users
   - Training materials and sessions
   - Quality assurance reports and testing documentation

5. **Project Timeline:**
    - Project Initiation: [Start Date]
    - Platform Selection and Infrastructure Setup: [1-2 months]
    - Integration and Development: [3-4 months]
    - Testing and Quality Assurance: [2-3 months]
    - Documentation, Training, and Deployment: [1-2 months]
    - Project Closure: [End Date]
6. **Project Team:**
    - Project Manager
    - Solution Architects
    - Developers and Programmers
    - Quality Assurance Testers
    - Content Creators and Managers
    - Network and Security Experts
    - User Interface Designers
    - Analytics and Monitoring Specialists
7. **Budget:**
    - The budget for this project will be determined based on the specific requirements and scope, including infrastructure costs, licensing fees, and personnel expenses.
8. **Risks and Mitigations:**
    - Identify potential risks related to technology, scalability, security, and user adoption and develop mitigation strategies.
9. **Stakeholder Communication:**
    - Establish a communication plan to regularly update stakeholders on
    - project progress and milestones.
10. **Project Evaluation:**
    - Define key performance indicators (KPIs) and success criteria to evaluate the effectiveness of the media streaming platform and make necessary improvements.

By executing the "Media Streaming with IBM Cloud Video Streaming" project, organizations can enhance their ability to deliver high-quality video content to a global audience while benefiting from the capabilities offered by IBM Cloud's video streaming services.

Media Streaming With IBM Cloud Video Streaming

1. **Project Idea and Objectives:** Start by defining the purpose and objectives of your media streaming project. What kind of content will you stream, and what is the desired outcome?

2. **Content Creation:**Create or gather the content you want to stream, ensuring it is of high quality and appropriate for your target audience.

3. **IBM Cloud Video Streaming Setup:**

- Sign up for an IBM Cloud account and access the Video Streaming service.
- Configure your streaming settings, such as video quality, encoding options, and security settings

4. **Streaming Infrastructure:**

- Consider the hardware and software infrastructure required to support streaming, including encoding servers, content delivery networks (CDNs), and redundancy measures.

5. **User Interface (UI):**

- Design an intuitive and attractive user interface for your streaming platform. Consider the user experience, including navigation, playback controls, and user interactions.

6. **Mobile Accessibility:**

- Ensure your streaming platform is accessible on mobile devices, as many users watch content on smartphones and tablets.

7. **Content Delivery Strategy:**

- Develop a strategy for content delivery to ensure low latency and a smooth viewing experience. Leverage CDNs and edge computing to reduce buffering.

8. **Interactivity:**
- Implement interactive features, such as live chat, comments, or polls, to engage your audience and make the streaming experience more interactive.

9. **Monetization Options:**
- If applicable, explore monetization options like pay-per-view, subscription models, or advertising integration.

