**CLOUD APPLICATION DEVELOPMENT (GROUP 1)**

PHASE 4 : ASSIGNMENT NOTEBOOK SUBMISSION

**NAME :** AJITH. B

**EMAIL :** ajithkumara91033@gmail.com

**GitHub Repository URL :** https://GitHub.com/Ajith8111/democodeExample.git

**PROJECT OF THE TITLE :**

Media streaming with IBM cloud video streaming

## DEVELOPMENT :

Continuing to build your platform with integrated video streaming services for on-demand playback involves several steps to ensure a seamless user experience and efficient content delivery. Here are the next steps:

### 1.Content Encoding and Transcoding:

Video files come in various formats and qualities. To ensure smooth playback across different devices and network conditions, it's essential to transcode your video content into multiple bitrates and formats (e.g., HLS, DASH). Many video streaming services offer built-in encoding

and transcoding capabilities. Use these services to prepare your video content for adaptive streaming.

**2.Video Asset Management:**

Implement a system for managing video assets. This includes metadata storage, categorization, and indexing. Create a database to keep track of video assets, their titles, descriptions, durations, and other relevant information.

**3.User Authentication and Authorization:**

Implement user authentication to restrict access to certain videos or features. This ensures that only authorized users can view specific content. You can also manage user roles and permissions to control who can upload, edit, or delete videos.

**4.Video Player Integration:**

Choose a video player library or service for on-demand video playback on your platform. Popular options include Video.js, Plyr, or JWPlayer. Integrate the selected video player into your frontend. Configure it to work with adaptive streaming formats and ensure that it can retrieve video content from your storage.

**5.Adaptive Streaming:**

Configure your video player to support adaptive streaming. This technology adjusts the video quality in real-time based on the user's network conditions and device capabilities. Implement adaptive streaming protocols like HLS or DASH.

**6.Content Delivery Network (CDN) Integration:**

Leverage a CDN to deliver video content efficiently to users around the world. CDNs cache your videos on servers located in various geographic regions, reducing latency and ensuring faster video loading. Popular CDNs include Akamai, Cloudflare, and AWS CloudFront.

**7.Streaming Analytics and Metrics:**

Implement analytics tools to monitor video performance and user engagement. Track metrics such as video views, drop-off rates, and user interactions. Services like Google Analytics, Mixpanel, or custom analytics solutions can help with this.

**8.Search and Recommendation Engine:**

Implement a search and recommendation system to help users discover content. Use metadata and user behavior data to suggest related videos or create personalized recommendations.

**9.Video Thumbnails and Previews:**

Enhance the user experience with video thumbnails and previews. Create thumbnail images for each video and enable video previews when users hover over video thumbnails.

**10.Content Management and Moderation:**

Implement a content management system to allow content administrators to review and moderate uploaded videos. This helps maintain content quality and ensure compliance with your platform's policies.

**11.Monetization (Optional):**

If your platform includes premium content, integrate monetization features. Implement subscription models, pay-per-view options, or advertising solutions to generate revenue from your videos.

**12.Mobile Apps and Responsive Design:**

Ensure your platform is accessible on both desktop and mobile devices. Develop mobile apps for iOS and Android if necessary.

**13.Testing and Quality Assurance:**

Thoroughly test your platform to ensure video playback is smooth and error-free across various devices, browsers, and network conditions.

**14.Scalability and Performance Optimization:**

As your user base grows, ensure your platform can scale horizontally to handle increased traffic. Optimize performance by minimizing server response times and leveraging caching mechanisms.

**15.Legal Considerations:**

Be aware of copyright and licensing issues related to video content. Implement copyright infringement reporting and takedown procedures if necessary.

**16.User Engagement and Community Building:**

Encourage user interaction through comments, likes, shares, and social media integration. Build a community around your platform.

**17.User Support and Feedback:**

Offer user support and feedback channels to address issues and improve the user experience continuously.

Remember to keep security a top priority throughout the development process. Secure video access, user data, and the platform as a whole to protect against potential threats and breaches.

## VIDEO UPLOAD :

Creating a platform for users to upload movies and videos involves several steps, including setting up a server, implementing a user interface, and handling the actual video upload and storage. Below, I'll outline a simplified web-based solution using popular technologies like Node.js, Express, and MongoDB for backend, and HTML and JavaScript for the frontend. Please note that this is a high-level overview, and real-world implementations may require more features and security considerations.

Backend (Node.js with Express and MongoDB):

### 1.Setting Up Dependencies:

Start by initializing a Node.js project, installing necessary dependencies, and setting up your MongoDB database.

```bash npm init npm install express
mongoose multer
```

### 2.Create Server:

Create an Express server and set up routes to handle file uploads.

```javascript const express =
require('express'); const mongoose =
require('mongoose'); const multer =
require('multer'); const app = express();

// Connect to MongoDB mongoose.connect('mongodb://localhost/video_platform',
{ useNewUrlParser: true,
useUnifiedTopology: true });

// Define a Schema and Model for Video const
videoSchema = new mongoose.Schema({ title:
String, description: String, videoUrl: String,
});
const Video = mongoose.model('Video', videoSchema);

// Configure multer to handle file uploads
const storage = multer.diskStorage({
  destination: (req, file, cb) => {
    cb(null, 'uploads/'); // Folder to store uploaded videos
```

```
  },
  filename: (req, file, cb) => {
    cb(null, Date.now() + '-' + file.originalname);
  },
});
const upload = multer({ storage: storage });

// Create an API endpoint for video uploads
app.post('/upload', upload.single('video'), (req, res) => {
  const { title, description } = req.body; const videoUrl =
  req.file.path; const video = new Video({ title,
  description, videoUrl });

  video.save((err) => {
    if (err) {
      return res.status(500).send(err);
    }
    return res.status(200).send('Video uploaded successfully!');
  });
});

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

### 3.Frontend (HTML & JavaScript):

Create a simple HTML form for users to upload videos.