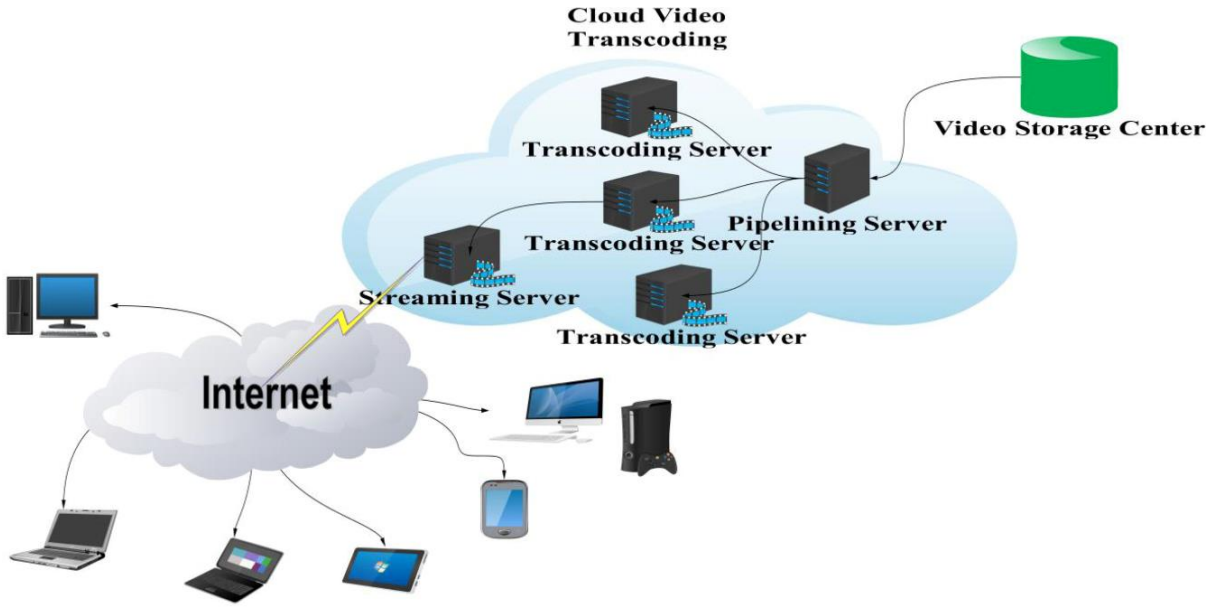


Fig. 1. Generic architecture of cloud-based media streaming.

10. **Security:**
- Implement security measures to protect your content and the privacy of your viewers. This may include DRM (Digital Rights Management) and encryption.

11. **Testing and Quality Assurance:**
- Rigorously test your streaming platform to ensure it works smoothly on various devices and under different network conditions.

12. **Marketing and Promotion:**
- Develop a marketing strategy to attract and retain viewers. Utilize social media, email marketing, and partnerships to promote your content.

13. Analytics and Insights:

- Integrate analytics tools to gather data on viewer behavior, engagement, and performance. Use this data to refine your streaming strategy.

14. Feedback Loop:

- Create a system for receiving and acting on user feedback to continuously improve your streaming service.

15. Scaling and Future Development:

- Plan for scalability as your project grows. Consider future enhancements, such as virtual reality streaming or 360-degree video.

16. Compliance and Legal Considerations:

- Ensure that your project complies with copyright laws and any other legal requirements in your region.

17. Documentation and Support:

- Provide clear documentation for users and offer customer support to address any issue.

MEDIA STREAMING WITH IBM VIDEO STREAMING

We'll use **FLASK, IBM-DB2** for the **backend** and **HTML, CSS, JAVASCRIPT** for the **frontend**, and a basic in-**memory data structure** for the product catalog with **IBM CLOUD OBJECT STORAGE** using **DOCKER and KUBERNETES** in containers.

Media Streaming Features:

1. **Live Streaming:** Ability to broadcast live events in real-time to audiences.
2. **On-Demand Streaming:** Store and stream pre-recorded content for users to access at their convenience.
3. **Multi-bitrate Streaming:** Deliver content at different quality levels to accommodate various internet connection speeds.
4. **Content Management:** Organize and categorize videos into playlists or libraries for easy access.
5. **Monetization Options:** Support for subscription models, pay-per-view, or ad-based monetization.
6. **Analytics and Insights:** Track viewer engagement, audience demographics, and video performance.
7. **Custom Branding:** Customize the video player and interface with your brand's logo, colors, and themes.
8. **Security Features:** Implement content protection measures such as encryption, access control, and DRM (Digital Rights Management).
9. **Live Chat and Interaction:** Engage with viewers through live chat, comments, and interactive features.
10. **Mobile Compatibility:** Support for streaming across various devices and operating systems.

User Interface Design:

An intuitive user interface (UI) for IBM Video Streaming can include the following elements:

1. **Homepage:** Featuring categorized sections for live events, recommended content, and trending videos.
2. **Navigation Menu:** Clear and simple navigation for Live, On-Demand, Categories, Search, and User Profile sections.
3. **Video Player:** Clean, customizable, and responsive player with play/pause, volume control, and playback options.
4. **Search Bar:** Easy-to-use search functionality to find specific content.
5. **User Profile:** Allows users to manage their subscriptions, preferences, watch history, and account settings.
6. **Content Categories:** Sections dedicated to different types of content (e.g., sports, news, entertainment) for easy browsing.
7. **Call to Action Buttons:** Prominently placed buttons for signing up, subscribing, or accessing premium content.

User Registration and Authentication Mechanisms:

To ensure secure access to the platform, robust registration and authentication mechanisms should be in place:

1. **Registration:** Users sign up with an email, username, and password.
2. **Email Verification:** Send a verification link to the user's provided email for account confirmation.
3. **Two-Factor Authentication (2FA):** Offer an additional layer of security with 2FA via SMS or authenticator apps.
4. **Account Management:** Allow users to reset passwords, update account information, and manage security settings.
5. **Access Control:** Implement user roles and permissions to manage access to different features or content based on user types (e.g., viewers, administrators).
6. **Encryption:** Employ encryption protocols to secure data transmission and storage.

7. **OAuth Integration:** Enable social media or single sign-on options for simplified login.

To Setup User Registration and Authentication:

Frontend Code of HTML and CSS is used

```
<!DOCTYPE html>
<html>
<head>
  <title>User Registration</title>
  <style>
    /* Basic CSS for styling the form */
    /* Add your own styling as needed */
    .form-container {
      width: 300px;
      margin: 0 auto;
    }
    .form-group {
      margin-bottom: 15px;
    }
  </style>
</head>
<body>
  <div class="form-container">
```

```
    <h2>User Registration</h2>
    <form id="registrationForm">
      <div class="form-group">
        <label for="username">Username:</label>
        <input type="text" id="username" name="username" required>
      </div>
      <div class="form-group">
        <label for="email">Email:</label>
        <input type="email" id="email" name="email" required>
      </div>
      <div class="form-group">
        <label for="password">Password:</label>
        <input type="password" id="password" name="password"
required>
      </div>
      <div class="form-group">
        <button type="submit">Register</button>
      </div>
    </form>
  </div>

  <script>
```

```
    // JavaScript for handling form submission

document.getElementById('registrationForm').addEventListener('submit', function(event) {
        event.preventDefault();
        const username = document.getElementById('username').value;
        const email = document.getElementById('email').value;
        const password = document.getElementById('password').value;

        // Here, you can perform further actions like sending this data to the backend for registration process
        // For the example, you can log the data to the console
        console.log('Username: ', username);
        console.log('Email: ', email);
        console.log('Password: ', password);

        // Additional steps: Use Fetch API or other means to send data to the backend for user registration
        // This frontend code simulates the form submission and logging the entered data.
    });
  </script>
</body>
</html>
```