```html
<!DOCTYPE html>
<html>
<head>
  <title>Video Upload</title>
</head>
<body>
  <h1>Upload Your Video</h1>
  <form action="/upload" method="POST" enctype="multipart/form-data">
    <label for="title">Title:</label>
```

```
    <input type="text" name="title" required><br><br>
    <label for="description">Description:</label>
    <textarea name="description"></textarea><br><br>
    <label for="video">Choose a Video:</label>
    <input type="file" name="video" accept="video/*" required><br><br>
    <input type="submit" value="Upload">
  </form>
</body>
</html>
```

### 4.Storing and Retrieving Videos:

Extend the backend to store video metadata in your MongoDB database and create routes to retrieve and display videos on your platform.

### 5.User Authentication and Authorization:

Implement user authentication and authorization to ensure only authorized users can upload and manage videos.

### 6.Security Considerations:

Make sure to validate and sanitize user input, handle file type and size restrictions, and protect against common security issues like Cross-Site Request Forgery (CSRF).

### 7.Scalability and Performance:

Consider using a content delivery network (CDN) for video delivery to ensure optimal performance.

### 8.User Experience:

Enhance the user experience with features like video previews, thumbnail generation, and video categorization.

This is a basic outline to get you started. Depending on your platform's requirements, you may need to add more features, such as video processing, user profiles, comments, and likes. Additionally, it's crucial to ensure data privacy and security, especially when handling user-generated content.

## INTEGRATING :

Integrating IBM Cloud Video Streaming services into your platform can help ensure smooth and high-quality video playback. IBM Video Streaming services provide a robust infrastructure for delivering video content to users. Below, I'll outline the steps to integrate IBM Cloud Video Streaming services into your platform:

### 1.Create an IBM Cloud Account:

If you don't already have one, sign up for an IBM Cloud account at [https://cloud.ibm.com/](https://cloud.ibm.com/).

### 2.Provision the IBM Video Streaming Service:

- Log in to your IBM Cloud account.
- Navigate to the IBM Cloud Catalog and find the "Video Streaming" service.
- Click on the service and follow the instructions to provision it. You may need to select a plan based on your usage.

### 3.Get API Credentials:

- Once the service is provisioned, you'll need API credentials to interact with the service programmatically. You can find these credentials in the IBM Cloud Dashboard.
- Note down the API Key, Service Instance ID, and Endpoint URL.

### 4.Integrate with the Backend (Node.js):

In your Node.js backend, use the IBM Cloud Video Streaming SDK to interact with the service. You can use the `ibm-watson` Node.js SDK to connect to IBM services. Install it using npm:

```bash
npm install ibm-watson
```

Here's an example of how to use the SDK to create a video asset:

```javascript const IBMCloudVideo = require('ibm-
watson/video');

const videoClient = new IBMCloudVideo({
  apiKey: 'YOUR_API_KEY', serviceInstanceID:
  'YOUR_SERVICE_INSTANCE_ID', url:
  'https://YOUR_SERVICE_INSTANCE_URL',
});
```

```
// Create a video asset const
createAssetParams = {
  asset: 'asset-name',
  type: 'asset-type',
};

videoClient.createAsset(createAssetParams)
  .then(response => { console.log('Created video asset:',
    JSON.stringify(response, null, 2));
  })
  .catch(error => { console.error('Error creating
    video asset:', error);
  });
```

Make sure to replace `'YOUR_API_KEY'`, `'YOUR_SERVICE_INSTANCE_ID'`, and `'YOUR_SERVICE_INSTANCE_URL'` with the actual credentials and URL.

### 5.Upload Videos to IBM Cloud Video Streaming:

Use the IBM Cloud Video Streaming SDK to upload your video content to the platform. You can upload video files to your created assets. You may also want to use the `ibm-watson` SDK for this purpose.

### 6.Configure Video Playback:

Use the video assets' information provided by IBM Cloud Video Streaming to configure video playback on your platform. You can embed videos using HTML5 video players and set up adaptive streaming for different resolutions and devices.

### 7. Security and Access Control:

IBM Cloud Video Streaming services offer security features like token-based access and encryption. Implement these features to protect your videos from unauthorized access.

### 8.Monitoring and Analytics:

IBM Cloud Video Streaming provides monitoring and analytics tools to track the performance of your videos. Set up monitoring to ensure the quality of video playback.

### 9.Scalability and Content Delivery:

IBM Cloud services often include content delivery networks (CDNs) for efficient content delivery. Make sure to configure the CDN settings to optimize video delivery to users globally.

### 10.User Interface:

Enhance the user interface of your platform to provide a seamless video-watching experience, including features like video playlists, recommendations, and user engagement analytics.

Remember that the specific implementation details can vary depending on your application's technology stack and requirements. IBM Cloud Video Streaming services offer flexibility and scalability, making it a powerful choice for delivering high-quality video content.

**GitHub Repository URL:** https://GitHub.com/Ajith8111/democodeExample.git