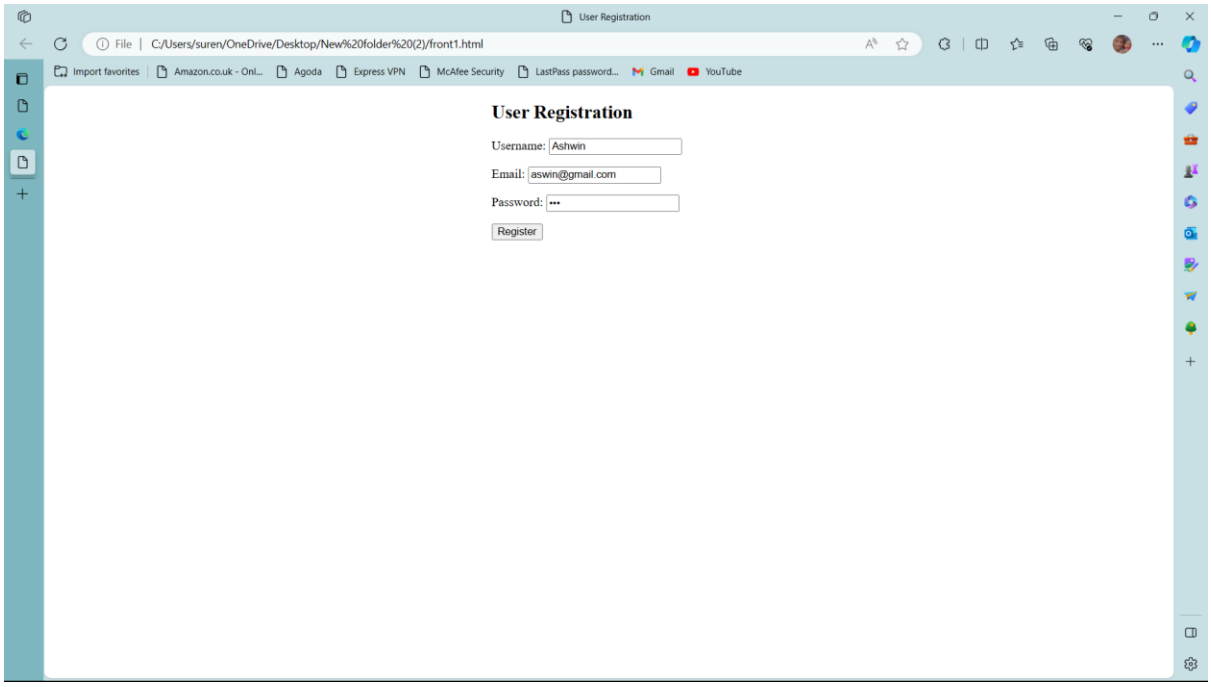
Backend of Javascript is used:

```
// Required dependencies
const express = require('express');
const bodyParser = require('body-parser');
const mongoose = require('mongoose');
const bcrypt = require('bcrypt'); // For password hashing

const app = express();

// Body parser middleware
app.use(bodyParser.urlencoded({ extended: false }));
app.use(bodyParser.json());

// MongoDB connection setup
mongoose.connect('mongodb://localhost:27017/mydatabase', {
  useNewUrlParser: true,
  useUnifiedTopology: true,
});

// User model schema
const User = mongoose.model('User', {
```

```
  username: String,
  email: String,
  password: String,
});

// User registration endpoint
app.post('/register', async (req, res) => {
  const { username, email, password } = req.body;

  // Check if the user already exists
  const existingUser = await User.findOne({ email });

  if (existingUser) {
    return res.status(409).json({ message: 'User already exists' });
  }

  // Hash the password
  const hashedPassword = await bcrypt.hash(password, 10);

  // Create a new user
  const newUser = new User({
```

```
    username,
    email,
    password: hashedPassword,
  });

  await newUser.save();

  res.status(201).json({ message: 'User registered successfully' });
});

// User login endpoint
app.post('/login', async (req, res) => {
  const { email, password } = req.body;

  // Find the user by email
  const user = await User.findOne({ email });

  if (!user) {
    return res.status(404).json({ message: 'User not found' });
  }
```

```
  // Compare hashed password
  const passwordMatch = await bcrypt.compare(password, user.password);

  if (!passwordMatch) {
    return res.status(401).json({ message: 'Invalid password' });
  }

  res.status(200).json({ message: 'Login successful' });
});

// Server setup
const PORT = 3000;
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}`);
});vv
```



THEREFORE, LOGIN PAGE HAS BEEN DEPLOYED SUCCESSFULLY

MEDIA STREAMING WITH IBM CLOUD VIDEO STREAMING

PHASE-4: DEVELOPMENT PART – 2

Continue building the platform by integrating video streaming services and enabling on-demand playback. Implement the functionality for users to upload their movies and videos to the platform. Integrate IBM Cloud Video Streaming services to enable smooth and high-quality video playback.

To enable on-demand playback and allow users to upload their movies and videos, along with integrating IBM Cloud Video Streaming services for smooth playback, you'd typically involve both frontend and backend components. Here's a simplified guide:

Frontend Integration for Uploading Videos:

HTML (Upload Form):

```
<!DOCTYPE html>
<html>
<head>
  <title>Video Upload</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      background-color: #f4f4f4;
      margin: 0;
      padding: 0;
```

```
  display: flex;
  justify-content: center;
  align-items: center;
  height: 100vh;
}
.upload-container {
  background-color: #fff;
  padding: 30px;
  border-radius: 8px;
  box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}
.upload-container h2 {
  margin-top: 0;
  text-align: center;
}
.upload-form {
  display: flex;
  flex-direction: column;
  align-items: center;
}
.upload-form input[type="file"] {
```

```
      margin-bottom: 20px;
    }
    .upload-form button {
      padding: 10px 20px;
      background-color: #3498db;
      color: #fff;
      border: none;
      border-radius: 4px;
      cursor: pointer;
      transition: background-color 0.3s;
    }
    .upload-form button:hover {
      background-color: #2980b9;
    }
  </style>
</head>
<body>
  <div class="upload-container">
    <h2>Upload Your Video</h2>
    <form class="upload-form" id="videoUploadForm"
enctype="multipart/form-data">
      <input type="file" id="videoFile" accept="video/*" required>
```

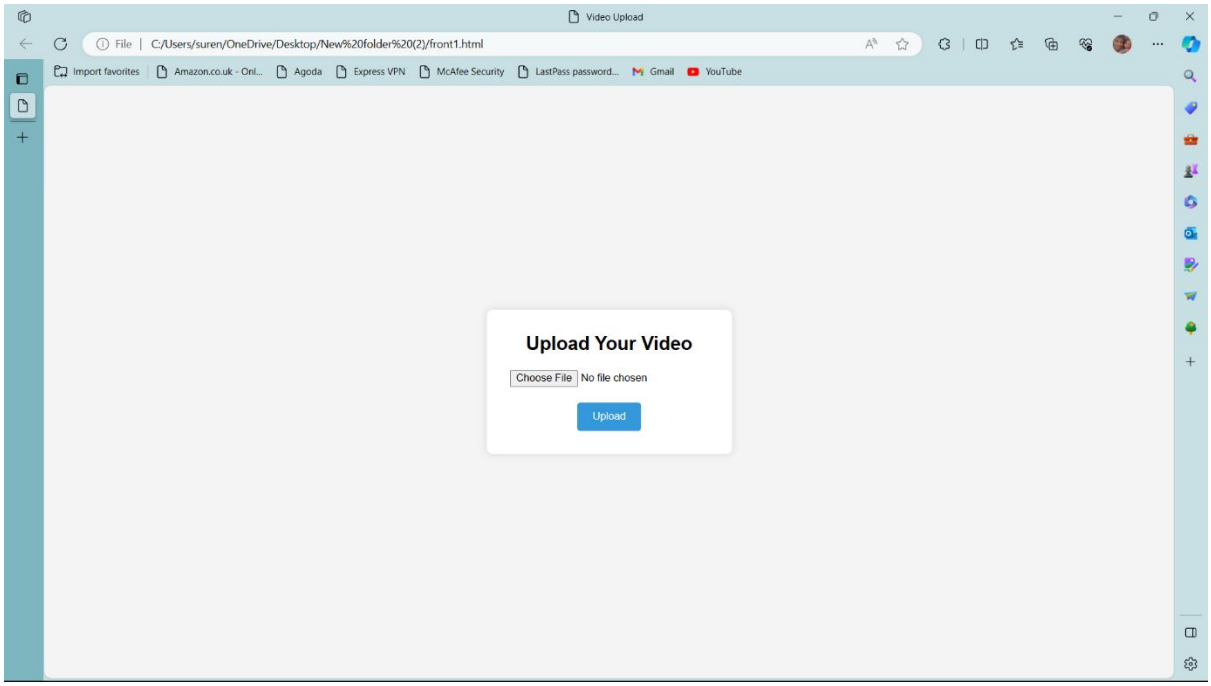
```
    <button type="submit">Upload</button>
  </form>
 </div>

 <script>

document.getElementById('videoUploadForm').addEventListener('submit', function(event) {
    event.preventDefault();
    const videoFile = document.getElementById('videoFile').files[0];

    // Simulate video upload by logging file details
    console.log('Uploaded Video:', videoFile);

    // Additional steps: Use Fetch API or other means to send the video file to the backend
    // For this example, we simulate file upload and log the file details.
  });
 </script>
</body>
</html>
```

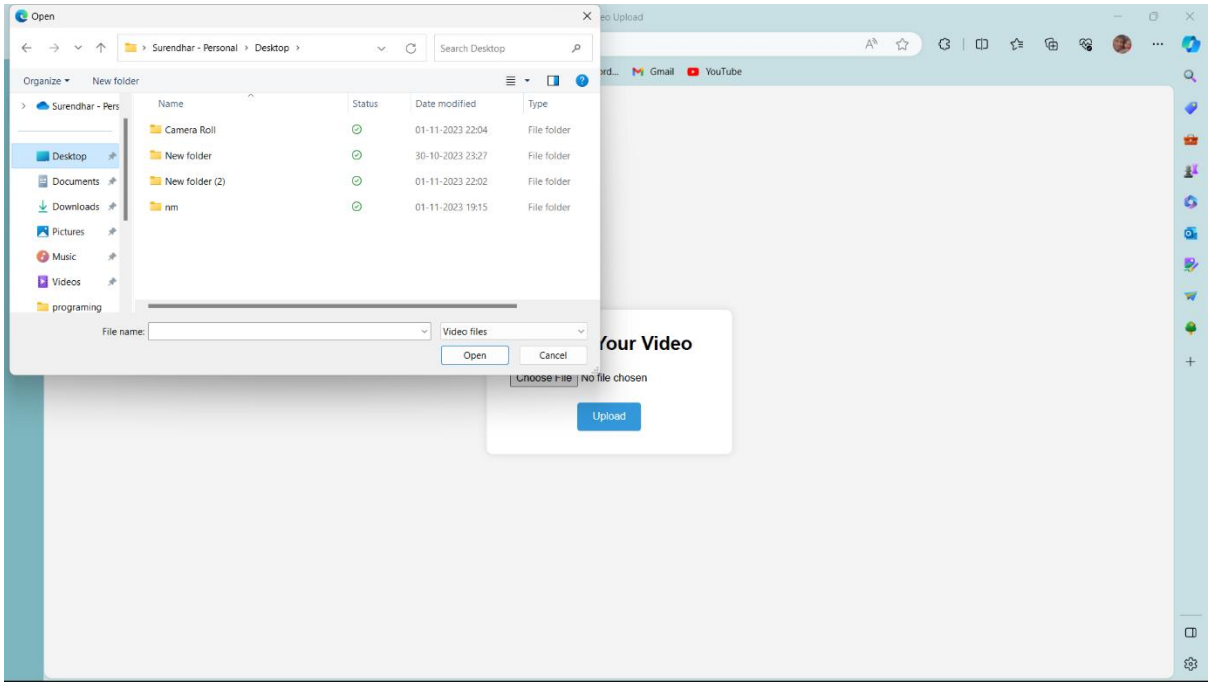
Backend Integration for Video Storage and IBM Cloud Video Streaming:

1. Backend Server: Set up a backend server (using Node.js, Python, etc.) to handle file uploads, process the videos, and store them securely. Use a framework like Express.js (for Node.js) to create endpoints for handling file uploads.
2. IBM Cloud Video Integration: Use IBM Cloud Video Streaming API to manage and deliver videos. The IBM Cloud Video API allows you to create channels, upload videos, manage video metadata, and retrieve playback URLs.
3. Steps for Backend:
    - Receive uploaded video files from the frontend.
    - Store these files securely (consider using a cloud storage service like IBM Cloud Object Storage).
    - Utilize IBM Cloud Video API to create video assets, upload videos, and generate playback URLs.

```
const express = require('express');

const multer = require('multer');

const ibmVideoService = require('ibm-video-service-library'); // Import the library for IBM Video Service


const app = express();

const upload = multer({ dest: 'uploads/' });


// Endpoint to handle video uploads

app.post('/upload-video', upload.single('video'), async (req, res) => {

  const videoFile = req.file;
```

```
  // Handle storing the file securely and then use IBM Video API to
upload the video
  const ibmVideo = new ibmVideoService();
  const videoAsset = await ibmVideo.uploadVideo(videoFile.path);


  const playbackURL = videoAsset.playbackURL;


  res.json({ playbackURL });
});

app.listen(3000, () => {
  console.log('Server is running on port 3000');
});
```




Open
Surendhar - Personal › Desktop ›
Search Desktop
Organize
New folder
Surendhar - Pers
Desktop
Documents
Downloads
Pictures
Music
Videos
programing
Name
Status
Date modified
Type
Camera Roll
01-11-2023 22:04
File folder
New folder
30-10-2023 23:27
File folder
New folder (2)
01-11-2023 22:02
File folder
nm
01-11-2023 19:15
File folder
File name:
Video files
Open
Cancel
Gmail
YouTube
Your Video
Choose File
No file chosen
Upload